Lecture Notes for STAT 574

DECISION TREES

A **decision tree** is a type of flowchart that shows a clear pathway to a decision. It starts at a single point (or **root node**) which then branches (or **splits**) in two or more directions. Each branch offers different possible outcomes until a final outcome is achieved. When plotted as a graph, it resembles a tree. The nodes at the end of the decision path are called **leaf nodes** (or **terminal nodes**). Between the root node and the leaf nodes, there are a number of **internal nodes** (or **decision nodes**).

Decision trees work for both categorical and numerical variables. Their objective is to partition the population into homogeneous non-overlapping sets, based on the most significant **input** (or **explanatory** or **predictor**) **variables**. The following two types of trees are commonly used in practice: **regression tree** (for a continuous **target variable**), and **classification tree**, for the categorical target variable.

Classification and Regression Trees

Different rules apply to "grow" regression and classification trees. For regression trees, residual sum of squares (RSS) and chi-squared automatic interaction detection (CHAID) splitting criteria are used. For classification trees, entropy, Gini, and CHAID criteria are used.

Sometimes created regression and classification trees are too complex, with redundant splits, and a very small number of cases in leaf nodes. In this case **pruning** of the tree is advisable, which results in a smaller number of leaves with more substantial splitting and easier interpretation. The cost-complexity pruning technique is commonly used.

To compare the performance of fitted regression trees and choose the best-performing tree, it is customary to split randomly the original data set into a training set (containing typically 70% or 80% of the data rows) and a testing set (containing the remaining 30% or 20% of the rows). A decision tree is developed on the training set, and the goodness-of-fit is verified on the testing set. For regression trees (when the response is continuous), the response is predicted on the testing data set, and proportions of predicted values that are within, say, 10%, 15%, and 20% of the true values are computed and compared for different models. The model for which these proportions are the highest is the best predicting model. For classification trees, prediction is in the form of the probability of being in each category. We can assume that the category with the highest probability is the one that is being predicted, and the measure of performance is the proportion of predictions that coincide with the true observations in the testing data set. The model with the highest proportion of correctly predicted categories should be chosen, or, equivalently, the model with the smallest **misclassification rate** should be chosen.

Regression Tree

The RSS Splitting Criterion

For Residual-Sum-of-Squares (RSS) splitting criterion, the predictor space is partitioned into multi-dimensional rectangles $R_1, \ldots, R_J$ that minimize the **residual sum of squares** (RSS) $\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$ where y_i is the observed target variable for individual i, $i = 1, \ldots, n$, and $\hat{y}_{R_j}$ is the estimated average response for all observations in the set R_j. In the tree-building process (also termed **training a decision tree**), a top-down binary splitting is used to obtain two new branches at each decision node. For a predictor variable X, the split $X = s$ is chosen in such a way that it leads to the largest reduction of RSS for the two subregions $\{X < s\}$ and $\{X \geq s\}$.

Remark. It is easy to see that the RSS is minimized at the mean value. Indeed, to minimize $\sum_{i=1}^{n} (y_i - \hat{y})^2$, we take the derivative with respect to $\hat{y}$ and set it equal to zero. We get $-2\sum_{i=1}^{n} (y_i - \hat{y}) = 0$ or $\sum_{i=1}^{n} y_i - n\hat{y} = 0$. From here, $\hat{y} = \frac{1}{n}\sum_{i=1}^{n} y_i = \bar{y}$. $\square$

Remark. The splitting procedure based on minimizing the RSS is an example of the **classification and regression tree (CART)** algorithm. More generally, this algorithm involves splitting based on the minimization of some quantity. Note that in the CART algorithm, only binary splitting is allowed.

Historical Note. The CART algorithm was introduced in the book "Classification and Regression Trees" by Breiman, L., Friedman, J., Olshen, R. and Stone, C, Chapman and Hall, Wadsworth, New York, 1984.

The CHAID Splitting Criterion

The chi-squared automatic interaction detection (CHAID) splitting criterion is based on statistical tests. To grow a tree, the response variable can be either continuous of categorical but the predictor variables must be all categorical only (that is, continuous predictors are automatically segmented).

While training a tree, the next best split is determined by an F-test for the continuous response variable and by a chi-squared test for the categorical response variable. The predictor variable with the smallest (Bonferroni adjusted) p-value, i.e., the predictor variable that will yield the most significant split will be considered for the next split in the tree. If the smallest p-value for any predictor is greater than some pre-specified value of alpha, then no further splits will be performed, and the respective node will become a terminal node.

The Bonferroni Adjusted P-value

The Bonferroni correction helps to protect against inflation of the probability of Type I error when performing multiple simultaneous hypotheses testing (it, however, inflates the probability of Type II error). Suppose we would like to test simultaneously k pairs of hypotheses and maintain the probability of Type I error equal to α. The probability of Type I error is then the probability that at least one test shows significance when no significance exists. That is, at least one test statistic falls in the (adjusted) rejection region. The complement of that is that all test statistics lie in the (adjusted) acceptance region, and this translates into the identity: $1 - \alpha = \mathbb{P}$(all test statistics are in the (adjusted) acceptance region) $= (1 - \alpha_{adjusted})^k$, assuming that tests are independent. From here, the Bonferonni adjusted p-value is a p-value that should be compared to $\alpha_{adjusted} = 1 - (1 - \alpha)^{(1/k)} \approx \alpha/k$. For example, if $k = 5$ and five simultaneous hypothesis testings are carried out, then each p-value should be compared not to $\alpha = 0.05$ but $\alpha/k = 0.05/5 = 0.01$.

Remark. Note that with the CHAID splitting criterion, each internal node can have more than two emanating branches, that is, a tree is not limited to binary splitting.

Historical Note. The CHAID splitting criterion was proposed by G.V Kass in "An Exploratory Technique for Investigating Large Quantities of Categorical Data", Journal of the Royal Statistical Society, Series C (Applied Statistics), Vol. 29, No. 2 (1980), pp. 119-127.

The Cost-complexity Pruning Technique

For a decision tree, the **complexity of the tree** T is defined as $|T|$, the number of terminal nodes (leaves). The **cost** of a decision tree, denoted by $R(T)$, is the RSS for regression trees and the misclassification rate for classification trees. The **cost-complexity (CC) measure** is defined as $CC(T) = R(T) + \alpha \cdot |T|$, a linear combination of the cost and complexity, where the term $\alpha \cdot |T|$ is called the **complexity penalty term**. The parameter α is known as the **cost-complexity parameter (ccp)**.

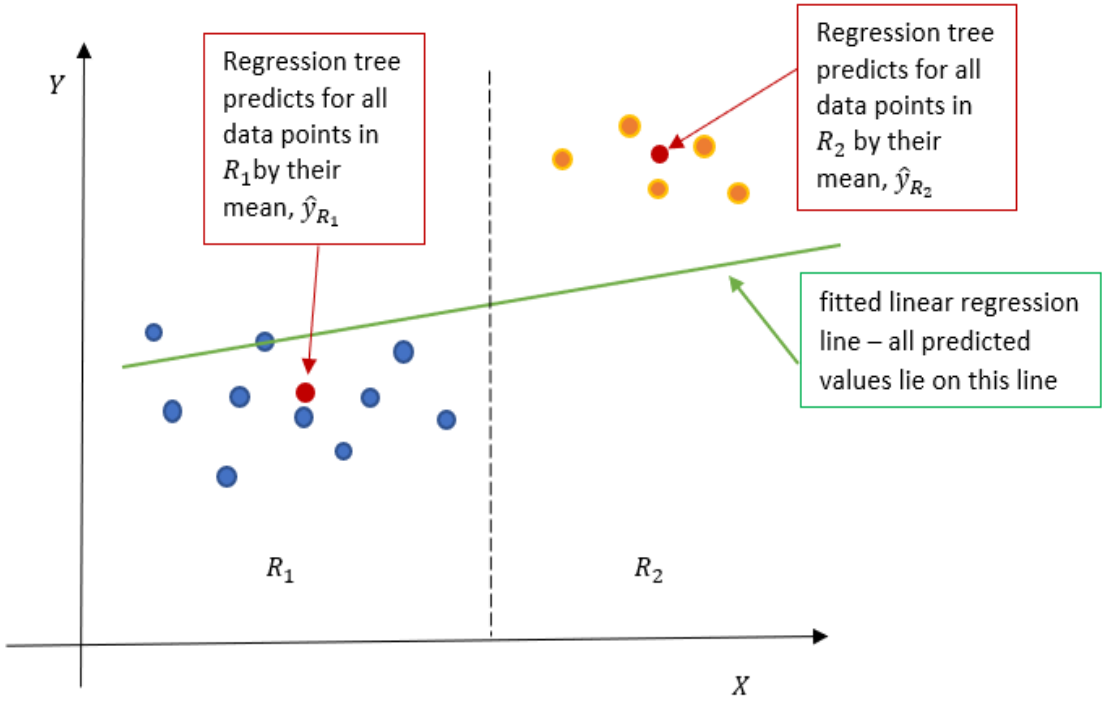
The cost-complexity pruning algorithm (also known as the **minimal cost-complexity** pruning algorithm) works as follows. The cost-complexity of a single node is $CC(t) = R(t) + \alpha$. The branch T_t is defined to be a tree with t as the root node. In general, the cost of a node is greater than the sum of the costs of its terminal nodes, that is, $R(T_t) < R(t)$. However, the cost-complexity measure of a node t and its branch T_t can be made equal, depending on the value of α. We define the **effective** α of a node to be the value where $CC(T_t) = CC(t)$, or $R(T_t) + \alpha_{eff} |T_t| = R(t) + \alpha_{eff}$, giving

$$\alpha_{eff} = \frac{R(t) - R(T_t)}{|T_t| - 1}.$$

An internal node with the smallest value of α_eff is the **weakest link** and will be pruned. The pruning process continues until there are no more effective α's smaller than a pre-specified value. More about this pruning algorithm will be explained in the example below.

Historical Note. The cost-complexity pruning algorithm was first described in the book "Classification and Regression Trees" by Breiman, L., Friedman, J., Olshen, R. and Stone, C, Chapman and Hall, Wadsworth, New York, 1984.

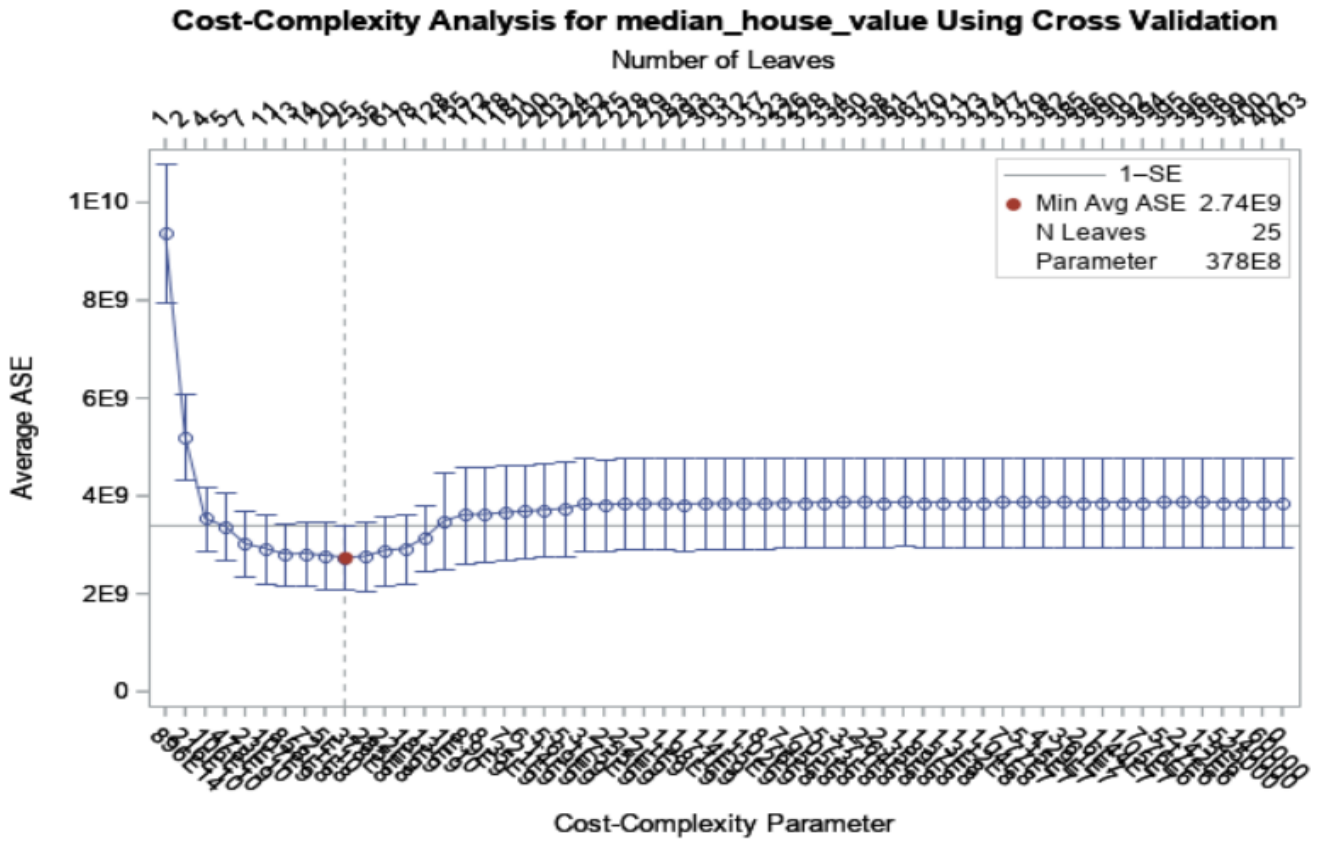
Remark. When data are not distributed in a linear pattern, fitting a regression tree has a clear advantage over a linear regression model as illustrated below.



Example. The data set "housing_data.csv" contains variables aggregated by residential neighborhoods: housing median age, total number of rooms, total number of bedrooms, total population, total number of households, median income, ocean proximity, and median house value. There are 2842 rows in this data set. We model median house value using a regression tree with the RSS splitting and cost-complexity pruning. First, we split the data into 80% training and 20% testing sets. Then we run a full regression tree (without pruning). We use the RSS splitting criterion. Note that since the tree is large, we need to specify a seed. The RSS splitting algorithm is deterministic, but if two splits are equivalent, the order of splits is carried out at random.

In SAS:

```
proc import out=housing datafile="./housing_data.csv" dbms=csv replace;
run;

/*SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS*/
proc surveyselect data=housing rate=0.8 seed=677530
out=housing outall method=srs;
run;

/*RSS SPLITTING CRITERION - FULL TREE*/
proc hpsplit data=housing seed=304576;
class ocean_proximity;
model median_house_value = housing_median_age total_rooms total_bedrooms
population households median_income ocean_proximity;
grow RSS;
partition rolevar=selected(train="1");
run;
```

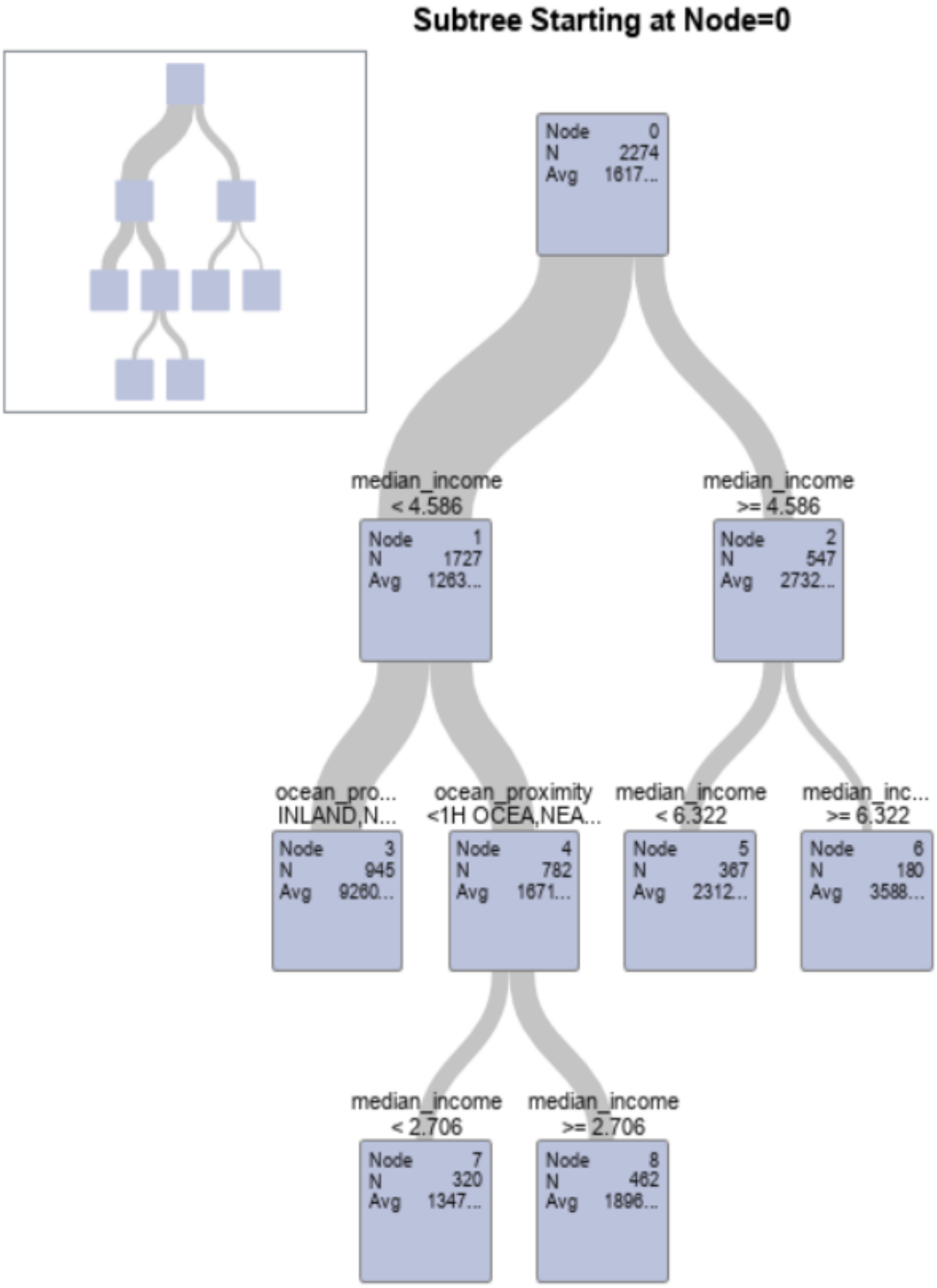


As part of the output, SAS produces a cost-complexity analysis plot. In this plot, the Average ASE (average squared error= RSS/n where n is the sample size of the training set) is plotted against the number of leaves in a tree (top scale on the x-axis). The additional scale on the x-axis (at the bottom) is for the cost-complexity parameter. To compute Average ASE, SAS does (by default) **10-fold cross-validation** by randomly dividing the training set into 10 equal parts and fitting the tree iteratively 10 times, every time holding 1/10th of the data as a **validation set**. ASE is computed for each of the 10 validation sets and then averaged.

As suggested by the plot, the global minimum corresponds to 25 leaf nodes, but this is obviously a very large tree (typically with very poor predictive accuracy). As suggested by Breiman, et. al (1984), it is more reasonable to prune a tree to the number of leaves corresponding to the smallest value of Average ASE below one standard error. The standard error is a standard deviation of the 10 ASE values computed during the cross-validation process. On the plot, the 1-SE line is the horizontal line, and the first value below it corresponds to 5 leaves. Next, we fit a pruned tree with 5 leaf nodes, using the cost-complexity pruning method.

```
/*RSS SPLITTING AND COST-COMPLEXITY PRUNING*/
proc hpsplit data=housing;
class ocean_proximity;
model median_house_value = housing_median_age total_rooms total_bedrooms
population households median_income ocean_proximity;
grow RSS;
prune costcomplexity(leaves=5);
partition rolevar=selected(train="1");
output out=predicted;
ID selected;
run;
```

Studying the tree on the next page, we can see that all the splits are done on two predictors only. The first split is done with respect to the median income (<4.586, 1727 rows vs. >=4.586, 547), then the latter node is split on the median income again (<6.322, 367 rows vs. >=6.322, 180 rows), and the former node is split on ocean proximity (inland and near ocean, 945 rows vs. others, 782 rows). The latter node is also split on median income (<2.706, 320 rows vs. >=2.706, 462 rows). There are 5 leaf nodes in this pruned tree.


Subtree Starting at Node=0
Node 0
N 2274
Avg 1617...
median_income
< 4.586
median_income
>= 4.586
Node 1
N 1727
Avg 1263...
Node 2
N 547
Avg 2732...
ocean_pro...
INLAND,N...
ocean_proximity
<1H OCEA,NEA...
median_income
< 6.322
median_inc...
>= 6.322
Node 3
N 945
Avg 9260...
Node 4
N 782
Avg 1671...
Node 5
N 367
Avg 2312...
Node 6
N 180
Avg 3588...
median_income
< 2.706
median_income
>= 2.706
Node 7
N 320
Avg 1347...
Node 8
N 462
Avg 1896...

Further, the data set "predicted" contains predicted responses for both training and testing data (identified by the variable "selected"). We limit the data to the testing set and compute proportions of predictions within 10%, 15%, and 20% of the observed values.

```
/*COMPUTING PREDICTION ACCURACY FOR TESTING DATA*/
data test;
set predicted;
if(selected="0");
keep _leaf_ median_house_value P_median_house_value;
run;

data accuracy;
set test;
if(abs(median_house_value-P_median_house_value)<0.10*median_house_value)
then ind10=1; else ind10=0;
if(abs(median_house_value-P_median_house_value)<0.15*median_house_value)
then ind15=1; else ind15=0;
if(abs(median_house_value-P_median_house_value)<0.20*median_house_value)
then ind20=1; else ind20=0;
run;

proc sql;
 select mean(ind10) as accuracy10, mean(ind15) as accuracy15,
 mean(ind20) as accuracy20
 from accuracy;
 quit;
```

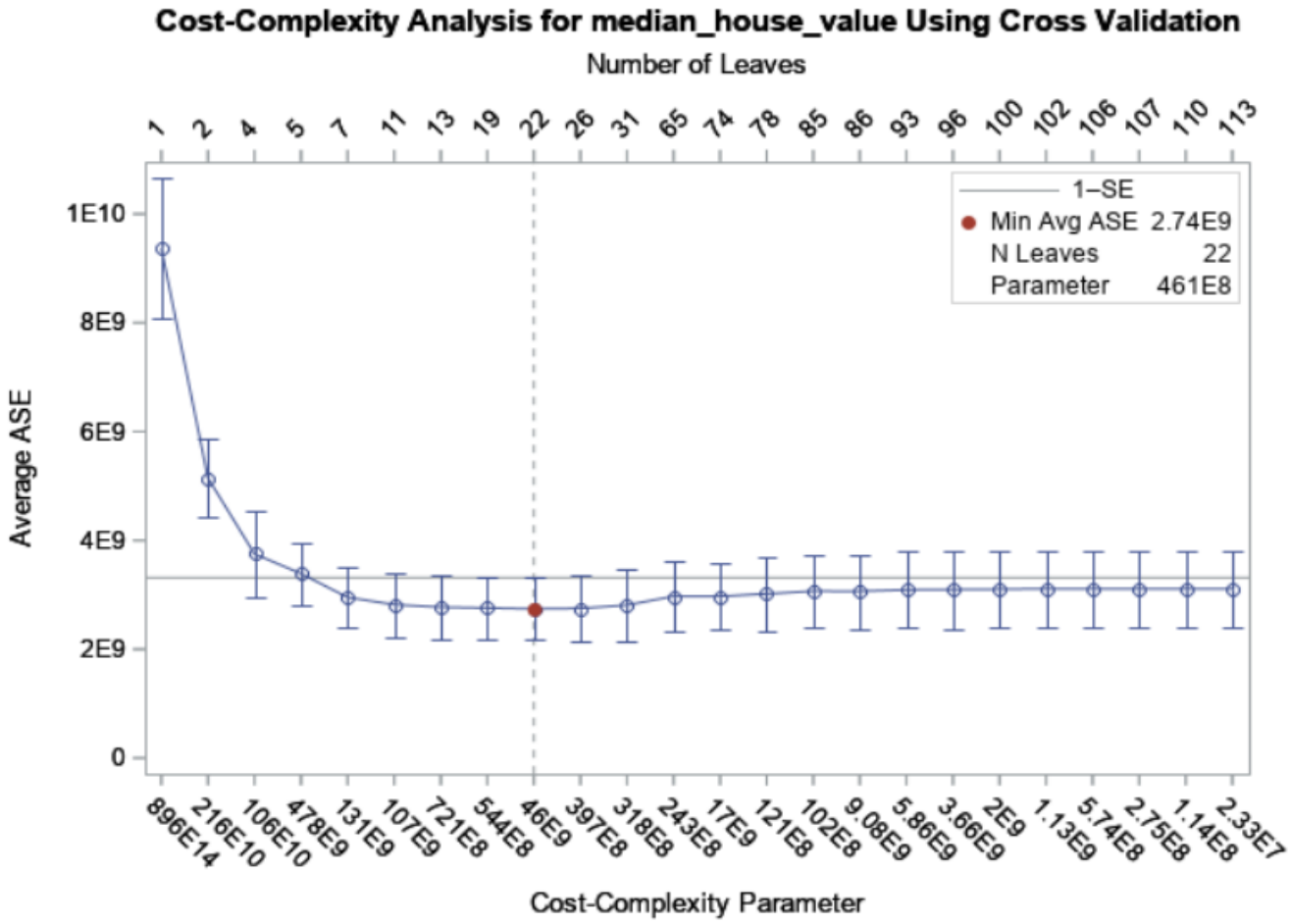
accuracy10	accuracy15	accuracy20
0.257042	0.362676	0.471831

From here, we can see that roughly 27.5% of predictions are within 10%, 36.3% are within 15%, and 47.2% are within 20% of the true observed values in the testing data set.

Now we grow a full regression tree using the CHAID splitting criterion. Note that a seed has to be specified because the algorithm involves random segmentation.

```
/*CHAID SPLITTING CRITERION - FULL TREE*/
```

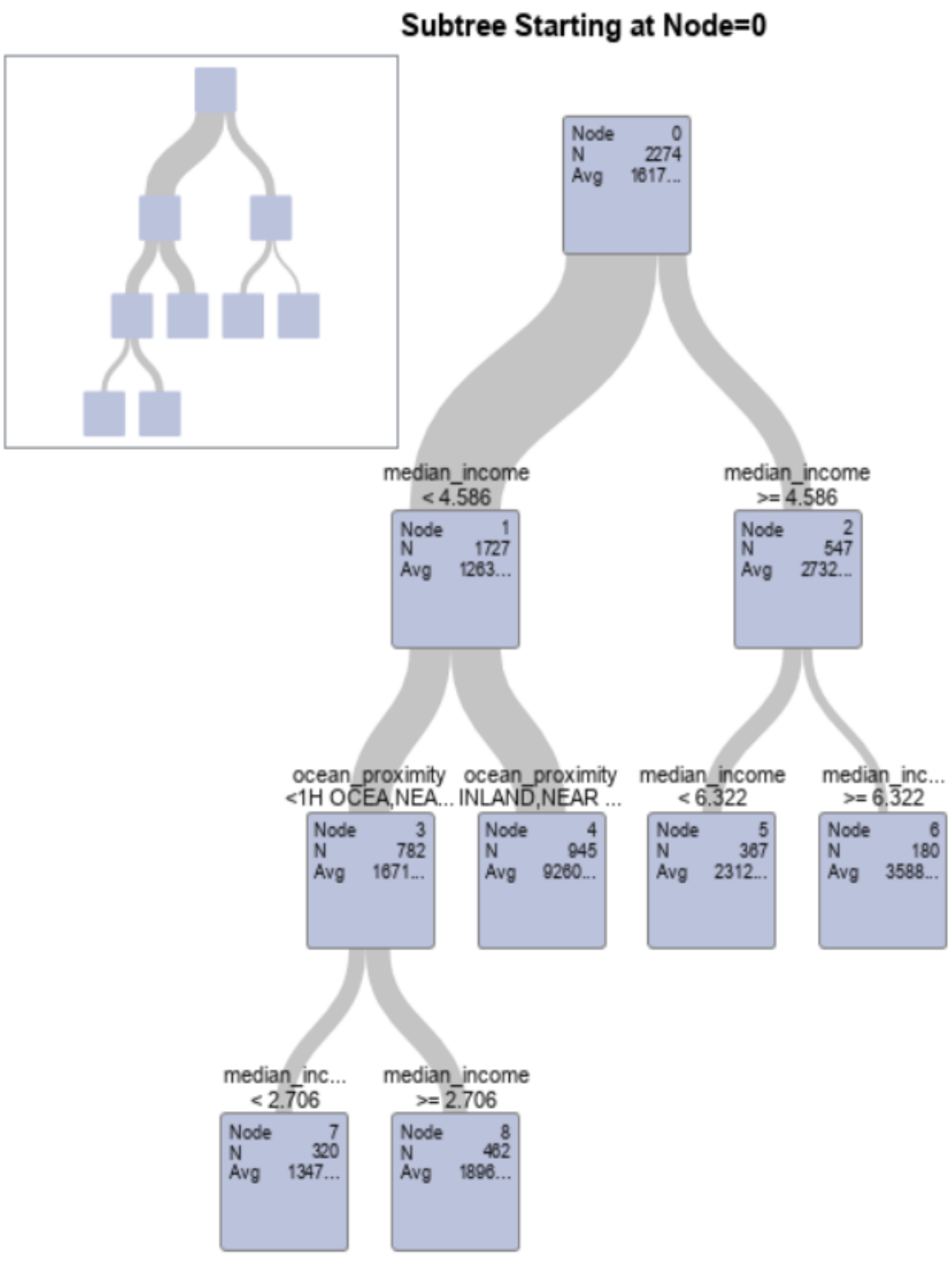
```
proc hpsplit data=housing seed=501231;
class ocean_proximity;
model median_house_value = housing_median_age total_rooms total_bedrooms
population households median_income ocean_proximity;
grow CHAID;
partition rolevar=selected(train="1");
run;
```



As suggested by the graph, the number of leaves in the pruned tree is 5 (on the 1-SE line). We run the following statement and obtain exactly the same regression tree as with the RSS splitting criterion.

```
/*CHAID SPLITTING AND COST-COMPLEXITY PRUNING */
proc hpsplit data=housing seed=501231;
```

```
class ocean_proximity;
model median_house_value = housing_median_age total_rooms total_bedrooms
population households median_income ocean_proximity;
grow CHAID;
prune costcomplexity (leaves=5);
partition rolevar=selected(train="1");
output out=predicted;
ID selected;
run;
```


Subtree Starting at Node=0
Node 0
N 2274
Avg 1617...
median_income
< 4.586
median_income
>= 4.586
Node 1
N 1727
Avg 1263...
Node 2
N 547
Avg 2732...
ocean_proximity
<1H OCEA,NEA...
ocean_proximity
INLAND,NEAR ...
median_income
< 6.322
median_inc...
>= 6.322
Node 3
N 782
Avg 1671...
Node 4
N 945
Avg 9260...
Node 5
N 367
Avg 2312...
Node 6
N 180
Avg 3588...
median_inc...
< 2.706
median_income
>= 2.706
Node 7
N 320
Avg 1347...
Node 8
N 462
Avg 1896...

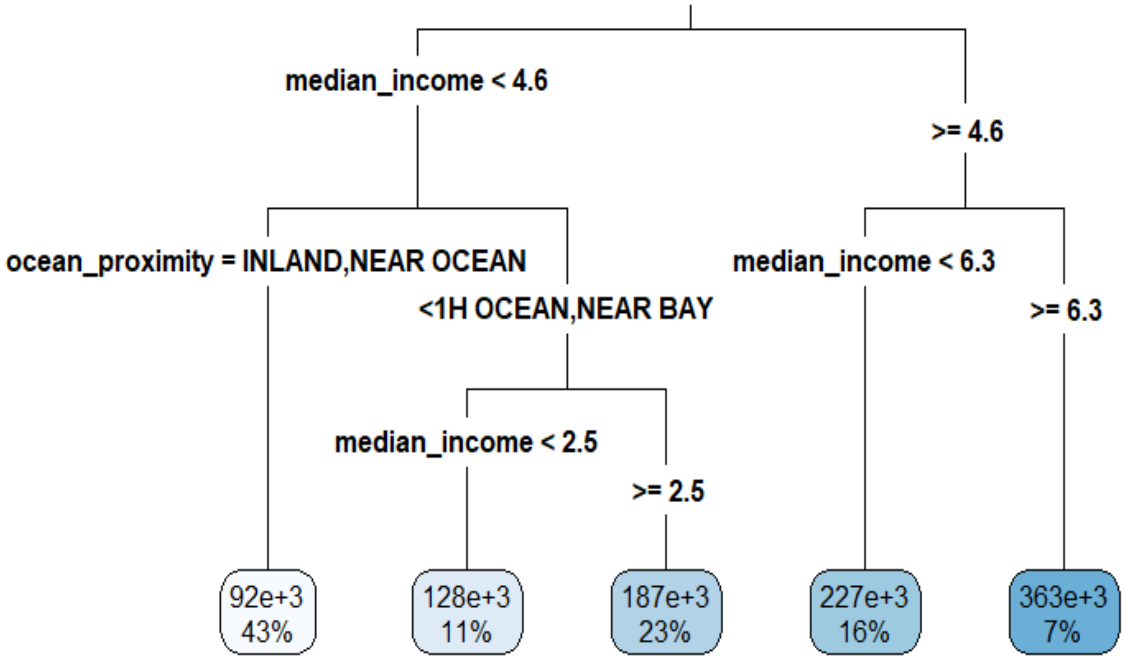
Next, we fit the regression tree with the RSS splitting criteria, using R. We prune the tree to 5 leaves.

In R:

```
housing.data<- read.csv(file="./housing_data.csv", header=TRUE, sep=",")

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
set.seed(105388)
sample <- sample(c(TRUE, FALSE), nrow(housing.data), replace=TRUE, prob=c(0.8,0.2))
train<- housing.data[sample,]
test<- housing.data[!sample,]

#FITTING FULL REGRESSION TREE WITH RSS SPLITTING
install.packages("rpart")
library(rpart) #recursive partitioning and regression
reg.tree.full<- rpart(median_house_value ~ housing_median_age + total_rooms + total_bedrooms
+ population + households + median_income + ocean_proximity, data=train, method="anova",
xval=10, cp=0) #xval=number of cross-validations

printcp(reg.tree.full)
```

```
            CP nsplit rel error  xerror     xstd
1   4.0377e-01      0   1.00000 1.00114 0.038634
2   1.1710e-01      1   0.59623 0.61587 0.024985
3   1.0159e-01      2   0.47913 0.48931 0.022913
4   2.8290e-02      3   0.37754 0.39223 0.019864
5   2.5166e-02      4   0.34925 0.37005 0.020814
6   1.5738e-02      5   0.32408 0.34350 0.020419
7   1.4741e-02      6   0.30834 0.32689 0.020259
8   7.0612e-03      7   0.29360 0.31084 0.019927
9   4.4519e-03      8   0.28654 0.30528 0.019780
10  4.3618e-03      9   0.28209 0.30798 0.020052
```

```
#FITTING REGRESSION TREE WITH RSS SPLITTING AND COST-COMPLEXITY PRUN-
ING
reg.tree.RSS<- rpart(median_house_value ~ housing_median_age + total_rooms + total_bedrooms
+ population + households + median_income + ocean_proximity, data=train, method="anova",
cp=0.026) #pruned tree with 4 splits and 5 leaves

#install.packages("rpart.plot")
library(rpart.plot)
rpart.plot(reg.tree.RSS, type=3)
```

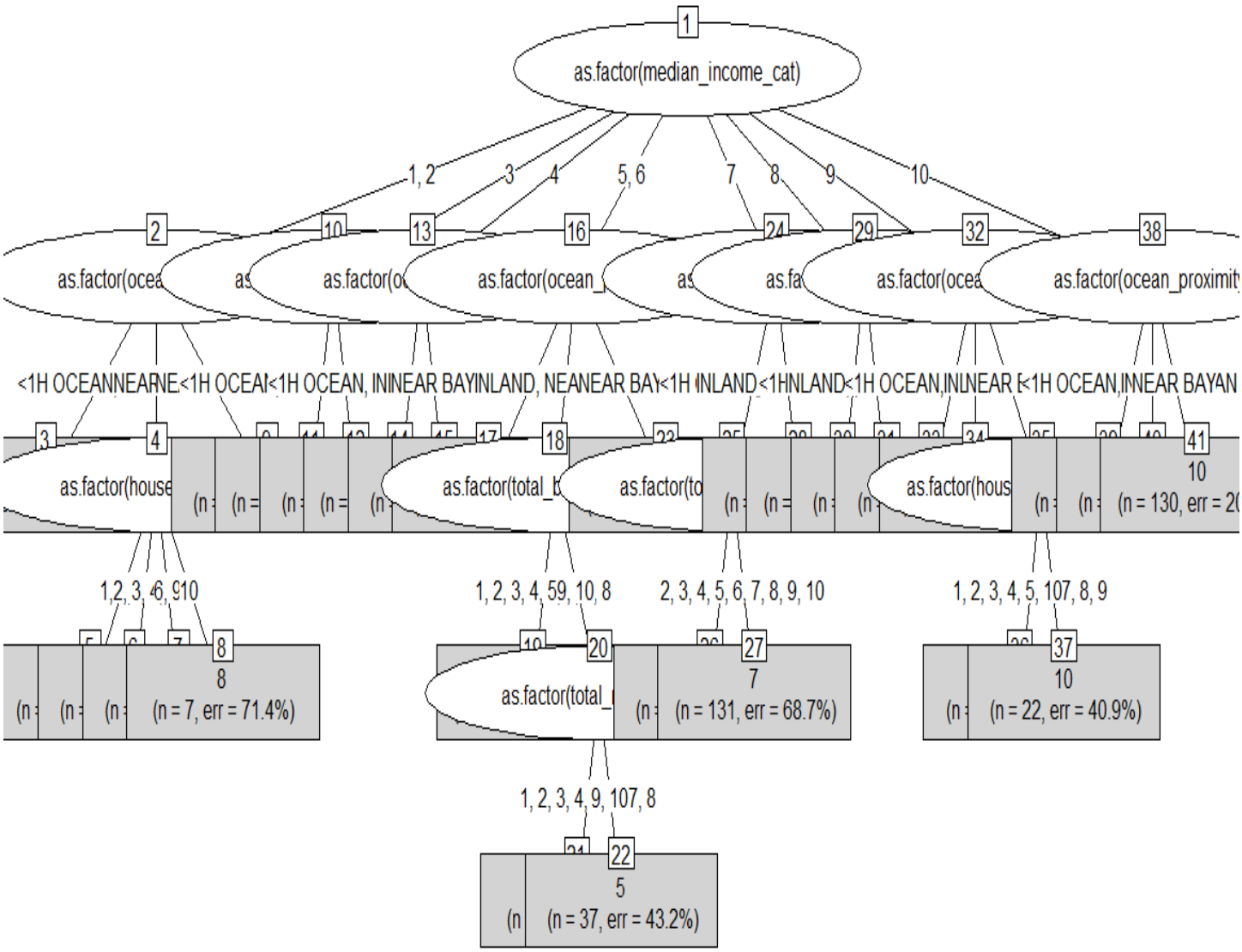



```
#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
P_median_house_value<- predict(reg.tree.RSS, newdata=test)

#accuracy within 10%
accuracy10<-
ifelse(abs(test$median_house_value-P_median_house_value)<0.10*test$median_house_value,1,0)
print(mean(accuracy10))
0.2422018
#accuracy within 15%
accuracy15<-
ifelse(abs(test$median_house_value-P_median_house_value)<0.15*test$median_house_value,1,0)
print(mean(accuracy15))
0.346789
#accuracy within 20%
accuracy20<-
ifelse(abs(test$median_house_value-P_median_house_value)<0.20*test$median_house_value,1,0)
print(mean(accuracy20))
0.440367
```

Note that this tree is similar to the one produced by SAS, and the proportions of accurate predic-

tions are close to the ones in SAS. Now we fit a regression tree with five terminal nodes based on the CHAID splitting criterion.

```
#FITTING REGRESSION TREE WITH CHAID SPLITTING AND COST-COMPLEXITY PRUN-
ING
install.packages("CHAID", repos="http://R-Forge.R-project.org", type="source")

#BINNING CONTINUOUS PREDICTOR VARIABLES
install.packages("dplyr")
library(dplyr)
housing.data<- mutate(housing.data, housing_median_age_cat=ntile(housing_median_age,10),
total_rooms_cat=ntile(total_rooms,10), total_bedrooms_cat=ntile(total_bedrooms,10), popu-
lation_cat=ntile(population,10), households_cat=ntile(households,10),
median_income_cat=ntile(median_income,10), median_house_value_cat=ntile(median_house_value,10))

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
set.seed(105388)
sample <- sample(c(TRUE, FALSE), nrow(housing.data), replace=TRUE, prob=c(0.8,0.2))
train<- housing.data[sample,]
test<- housing.data[!sample,]

library(CHAID)

reg.tree.CHAID<- chaid(as.factor(median_house_value_cat) ~ as.factor(housing_median_age_cat)
+ as.factor(total_rooms_cat) + as.factor(total_bedrooms_cat) + as.factor(population_cat) +
as.factor(households_cat) + as.factor(median_income_cat) + as.factor(ocean_proximity), data=train,
control=chaid_control(maxheight=4))

plot(reg.tree.CHAID, type="simple")
```



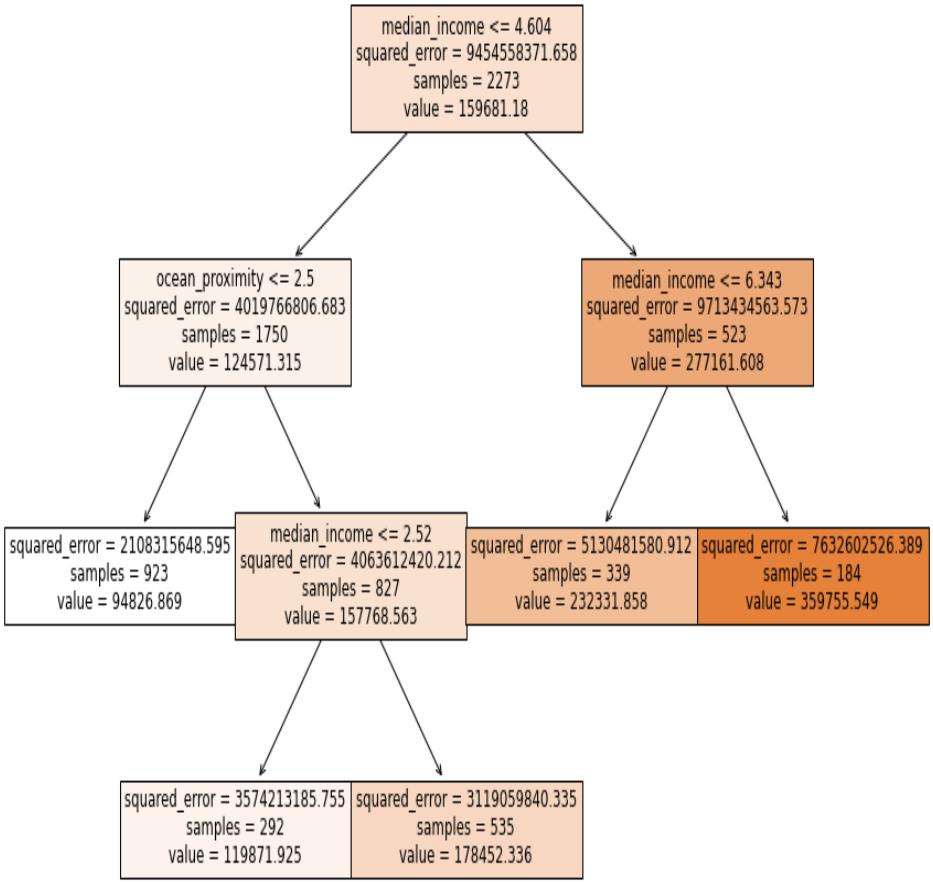

```
#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
predclass<- as.numeric(predict(reg.tree.CHAID, newdata=test))
test<- cbind(test,predclass)

#computing predicted values (mean values per decile in the training set)

aggr.data<- aggregate(train$median_house_value, by=list(train$median_house_value_cat), FUN=mean)
#combining observed and predicted values in the testing set
```

```
aggr.data$predclass<- aggr.data$Group.1
aggr.data$P_median_house_value<- aggr.data$x
test<- left_join(test, aggr.data, by='predclass')

#accuracy within 10%
accuracy10<-
ifelse(abs(test$median_house_value-test$P_median_house_value) <0.10*test$median_house_value,1,0)
print(mean(accuracy10))
0.2880734
#accuracy within 15%
accuracy15<-
ifelse(abs(test$median_house_value-test$P_median_house_value) <0.15*test$median_house_value,1,0)
print(mean(accuracy15))
0.4091743
#accuracy within 20%
accuracy20<-
ifelse(abs(test$median_house_value-test$P_median_house_value) <0.20*test$median_house_value,1,0)
print(mean(accuracy20))
0.5045872
```

Finally, we write a Python code to fit the regression tree using RSS and CHAID splitting criteria.

In Python:

```
import numpy
import pandas
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeRegressor
from sklearn import tree
from sklearn.model_selection import train_test_split

housing=pandas.read_csv('C:/Users/000110888/Desktop/housing_data.csv')
coding={'<1H OCEAN': 1, 'INLAND': 2, 'NEAR BAY': 3, 'NEAR OCEAN': 4}
housing['ocean_proximity']=housing['ocean_proximity'].map(coding)
X=housing.iloc[:,0:7].values
y=housing.iloc[:,7].values

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.20,
random_state=348644)

#FITTING REGRESSION TREE WITH RSS SPLITTING CRITERION
rtree = DecisionTreeRegressor(random_state=907420,
criterion="squared_error", max_leaf_nodes=5)
reg_tree_RSS = rtree.fit(X_train, y_train)

#PLOTTING FITTED TREE
fig=plt.figure(figsize=(15,10))
fn=['housing_median_age','total_rooms','total_bedrooms','population',
'households','median_income','ocean_proximity']
tree.plot_tree(reg_tree_RSS, feature_names=fn, filled=True)
```


median_income <= 4.604
squared_error = 9454558371.658
samples = 2273
value = 159681.18
ocean_proximity <= 2.5
squared_error = 4019766806.683
samples = 1750
value = 124571.315
median_income <= 6.343
squared_error = 9713434563.573
samples = 523
value = 277161.608
squared_error = 2108315648.595
samples = 923
value = 94826.869
median_income <= 2.52
squared_error = 4063612420.212
samples = 827
value = 157768.563
squared_error = 5130481580.912
samples = 339
value = 232331.858
squared_error = 7632602526.389
samples = 184
value = 359755.549
squared_error = 3574213185.755
samples = 292
value = 119871.925
squared_error = 3119059840.335
samples = 535
value = 178452.336

```
#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
y_pred=reg_tree_RSS.predict(X_test)

ind10=[]
ind15=[]
ind20=[]

for sub1, sub2 in zip(y_pred, y_test):
    ind10.append(1) if abs(sub1-sub2)<0.10*sub2 else ind10.append(0)
    ind15.append(1) if abs(sub1-sub2)<0.15*sub2 else ind15.append(0)
    ind20.append(1) if abs(sub1-sub2)<0.20*sub2 else ind20.append(0)

#accuracy within 10%
accuracy10=sum(ind10)/len(ind10)
print(accuracy10)

#accuracy within 15%
accuracy15=sum(ind15)/len(ind15)
print(accuracy15)

#accuracy within 20%
accuracy20=sum(ind20)/len(ind20)
print(accuracy20)
```

```
0.2671353251318102
0.36731107205623903
0.46045694200351495
```

For the CHAID splitting criterion, we first split the response variable into deciles and then turn the deciles (0, ..., 9) into nominal classes (0th, ..., 9th). The tree is fitted using the Chefboost package, which fits CHAID trees with nominal response only.

```
#FITTING REGRESSION TREE WITH CHAID SPLITTING CRITERION

#SPLITTING RESPONSE VARIABLE INTO DECILES AND MAKING IT NOMINAL
housing['deciles']=pandas.qcut(housing['median_house_value'], 10, labels=False)
deciles_coding={0:'0th',1:'1st',2:'2nd',3:'3rd',4:'4th',5:'5th',6:'6th',7:'7th',8:'8th',9:'9th'}
housing['deciles']=housing['deciles'].map(deciles_coding)

X=housing.iloc[:,0:7].values
y=housing.iloc[:,7:9].values

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.20, random_state=348644)

X_train=pandas.DataFrame(X_train, columns=['housing_median_age','total_rooms',
'total_bedrooms','population','households','median_income','ocean_proximity'])
y_train=pandas.DataFrame(y_train[:,1], columns=['deciles'])
train_data=pandas.concat([X_train, y_train],axis=1)

#FITTING TREE
from chefboost import Chefboost

config={'algorithm': 'CHAID', 'max_depth': 4}
tree_chaid=Chefboost.fit(train_data, config, target_label='deciles')
```

The fitted tree is not plotted but stored in the `/outputs/rules/rules.py` file. Here are the decision rules for the fitted tree:

```
def findDecision(obj): #obj[0]: housing_median_age, obj[1]: total_rooms, obj[2]: total_bedroo
# {"feature": "median_income", "instances": 2273, "metric_value": 123.2024, "depth": 1}
if obj[5]<=3.553185833699958:
# {"feature": "ocean_proximity", "instances": 1371, "metric_value": 110.1406, "depth": 2}
if obj[6]<=2.0:
# {"feature": "housing_median_age", "instances": 764, "metric_value": 109.5716, "depth": 3}
if obj[0]<=26.785340314136125:
# {"feature": "total_rooms", "instances": 406, "metric_value": 72.5045, "depth": 4}
if obj[1]<=2437.9162561576354:
# {"feature": "population", "instances": 252, "metric_value": 54.3406, "depth": 5}
if obj[3]<=1262.9637320858756:
# {"feature": "households", "instances": 218, "metric_value": 44.3203, "depth": 6}
```

```
if obj[4]>266.10091743119267:
# {"feature": "total_bedrooms", "instances": 118, "metric_value": 25.8022, "depth": 7}
if obj[2]<=480.1384663358453:
return '2nd'
elif obj[2]>480.1384663358453:
return '1st'
else: return '1st'
elif obj[4]<=266.10091743119267:
# {"feature": "total_bedrooms", "instances": 100, "metric_value": 25.441, "depth": 7}
if obj[2]<=415.051670879008:
return '0th'
elif obj[2]>415.051670879008:
return '3rd'
else: return '3rd'
else: return '0th'
elif obj[3]>1262.9637320858756:
# {"feature": "households", "instances": 34, "metric_value": 17.5386, "depth": 6}
if obj[4]<=411.05882352941177:
# {"feature": "total_bedrooms", "instances": 21, "metric_value": 10.9193, "depth": 7}
if obj[2]>358.0:
return '0th'
elif obj[2]<=358.0:
return '0th'
else: return '0th'
elif obj[4]>411.05882352941177:
# {"feature": "total_bedrooms", "instances": 13, "metric_value": 7.8621, "depth": 7}
if obj[2]<=490.0:
return '1st'
elif obj[2]>490.0:
return '0th'
else: return '0th'
else: return '0th'
else: return '0th'

          ***  LINES OMITTED ***

if obj[2]>1294.0:
# {"feature": "population", "instances": 5, "metric_value": 2.8284, "depth": 7}
if obj[3]>3283.0:
return '9th'
elif obj[3]<=3283.0:
return '9th'
```

```
else: return '9th'
elif obj[2]<=1294.0:
return '9th'
else: return '9th'
else: return '9th'
else: return '9th'
else: return '9th'
else: return '9th'
else: return '9th'
```

```python
#USING FITTED MODEL TO PREDICT FOR TESTING DATA
X_test=pandas.DataFrame(X_test, columns=['housing_median_age','total_rooms',
'total_bedrooms','population','households','median_income','ocean_proximity'])

y_pred=[]
for i in range(len(y_test)):
    y_pred.append(Chefboost.predict(tree_chaid, X_test.iloc[i,:]))

#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
y_test=pandas.DataFrame(y_test[:,0], columns=['median_house_value'])
y_pred=pandas.DataFrame(y_pred, columns=['predclass'])
pred_data=pandas.concat([y_test,y_pred],axis=1)

df_new=pred_data.groupby('predclass')['median_house_value'].mean()#predicted value=class mean
inner_join = pandas.merge(pred_data, df_new, on='predclass', how ='inner')

ind10=[]
ind15=[]
ind20=[]
#median_house_value_x=observed value, median_house_value_y=predicted value
for sub1, sub2 in zip(inner_join['median_house_value_x'], inner_join['median_house_value_y']):
    ind10.append(1) if abs(sub1-sub2)<0.10*sub1 else ind10.append(0)
    ind15.append(1) if abs(sub1-sub2)<0.15*sub1 else ind15.append(0)
    ind20.append(1) if abs(sub1-sub2)<0.20*sub1 else ind20.append(0)

#accuracy within 10%
accuracy10=sum(ind10)/len(ind10)
print(accuracy10)

#accuracy within 15%
accuracy15=sum(ind15)/len(ind15)
print(accuracy15)

#accuracy within 20%
accuracy20=sum(ind20)/len(ind20)
print(accuracy20)
```

```
0.20210896309314588
0.3022847100175747
0.39015817223198596
```

□

Classification Tree

A classification tree is a decision tree for a categorical (or even binary) target (response) variable. In classification trees, a natural alternative to the RSS is the **classification error rate**, defined as the fraction of observations in a region that do not belong to the most common class (that is, are misclassified by the tree). Two splitting methods are usually used, one is based on the Gini impurity index and the other is based on entropy.

Gini Impurity Index

The following **Gini impurity index** is commonly used in practice: $G = \sum_{k=1}^{K} p_k(1-p_k)$ where p_k is the proportion of observations in the kth class (that is, the probability of randomly picking a data point in the kth class).

Historical Note. The Gini impurity index is named after an Italian statistician Corrado Gini who proposed the idea in his 1912 paper "Variability and Mutability".

Example. Suppose we observe 10 data points, 5 of which we color blue and the other 5 we color green (see the picture). Suppose we randomly pick a data point and then randomly classify it as blue or green according to the class distribution in the data set. That is, we classify it as blue (or green) with probability $5/10 = 1/2$, since we have 5 data points of each color. What's the probability we classify our data point incorrectly?

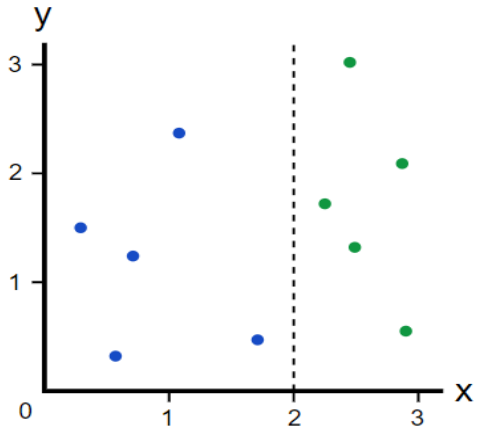
$$\mathbb{P}(\text{pick blue, classify blue}) = (1/2)(1/2) = 1/4,$$
$$\mathbb{P}(\text{pick blue, classify green}) = (1/2)(1/2) = 1/4,$$
$$\mathbb{P}(\text{pick green, classify blue}) = (1/2)(1/2) = 1/4,$$
$$\mathbb{P}(\text{pick green, classify green}) = (1/2)(1/2) = 1/4.$$

Therefore, the probability of misclassification is $\mathbb{P}(\text{pick blue, classify green}) + \mathbb{P}(\text{pick green, classify blue}) = 1/4 + 1/4 = 1/2 = 0.5$.

The Gini impurity index is computed as $G = \mathbb{P}(\text{pick blue})\big(1 - \mathbb{P}(\text{pick blue})\big) + \mathbb{P}(\text{pick green})\big(1 - \mathbb{P}(\text{pick green})\big) = (0.5)(1-0.5)+(0.5)(1-0.5) = 0.25+0.25 = 0.5$ and coincides with the probability of misclassification.

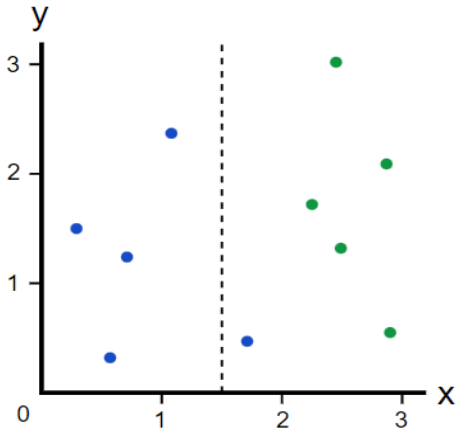
If we now make a "perfect" vertical split at $X = 2$ (see the figure below), then 100% of blue dots will be classified correctly as blue and 100% of green dots will be classified correctly as green, so the Gini impurity index for the region on the left is $G_{left} = \mathbb{P}(\text{pick blue})\big(1-\mathbb{P}(\text{pick blue})\big)+\mathbb{P}(\text{pick green})\big(1-$

$\mathbb{P}(\text{pick green})\big) = (1)(1-1) + (0)(1-0) = 0$. The Gini index for the region on the right is $G_{right} = \mathbb{P}(\text{pick blue})\big(1 - \mathbb{P}(\text{pick blue})\big) + \mathbb{P}(\text{pick green})\big(1 - \mathbb{P}(\text{pick green})\big) = (0)(1-0) + (1)(1-1) = 0$. Thus, the "perfect" split turned a data set with a 0.5 impurity into two branches with zero impurity, which is the lowest and the best possible impurity.



A Perfect Split

To illustrate further, consider now an "imperfect" split at $X = 1.5$ where one blue data point falls into the region on the right (as seen in the picture below). The region on the left has only blue data points, so we know that $G_{left} = 0$. The region on the right has 1 blue and 5 green data points, and hence, $G_{right} = (1/6)(1-1/6) + (5/6)(1-5/6) = 10/36 = 0.2778$.



An Imperfect Split

Finally, the **quality of the split** is determined by weighting the impurity of each region (branch) by how many data points it has. Since the region on the left has 4 data points and the region on the right has 6, we get $(0.4)(0) + (0.6)(0.2778) = 0.1667$. Thus, the amount of impurity that is "removed" by this split is $0.5 - 0.1667 = 0.3333$. This value is called the **Gini gain**. The Gini gain for the "perfect" split is $0.5 - ((0.5)(0) + (0.5)(0)) = 0.5$. When training a decision tree, the best split is chosen by maximizing the Gini gain. The larger the Gini gain, the better the split. □

Cross-entropy Loss Function

An alternative to the Gini impurity index is the **cross-entropy** (or **cross-entropy loss function** or **Shannon's entropy**) defined by the formula:

$$E = \begin{cases} -\sum_{k=1}^{K} p_k \ln(p_k), & \text{if } 0 < p_k \leq 1, \\ 0, & \text{if } p_k = 0. \end{cases}$$

Historical Note. Claude Shannon (1916-2001) was an American mathematician, electrical engineer, and cryptographer and is known as a "father of information theory".

Example. In our example, the cross-entropy for the entire data set is $E = -(0.5)\ln(0.5) - (0.5)\ln(0.5) = -\ln(0.5) = 0.6931$. For the "perfect" split, the cross-entropy for the left region is $E_{left} = -(1)\ln(1) - 0 = 0$, and so it is for the right region: $E_{right} = 0 - (1)\ln(1) = 0$. For the "imperfect" split, the cross-entropy for the left region is $E_{left} = -(1)\ln(1) - 0 = 0$ and that for the right region is $E_{right} = -(1/6)\ln(1/6) - (5/6)\ln(5/6) = 0.4506$. A split is better if the reduction in the cross-entropy is larger. For the "perfect" split, the cross-entropy is reduced by $0.6931 - [(0.4)(0) + (0.6)(0)] = 0.6931$, whereas for the "imperfect split", it is $0.6931 - [(0.4)(0) + (0.6)(0.4506)] = 0.4227 < 0.6931$, so the "perfect" split wins.

With the cross-entropy loss function, the amount of impurity that is "removed" by a split is termed the **information gain**. The splitting algorithm that uses entropy and information gain as the metric is called the **Iterative Dichotomiser 3 (ID3) algorithm**. A successor of the ID3 algorithm (a recognized improved version of ID3) is **C4.5 algorithm** that uses entropy and gain ratio as the measures. The **gain ratio** is defined by the ratio of information gain and **split information**. How to calculate split information is easier to explain by example.

Example. In the above example, the "imperfect" split can be summarized as follows:

Split	Blue	Green	Total
left	4	0	4
right	1	5	6

Split Information= $-\frac{4}{10}\ln\frac{4}{10} - \frac{6}{10}\ln\frac{6}{10} = 0.673012$, and Gain Ratio=Information Gain/Split

Information= 0.4227/0.673012 = 0.628072. Gain ratios are compared for all candidate variables, and the one with the largest value is selected for the next split. Note that the gain ratio penalizes having too many branches that a split would result in (that is, splitting in a high number of branches results in high information gain, but the split information also increases, so the gain ratio reduces the bias). □

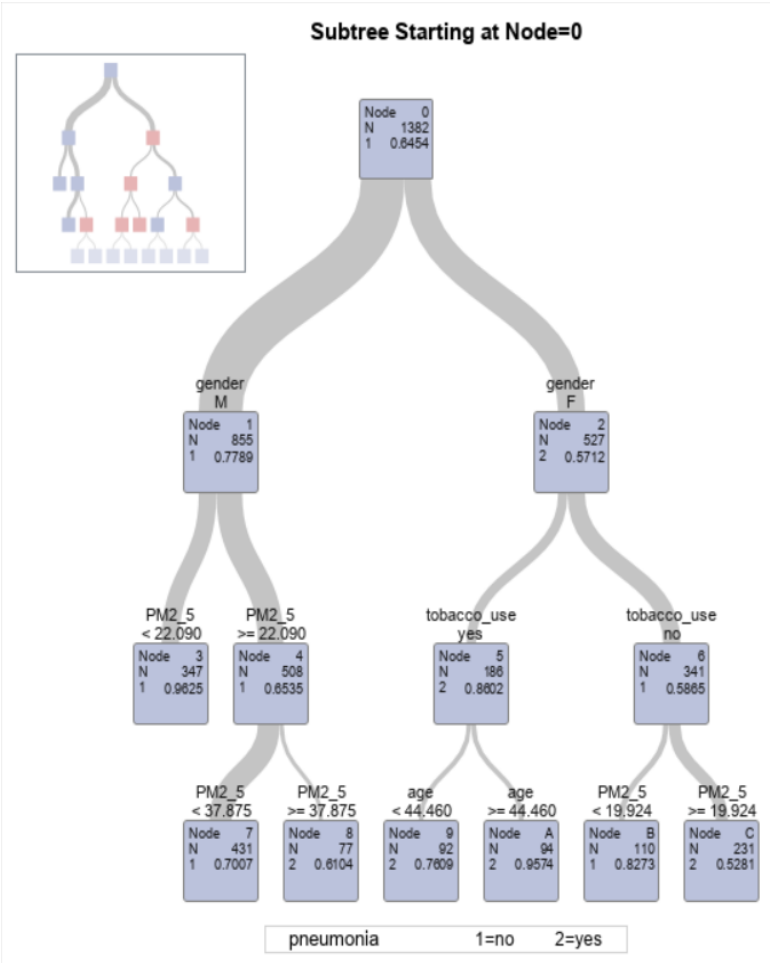
Historical Note. The ID3 algorithm was invented by John Ross Quinlan in 1979, and in 1993 he published "C4.5 Programs for Machine Learning" where he introduced the C4.5 algorithm.

Remark. Note that at any given node, there may be a number of splits on different variables, all of which give almost the same decrease in impurity. Since data are noisy, the choice between competing splits is almost random. However, choosing an alternative split that is almost as good will lead to a different evolution of the tree from that node downward. That's why when a classification tree is developed, a seed should be specified for reproducibility of results.

Example. The data file "pneumonia_data.csv" contains data on individuals' age, gender, an indicator of tobacco use, PM2.5 measurement for the place of residence (atmospheric particulate matter that has a diameter of fewer than 2.5 micrometers, in micro grams per cubic meter), and an indicator of pneumonia. We model pneumonia diagnosis using binary classification trees with the Gini, entropy, and CHAID splitting criteria.

In SAS:

```
proc import out=pneumonia datafile="./pneumonia_data.csv"
dbms=csv replace;

/*SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS*/
proc surveyselect data=pneumonia rate=0.8 seed=6132208
out=pneumonia outall method=srs;
run;

/*GINI SPLITTING AND COST-COMPLEXITY PRUNING */
proc hpsplit data=pneumonia maxdepth=4;
class pneumonia gender tobacco_use;
   model pneumonia(event="yes")= age gender tobacco_use PM2_5;
grow gini;
prune costcomplexity;
partition rolevar=selected(train="1");
output out=predicted;
ID selected;
run;
```



Further, we want to measure the performance of the tree based on the proportion of observations in the testing set that are predicted correctly. The data set `predicted` contains predicted probabilities of pneumonia for both training and testing sets (indexed by the variable `selected`) (see below).

```
proc print data=predicted (obs=10);
run;
```

Obs	pneumonia	Selected	_Node_	_Leaf_	P_pneumoniano	P_pneumoniayes
1	yes	0	3	0	0.96254	0.03746
2	no	0	7	1	0.70070	0.29930
3	no	1	7	1	0.70070	0.29930
4	no	1	15	5	0.20455	0.79545
5	no	0	19	9	0.51185	0.48815
6	yes	0	19	9	0.51185	0.48815
7	no	1	3	0	0.96254	0.03746
8	no	1	7	1	0.70070	0.29930
9	no	1	7	1	0.70070	0.29930
10	no	1	15	5	0.20455	0.79545

We introduce a cutoff, a value between 0 and 1, such that if a predicted probability of pneumonia is above it, we assume that pneumonia is predicted as present, otherwise absent. We search for an optimal value of cutoff that gives the highest correctly predicted proportion.

```
/*COMPUTING PREDICTION ACCURACY FOR TESTING DATA*/
data test;
set predicted;
if(selected="0");
keep pneumonia P_pneumoniayes;
run;

data cutoffs;
set test;
do i=1 to 99;
tp=(P_pneumoniayes > 0.01*i and pneumonia="yes");
tn=(P_pneumoniayes < 0.01*i and pneumonia="no");
output;
end;
run;

proc sql;
create table rates as
select i, sum(tp+tn)/count(*) as trueclassrate
from cutoffs
group by i;
```

```
select 0.01*i as cutoff, trueclassrate
 from rates
  having trueclassrate=max(trueclassrate);
quit;
```

cutoff	trueclassrate
0.49	0.805797
0.5	0.805797
0.51	0.805797
0.52	0.805797
0.53	0.805797
0.54	0.805797
0.55	0.805797
0.56	0.805797
0.57	0.805797
0.58	0.805797
0.59	0.805797

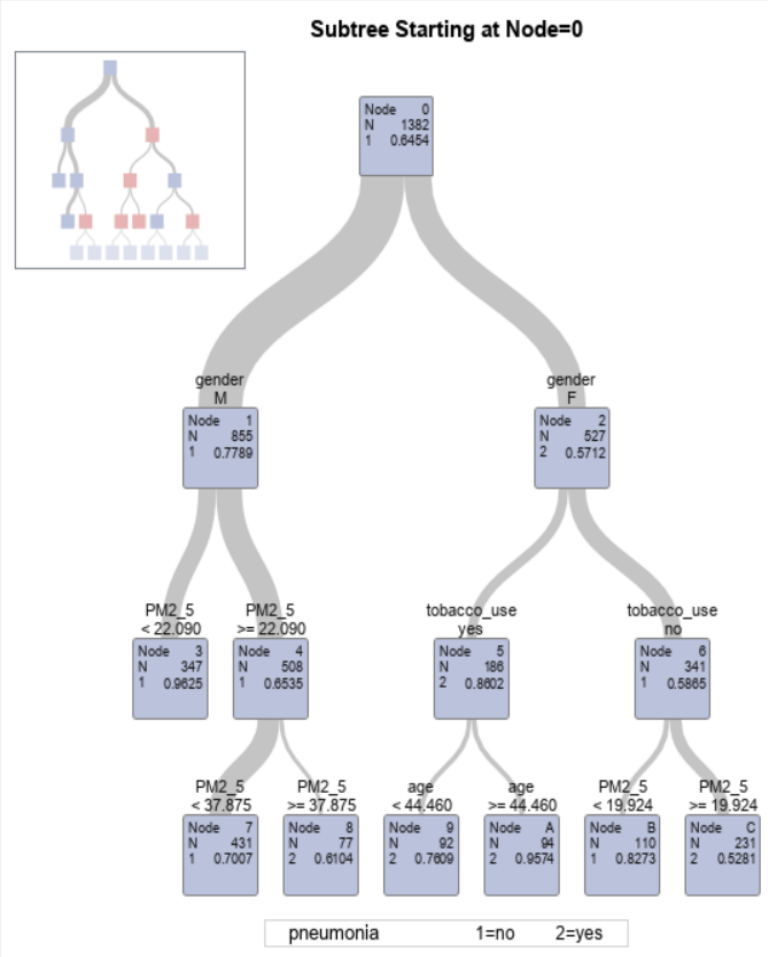
...

0.72	0.805797
0.73	0.805797
0.74	0.805797
0.75	0.805797
0.76	0.805797
0.77	0.805797
0.78	0.805797
0.79	0.805797

From this output, we can see that the largest proportion of correct predictions (80.5797%) is achieved for any cutoff between 0.49 and 0.79.

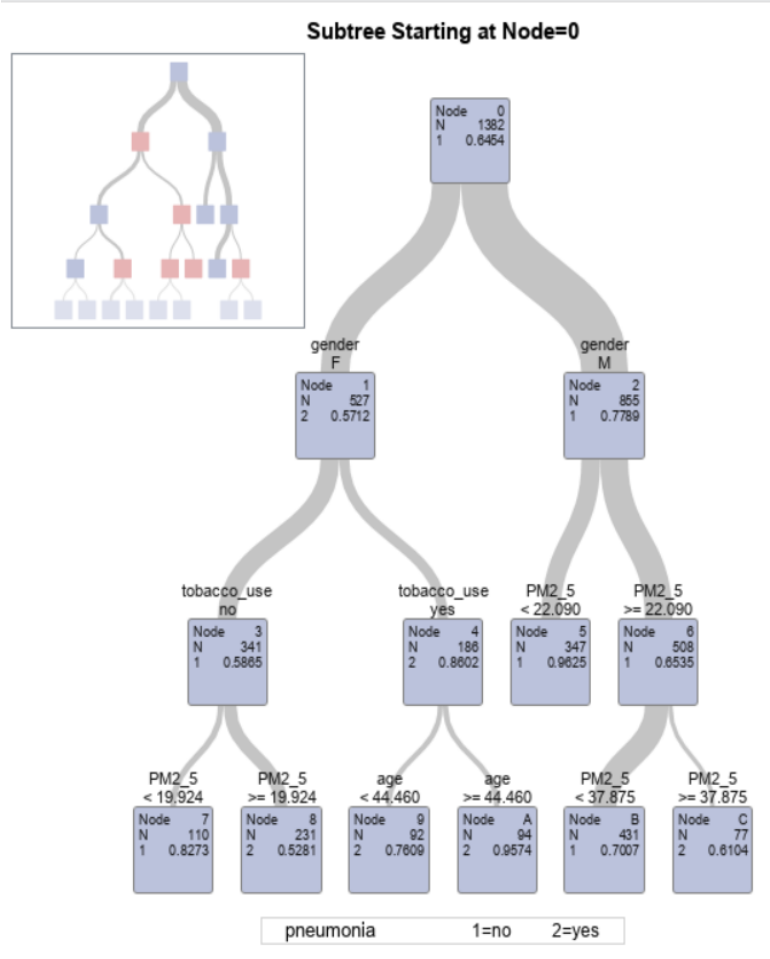
Moving forward, we fit binary classification trees using the entropy and CHAID splitting criteria. We use the cost-complexity pruning algorithm. The trees come out to be the same as the one fitting using the Gini impurity index.

```
/*ENTROPY SPLITTING AND COST-COMPLEXITY PRUNING */
proc hpsplit data=pneumonia maxdepth=4;
class pneumonia gender tobacco_use;
   model pneumonia(event="yes")= age gender tobacco_use PM2_5;
grow entropy;
prune costcomplexity;
partition rolevar=selected(train="1");
output out=predicted;
ID selected;
run;
```




```
/*CHAID SPLITTING AND COST-COMPLEXITY PRUNING */
proc hpsplit data=pneumonia maxdepth=4;
class pneumonia gender tobacco_use;
   model pneumonia(event="yes")= age gender
```

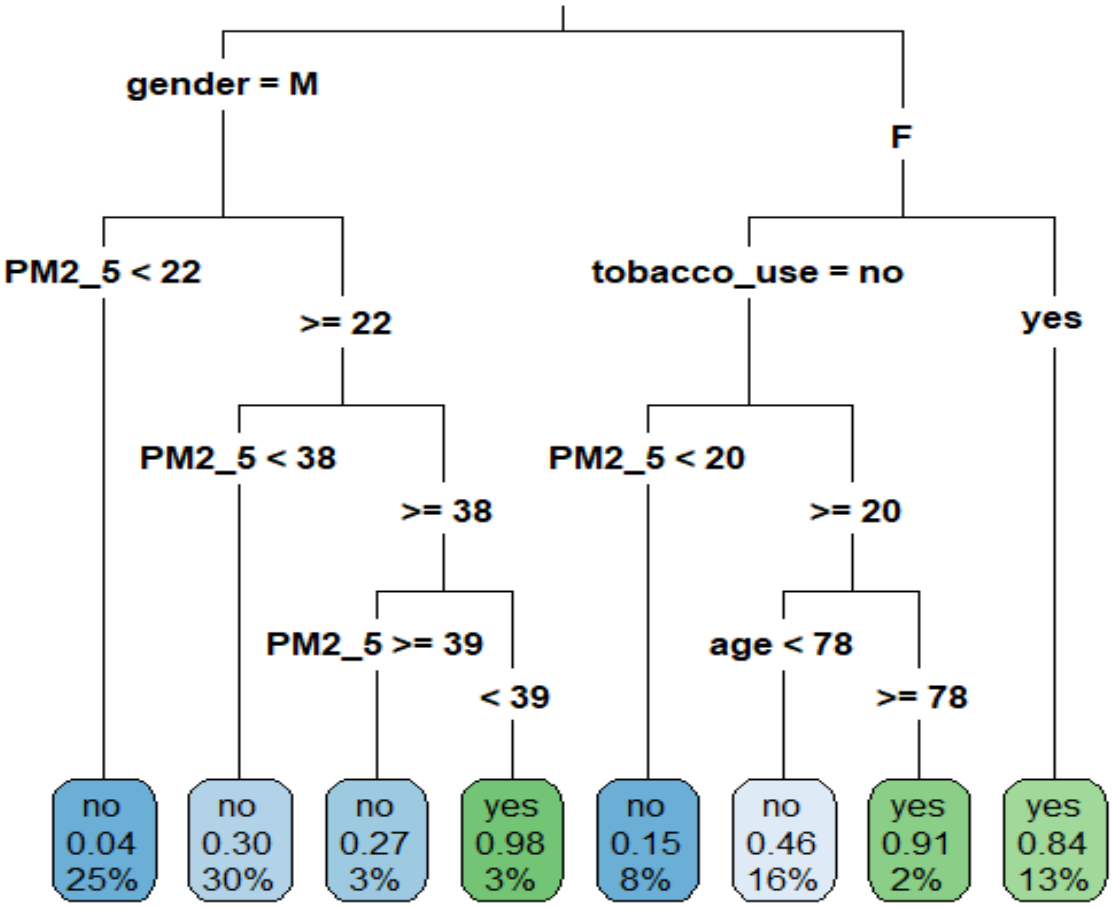
```
tobacco_use PM2_5;
grow CHAID;
prune costcomplexity;
partition rolevar=selected(train="1");
output out=predicted;
ID selected;
run;
```



In R:

```
pneumonia.data<- read.csv(file="./pneumonia_data.csv", header=TRUE, sep=",")

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
set.seed(283605)
sample <- sample(c(TRUE, FALSE), nrow(pneumonia.data), replace=TRUE, prob=c(0.8,0.2))
```

```
train<- pneumonia.data[sample,]
test<- pneumonia.data[!sample,]

#FITTING PRUNED BINARY TREE WITH GINI SPLITTING
library(rpart)
tree.gini<- rpart(pneumonia ~ age + gender + tobacco_use + PM2_5, data=train, method="class",
parms=list(split="Gini"), maxdepth=4)

library(rpart.plot)
rpart.plot(tree.gini, type=3)
```




```
#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
pred.values<- predict(tree.gini, test)
test<- cbind(test,pred.values)
```

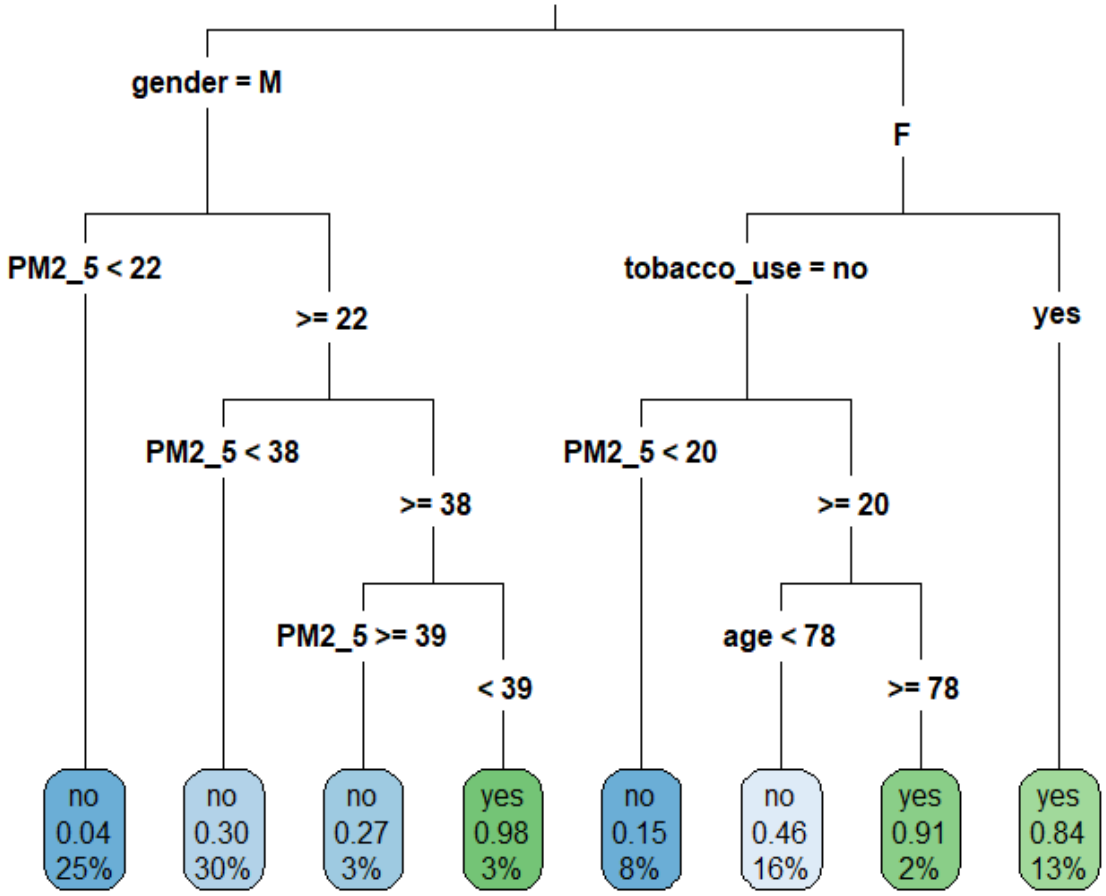
```
tp<- matrix(NA, nrow=nrow(test), ncol=99)
tn<- matrix(NA, nrow=nrow(test), ncol=99)

for (i in 1:99) {
    tp[,i]<- ifelse(test$pneumonia=="yes" & test$yes>0.01*i,1,0)
    tn[,i]<- ifelse(test$pneumonia=="no" & test$yes<=0.01*i,1,0)
}

trueclassrate<- matrix(NA, nrow=99, ncol=2)
for (i in 1:99){
    trueclassrate[i,1]<- 0.01*i
    trueclassrate[i,2]<- sum(tp[,i]+tn[,i])/nrow(test)
}

print(trueclassrate[which(trueclassrate[,2]==max(trueclassrate[,2])),])
```

```
      [,1]      [,2]
 [1,] 0.30 0.7886905
 [2,] 0.31 0.7886905
 [3,] 0.32 0.7886905
 [4,] 0.33 0.7886905
 [5,] 0.34 0.7886905
 [6,] 0.35 0.7886905
 [7,] 0.36 0.7886905
 [8,] 0.37 0.7886905
 [9,] 0.38 0.7886905
[10,] 0.39 0.7886905
[11,] 0.40 0.7886905
[12,] 0.41 0.7886905
[13,] 0.42 0.7886905
[14,] 0.43 0.7886905
[15,] 0.44 0.7886905
[16,] 0.45 0.7886905
```

The prediction accuracy for this tree is 78.86905%, which corresponds to any cutoff between 0.30 and 0.45. Next, we fit a binary tree using the entropy splitting criterion. The tree is identical to the one produced by the Gini criterion.

```
#FITTING PRUNED BINARY TREE WITH ENTROPY SPLITTING
tree.entropy<- rpart(pneumonia ~ age + gender + tobacco_use + PM2_5, data=train, method="class",
parms=list(split="entropy"), maxdepth=4)
```

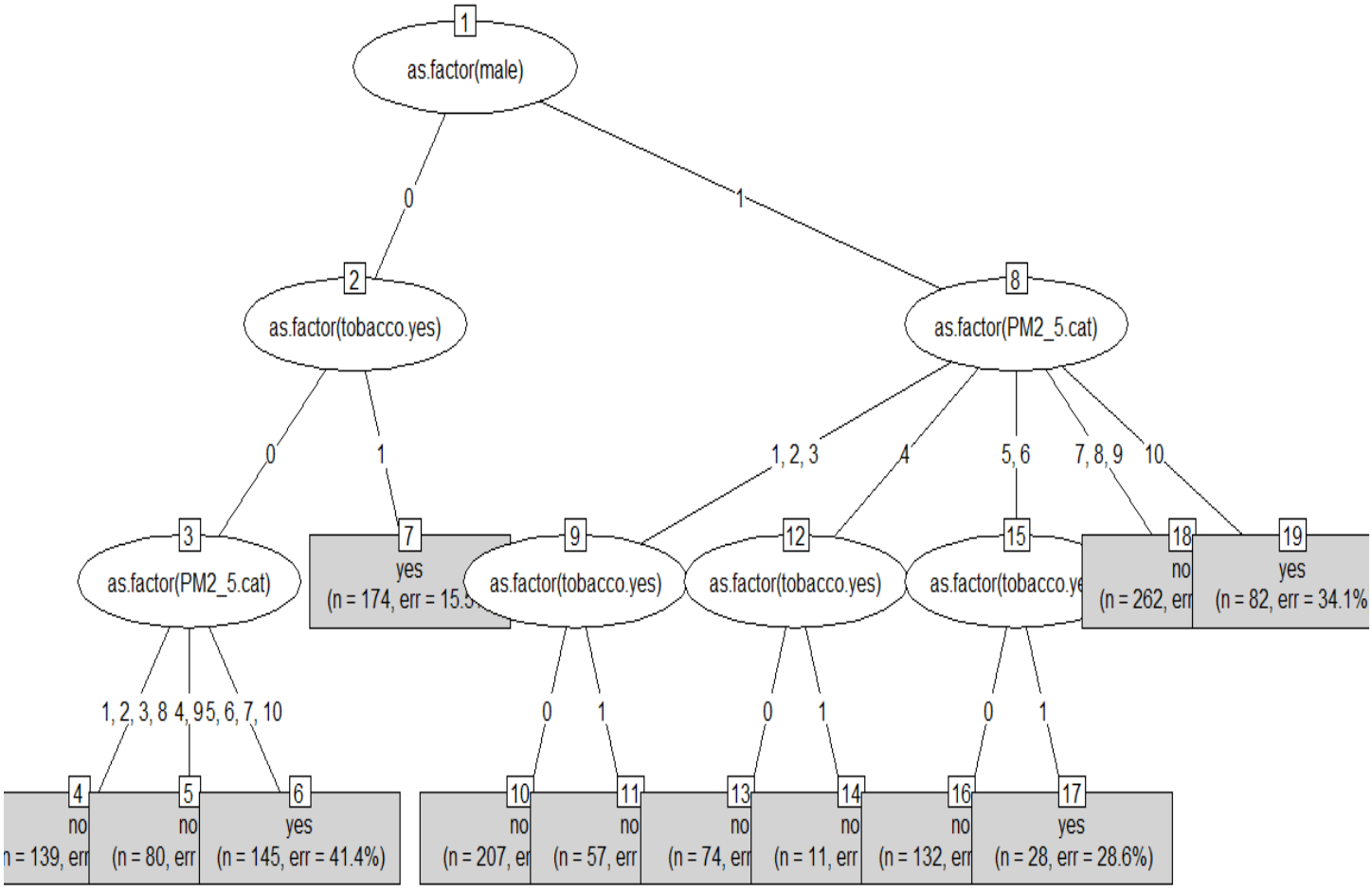
```
rpart.plot(tree.entropy, type=3)
```



Finally, we produce a binary classification tree based on the CHAID splitting criterion.

```
#FITTING PRUNED BINARY TREE WITH CHAID SPLITTING
#BINNING CONTINUOUS PREDICTOR VARIABLES
library(dplyr)

pneumonia.data<- mutate(pneumonia.data, age.cat=ntile(age,10), PM2_5.cat=ntile(PM2_5,10))

#CREATING INDICATORS FOR CATEGORICAL VARIABLES
pneumonia.data$male<- ifelse(pneumonia.data$gender=="M",1,0)
pneumonia.data$tobacco.yes<- ifelse(pneumonia.data$tobacco_use=="yes",1,0)
```

```
#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
set.seed(283605)
sample <- sample(c(TRUE, FALSE), nrow(pneumonia.data), replace=TRUE, prob=c(0.8,0.2))
train<- pneumonia.data[sample,]
test<- pneumonia.data[!sample,]

#FITTING BINARY CLASSIFICATION TREE
library(CHAID)
tree.CHAID<- chaid(as.factor(pneumonia) ~ as.factor(age.cat) + as.factor(male) + as.factor(tobacco.yes)
+ as.factor(PM2_5.cat), data=train, control=chaid_control(maxheight=3))

#PLOTTING FITTED TREE
plot(tree.CHAID, type="simple")
```




```
#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
```

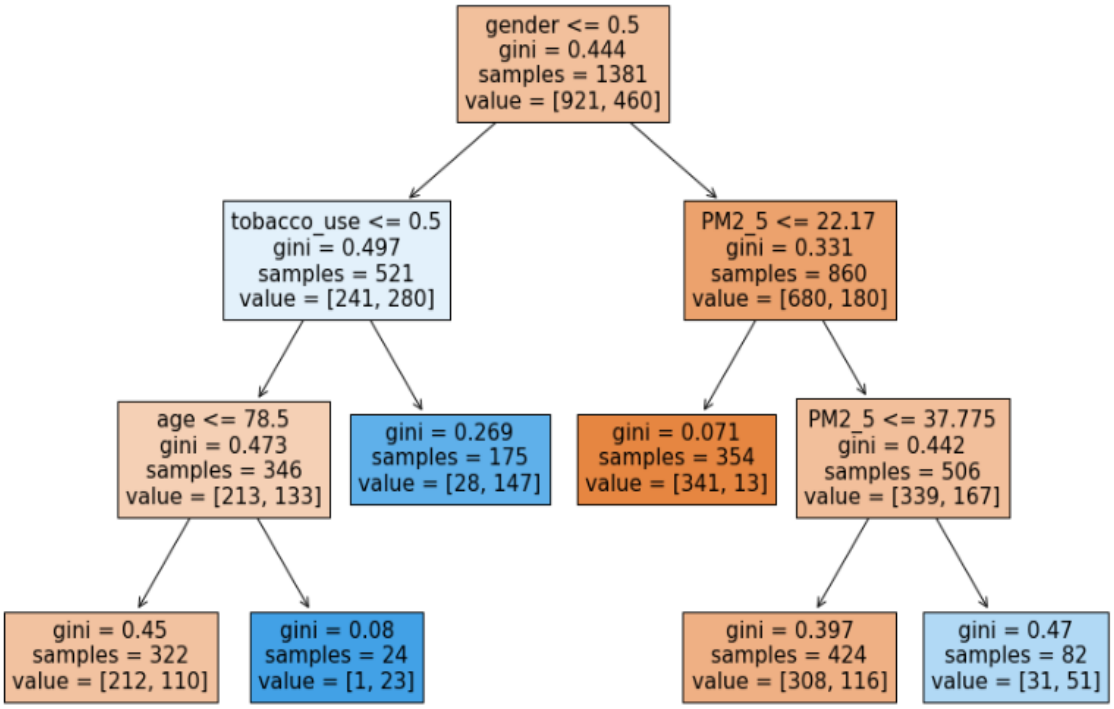
```
pred.pneumonia<- predict(tree.CHAID, newdata=test)
test<- cbind(test,pred.pneumonia)

truepred<- c()
n<- nrow(test)
for (i in 1:n)
    truepred[i]<- ifelse(test$pneumonia[i]==test$pred.pneumonia[i],1,0)

print(truepredrate<- sum(truepred)/length(truepred))
```
0.8035714

Note that the predicted values are yes/no, not probabilities. This tree gives 80.35714% correct predictions.

Next, we employ Python to build binary classification trees with the Gini and entropy slitting criteria.

In Python:

```
import numpy
import pandas
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn.model_selection import train_test_split

pneumonia_data=pandas.read_csv('C:/Users/000110888/Desktop/pneumonia_data.csv')
code_gender={'M':1,'F':0}
code_tobacco_use={'yes':1,'no':0}
code_pneumonia={'yes':1,'no':0}

pneumonia_data['gender']=pneumonia_data['gender'].map(code_gender)
pneumonia_data['tobacco_use']=pneumonia_data['tobacco_use'].map(code_tobacco_use)
pneumonia_data['pneumonia']=pneumonia_data['pneumonia'].map(code_pneumonia)

X=pneumonia_data.iloc[:,0:4].values
y=pneumonia_data.iloc[:,4].values

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.20,
random_state=786756)

#FITTING BINARY TREE WITH GINI SPLITTING CRITERION
gini_tree=DecisionTreeClassifier(max_leaf_nodes=6, criterion='gini', random_state=199233)
gini_tree.fit=gini_tree.fit(X_train,y_train)

#PLOTTING FITTED TREE
fig = plt.figure(figsize=(15,10))
tree.plot_tree(gini_tree.fit, feature_names=['gender','age','tobacco_use','PM2_5'], filled=True)
```


gender <= 0.5
gini = 0.444
samples = 1381
value = [921, 460]
tobacco_use <= 0.5
gini = 0.497
samples = 521
value = [241, 280]
PM2_5 <= 22.17
gini = 0.331
samples = 860
value = [680, 180]
age <= 78.5
gini = 0.473
samples = 346
value = [213, 133]
gini = 0.269
samples = 175
value = [28, 147]
gini = 0.071
samples = 354
value = [341, 13]
PM2_5 <= 37.775
gini = 0.442
samples = 506
value = [339, 167]
gini = 0.45
samples = 322
value = [212, 110]
gini = 0.08
samples = 24
value = [1, 23]
gini = 0.397
samples = 424
value = [308, 116]
gini = 0.47
samples = 82
value = [31, 51]

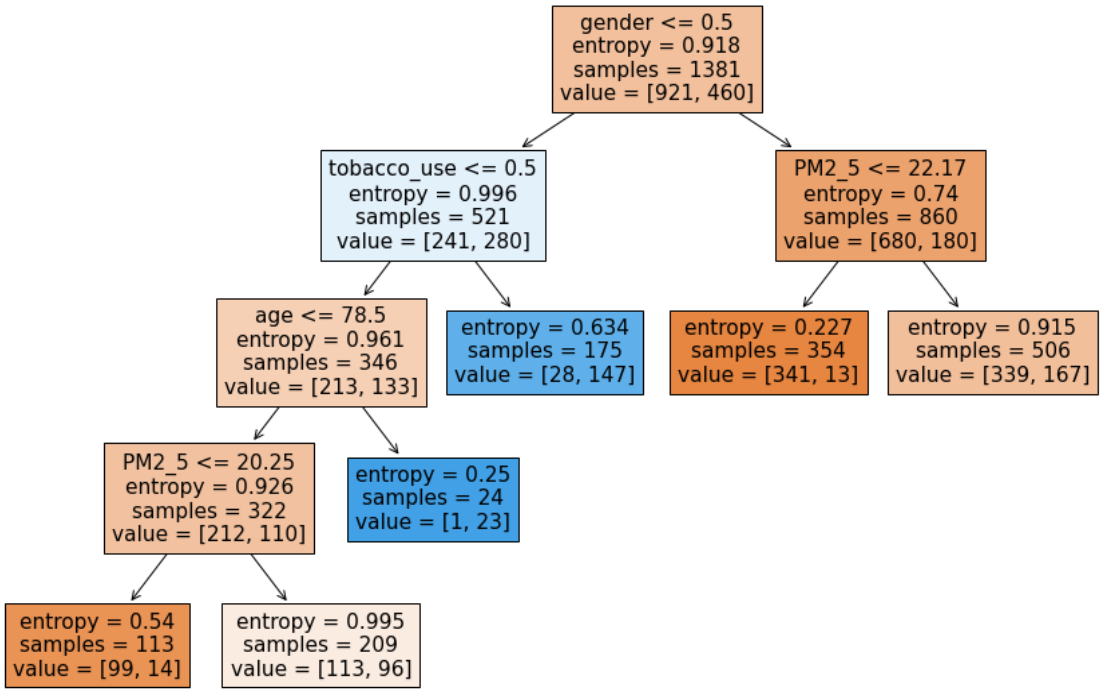
```
#COMPUTING PREDICTION ACCURACY FOR TESTING DATA

def accuracy():
    y_pred=gini_tree.predict_proba(X_test)

#y_pred[::,1] are predicted probabilities of "yes"

    total=len(y_pred)
    trueclassrate=[]
    cutoff=[]

    for i in range(99):
        tp=0
        tn=0
        cutoff.append(0.01*(i+1))
        for sub1, sub2 in zip(y_pred[::,1], y_test):
            tp_ind=1 if (sub1>0.01*(i+1) and sub2==1) else 0
            tn_ind=1 if (sub1<0.01*(i+1) and sub2==0) else 0
            tp+=tp_ind
            tn+=tn_ind
        rate=(tp+tn)/total
        trueclassrate.append(rate)

    df=pandas.DataFrame({'trueclassrate': trueclassrate,'cutoff': cutoff})
    max_rate=max(trueclassrate)
    optimal=df[df['trueclassrate']==max_rate]
    print(optimal)

accuracy()
```

```
    trueclassrate  cutoff
34       0.728324    0.35
35       0.728324    0.36
36       0.728324    0.37
37       0.728324    0.38
38       0.728324    0.39
39       0.728324    0.40
40       0.728324    0.41
41       0.728324    0.42
42       0.728324    0.43
43       0.728324    0.44
44       0.728324    0.45
45       0.728324    0.46
46       0.728324    0.47
47       0.728324    0.48
48       0.728324    0.49
49       0.728324    0.50
50       0.728324    0.51
51       0.728324    0.52
52       0.728324    0.53
```

```
53      0.728324    0.54
54      0.728324    0.55
55      0.728324    0.56
56      0.728324    0.57
57      0.728324    0.58
58      0.728324    0.59
59      0.728324    0.60
60      0.728324    0.61
61      0.728324    0.62
```

The true classification rate is 72.8324%, which corresponds to any cutoff between 0.35 and 0.62.

```
#FITTING BINARY TREE WITH ENTROPY SPLITTING CRITERION
gini_tree=DecisionTreeClassifier(max_leaf_nodes=6, criterion='entropy', random_state=199233)
gini_tree.fit=gini_tree.fit(X_train,y_train)

#PLOTTING FITTED TREE
fig = plt.figure(figsize=(15,10))
tree.plot_tree(gini_tree.fit, feature_names=['gender','age','tobacco_use','PM2_5'], filled=True)
```



```
#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
accuracy()
```

```
    trueclassrate  cutoff
33       0.728324    0.34
34       0.728324    0.35
35       0.728324    0.36
36       0.728324    0.37
37       0.728324    0.38
38       0.728324    0.39
39       0.728324    0.40
40       0.728324    0.41
41       0.728324    0.42
42       0.728324    0.43
43       0.728324    0.44
44       0.728324    0.45
```

The true classification rate this three is also 72.8324%, which corresponds to any cutoff between 0.34 and 0.45.

Finally, we use `Chefboost` decision tree framework in Python to fit a binary classification tree with the CHAID splitting criterion.

```
import pandas
from sklearn.model_selection import train_test_split
from chefboost import Chefboost

pneumonia_data=pandas.read_csv('C:/Users/000110888/Desktop/pneumonia_data.csv')
code_gender={'M':1,'F':0}
code_tobacco_use={'yes':1,'no':0}

pneumonia_data['gender']=pneumonia_data['gender'].map(code_gender)
pneumonia_data['tobacco_use']=pneumonia_data['tobacco_use'].map(code_tobacco_use)

X=pneumonia_data.iloc[:,0:4].values
y=pneumonia_data.iloc[:,4].values

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.20,
random_state=786756)

X_train=pandas.DataFrame(X_train, columns=['gender','age','tobacco_use','PM2_5'])
y_train=pandas.DataFrame(y_train, columns=['pneumonia'])
train_data=pandas.concat([X_train, y_train], axis=1) #one-to-one concatenation

#FITTING BINARY TREE WITH CHAID SPLITTING ALGORITHM
config={'algorithm': 'CHAID', "max_depth": 4}
tree_chaid=Chefboost.fit(train_data, config, target_label='pneumonia')
```

The fitted tree is not plotted but stored in the `/outputs/rules/rules.py` file. Here are the decision rules for the fitted tree:

```
def findDecision(obj): #obj[0]: gender, obj[1]: age, obj[2]: tobacco_use, obj[3]: PM2_5
# {"feature": "tobacco_use", "instances": 1381, "metric_value": 28.9752, "depth": 1}
if obj[2]<=0.0:
# {"feature": "PM2_5", "instances": 1052, "metric_value": 35.0779, "depth": 2}
if obj[3]>16.381331614375146:
# {"feature": "age", "instances": 838, "metric_value": 24.3317, "depth": 3}
if obj[1]>31.85100857903719:
# {"feature": "gender", "instances": 674, "metric_value": 16.9648, "depth": 4}
if obj[0]>0.0:
return 'no'
elif obj[0]<=0.0:
return 'no'
else: return 'no'
elif obj[1]<=31.85100857903719:
```

```
# {"feature": "gender", "instances": 164, "metric_value": 14.0331, "depth": 4}
if obj[0]>0.0:
return 'no'
elif obj[0]<=0.0:
return 'no'
else: return 'no'
else: return 'no'
elif obj[3]<=16.381331614375146:
# {"feature": "age", "instances": 214, "metric_value": 25.7088, "depth": 3}
if obj[1]<=50.52336448598131:
# {"feature": "gender", "instances": 109, "metric_value": 17.447, "depth": 4}
if obj[0]>0.0:
return 'no'
elif obj[0]<=0.0:
return 'no'
else: return 'no'
elif obj[1]>50.52336448598131:
# {"feature": "gender", "instances": 105, "metric_value": 17.3185, "depth": 4}
if obj[0]>0.0:
return 'no'
elif obj[0]<=0.0:
return 'no'
else: return 'no'
else: return 'no'
else: return 'no'
elif obj[2]>0.0:
# {"feature": "gender", "instances": 329, "metric_value": 17.9638, "depth": 2}
if obj[0]<=0.0:
# {"feature": "PM2_5", "instances": 175, "metric_value": 17.984, "depth": 3}
if obj[3]>26.570514285714314:
# {"feature": "age", "instances": 89, "metric_value": 13.2291, "depth": 4}
if obj[1]>41.59550561797753:
return 'yes'
elif obj[1]<=41.59550561797753:
return 'yes'
else: return 'yes'
elif obj[3]<=26.570514285714314:
# {"feature": "age", "instances": 86, "metric_value": 11.8925, "depth": 4}
if obj[1]>43.31395348837209:
return 'yes'
elif obj[1]<=43.31395348837209:
return 'yes'
```

```
else: return 'yes'
else: return 'yes'
elif obj[0]>0.0:
# {"feature": "PM2_5", "instances": 154, "metric_value": 9.3417, "depth": 3}
if obj[3]>16.351659726119824:
# {"feature": "age", "instances": 120, "metric_value": 4.2591, "depth": 4}
if obj[1]<=62.55213873641577:
return 'no'
elif obj[1]>62.55213873641577:
return 'yes'
else: return 'yes'
elif obj[3]<=16.351659726119824:
# {"feature": "age", "instances": 34, "metric_value": 10.2831, "depth": 4}
if obj[1]>46.470588235294116:
return 'no'
elif obj[1]<=46.470588235294116:
return 'no'
else: return 'no'
else: return 'no'
else: return 'no'
else: return 'yes'
```

We use the built tree for prediction on the testing set.

```
#COMPUTING PREDICTION ACCURACY
X_test=pandas.DataFrame(X_test, columns=['gender','age','tobacco_use','PM2_5'])

y_pred=[]
for i in range(len(y_test)):
    y_pred.append(Chefboost.predict(tree_chaid, X_test.iloc[i,:]))

y_test=pandas.DataFrame(y_test,columns=['pneumonia'])
y_pred=pandas.DataFrame(y_pred,columns=['predicted'])
df=pandas.concat([y_test,y_pred],axis=1)

match=[]
for i in range(len(df)):
    if df['pneumonia'][i] == df['predicted'][i]:
        match.append(1)
    else:
        match.append(0)

trueclassrate=sum(match)/len(match)

print(trueclassrate)
```

0.7052023121387283

About 70.52% of observations in the testing set are predicted correctly by this classification tree. □

CONFUSION MATRIX

For a binary classification tree, we used the correct classification rate as a measure of model performance. Traditionally, other quantities are also used. We define them below. Suppose, hypothetically speaking, of 100 observations in a testing set, 50 yes's are predicted correctly, 10 yes's are predicted incorrectly, 35 no's are predicted correctly, and 5 no's are predicted incorrectly. We summarize this information in the following table (called **confusion matrix** or **classification matrix**):

	True "Yes"	True "No"	Total
Predicted "Yes"	50	5	55
Predicted "No"	10	35	45
Total	60	40	100

For simplicity of notation, correctly predicted yes's are called "true positive" (TP), incorrectly predicted yes's are called "false negative" (FN), correctly predicted no's are called "true negative" (TN), and incorrectly predicted no's are called "false positive" (FP).

Example. In our example, $TP = 50, FN = 10, TN = 35$, and $FP = 5$. □

The numbers of cases in these categories are used to calculate various measures of model fit:

- **Accuracy (or Correct Classification Rate):** Overall, how often is the classifier correct?

  $(TP + TN)/Total = (TP + TN)/(TP + TN + FP + FN) = (50 + 35)/100 = 0.85$

- **Misclassification Rate:** Overall, how often is it wrong?

  $(FP+FN)/Total = (FP+FN)/(TP+TN+FP+FN) = (5+10)/100 = 0.15 = 1-Accuracy$

- **True Positive Rate (or Sensitivity or Recall):** When it's actually yes, how often does it predict yes?

  $TP/True\ yes = TP/(TP + FN) = 50/60 = 0.8333$

- **False Negative Rate (FNR):** When it's actually yes, how often does it predict no?

  $FN/True\ yes = FN/(TP + FN) = 10/60 = 0.1667 = 1 - Sensitivity$

- **True Negative Rate (or Specificity):** When it's actually no, how often does it predict no?

  $TN/True\ no = TN/(FP + TN) = 35/40 = 0.875$

- **False Positive Rate (FPR):** When it's actually no, how often does it predict yes?

$$FP/True\ no = FP/(FP + TN) = 5/40 = 0.125 = 1 - Specificity$$

• **Positive Predictive Value (PPV, or Precision):** When it predicts yes, how often is it correct?

$$TP/Predicted\ yes = TP/(TP + FP) = 50/55 = 0.9091$$

• **Negative Predictive Value (NPV):** When it predicts no, how often is it correct?

$$TN/Predicted\ no = TN/(FN + TN) = 35/45 = 0.7778$$

These definitions can be presented in a theoretical confusion matrix:

	True "Yes"	True "No"	
Predicted "Yes"	True Positive (TP)	False Positive (FP)	Precision $= \frac{TP}{TP + FP}$
Predicted "No"	False Negative (FN)	True Negative (TN)	Negative Predictive Value $= \frac{TN}{FN + TN}$
	Sensitivity $= \frac{TP}{TP + FN}$, False Negative Rate = 1 − Sensitivity	Specificity $= \frac{TN}{FP + TN}$, False Positive Rate = 1 − Specificity	Accuracy $= \frac{TP + TN}{TP + TN + FP + FN}$, Misclassification Rate = 1 − Accuracy

Another performance measure that is often utilized is the F1-score. It combines recall and precision into a single measure.

• **F1-score:** It is a harmonic mean of sensitivity and precision and can be calculated as follows:

$$\text{F1-score} = \frac{2}{\frac{1}{sensitivity} + \frac{1}{precision}} = \frac{2}{\frac{TP+FN}{TP} + \frac{TP+FP}{TP}}$$
$$= \frac{2TP}{2TP + FN + FP} = \frac{(2)(50)}{(2)(50) + 10 + 5} = \frac{100}{115} = 0.869565.$$

Example. Going back to the pneumonia example, we take the binary classification tree with Gini splitting and cost-complexity pruning and compute the confusion matrix and the performance measures for the test data, using 0.5 as the cutoff.

In SAS:

```
proc import out=pneumonia datafile="./pneumonia_data.csv"
dbms=csv replace;

/*SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS*/
proc surveyselect data=pneumonia rate=0.8 seed=6132208
out=pneumonia outall method=srs;
run;

/*GINI SPLITTING AND COST-COMPLEXITY PRUNING */
proc hpsplit data=pneumonia maxdepth=4;
class pneumonia gender tobacco_use;
   model pneumonia(event="yes")= age gender tobacco_use PM2_5;
grow gini;
prune costcomplexity;
partition rolevar=selected(train="1");
output out=predicted;
ID selected;
run;

/*COMPUTING CONFUSION MATRIX AND PERFORMANCE MEASURES FOR TESTING SET*/
data test;
set predicted;
if(selected="0");
tp=(P_pneumoniayes > 0.5 and pneumonia="yes");
fp=(P_pneumoniayes > 0.5 and pneumonia="no");
tn=(P_pneumoniano > 0.5 and pneumonia="no");
fn=(P_pneumoniano > 0.5 and pneumonia="yes");
run;
```

```
proc sql;
create table confusion as
select sum(tp) as tp, sum(fp) as fp, sum(tn) as tn,
sum(fn) as fn, count(*) as total
from test;
select * from confusion;
quit;
```

tp	fp	tn	fn	total
55	9	223	58	345

```
proc sql;
select (tp+tn)/total as accuracy, (fp+fn)/total as
misclassrate, tp/(tp+fn) as sensitivity,
fn/(tp+fn) as FNR, tn/(fp+tn) as specificity,
fp/(fp+tn) as FPR, tp/(tp+fp) as precision,
tn/(fn+tn) as NPV, 2*tp/(2*tp+fn+fp) as F1score
from confusion;
quit;
```

accuracy	misclassrate	sensitivity	FNR	specificity	FPR	precision	NPV	F1score
0.805797	0.194203	0.486726	0.513274	0.961207	0.038793	0.859375	0.793594	0.621469

In R:

```
pneumonia.data<- read.csv(file="./pneumonia_data.csv", header=TRUE, sep=",")

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
set.seed(283605)
sample <- sample(c(TRUE, FALSE), nrow(pneumonia.data), replace=TRUE, prob=c(0.8,0.2))
train<- pneumonia.data[sample,]
test<- pneumonia.data[!sample,]
```

```
#FITTING PRUNED BINARY TREE WITH GINI SPLITTING
library(rpart)
tree.gini<- rpart(pneumonia ~ age + gender + tobacco_use + PM2_5, data=train, method="class",
parms=list(split="Gini"), maxdepth=4)

#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
pred.values<- predict(tree.gini, test)
test<- cbind(test,pred.values)

tp<- c()
fp<- c()
tn<- c()
fn<- c()

total<- nrow(test)
for (i in 1:total){
    tp[i]<- ifelse(test$yes[i]>0.5 & test$pneumonia[i]=="yes",1,0)
    fp[i]<- ifelse(test$yes[i]>0.5 & test$pneumonia[i]=="no",1,0)
    tn[i]<- ifelse(test$no[i]>0.5 & test$pneumonia[i]=="no",1,0)
    fn[i]<- ifelse(test$no[i]>0.5 & test$pneumonia[i]=="yes",1,0)
}

print(tp<- sum(tp))
```

56

```
print(fp<- sum(fp))
```

8

```
print(tn<- sum(tn))
```

206

```
print(fn<- sum(fn))
```

66

```
print(total)
```

336

```
print(accuracy<- (tp+tn)/total)
```

0.7797619

```
print(misclassrate<- (fp+fn)/total)
```

0.2202381

```
print(sensitivity<- tp/(tp+fn))
```

0.4590164

```
print(FNR<- fn/(tp+fn))
```

0.5409836

```
print(specificity<- tn/(fp+tn))
```

0.9626168

```
print(FPR<- fp/(fp+tn))
```

0.03738318

```
print(precision<- tp/(tp+fp))
```

0.875

```
print(NPV<- tn/(fn+tn))
```

0.7573529

```
print(F1score<- 2*tp/(2*tp+fn+fp))
```

0.6021505

In Python:

```
import numpy
import pandas
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn.model_selection import train_test_split

pneumonia_data=pandas.read_csv('C:/Users/000110888/Desktop/pneumonia_data.csv')
code_gender={'M':1,'F':0}
code_tobacco_use={'yes':1,'no':0}
code_pneumonia={'yes':1,'no':0}

pneumonia_data['gender']=pneumonia_data['gender'].map(code_gender)
pneumonia_data['tobacco_use']=pneumonia_data['tobacco_use'].map(code_tobacco_use)
pneumonia_data['pneumonia']=pneumonia_data['pneumonia'].map(code_pneumonia)

X=pneumonia_data.iloc[:,0:4].values
y=pneumonia_data.iloc[:,4].values

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.20,
random_state=786756)

#FITTING BINARY TREE WITH GINI SPLITTING CRITERION
gini_tree=DecisionTreeClassifier(max_leaf_nodes=6, criterion='gini', random_state=199233)
gini_tree.fit=gini_tree.fit(X_train,y_train)
```

```
#COMPUTING CONFUSION MATRIX AND PERFORMANCE MEASURES FOR TESTING SET
y_pred=gini_tree.predict_proba(X_test)

total=len(y_pred)

tpos=[]
fpos=[]
tneg=[]
fneg=[]

for sub1, sub2 in zip(y_pred[::,1], y_test):
    tpos.append(1) if (sub1>0.5 and sub2==1) else tpos.append(0)
    fpos.append(1) if (sub1>0.5 and sub2==0) else fpos.append(0)
    tneg.append(1) if (sub1<0.5 and sub2==0) else tneg.append(0)
    fneg.append(1) if (sub1<0.5 and sub2==1) else fneg.append(0)
    tp=sum(tpos)
    fp=sum(fpos)
    tn=sum(tneg)
    fn=sum(fneg)

print('tp:', tp)
print('fp:', fp)
print('tn:', tn)
print('fn:', fn)
print('total:', total)

accuracy=(tp+tn)/total
misclassrate=(fp+fn)/total
sensitivity=tp/(tp+fn)
FNR=fn/(tp+fn)
specificity=tn/(fp+tn)
FPR=fp/(fp+tn)
precision=tp/(tp+fp)
NPV=tn/(fn+tn)
F1score=2*tp/(2*tp+fn+fp)

print('accuracy:', accuracy)
print('misclassrate:', misclassrate)
print('sensitivity:', sensitivity)
print('FNR:', FNR)
print('specificity:', specificity)
print('FPR:', FPR)
print('precision:', precision)
print('NPV:', NPV)
print('F1score:', F1score)
```

```
tp: 63
fp: 14
tn: 189
fn: 80
total: 346
```

```
accuracy: 0.7283236994219653
misclassrate: 0.27167630057803466
sensitivity: 0.4405594405594406
FNR: 0.5594405594405595
specificity: 0.9310344827586207
FPR: 0.06896551724137931
precision: 0.8181818181818182
NPV: 0.7026022304832714
F1score: 0.5727272727272728
```

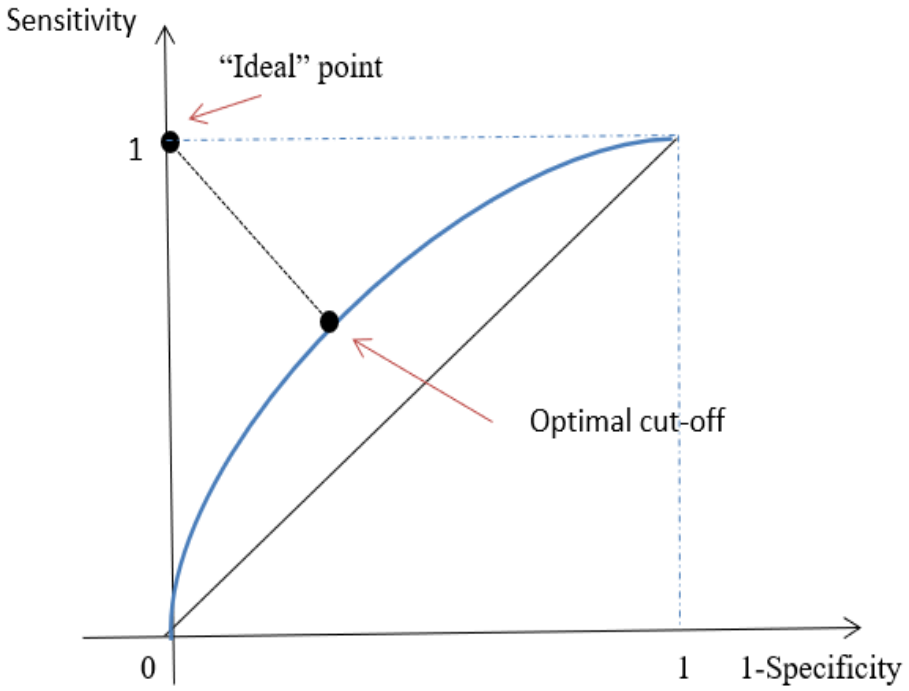
□

RECEIVER OPERATING CHARACTERISTIC (ROC) CURVE

Consider the situation of a binary classification tree. We are given actual observations (yes/no) and predicted probabilities of yes/no. Before we introduced a cut-off, a number between 0 and 1, such that if the predicted probability of "yes" is above the cut-off, then we assume that the predicted value is "yes". We used cut-offs ranging between 0.01 and 0.99 with a step of 0.01 to choose the optimal cutoff that maximizes the true (correct) classification rate (equivalently, minimizes, the misclassification rate). In the previous section, we computed the performance measures assuming the cut-off is 0.5.

A more sophisticated approach relies on the Receiver Operating Characteristic (ROC) curve, schematically presented in the figure below. A ROC curve is a plot of sensitivity (true positive rate) against 1-specificity (false positive rate) for different cut-off points. The cut-off points are often termed *classification thresholds*. A ROC curve connects the origin $(0, 0)$ and the point $(1, 1)$ as a curve that lies above the bisector. Note that the bisector represents a segment on which both sensitivity and specificity are equal to 0.5, corresponding to a random guess.

An ROC curve shows a trade-off between sensitivity and specificity, that is, an increase in sensitivity is accompanied by a decrease in specificity. These two quantities are known to work reciprocally.

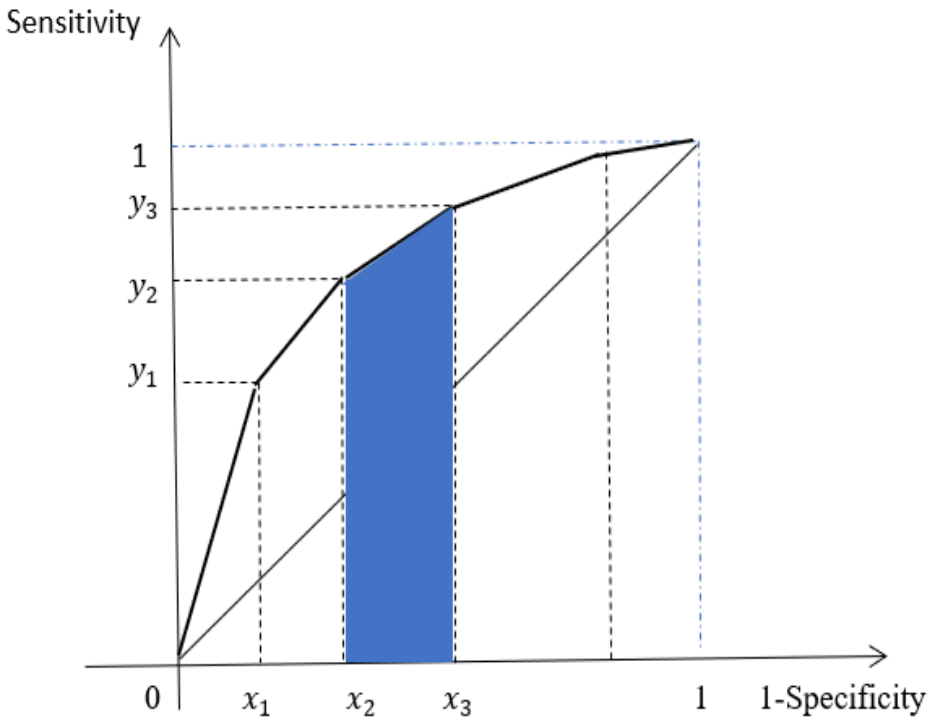


Define the point (0,1) as the "ideal" point. Indeed, at this point sensitivity (true positive rate) and specificity (true negative rate) would both be equal to one, which is clearly a hypothetical situation. Nonetheless, the point on the ROC curve closest to this"ideal" point gives the optimal cut-off for the binary classification.

Area Under the ROC Curve

Often in practice another model performance measure is computed. It is the **area under the curve (AUC)** (or **area under the ROC curve (AUROC)**). The larger the area under the ROC curve, the further the curve is from the bisector, and thus a higher value of AUC indicates a better overall fit of the model (for any classification threshold).

In theory, the ROC curve is smooth, and computing AUC would involve calculus. In practice, however, the cut-offs are chosen on a discrete scale, so the fitted ROC curve is piece-wise linear, and thus, the area can be computed as the sum of areas of the trapezoids (see the figure below). The height of each trapezoid is the distance between two distinct consecutive values of $1 - Specificity$, and the top and bottom sides are distinct consecutive values of $Sensitivity$. Hence, the formula for the area of one trapezoid is $\big((1-Specificity)_2 - (1-Specificity)_1\big)\Big(Sensivity_1 + Sensitivity_2\Big)/2$.

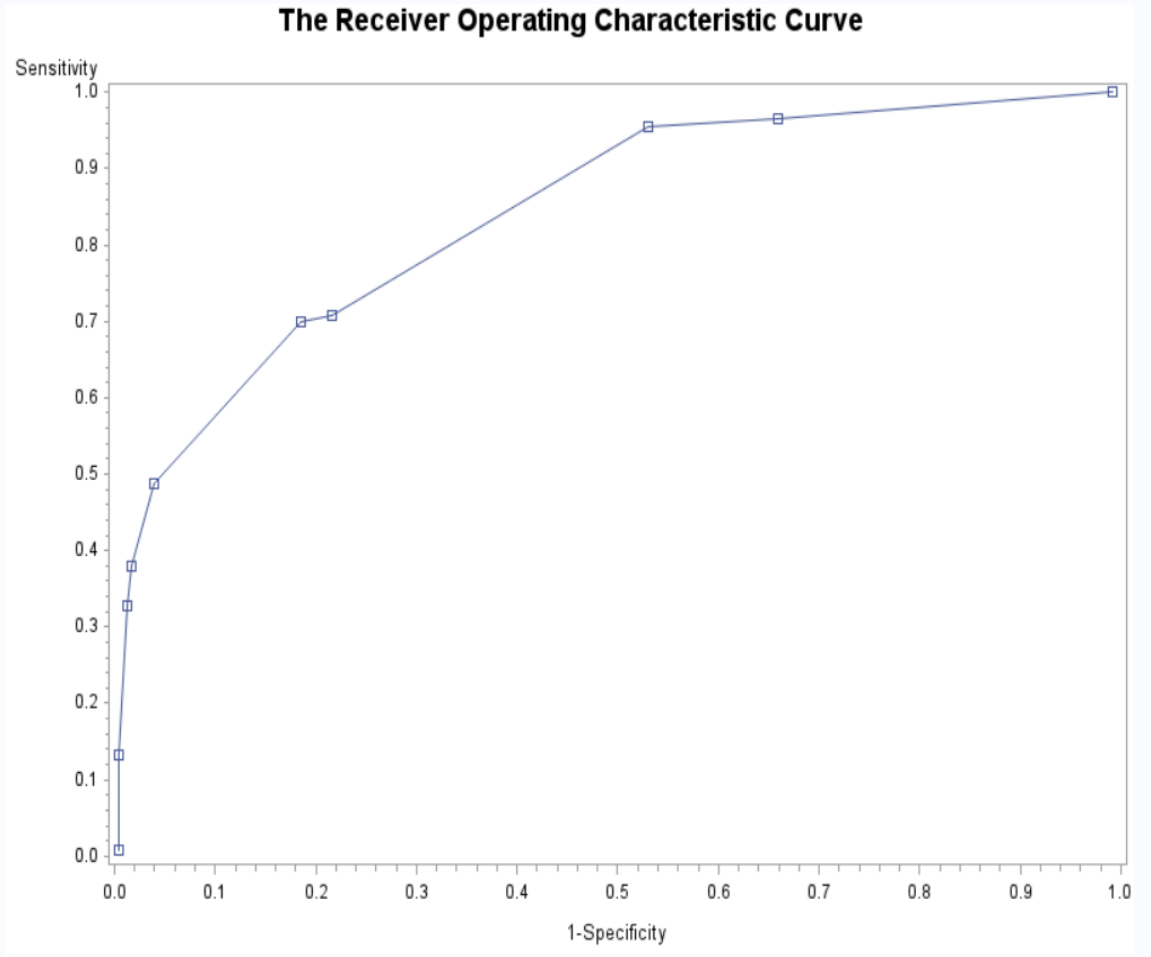

Sensitivity
1
y_3
y_2
y_1
0
x_1
x_2
x_3
1
1-Specificity

Example. Consider the binary classification tree with the Gini splitting algorithm and cost-complexity pruning. We plot the ROC curve and find the optimal cut-off that corresponds to the minimal distance to the "ideal" point.

In SAS:

```
proc import out=pneumonia datafile="./pneumonia_data.csv"
dbms=csv replace;

/*SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS*/
proc surveyselect data=pneumonia rate=0.8 seed=6132208
out=pneumonia outall method=srs;
run;

/*GINI SPLITTING AND COST-COMPLEXITY PRUNING */
proc hpsplit data=pneumonia maxdepth=4;
class pneumonia gender tobacco_use;
   model pneumonia(event="yes")= age gender tobacco_use PM2_5;
grow gini;
prune costcomplexity;
partition rolevar=selected(train="1");
output out=predicted;
ID selected;
run;

/*COMPUTING CONFUSION MATRICES AND PERFORMANCE MEASURES
FOR TESTING SET FOR A RANGE OF CUTOFFS*/
data test;
set predicted;
if(selected="0");
run;

data cutoffs;
set test;
do i=0 to 101;
tp=(P_pneumoniayes >= 0.01*i and pneumonia="yes");
fp=(P_pneumoniayes >= 0.01*i and pneumonia="no");
tn=(P_pneumoniayes <  0.01*i and pneumonia="no");
fn=(P_pneumoniayes <  0.01*i and pneumonia="yes");
output;
end;
run;
```

```
proc sql;
create table confusion as
select i, sum(tp) as tp, sum(fp) as fp, sum(tn) as tn,
sum(fn) as fn, count(*) as total
from cutoffs
group by i;
quit;

proc sql;
create table measures as
select i, (tp+tn)/total as accuracy, (fp+fn)/total as
misclassrate, tp/(tp+fn) as sensitivity, tn/(fp+tn) as specificity,
fp/(fp+tn) as oneminusspec
from confusion
group by i;
quit;

/*PLOTTING ROC CURVE*/
title 'The Receiver Operating Characteristic Curve';
proc gplot data=measures;
symbol v=square interpol=join;
plot sensitivity*oneminusspec/ vaxis=0 to 1 by 0.1 haxis=0 to 1 by 0.1;
label sensitivity="Sensitivity" oneminusspec="1-Specificity";
run;
```




```
/*REPORTING MEASURES FOR THE POINT ON ROC CURVE CLOSEST TO THE IDEAL POINT (0,1)*/
proc sql;
select accuracy, misclassrate, sensitivity, specificity,
sqrt(oneminusspec**2+(1-sensitivity)**2) as distance, i*0.01 as cutoff
from measures
having distance=min(distance);
quit;
```

accuracy	misclassrate	sensitivity	specificity	distance	cutoff
0.776812	0.223188	0.699115	0.814655	0.35339	0.31
0.776812	0.223188	0.699115	0.814655	0.35339	0.32
0.776812	0.223188	0.699115	0.814655	0.35339	0.33
0.776812	0.223188	0.699115	0.814655	0.35339	0.34
0.776812	0.223188	0.699115	0.814655	0.35339	0.35
0.776812	0.223188	0.699115	0.814655	0.35339	0.36
0.776812	0.223188	0.699115	0.814655	0.35339	0.37
0.776812	0.223188	0.699115	0.814655	0.35339	0.38
0.776812	0.223188	0.699115	0.814655	0.35339	0.39
0.776812	0.223188	0.699115	0.814655	0.35339	0.4
0.776812	0.223188	0.699115	0.814655	0.35339	0.41
0.776812	0.223188	0.699115	0.814655	0.35339	0.42
0.776812	0.223188	0.699115	0.814655	0.35339	0.43
0.776812	0.223188	0.699115	0.814655	0.35339	0.44
0.776812	0.223188	0.699115	0.814655	0.35339	0.45
0.776812	0.223188	0.699115	0.814655	0.35339	0.46
0.776812	0.223188	0.699115	0.814655	0.35339	0.47
0.776812	0.223188	0.699115	0.814655	0.35339	0.48

From this output, we see that any cut-off between 0.31 and 0.48 gives the minimal distance between the ROC curve and the "ideal" point.

```
/*COMPUTING AREA UNDER THE ROC CURVE*/
proc sort data=measures;
by oneminusspec;
run;

data AUC;
set measures;
lagx=lag(oneminusspec);
lagy=lag(sensitivity);
if lagx=. then lagx=0;
if lagy=. then lagy=0;
trapezoid=(oneminusspec-lagx)*(sensitivity+lagy)/2;
AUC+trapezoid;
run;
```

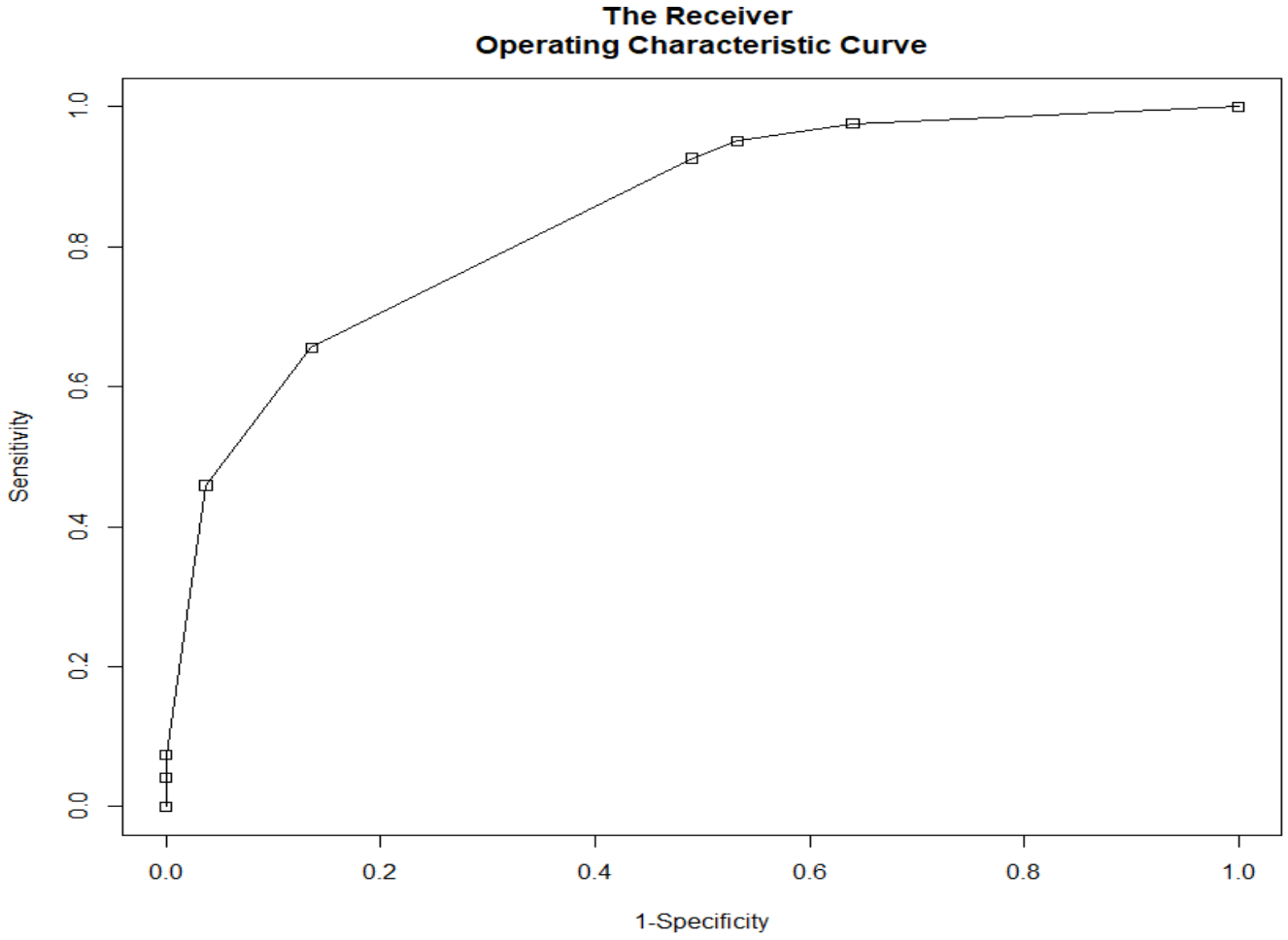
```
proc print data=AUC (firstobs=102) noobs;
var AUC;
run;
```

AUC
0.84082

In R:

```
pneumonia.data<- read.csv(file="./pneumonia_data.csv", header=TRUE, sep=",")

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
set.seed(283605)
sample <- sample(c(TRUE, FALSE), nrow(pneumonia.data), replace=TRUE, prob=c(0.8,0.2))
train<- pneumonia.data[sample,]
test<- pneumonia.data[!sample,]

#FITTING PRUNED BINARY TREE WITH GINI SPLITTING
library(rpart)
tree.gini<- rpart(pneumonia ~ age + gender + tobacco_use + PM2_5, data=train, method="class",
parms=list(split="Gini"), maxdepth=4)

#COMPUTING CONFUSION MATRICES AND PERFORMANCE MEASURES FOR TEST-
ING DATA
#for a range of cut-offs
pred.values<- predict(tree.gini, test)
test<- cbind(test,pred.values)

tpos<- matrix(NA, nrow=nrow(test), ncol=102)
fpos<- matrix(NA, nrow=nrow(test), ncol=102)
tneg<- matrix(NA, nrow=nrow(test), ncol=102)
fneg<- matrix(NA, nrow=nrow(test), ncol=102)

for (i in 0:101) {
    tpos[,i+1]<- ifelse(test$pneumonia=="yes" & test$yes>=0.01*i,1,0)
    fpos[,i+1]<- ifelse(test$pneumonia=="no" & test$yes>=0.01*i, 1,0)
    tneg[,i+1]<- ifelse(test$pneumonia=="no" & test$yes<0.01*i,1,0)
    fneg[,i+1]<- ifelse(test$pneumonia=="yes" & test$yes<0.01*i,1,0)
```

```
}

tp<- c()
fp<- c()
tn<- c()
fn<- c()
accuracy<- c()
misclassrate<- c()
sensitivity<- c()
specificity<- c()
oneminusspec<- c()
cutoff<- c()

for (i in 1:102) {
tp[i]<- sum(tpos[,i])
fp[i]<- sum(fpos[,i])
tn[i]<- sum(tneg[,i])
fn[i]<- sum(fneg[,i])
total<- nrow(test)
accuracy[i]<- (tp[i]+tn[i])/total
misclassrate[i]<- (fp[i]+fn[i])/total
sensitivity[i]<- tp[i]/(tp[i]+fn[i])
specificity[i]<- tn[i]/(fp[i]+tn[i])
oneminusspec[i]<- fp[i]/(fp[i]+tn[i])
cutoff[i]<- 0.01*(i-1)
}

#PLOTTING ROC CURVE
plot(oneminusspec, sensitivity, type="l", lty=1, main="The Receiver Operating Characteristic
Curve", xlab="1-Specificity", ylab="Sensitivity")
points(oneminusspec, sensitivity, pch=0) #pch=plot character, 0=square
```




```
#REPORTING MEASURES FOR THE POINT ON ROC CURVE CLOSEST TO THE IDEAL
POINT (0,1)
distance<- c()
for (i in 1:102)
    distance[i]<- sqrt(oneminusspec[i]^2+(1-sensitivity[i])^2)

measures<- cbind(accuracy, misclassrate, sensitivity, specificity, distance, cutoff)
min.dist<- min(distance)
print(measures[which(measures[,5]==min.dist),])
```

```
 accuracy misclassrate sensitivity specificity  distance cutoff
0.7886905    0.2113095   0.6557377    0.864486 0.3699738   0.30
0.7886905    0.2113095   0.6557377    0.864486 0.3699738   0.31
0.7886905    0.2113095   0.6557377    0.864486 0.3699738   0.32
0.7886905    0.2113095   0.6557377    0.864486 0.3699738   0.33
```

```
0.7886905    0.2113095   0.6557377    0.864486 0.3699738   0.34
0.7886905    0.2113095   0.6557377    0.864486 0.3699738   0.35
0.7886905    0.2113095   0.6557377    0.864486 0.3699738   0.36
0.7886905    0.2113095   0.6557377    0.864486 0.3699738   0.37
0.7886905    0.2113095   0.6557377    0.864486 0.3699738   0.38
0.7886905    0.2113095   0.6557377    0.864486 0.3699738   0.39
0.7886905    0.2113095   0.6557377    0.864486 0.3699738   0.40
0.7886905    0.2113095   0.6557377    0.864486 0.3699738   0.41
0.7886905    0.2113095   0.6557377    0.864486 0.3699738   0.42
0.7886905    0.2113095   0.6557377    0.864486 0.3699738   0.43
0.7886905    0.2113095   0.6557377    0.864486 0.3699738   0.44
0.7886905    0.2113095   0.6557377    0.864486 0.3699738   0.45
```

We can see that for the binary tree built in R, any cut-off between 0.30 and 0.45 is an optimal one.

```
#COMPUTING AREA UNDER THE ROC CURVE
sensitivity<- sensitivity[order(sensitivity)]
oneminusspec<- oneminusspec[order(oneminusspec)]

library(Hmisc) #Harrell Miscellaneous packages
lagx<- Lag(oneminusspec,shift=1)
lagy<- Lag(sensitivity, shift=1)
lagx[is.na(lagx)]<- 0
lagy[is.na(lagy)]<- 0
trapezoid<- (oneminusspec-lagx)*(sensitivity+lagy)/2
print(AUC<- sum(trapezoid))
```

0.8439367

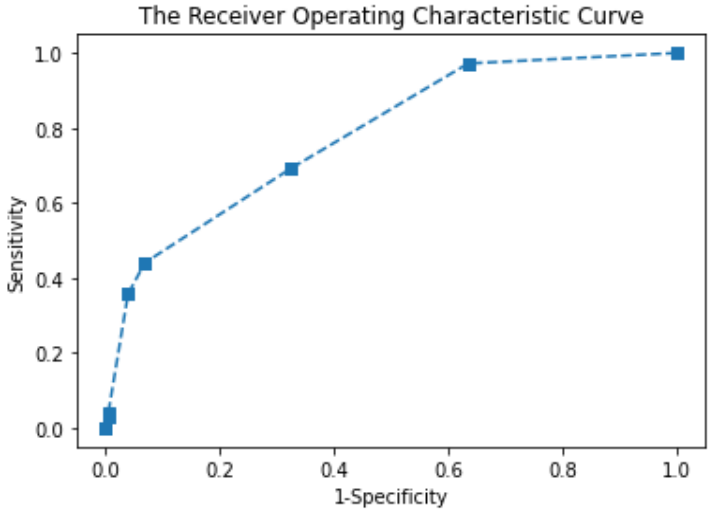
In Python:

```
import numpy
import pandas
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn.model_selection import train_test_split

pneumonia_data=pandas.read_csv('C:/Users/000110888/Desktop/pneumonia_data.csv')
code_gender={'M':1,'F':0}
code_tobacco_use={'yes':1,'no':0}
code_pneumonia={'yes':1,'no':0}

pneumonia_data['gender']=pneumonia_data['gender'].map(code_gender)
pneumonia_data['tobacco_use']=pneumonia_data['tobacco_use'].map(code_tobacco_use)
pneumonia_data['pneumonia']=pneumonia_data['pneumonia'].map(code_pneumonia)

X=pneumonia_data.iloc[:,0:4].values
y=pneumonia_data.iloc[:,4].values

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.20, random_state=786756)

#FITTING BINARY TREE WITH GINI SPLITTING CRITERION
gini_tree=DecisionTreeClassifier(max_leaf_nodes=6, criterion='gini', random_state=199233)
gini_tree.fit=gini_tree.fit(X_train,y_train)
```

```
#COMPUTING CONFUSION MATRICES AND PERFORMANCE MEASURES FOR TESTING SET FOR A RANGE OF CUTOFFS
y_pred=gini_tree.predict_proba(X_test)

total=len(y_pred)

cutoff=[]
accuracy=[]
misclassrate=[]
sensitivity=[]
specificity=[]
oneminusspec=[]
distance=[]

for i in range(99):
    tp=0
    fp=0
    tn=0
    fn=0
    cutoff.append(0.01*(i+1))
    for sub1, sub2 in zip(y_pred[::,1], y_test):
        tp_ind=1 if (sub1>0.01*(i+1) and sub2==1) else 0
        fp_ind=1 if (sub1>0.01*(i+1) and sub2==0) else 0
        tn_ind=1 if (sub1<0.01*(i+1) and sub2==0) else 0
        fn_ind=1 if (sub1<0.01*(i+1) and sub2==1) else 0
        tp+=tp_ind
        fp+=fp_ind
        tn+=tn_ind
        fn+=fn_ind
```

```
    accuracy_i=(tp+tn)/total
    misclassrate_i=(fp+fn)/total
    sensitivity_i=tp/(tp+fn)
    specificity_i=tn/(fp+tn)
    oneminusspec_i=fp/(fp+tn)
    distance_i=numpy.sqrt(pow(oneminusspec_i,2)+pow(1-sensitivity_i,2))

    accuracy.append(accuracy_i)
    misclassrate.append(misclassrate_i)
    sensitivity.append(sensitivity_i)
    specificity.append(specificity_i)
    oneminusspec.append(oneminusspec_i)
    distance.append(distance_i)

#PLOTTING ROC CURVE
import matplotlib.pyplot as plot
plt.plot(oneminusspec, sensitivity, linestyle='--', marker='s')
plt.title('The Receiver Operating Characteristic Curve')
plt.xlabel('1-Specificity')
plt.ylabel('Sensitivity')

#REPORTING MEASURES FOR THE POINT ON ROC CURVE CLOSEST TO THE IDEAL POINT (0,1)
df=pandas.DataFrame({'accuracy': accuracy,'misclassrate': misclassrate,'sensitivity':
sensitivity,'specificity': specificity, 'oneminusspec': oneminusspec,'distance': distance,'cut-off': cutoff})
min_distance=min(distance)
optimal=df[df['distance']==min_distance]
print(optimal)

#COMPUTING AREA UNDER THE ROC CURVE
df=df.sort_values('oneminusspec', ascending=True)
df['lagx']=df['oneminusspec'].shift(1)
df['lagy']=df['sensitivity'].shift(1)
df['lagx']=numpy.nan_to_num(df['lagx'],nan=0)
df['lagy']=numpy.nan_to_num(df['lagy'],nan=0)
df['trapezoid']=(df['oneminusspec']-df['lagx'])*(df['sensitivity']+df['lagy'])/2;
AUC=sum(df['trapezoid'])
print(AUC)
```

```
    accuracy  misclassrate  sensitivity  specificity  oneminusspec  distance
27  0.682081      0.317919     0.692308     0.674877      0.325123  0.447638
28  0.682081      0.317919     0.692308     0.674877      0.325123  0.447638
29  0.682081      0.317919     0.692308     0.674877      0.325123  0.447638
30  0.682081      0.317919     0.692308     0.674877      0.325123  0.447638
31  0.682081      0.317919     0.692308     0.674877      0.325123  0.447638
32  0.682081      0.317919     0.692308     0.674877      0.325123  0.447638
33  0.682081      0.317919     0.692308     0.674877      0.325123  0.447638

    cut-off
27     0.28
28     0.29
29     0.30
30     0.31
31     0.32
32     0.33
33     0.34
0.7814776561697745
```



From this output, we see that any cut-off between 0.28 and 0.34 gives the minimal distance between the ROC curve and the "ideal" point. The area under the ROC curve is about 0.78. □

MULTINOMIAL CLASSIFICATION TREE

A **multi-class** (or **multinomial**) **classification tree** classifies observations into one of three or more classes. The same algorithms as for the binary classification tree apply to this case as well.

The output for prediction contains predicted probabilities of each of the multiple classes. We assume that the class with the highest predicted probability is the class actually predicted by the fitted model.

Performance Measures for Individual Classes

The performance measures used for binary classification can be extended to a multi-class classification. Unlike binary classification, there are no positive or negative classes but we can compute TP, TN, FP, and FN for each individual class.

For example, in a data set, there are 50 greens, 40 blues, and 10 reds. The predicted colors are summarized in the following confusion matrix:

True color	Predicted Color		
	green	blue	red
green	35	5	10
blue	5	30	5
red	3	2	5

We consider each color separately and compute all the performance measures. For the green color, the 2-by-2 confusion matrix has the form:

True color	Predicted Color	
	green	not green
green	35	15
not green	8	42

Now we compute the performance measures for this confusion matrix: $TP = 35, TN = 42, FP = 8, FN = 15$, Accuracy $= (TP+TN)/(TP+TN+FP+FN) = (35+42)/100 = 0.77$, Misclassification Rate $= (FP+FN)/(TP+TN+FP+FN) = 1-$Accuracy $= 0.23$, Sensitivity $= TP/(TP+FN) = 35/(35+15) = 0.70$, FNR $= FN/(TP+FN) = 1-$ Sensitivity $= 0.30$, Specificity $= TN/(FP+TN) = 42/(8+42) = 0.84$, FPR $= FP/(FP+TN) = 1-$Specificity $= 0.16$, Precision $= TP/(TP+FP) = 35/(35+8) = 0.813953$, NPV $= TN/(FN+TN) = 42/(15+42) = 0.736842$, F1-score $= \dfrac{2TP}{2TP+FN+FP} = \dfrac{(2)(35)}{(2)(35)+15+8} = 0.752688.$

For the blue color, the 2-by-2 confusion matrix is

True color	Predicted Color	
	blue	not blue
blue	30	10
not blue	7	53

The performance measures for this binary classification are: $TP = 30, TN = 53, FP = 7, FN = 10$, Accuracy $= (TP+TN)/(TP+TN+FP+FN) = (30+53)/100 = 0.83$, Misclassification

Rate $= (FP+FN)/(TP+TN+FP+FN) = 1-\text{Accuracy} = 0.17, \text{Sensitivity} = TP/(TP+FN) = 30/(30+10) = 0.75, \text{FNR} = FN/(TP+FN) = 1-\text{Sensitivity} = 0.25, \text{Specificity} = TN/(FP+TN) = 53/(7+53) = 0.95, \text{FPR} = FP/(FP+TN) = 1-\text{Specificity} = 0.05, \text{Precision} = TP/(TP+FP) = 30/(30+7) = 0.810811, \text{NPV} = TN/(FN+TN) = 53/(10+53) = 0.84127, \text{F1-score} = \dfrac{2TP}{2TP+FN+FP} = \dfrac{(2)(30)}{(2)(30)+10+7} = 0.779221.$

Finally, for the red color, the 2-by-2 confusion matrix is

True color	Predicted Color	
	red	not red
red	5	5
not red	15	75

Now we compute the performance measures for this confusion matrix: $TP = 5, TN = 75, FP = 15, FN = 5, \text{Accuracy} = (TP+TN)/(TP+TN+FP+FN) = (5+75)/100 = 0.80, \text{Misclassification Rate} = (FP+FN)/(TP+TN+FP+FN) = 1-\text{Accuracy} = 0.20, \text{Sensitivity} = TP/(TP+FN) = 5/(5+5) = 0.50, \text{FNR} = FN/(TP+FN) = 1-\text{Sensitivity} = 0.50, \text{Specificity} = TN/(FP+TN) = 75/(15+75) = 0.8333, \text{FPR} = FP/(FP+TN) = 1-\text{Specificity} = 0.1667, \text{Precision} = TP/(TP+FP) = 5/(5+15) = 0.25, \text{NPV} = TN/(FN+TN) = 75/(5+75) = 0.9375, \text{F1-score} = \dfrac{2TP}{2TP+FN+FP} = \dfrac{(2)(5)}{(2)(5)+5+15} = 0.3333.$

Micro Measures

Turning now to the multinomial model, we compute performance measures based on total $TP = 35+30+5 = 70$, total $TN = 42+53+75 = 170$, total $FP = 8+7+15 = 30$, and total $FN = 15+10+5 = 30$. These are global measures for the whole model, called **micro-averaged** measures. The micro-averaged measures are: $\text{Accuracy} = (TP+TN)/(TP+TN+FP+FN) = (70+170)/300 = 0.80, \text{Misclassification Rate} = 1-\text{Accuracy} = 0.20, \text{Sensitivity} = TP/(TP+FN) = 70/(70+30) = 0.70, \text{FNR} = 1-\text{Sensitivity} = 0.30, \text{Specificity} = TN/(FP+TN) = 170/(30+170) = 0.85, \text{FPR} = 1-\text{Specificity} = 0.15, \text{Precision} = TP/(TP+FP) = 70/(70+30) = 0.70, \text{NPV} = TN/(FN+TN) = 170/(30+170) = 0.85, \text{F1-score} = \dfrac{2TP}{2TP+FN+FP} = \dfrac{(2)(70)}{(2)(70)+30+30} = 0.70.$

Macro Measures

Macro measures are computed as an arithmetic average of individual measures for each class. The macro measures are: $\text{Accuracy} = (0.77+0.83+0.80)/3 = 0.80, \text{Misclassification Rate} = 1-\text{Accuracy} = 0.20, \text{Sensitivity} = (0.70+0.75+0.50)/3 = 0.65, \text{FNR} = 1-\text{Sensitivity} = 0.35, \text{Specificity} = (0.84+0.95+0.8333)/3 = 0.87, \text{FPR} = 1-\text{Specificity} = 0.13, \text{Precision} =$

$(0.813953 + 0.810811 + 0.25)/3 = 0.624921, \text{NPV} = (0.736842 + 0.84127 + 0.9375)/3 = 0.838537,$
$\text{F1-score} = (0.752688 + 0.779221 + 0.3333)/3 = 0.621736.$

Weighted Macro Measures

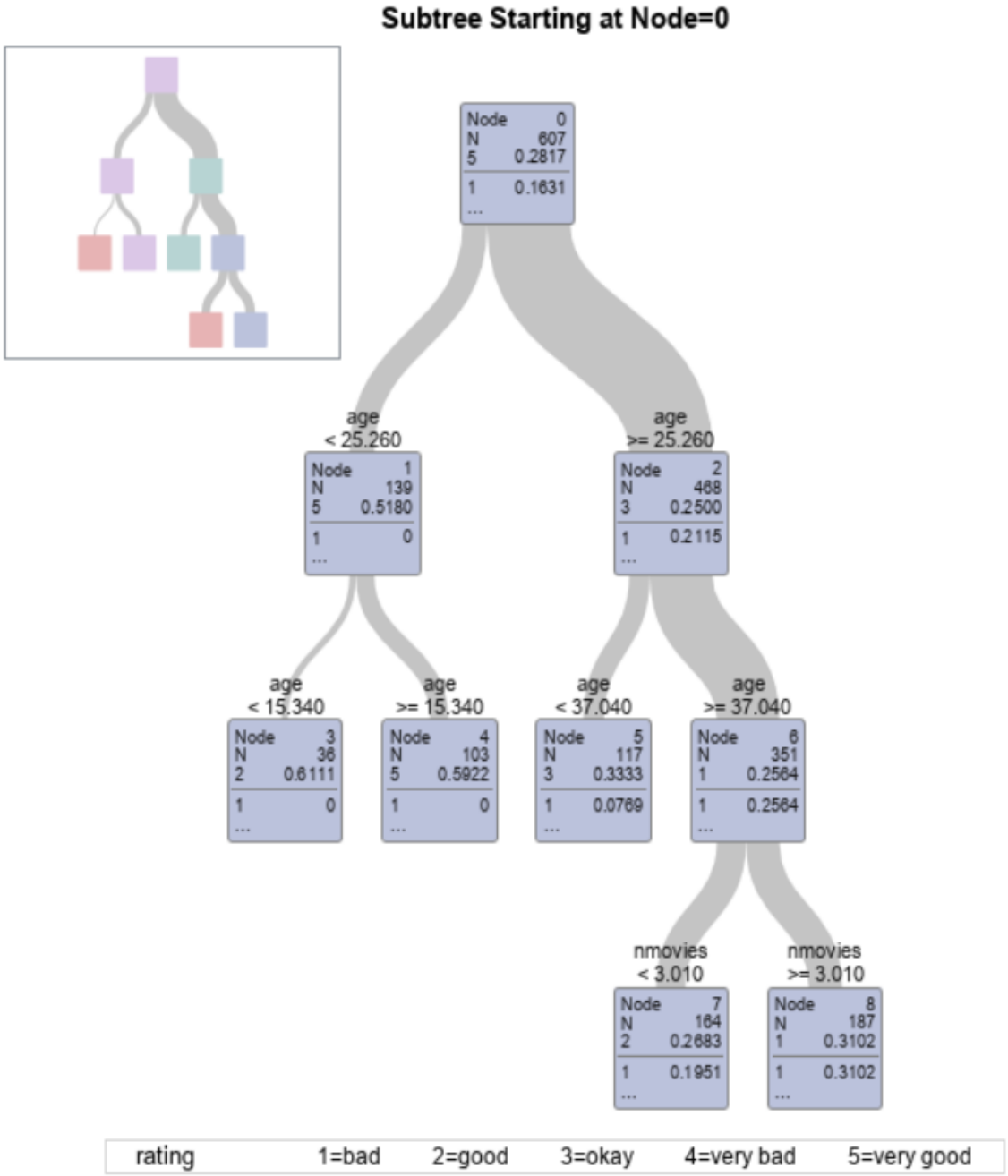
The **weighted macro** measures are weighted means of the measures for individual classes, where weights are proportional to the total number of samples in the class. There are 50 greens, 40 blues, and 10 reds, therefore, the weights are 50/(50+40+10)=0.5 for greens, 40/100=0.4 for blues, and 10/100=0.1 for reds. The weighted macro measures are computed as: $\text{Accuracy} = (0.77)(0.5) + (0.83)(0.4) + (0.80)(0.1) = 0.797, \text{Misclassification Rate} = 1 - \text{Accuracy} = 0.203, \text{Sensitivity} = (0.70)(0.5)+(0.75)(0.4)+(0.50)(0.1) = 0.70, \text{FNR} = 1-\text{Sensitivity} = 0.30, \text{Specificity} = (0.84)(0.5)+(0.95)(0.4)+(0.8333)(0.1) = 0.88333, \text{FPR} = 1-\text{Specificity} = 0.11667, \text{Precision} = (0.813953)(0.5)+(0.810811)(0.4) + (0.25)(0.1) = 0.756301, \text{NPV} = (0.736842)(0.5) + (0.84127)(0.4) + (0.9375)(0.1) = 0.798679, \text{F1-score} = (0.752688)(0.5) + (0.779221)(0.4) + (0.3333)(0.1) = 0.721362.$

Example. The data set "movie_data.csv" contains data on movie-goers (age, gender, membership, and the number of movies watched in the previous 4 weeks), and their rating of a new movie (very bad/bad/okay/good/very good). There are 758 rows in this data set. We split the data into 80% training and 20% testing sets, and fit a multinomial classification tree using Gini, entropy, and CHAID splitting criteria and the cost-complexity pruning algorithm. We then compute performance measures for individual classes, micro measures, macro measures, and weighted macro measures, and compare the three models based on these measures.

In SAS:

```
proc import out=movie
datafile="./movie_data.csv" dbms=csv replace;
run;

/*SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS*/
proc surveyselect data=movie rate=0.8 seed=118607
out=movie outall method=srs;
run;

/*GINI SPLITTING AND COST-COMPLEXITY PRUNING */
proc hpsplit data=movie;
class rating gender member;
   model rating=age gender member nmovies;
grow gini;
prune costcomplexity;
partition rolevar=selected(train="1");
```

```
output out=predicted;
ID selected;
run;
```




```
/*MACRO FOR COMPUTING PERFORMANCE MEASURES*/
```

```
%macro perf_measures(dataset);

/*computing confusion matrix*/
data test;
set predicted;
if(selected="0");
maxprob=max(P_ratingbad, P_ratinggood, P_ratingokay,
P_ratingvery_bad, P_ratingvery_good);
if maxprob=P_ratingvery_good then predclass='very good';
if maxprob=P_ratinggood then predclass='good';
if maxprob=P_ratingokay then predclass='okay';
if maxprob=P_ratingbad then predclass='bad';
if maxprob=P_ratingvery_bad then predclass='very bad';
run;

/*computing total number of rows in test set*/
proc sql;
create table totalrows as
select count(*) as nrows
from test;
quit;

data _null_;
set totalrows;
 call symput('totalrows', nrows);
run;

/*computing performance measures for individual classes*/
%macro class_metrics(class);
data indiv_class;
set test;
tp=(predclass=&class and rating=&class);
fp=(predclass=&class and rating ne &class);
tn=(predclass ne &class and rating ne &class);
fn=(predclass ne &class and rating=&class);
run;

proc sql;
create table confusion as
select sum(tp) as tp, sum(fp) as fp, sum(tn) as tn,
sum(fn) as fn, count(*) as total
from indiv_class;
```

```
quit;

proc sql;
create table measures as
select &class as class, tp, fp, tn, fn,
(tp+tn)/total as accuracy, (fp+fn)/total as
misclassrate, tp/(tp+fn) as sensitivity,
fn/(tp+fn) as FNR, tn/(fp+tn) as specificity,
fp/(fp+tn) as FPR, tp/(tp+fp) as precision,
tn/(fn+tn) as NPV, 2*tp/(2*tp+fn+fp) as F1score
from confusion;
select * from measures;
quit;

proc append base=&dataset data=measures;
run;

%mend;

%class_metrics('very bad')
%class_metrics('bad')
%class_metrics('okay')
%class_metrics('good')
%class_metrics('very good')

/*computing micro measures*/
proc sql;
create table totals as
select sum(tp) as tp, sum(fp) as fp, sum(tn) as tn,
sum(fn) as fn
from &dataset;
quit;

proc sql;
select 'micro measures',(tp+tn)/(tp+fp+tn+fn)
as accuracy, (fp+fn)/(tp+fp+tn+fn)
as misclassrate, tp/(tp+fn) as sensitivity,
fn/(tp+fn) as FNR, tn/(fp+tn) as specificity,
fp/(fp+tn) as FPR, tp/(tp+fp) as precision,
tn/(fn+tn) as NPV, 2*tp/(2*tp+fn+fp) as F1score
from totals;
quit;
```

```
/*computing macro measures*/
proc sql;
select 'macro measures', mean(accuracy) as accuracy,
mean(misclassrate) as misclassrate, mean(sensitivity) as sensitivity,
mean(FNR) as FNR, mean(specificity) as specificity,
mean(FPR) as FPR, mean(precision) as precision,
mean(NPV) as NPV, mean(F1score) as F1score
from &dataset;
quit;

/*computing weighted macro measures*/
data &dataset;
set &dataset;
weight=(tp+fn)/&totalrows;
w_accuracy=accuracy*weight;
w_misclassrate=misclassrate*weight;
w_sensitivity=sensitivity*weight;
w_FNR=FNR*weight;
w_specificity=specificity*weight;
w_FPR=FPR*weight;
w_precision=precision*weight;
w_FPR=FPR*weight;
w_precision=precision*weight;
w_NPV=NPV*weight;
w_F1score=F1score*weight;
run;

proc sql;
select 'weighted macro measures',sum(w_accuracy)
as accuracy, sum(w_misclassrate) as misclassrate,
sum(w_sensitivity) as sensitivity,
sum(w_FNR) as FNR, sum(w_specificity) as specificity,
sum(w_FPR) as FPR, sum(w_precision) as precision,
sum(w_NPV) as NPV, sum(w_F1score) as F1score
from &dataset;
quit;

%mend;

/*COMPUTING PERFORMANCE MEASURES FOR FITTED GINI TREE*/
%perf_measures(ginitree)
```

The SAS System

class	tp	fp	tn	fn	accuracy	misclassrate	sensitivity	FNR	specificity	FPR	precision	NPV	F1score
very bad	0	0	143	8	0.94702	0.05298	0	1	1	0	.	0.94702	0

The SAS System

class	tp	fp	tn	fn	accuracy	misclassrate	sensitivity	FNR	specificity	FPR	precision	NPV	F1score
bad	14	36	96	5	0.728477	0.271523	0.736842	0.263158	0.727273	0.272727	0.28	0.950495	0.405797

The SAS System

class	tp	fp	tn	fn	accuracy	misclassrate	sensitivity	FNR	specificity	FPR	precision	NPV	F1score
okay	9	52	79	11	0.582781	0.417219	0.45	0.55	0.603053	0.396947	0.147541	0.877778	0.222222

The SAS System

class	tp	fp	tn	fn	accuracy	misclassrate	sensitivity	FNR	specificity	FPR	precision	NPV	F1score
good	6	8	98	39	0.688742	0.311258	0.133333	0.866667	0.924528	0.075472	0.428571	0.715328	0.20339

The SAS System

class	tp	fp	tn	fn	accuracy	misclassrate	sensitivity	FNR	specificity	FPR	precision	NPV	F1score
very good	16	10	82	43	0.649007	0.350993	0.271186	0.728814	0.891304	0.108696	0.615385	0.656	0.376471

The SAS System

	accuracy	misclassrate	sensitivity	FNR	specificity	FPR	precision	NPV	F1score
micro measures	0.736755	0.263245	0.315217	0.684783	0.8125	0.1875	0.232	0.868476	0.267281

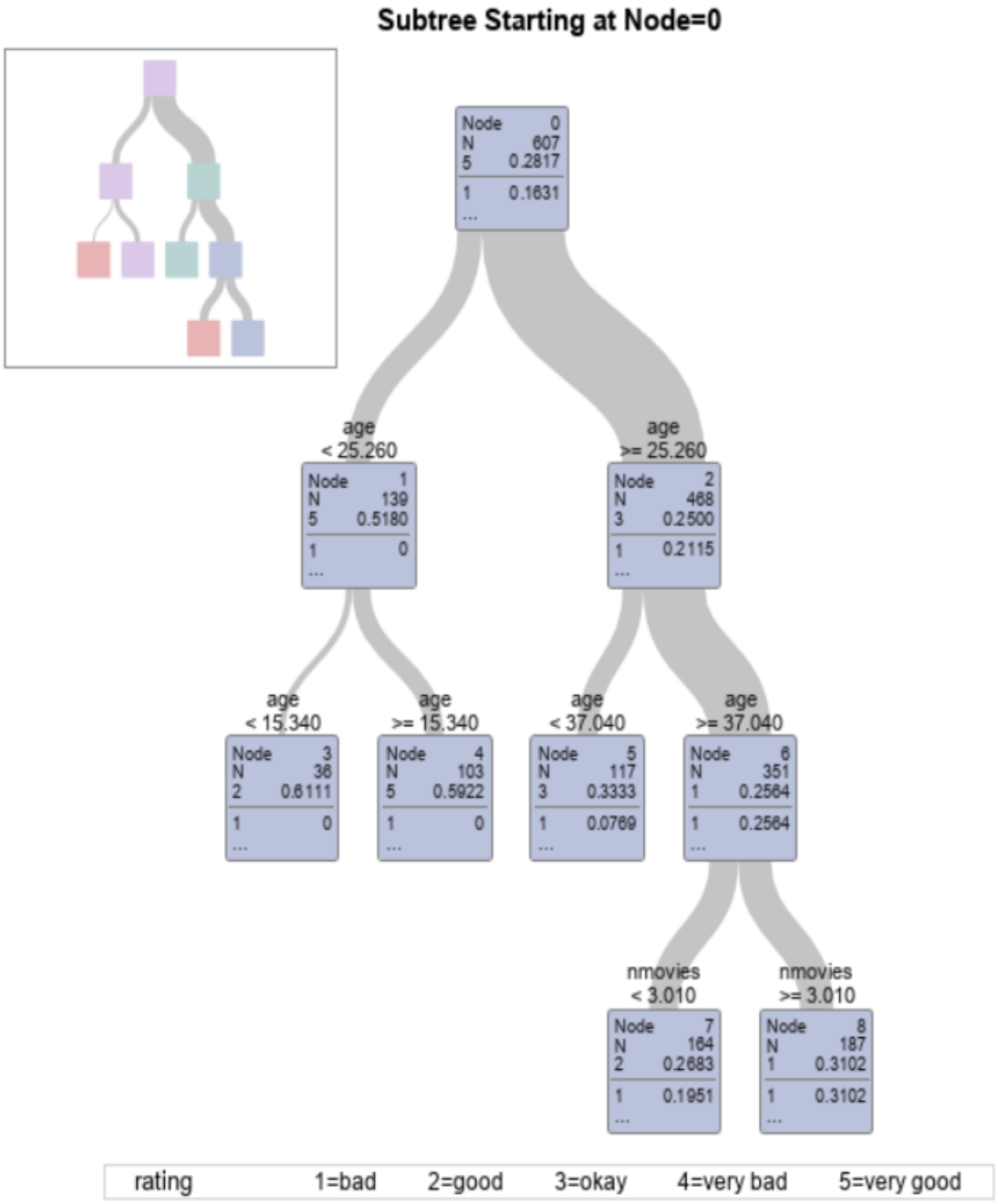
The SAS System

	accuracy	misclassrate	sensitivity	FNR	specificity	FPR	precision	NPV	F1score
macro measures	0.736755	0.263245	0.330044	0.669956	0.813714	0.186286	0.285371	0.872655	0.207852

The SAS System

	accuracy	misclassrate	sensitivity	FNR	specificity	FPR	precision	NPV	F1score
weighted macro measures	0.42428	0.184992	0.192053	0.417219	0.499888	0.109384	0.182494	0.499211	0.141107

```
/*ENTROPY SPLITTING AND COST-COMPLEXITY PRUNING*/
proc hpsplit data=movie;
class rating gender member;
   model rating=age gender member nmovies;
grow entropy;
prune costcomplexity;
partition rolevar=selected(train="1");
output out=predicted;
ID selected;
run;
```




```
/*COMPUTING PERFORMANCE MEASURES FOR FITTED ENTROPY TREE*/
%perf_measures(entropytree)
```

The SAS System

class	tp	fp	tn	fn	accuracy	misclassrate	sensitivity	FNR	specificity	FPR	precision	NPV	F1score
very bad	0	0	143	8	0.94702	0.05298	0	1	1	0	.	0.94702	0

The SAS System

class	tp	fp	tn	fn	accuracy	misclassrate	sensitivity	FNR	specificity	FPR	precision	NPV	F1score
bad	14	36	96	5	0.728477	0.271523	0.736842	0.263158	0.727273	0.272727	0.28	0.950495	0.405797

The SAS System

class	tp	fp	tn	fn	accuracy	misclassrate	sensitivity	FNR	specificity	FPR	precision	NPV	F1score
okay	9	52	79	11	0.582781	0.417219	0.45	0.55	0.603053	0.396947	0.147541	0.877778	0.222222

The SAS System

class	tp	fp	tn	fn	accuracy	misclassrate	sensitivity	FNR	specificity	FPR	precision	NPV	F1score
good	6	8	98	39	0.688742	0.311258	0.133333	0.866667	0.924528	0.075472	0.428571	0.715328	0.20339

The SAS System

class	tp	fp	tn	fn	accuracy	misclassrate	sensitivity	FNR	specificity	FPR	precision	NPV	F1score
very good	16	10	82	43	0.649007	0.350993	0.271186	0.728814	0.891304	0.108696	0.615385	0.656	0.376471

The SAS System

	accuracy	misclassrate	sensitivity	FNR	specificity	FPR	precision	NPV	F1score
micro measures	0.736755	0.263245	0.315217	0.684783	0.8125	0.1875	0.232	0.868476	0.267281

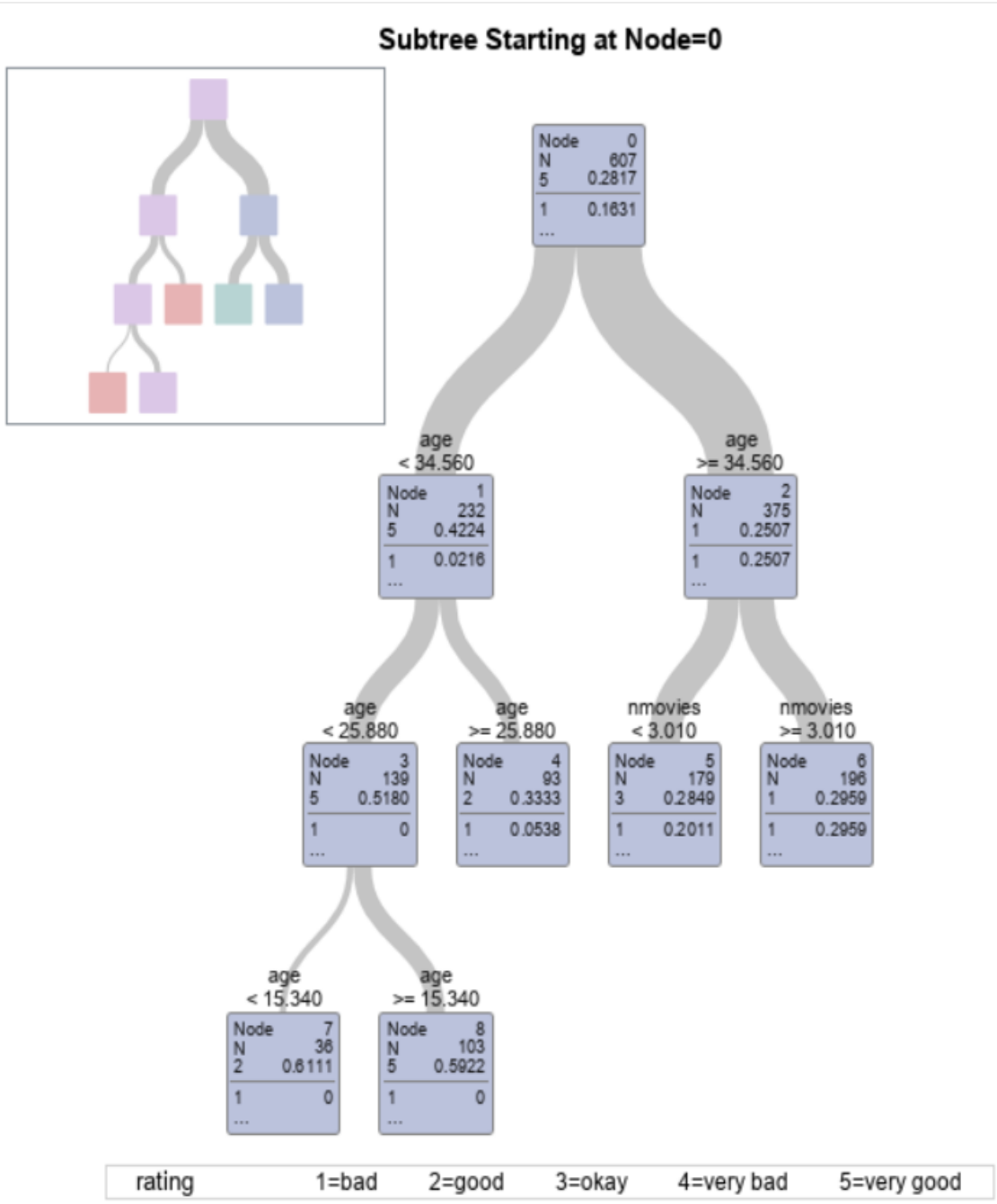
The SAS System

	accuracy	misclassrate	sensitivity	FNR	specificity	FPR	precision	NPV	F1score
macro measures	0.736755	0.263245	0.330044	0.669956	0.813714	0.186286	0.285371	0.872655	0.207852

The SAS System

	accuracy	misclassrate	sensitivity	FNR	specificity	FPR	precision	NPV	F1score
weighted macro measures	0.42428	0.184992	0.192053	0.417219	0.499888	0.109384	0.182494	0.499211	0.141107

```
/*CHAID SPLITTING AND COST-COMPLEXITY PRUNING*/
proc hpsplit data=movie;
class rating gender member;
   model rating=age gender member nmovies;
grow CHAID;
prune costcomplexity;
partition rolevar=selected(train="1");
output out=predicted;
ID selected;
run;
```




```
/*COMPUTING PERFORMANCE MEASURES FITTED CHAID TREE*/
%perf_measures(CHAIDtree)
```

The SAS System

class	tp	fp	tn	fn	accuracy	misclassrate	sensitivity	FNR	specificity	FPR	precision	NPV	F1score
very bad	0	0	143	8	0.94702	0.05298	0	1	1	0	.	0.94702	0

The SAS System

class	tp	fp	tn	fn	accuracy	misclassrate	sensitivity	FNR	specificity	FPR	precision	NPV	F1score
bad	14	39	93	5	0.708609	0.291391	0.736842	0.263158	0.704545	0.295455	0.264151	0.94898	0.388889

The SAS System

class	tp	fp	tn	fn	accuracy	misclassrate	sensitivity	FNR	specificity	FPR	precision	NPV	F1score
okay	9	49	82	11	0.602649	0.397351	0.45	0.55	0.625954	0.374046	0.155172	0.88172	0.230769

The SAS System

class	tp	fp	tn	fn	accuracy	misclassrate	sensitivity	FNR	specificity	FPR	precision	NPV	F1score
good	6	8	98	39	0.688742	0.311258	0.133333	0.866667	0.924528	0.075472	0.428571	0.715328	0.20339

The SAS System

class	tp	fp	tn	fn	accuracy	misclassrate	sensitivity	FNR	specificity	FPR	precision	NPV	F1score
very good	16	10	82	43	0.649007	0.350993	0.271186	0.728814	0.891304	0.108696	0.615385	0.656	0.376471

The SAS System

	accuracy	misclassrate	sensitivity	FNR	specificity	FPR	precision	NPV	F1score
micro measures	0.736755	0.263245	0.315217	0.684783	0.8125	0.1875	0.232	0.868476	0.267281

The SAS System

	accuracy	misclassrate	sensitivity	FNR	specificity	FPR	precision	NPV	F1score
macro measures	0.736755	0.263245	0.330044	0.669956	0.813757	0.186243	0.282632	0.873262	0.205762

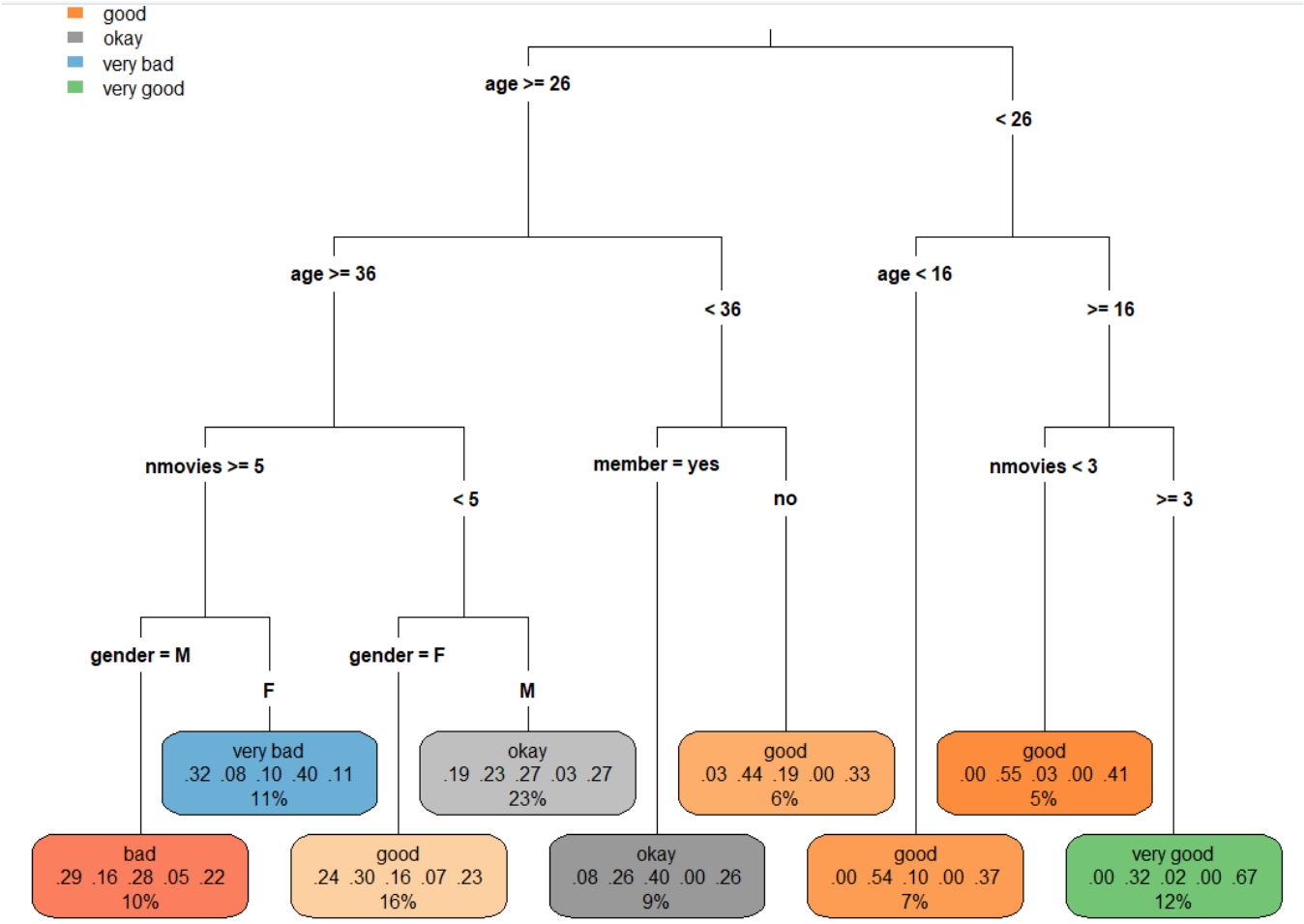
The SAS System

	accuracy	misclassrate	sensitivity	FNR	specificity	FPR	precision	NPV	F1score
weighted macro measures	0.424411	0.18486	0.192053	0.417219	0.500061	0.10921	0.18151	0.499543	0.140111

In R:

```
movie.data<- read.csv(file="./movie_data.csv", header=TRUE, sep=",")

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
set.seed(566222)
sample <- sample(c(TRUE, FALSE), nrow(movie.data),replace=TRUE, prob=c(0.8,0.2))
train<- movie.data[sample,]
test<- movie.data[!sample,]

#FITTING PRUNED MULTINOMIAL CLASSIFICATION TREE WITH GINI SPLITTING
library(rpart)
tree.gini<- rpart(rating ~ age + gender + member + nmovies, data=train, method="class",
```

```
parms=list(split="Gini"), maxdepth=4)

#PLOTTING FITTED TREE
library(rpart.plot)
rpart.plot(tree.gini, type=3)
```




```
#COMPUTING PREDICTED VALUES FOR TESTING DATA
pred.values<- predict(tree.gini, test)

#DETERMINING PREDICTED CLASSES
test<- cbind(test, pred.values)
test$maxprob<- pmax(test$'very bad',test$'bad',test$'okay', test$'good',test$'very good')

test$predclass<- ifelse(test$maxprob==test$'very bad', 'very bad', ifelse(test$maxprob==test$'bad','bad',
ifelse(test$maxprob==test$'okay','okay', ifelse(test$maxprob==test$'good','good','very good'))))
```

```
#DEFINING FUNCTION FOR COMPUTING PERFORMANCE MEASURES
perf.measures<- function() {

#COMPUTING PERFORMANCE MEASURES FOR INDIVIDUAL CLASSES
tp<- c()
fp<- c()
tn<- c()
fn<- c()
accuracy<- c()
misclassrate<- c()
sensitivity<- c()
FNR<- c()
specificity<- c()
FPR<- c()
precision<- c()
NPV<- c()
F1score<- c()

class.metrics<- function(class) {

    tp.class<- ifelse(test$predclass==class & test$liked==class,1,0)
    fp.class<- ifelse(test$predclass==class & test$liked!=class,1,0)
    tn.class<- ifelse(test$predclass!=class & test$liked!=class,1,0)
    fn.class<- ifelse(test$predclass!=class & test$liked==class,1,0)

    message('CLASS MEASURES:')
    message('class:', class)
    print(paste('tp:', tp[class]«- sum(tp.class)))
    #«- is global assignment, works outside the function
    print(paste('fp:', fp[class]«- sum(fp.class)))
    print(paste('tn:', tn[class]«- sum(tn.class)))
    print(paste('fn:', fn[class]«- sum(fn.class)))
    total«- nrow(test)

    print(paste('accuracy:', accuracy[class]«- (tp[class]+tn[class])/total))
    print(paste('misclassrate:', misclassrate[class]«- (fp[class]+fn[class])/total))
    print(paste('sensitivity:', sensitivity[class]«- tp[class]/(tp[class]+fn[class])))
    print(paste('FNR:', FNR[class]«- fn[class]/(tp[class]+fn[class])))
    print(paste('specificity:', specificity[class]«- tn[class]/(fp[class]+tn[class])))
    print(paste('FPR:', FPR[class]«- fp[class]/(fp[class]+tn[class])))
    print(paste('precision:', precision[class]«- tp[class]/(tp[class]+fp[class])))
```

```
    print(paste('NPV:', NPV[class]<- tn[class]/(fn[class]+tn[class])))
    print(paste('F1score:', F1score[class]<- 2*tp[class]/(2*tp[class]+fn[class]+fp[class])))
}

class.metrics(class='very bad')
class.metrics(class='bad')
class.metrics(class='okay')
class.metrics(class='good')
class.metrics(class='very good')

#COMPUTING MICRO MEASURES
tp.sum<- sum(tp)
fp.sum<- sum(fp)
tn.sum<- sum(tn)
fn.sum<- sum(fn)

message('MICRO MEASURES:')
print(paste('accuracy:', accuracy.micro<- (tp.sum+tn.sum)/(tp.sum+fp.sum+tn.sum+fn.sum)))
print(paste('misclassrate:', misclassrate.micro<- (fp.sum+fn.sum)/(tp.sum+fp.sum+tn.sum+fn.sum)))
print(paste('sensitivity:', sensitivity.micro<- tp.sum/(tp.sum+fn.sum)))
print(paste('FNR:', FNR.micro<- fn.sum/(tp.sum+fn.sum)))
print(paste('specificity:', specificity.micro<- tn.sum/(fp.sum+tn.sum)))
print(paste('FPR:', FPR.micro<- fp.sum/(fp.sum+tn.sum)))
print(paste('precision:', precision.micro<- tp.sum/(tp.sum+fp.sum)))
print(paste('NPV:', NPV.micro<- tn.sum/(fn.sum+tn.sum)))
print(paste('F1-score:', F1score.micro<- 2*tp.sum/(2*tp.sum+fn.sum+fp.sum)))

#COMPUTING MACRO MEASURES
message('MACRO MEASURES:')
print(paste('accuracy:', accuracy.macro<- mean(accuracy)))
print(paste('misclassrate:', misclassrate.macro<- mean(misclassrate)))
print(paste('sensitivity:', sensitivity.macro<- mean(sensitivity)))
print(paste('FNR:', FNR.macro<- mean(FNR)))
print(paste('specificity:', specificity.macro<- mean(specificity)))
print(paste('FPR:', FPR.macro<- mean(FPR)))
print(paste('precision:', precision.macro<- mean(precision, na.rm=TRUE)))
print(paste('NPV:', NPV.macro<- mean(NPV)))
print(paste('F1-score:', F1score.macro<- mean(F1score)))

#COMPUTING WEIGHTED MACRO MEASURES
weight<- c()
```

```
for (class in 1:5)
weight[class]<- (tp[class]+fn[class])/total

message('WEIGHTED MACRO MEASURES:')
print(paste('accuracy:', accuracy.wmacro<- weight%*%accuracy))
print(paste('misclassrate:', misclassrate.wmacro<- weight%*%misclassrate))
print(paste('sensitivity:', sensitivity.wmacro<- weight%*%sensitivity))
print(paste('FNR:', FNR.wmacro<- weight%*%FNR))
print(paste('specificity:', specificity.wmacro<- weight%*%specificity))
print(paste('FPR:', FPR.wmacro<- weight%*%FPR))
precision[is.na(precision)]<- 0
print(paste('precision:', precision.wmacro<- weight%*%precision))
print(paste('NPV:', NPV.wmacro<- weight%*%NPV))
print(paste('F1-score:', F1score.wmacro<- weight%*%F1score))

}

#COMPUTING PERFORMANCE MEASURES FOR FITTED GINI TREE
perf.measures()


CLASS MEASURES:
class:very bad
[1] "tp: 3"
[1] "fp: 10"
[1] "tn: 167"
[1] "fn: 13"
[1] "accuracy: 0.880829015544041"
[1] "misclassrate: 0.119170984455959"
[1] "sensitivity: 0.1875"
[1] "FNR: 0.8125"
[1] "specificity: 0.943502824858757"
[1] "FPR: 0.0564971751412429"
[1] "precision: 0.230769230769231"
[1] "NPV: 0.927777777777778"
[1] "F1score: 0.206896551724138"
CLASS MEASURES:
class:bad
[1] "tp: 10"
[1] "fp: 24"
[1] "tn: 139"
[1] "fn: 20"
[1] "accuracy: 0.772020725388601"
```

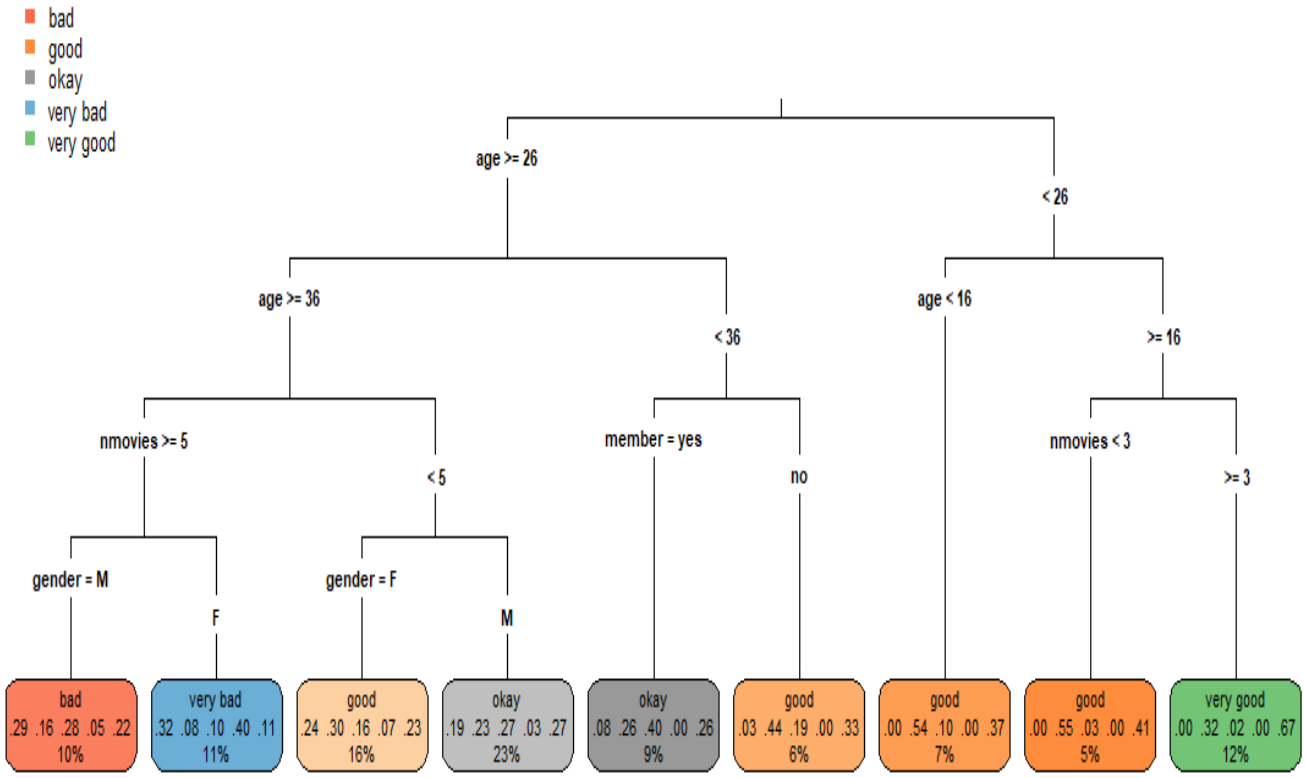
```
[1] "misclassrate: 0.227979274611399"
[1] "sensitivity: 0.333333333333333"
[1] "FNR: 0.666666666666667"
[1] "specificity: 0.852760736196319"
[1] "FPR: 0.147239263803681"
[1] "precision: 0.294117647058824"
[1] "NPV: 0.874213836477987"
[1] "F1score: 0.3125"
CLASS MEASURES:
class:okay
[1] "tp: 12"
[1] "fp: 49"
[1] "tn: 108"
[1] "fn: 24"
[1] "accuracy: 0.621761658031088"
[1] "misclassrate: 0.378238341968912"
[1] "sensitivity: 0.333333333333333"
[1] "FNR: 0.666666666666667"
[1] "specificity: 0.687898089171974"
[1] "FPR: 0.312101910828025"
[1] "precision: 0.19672131147541"
[1] "NPV: 0.818181818181818"
[1] "F1score: 0.247422680412371"
CLASS MEASURES:
class:good
[1] "tp: 20"
[1] "fp: 44"
[1] "tn: 95"
[1] "fn: 34"
[1] "accuracy: 0.595854922279793"
[1] "misclassrate: 0.404145077720207"
[1] "sensitivity: 0.37037037037037"
[1] "FNR: 0.62962962962963"
[1] "specificity: 0.683453237410072"
[1] "FPR: 0.316546762589928"
[1] "precision: 0.3125"
[1] "NPV: 0.736434108527132"
[1] "F1score: 0.338983050847458"
CLASS MEASURES:
class:very good
[1] "tp: 14"
[1] "fp: 7"
```

```
[1] "tn: 129"
[1] "fn: 43"
[1] "accuracy: 0.740932642487047"
[1] "misclassrate: 0.259067357512953"
[1] "sensitivity: 0.245614035087719"
[1] "FNR: 0.754385964912281"
[1] "specificity: 0.948529411764706"
[1] "FPR: 0.0514705882352941"
[1] "precision: 0.666666666666667"
[1] "NPV: 0.75"
[1] "F1score: 0.358974358974359"
MICRO MEASURES:
[1] "accuracy: 0.722279792746114"
[1] "misclassrate: 0.277720207253886"
[1] "sensitivity: 0.305699481865285"
[1] "FNR: 0.694300518134715"
[1] "specificity: 0.826424870466321"
[1] "FPR: 0.173575129533679"
[1] "precision: 0.305699481865285"
[1] "NPV: 0.826424870466321"
[1] "F1-score: 0.305699481865285"
MACRO MEASURES:
[1] "accuracy: 0.722279792746114"
[1] "misclassrate: 0.277720207253886"
[1] "sensitivity: 0.294030214424951"
[1] "FNR: 0.705969785575049"
[1] "specificity: 0.823228859880366"
[1] "FPR: 0.176771140119634"
[1] "precision: 0.340154971194026"
[1] "NPV: 0.821321508192943"
[1] "F1-score: 0.292955328391665"
WEIGHTED MACRO MEASURES:
[1] "accuracy: 0.694542135359339"
[1] "misclassrate: 0.305457864640661"
[1] "sensitivity: 0.305699481865285"
[1] "FNR: 0.694300518134715"
[1] "specificity: 0.810444817536544"
[1] "FPR: 0.189555182463457"
[1] "precision: 0.385869452420659"
[1] "NPV: 0.792968118413444"
[1] "F1-score: 0.312741888755092"
```

#FITTING PRUNED MULTINOMIAL CLASSIFICATION TREE WITH ENTROPY SPLIT-

TING
library(rpart)
tree.entropy<- rpart(rating ~ age + gender + member + nmovies, data=train, method="class", parms=list(split="entropy"))

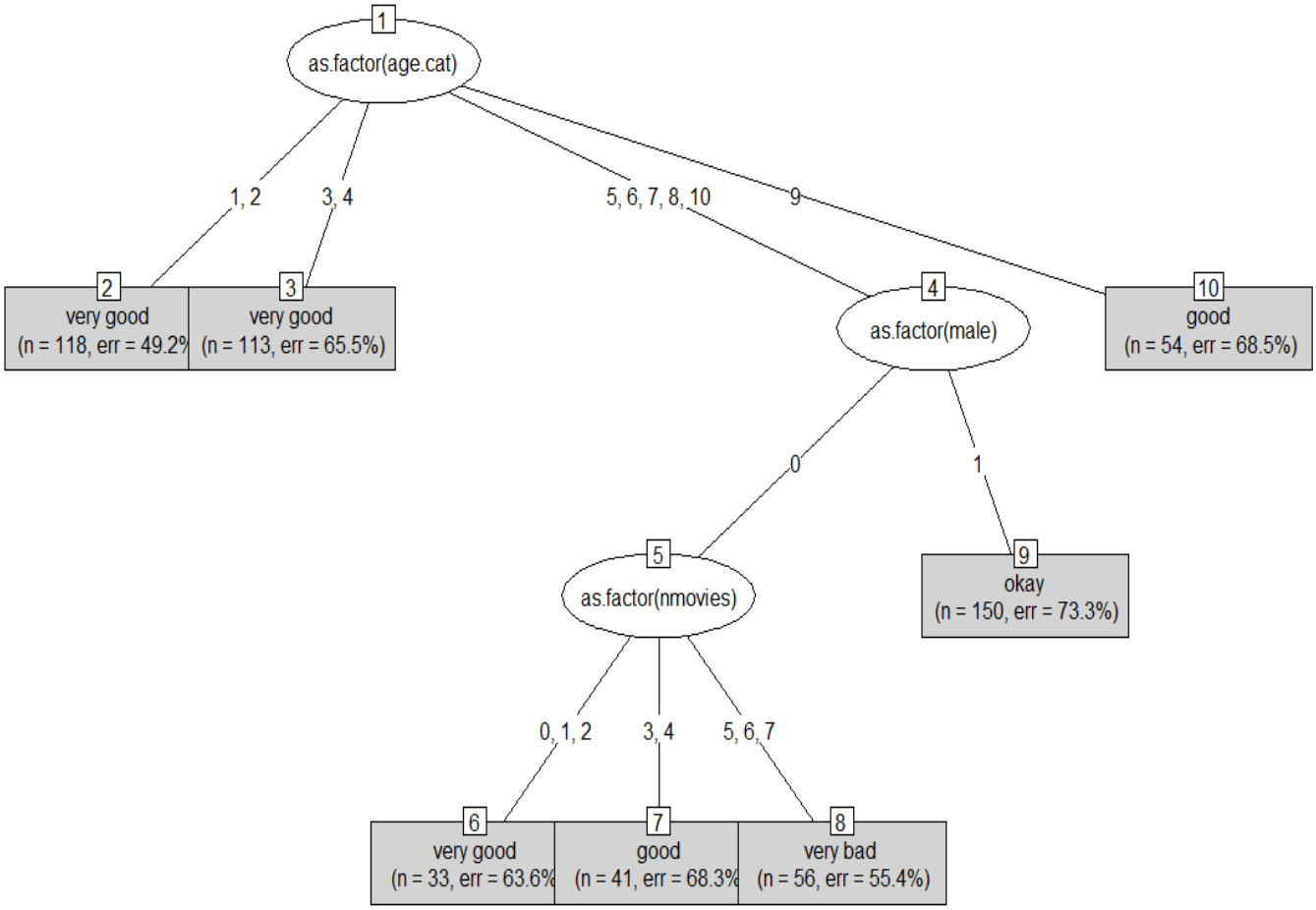
#PLOTTING FITTED TREE
rpart.plot(tree.entropy, type=3)
#Note: same as tree.gini



#FITTING PRUNED MULTINOMIAL CLASSIFICATION TREE WITH CHAID SPLITTING
#BINNING CONTINUOUS PREDICTOR VARIABLES
library(dplyr)
movie.data<- mutate(movie.data, age.cat=ntile(age,10))

#CREATING INDICATORS FOR CATEGORICAL VARIABLES
movie.data$male<- ifelse(movie.data$gender=="M",1,0)
movie.data$member.yes<- ifelse(movie.data$member=="yes",1,0)

```
#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
set.seed(566222)
sample <- sample(c(TRUE, FALSE), nrow(movie.data), replace=TRUE, prob=c(0.8,0.2))
train<- movie.data[sample,]
test<- movie.data[!sample,]

#FITTING BINARY CLASSIFICATION TREE
library(CHAID)
tree.CHAID<- chaid(as.factor(rating) ~ as.factor(age.cat) + as.factor(male) + as.factor(member.yes)
+ as.factor(nmovies), data=train)

#PLOTTING FITTED TREE
plot(tree.CHAID, type="simple")
```



```
#COMPUTING PREDICTED VALUES FOR TESTING DATA
pred.pneumonia<- predict(tree.CHAID, newdata=test)

#COMPUTING PERFORMANCE MEASURES FOR FITTED CHAID TREE
test$predclass<- pred.values
perf.measures()


CLASS MEASURES:
class:very bad
[1] "tp: 3"
[1] "fp: 8"
[1] "tn: 169"
[1] "fn: 13"
[1] "accuracy: 0.89119170984456"
[1] "misclassrate: 0.10880829015544"
[1] "sensitivity: 0.1875"
[1] "FNR: 0.8125"
[1] "specificity: 0.954802259887006"
[1] "FPR: 0.0451977401129944"
[1] "precision: 0.272727272727273"
[1] "NPV: 0.928571428571429"
[1] "F1score: 0.222222222222222"
CLASS MEASURES:
class:bad
[1] "tp: 0"
[1] "fp: 0"
[1] "tn: 163"
[1] "fn: 30"
[1] "accuracy: 0.844559585492228"
[1] "misclassrate: 0.155440414507772"
[1] "sensitivity: 0"
[1] "FNR: 1"
[1] "specificity: 1"
[1] "FPR: 0"
[1] "precision: NaN"
[1] "NPV: 0.844559585492228"
[1] "F1score: 0"
CLASS MEASURES:
class:okay
[1] "tp: 12"
[1] "fp: 47"
[1] "tn: 110"
```

```
[1] "fn: 24"
[1] "accuracy: 0.632124352331606"
[1] "misclassrate: 0.367875647668394"
[1] "sensitivity: 0.333333333333333"
[1] "FNR: 0.666666666666667"
[1] "specificity: 0.700636942675159"
[1] "FPR: 0.299363057324841"
[1] "precision: 0.203389830508475"
[1] "NPV: 0.82089552238806"
[1] "F1score: 0.252631578947368"
CLASS MEASURES:
class:good
[1] "tp: 9"
[1] "fp: 31"
[1] "tn: 108"
[1] "fn: 45"
[1] "accuracy: 0.606217616580311"
[1] "misclassrate: 0.393782383419689"
[1] "sensitivity: 0.166666666666667"
[1] "FNR: 0.833333333333333"
[1] "specificity: 0.776978417266187"
[1] "FPR: 0.223021582733813"
[1] "precision: 0.225"
[1] "NPV: 0.705882352941177"
[1] "F1score: 0.191489361702128"
CLASS MEASURES:
class:very good
[1] "tp: 36"
[1] "fp: 47"
[1] "tn: 89"
[1] "fn: 21"
[1] "accuracy: 0.647668393782383"
[1] "misclassrate: 0.352331606217617"
[1] "sensitivity: 0.631578947368421"
[1] "FNR: 0.368421052631579"
[1] "specificity: 0.654411764705882"
[1] "FPR: 0.345588235294118"
[1] "precision: 0.433734939759036"
[1] "NPV: 0.809090909090909"
[1] "F1score: 0.514285714285714"
MICRO MEASURES:
[1] "accuracy: 0.724352331606218"
```

```
[1] "misclassrate: 0.275647668393782"
[1] "sensitivity: 0.310880829015544"
[1] "FNR: 0.689119170984456"
[1] "specificity: 0.827720207253886"
[1] "FPR: 0.172279792746114"
[1] "precision: 0.310880829015544"
[1] "NPV: 0.827720207253886"
[1] "F1-score: 0.310880829015544"
MACRO MEASURES:
[1] "accuracy: 0.724352331606218"
[1] "misclassrate: 0.275647668393782"
[1] "sensitivity: 0.263815789473684"
[1] "FNR: 0.736184210526316"
[1] "specificity: 0.817365876906847"
[1] "FPR: 0.182634123093153"
[1] "precision: 0.283713010748696"
[1] "NPV: 0.82179995969676"
[1] "F1-score: 0.236125775431486"
WEIGHTED MACRO MEASURES:
[1] "accuracy: 0.683964670192488"
[1] "misclassrate: 0.316035329807512"
[1] "sensitivity: 0.310880829015544"
[1] "FNR: 0.689119170984456"
[1] "specificity: 0.77594855551869"
[1] "FPR: 0.22405144448131"
[1] "precision: 0.251598765949257"
[1] "NPV: 0.797834187071944"
[1] "F1-score: 0.271010381574411"
```

In Python:

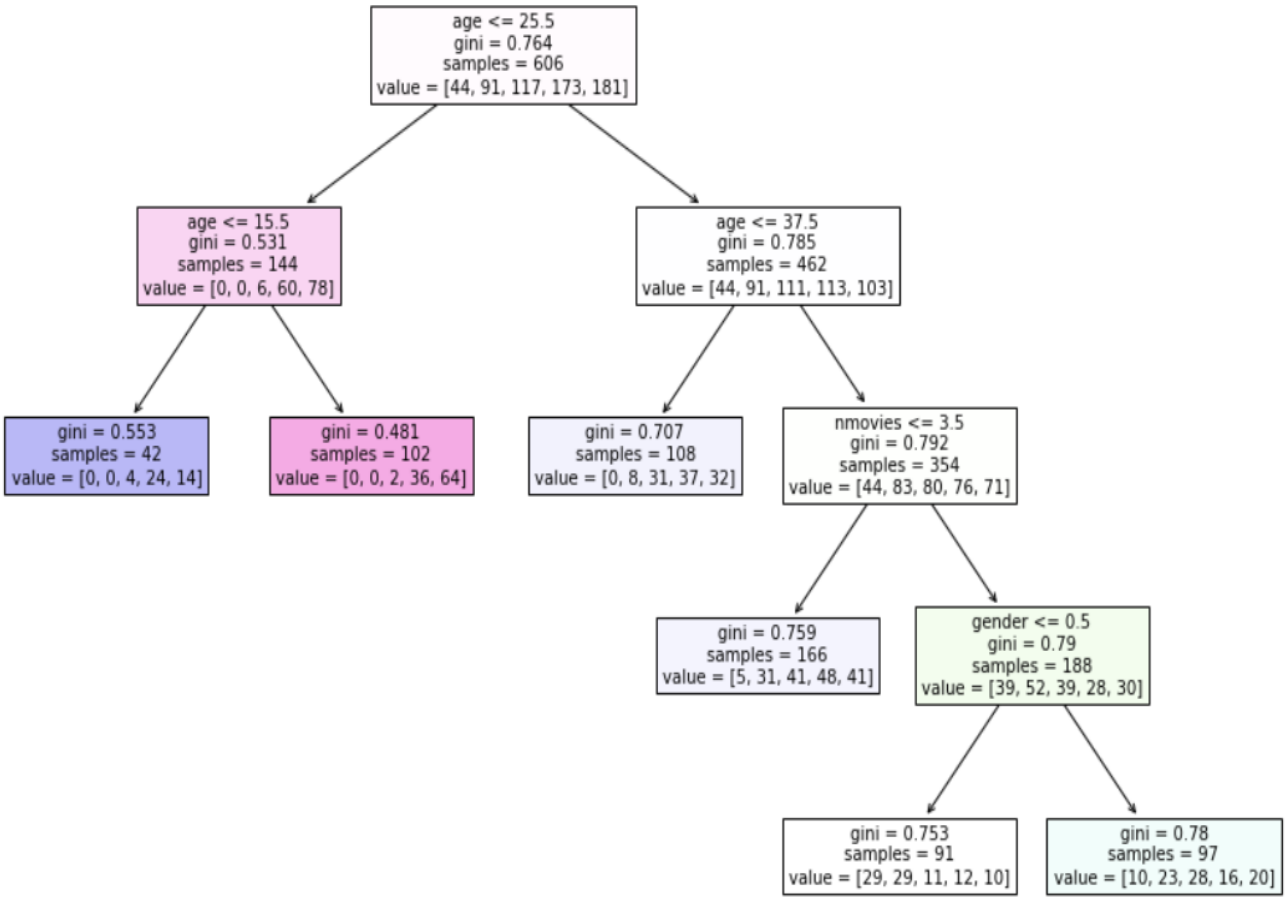
```
import numpy
import pandas
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn.model_selection import train_test_split

movie_data=pandas.read_csv('C:/Users/000110888/Desktop/movie_data.csv')
code_gender={'M':1,'F':0}
code_member={'yes':1,'no':0}
code_rating={'very bad':1,'bad':2,'okay':3,'good':4,'very good':5}

movie_data['gender']=movie_data['gender'].map(code_gender)
movie_data['member']=movie_data['member'].map(code_member)
movie_data['rating']=movie_data['rating'].map(code_rating)

X=movie_data.iloc[:,0:4].values
y=movie_data.iloc[:,4].values

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.20,
random_state=687088)

#########################################################################

#DEFINING FUNCTION FOR COMPUTING PERFORMANCE MEASURES
def perf_measures():

    #COMPUTING PERFORMANCE MEASURES FOR INDIVIDUAL CLASSES
    tp=[]
    fp=[]
    tn=[]
    fn=[]

    for cls in range(5):
        tp_sum=0
        fp_sum=0
        tn_sum=0
        fn_sum=0
        for sub1, sub2 in zip(predclass, y_test):
```

```
        if sub1==cls+1 and sub2==cls+1:
            tp_sum+=1

        if sub1==cls+1 and sub2!=cls+1:
            fp_sum+=1

        if sub1!=cls+1 and sub2!=cls+1:
            tn_sum+=1

        if sub1!=cls+1 and sub2==cls+1:
            fn_sum+=1

    tp.append(tp_sum)
    fp.append(fp_sum)
    tn.append(tn_sum)
    fn.append(fn_sum)


accuracy=[]
misclassrate=[]
sensitivity=[]
FNR=[]
specificity=[]
FPR=[]
precision=[]
NPV=[]
F1score=[]

print('CLASS MEASURES:')
for cls in range(5):
    accuracy_cls=(tp[cls]+tn[cls])/(tp[cls]+fp[cls]+tn[cls]+fn[cls])
    misclassrate_cls=(fp[cls]+fn[cls])/(tp[cls]+fp[cls]+tn[cls]+fn[cls])
    sensitivity_cls=tp[cls]/(tp[cls]+fn[cls])
```

```
        FNR.append(FNR_cls)
        specificity.append(specificity_cls)
        FPR.append(FPR_cls)
        precision.append(precision_cls)
        NPV.append(NPV_cls)
        F1score.append(F1score_cls)

        print()
        print('CLASS:', cls+1)
        print('tp:', tp[cls])
        print('fp:', fp[cls])
        print('tn:', tn[cls])
        print('fn:', fn[cls])
        print('accuracy:', accuracy[cls])
        print('misclassrate:', misclassrate[cls])
        print('sensitivity:', sensitivity[cls])
        print('FNR:', FNR[cls])
        print('specificity:', specificity[cls])
        print('FPR:', FPR[cls])
        print('precision:', precision[cls])
        print('NPV:', NPV[cls])
        print('F1score:', F1score[cls])

    #COMPUTING MICRO MEASURES
    tp_sum=numpy.sum(tp)
    fp_sum=numpy.sum(fp)
    tn_sum=numpy.sum(tn)
    fn_sum=numpy.sum(fn)

    print()
    print('MICRO MEASURES:')
    accuracy_micro=(tp_sum+tn_sum)/(tp_sum+fp_sum+tn_sum+fn_sum)
    misclassrate_micro=(fp_sum+fn_sum)/(tp_sum+fp_sum+tn_sum+fn_sum)
    sensitivity_micro=tp_sum/(tp_sum+fn_sum)
    FNR_micro=fn_sum/(tp_sum+fn_sum)
    specificity_micro=tn_sum/(fp_sum+tn_sum)
    FPR_micro=fp_sum/(fp_sum+tn_sum)
    precision_micro=tp_sum/(tp_sum+fp_sum)
    NPV_micro=tn_sum/(fn_sum+tn_sum)
    F1score_micro=2*tp_sum/(2*tp_sum+fn_sum+fp_sum)

    print('accuracy:', accuracy_micro)
    print('misclassrate:', misclassrate_micro)
    print('sensitivity:', sensitivity_micro)
    print('FNR:', FNR_micro)
    print('specificity:', specificity_micro)
    print('FPR:', FPR_micro)
    print('precision:', precision_micro)
    print('NPV:', NPV_micro)
    print('F1-score:', F1score_micro)
```

```
#COMPUTING MACRO MEASURES
accuracy_macro=numpy.mean(accuracy)
misclassrate_macro=numpy.mean(misclassrate)
sensitivity_macro=numpy.mean(sensitivity)
FNR_macro=numpy.mean(FNR)
specificity_macro=numpy.mean(specificity)
FPR_macro=numpy.mean(FPR)
precision_macro=numpy.mean(precision)
NPV_macro=numpy.mean(NPV)
F1score_macro=numpy.mean(F1score)

print()
print('MACRO MEASURES:')
print('accuracy:', accuracy_macro)
print('misclassrate:', misclassrate_macro)
print('sensitivity:', sensitivity_macro)
print('FNR:', FNR_macro)
print('specificity:', specificity_macro)
print('FPR:', FPR_macro)
print('precision:', precision_macro)
print('NPV:', NPV_macro)
print('F1-score:', F1score_macro)

#COMPUTING WEIGHTED MACRO MEASURES
weight=[]

for cls in range(5):
    weight_cls=(tp[cls]+fn[cls])/(tp[cls]+fp[cls]+tn[cls]+fn[cls])
    weight.append(weight_cls)

accuracy_wmacro=numpy.dot(weight,accuracy)
misclassrate_wmacro=numpy.dot(weight,misclassrate)
sensitivity_wmacro=numpy.dot(weight,sensitivity)
FNR_wmacro=numpy.dot(weight,FNR)
specificity_wmacro=numpy.dot(weight,specificity)
FPR_wmacro=numpy.dot(weight,FPR)
precision_wmacro=numpy.dot(weight,precision)
NPV_wmacro=numpy.dot(weight,NPV)
F1score_wmacro=numpy.dot(weight,F1score)

print()
print('WEIGHTED MACRO MEASURES:')
print('accuracy:', accuracy_wmacro)
print('misclassrate:', misclassrate_wmacro)
print('sensitivity:', sensitivity_wmacro)
print('FNR:', FNR_wmacro)
print('specificity:', specificity_wmacro)
print('FPR:', FPR_wmacro)
print('precision:', precision_wmacro)
print('NPV:', NPV_wmacro)
print('F1-score:', F1score_wmacro)
```

```
#FITTING MULTINOMIAL CLASSIFICATION TREE WITH GINI SPLITTING CRITERION
gini_tree=DecisionTreeClassifier(max_leaf_nodes=6, criterion='gini', random_state=707720)
gini_tree.fit=gini_tree.fit(X_train,y_train)

#PLOTTING FITTED TREE
fig=plt.figure(figsize=(15,10))
tree.plot_tree(gini_tree.fit, feature_names=['age','gender','member','nmovies'], filled=True)
```



```
#COMPUTING PREDICTED VALUES FOR TESTING DATA
y_pred=gini_tree.predict_proba(X_test)

#DETERMINING PREDICTED CLASSES
predclass=[]
for i  in range(0, len(y_pred)):
    list=[y_pred[i,0],y_pred[i,1], y_pred[i,2], y_pred[i,3], y_pred[i,4]]
    predclass.append(list.index(max(list))+1)
predclass=numpy.asarray(predclass)

#COMPUTING PERFORMANCE MEASURES FOR FITTED GINI TREE
print()
print('PERFORMANCE MEASURES FOR FITTED GINI TREE')
perf_measures()
```

```
PERFORMANCE MEASURES FOR FITTED GINI TREE
CLASS MEASURES:

CLASS: 1
tp: 8
fp: 15
tn: 127
fn: 2
accuracy: 0.8881578947368421
misclassrate: 0.1118421052631579
sensitivity: 0.8
FNR: 0.2
specificity: 0.8943661971830986
FPR: 0.1056338028169014
precision: 0.34782608695652173
NPV: 0.9844961240310077
F1score: 0.48484848484848486

CLASS: 2
tp: 0
fp: 0
tn: 125
fn: 27
accuracy: 0.8223684210526315
misclassrate: 0.17763157894736842
```

sensitivity: 0.0
FNR: 1.0
specificity: 1.0
FPR: 0.0
precision: 0
NPV: 0.8223684210526315
F1score: 0.0

CLASS: 3
tp: 5
fp: 21
tn: 106
fn: 20
accuracy: 0.7302631578947368
misclassrate: 0.26973684210526316
sensitivity: 0.2
FNR: 0.8
specificity: 0.8346456692913385
FPR: 0.16535433070866143
precision: 0.19230769230769232
NPV: 0.8412698412698413
F1score: 0.19607843137254902

CLASS: 4
tp: 23
fp: 53
tn: 58
fn: 18
accuracy: 0.5328947368421053
misclassrate: 0.46710526315789475
sensitivity: 0.5609756097560976
FNR: 0.43902439024390244
specificity: 0.5225225225225225
FPR: 0.4774774774774775
precision: 0.3026315789473684
NPV: 0.7631578947368421
F1score: 0.39316239316239315

CLASS: 5
tp: 13
fp: 14
tn: 89

fn: 36
accuracy: 0.6710526315789473
misclassrate: 0.32894736842105265
sensitivity: 0.2653061224489796
FNR: 0.7346938775510204
specificity: 0.8640776699029126
FPR: 0.13592233009708737
precision: 0.48148148148148145
NPV: 0.712
F1score: 0.34210526315789475

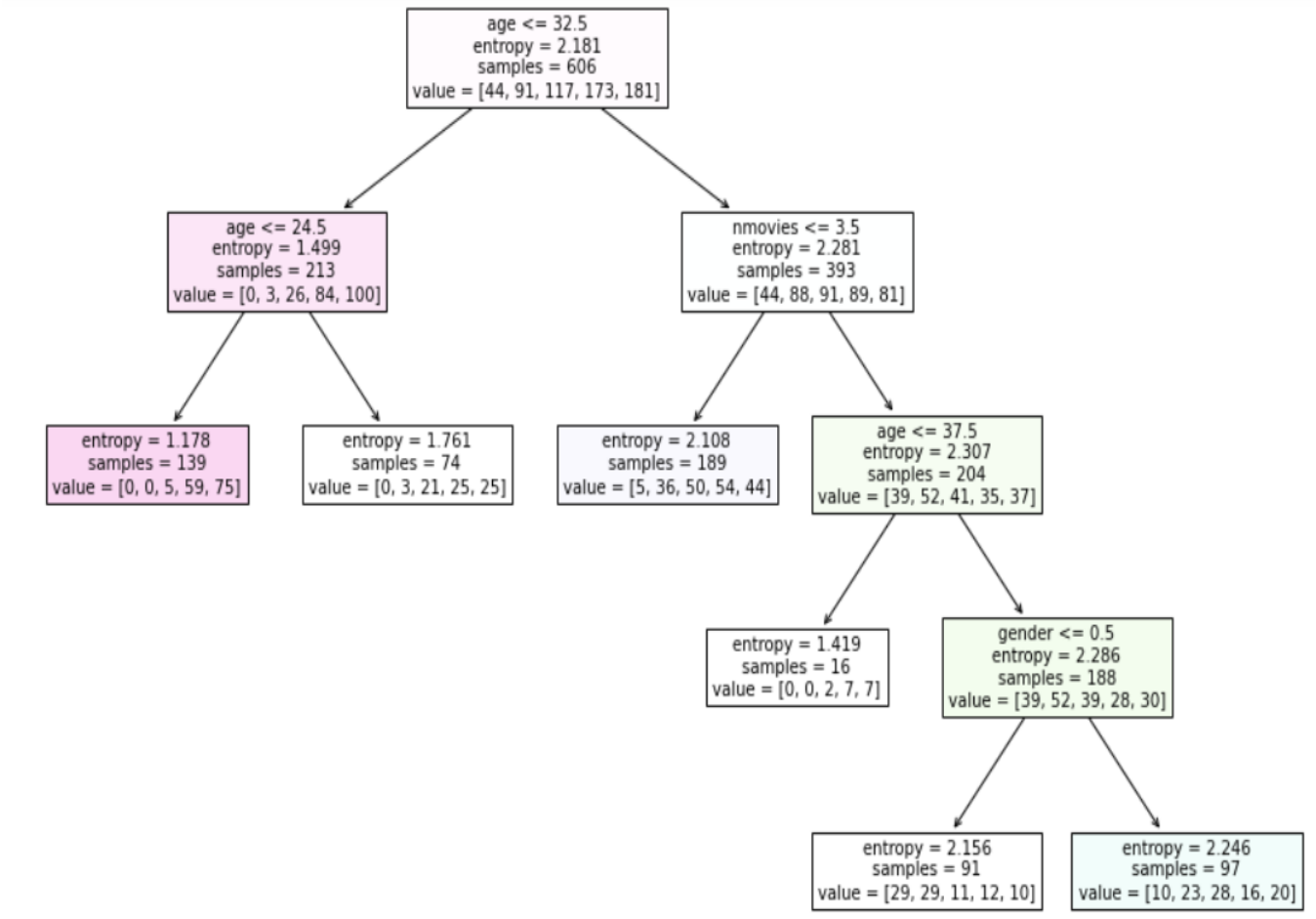
MICRO MEASURES:
accuracy: 0.7289473684210527
misclassrate: 0.2710526315789474
sensitivity: 0.3223684210526316
FNR: 0.6776315789473685
specificity: 0.8305921052631579
FPR: 0.16940789473684212
precision: 0.3223684210526316
NPV: 0.8305921052631579
F1-score: 0.3223684210526316

MACRO MEASURES:
accuracy: 0.7289473684210527
misclassrate: 0.2710526315789474
sensitivity: 0.36525634644101546
FNR: 0.6347436535589845
specificity: 0.8231224117799745
FPR: 0.17687758822002553
precision: 0.26484936793861275
NPV: 0.8246584562180646
F1-score: 0.28323891450826433

WEIGHTED MACRO MEASURES:
accuracy: 0.6846866343490305
misclassrate: 0.31531336565096957
sensitivity: 0.3223684210526316
FNR: 0.6776315789473684
specificity: 0.7932436378472407
FPR: 0.20675636215275928
precision: 0.2913581612282383
NPV: 0.7845929495045243

```
F1-score: 0.2804819845210101
```

```
#FITTING MULTINOMIAL CLASSIFICATION TREE WITH ENTROPY SPLITTING CRITERION
entropy_tree=DecisionTreeClassifier(max_leaf_nodes=6, criterion='entropy', random_state=707720)
entropy_tree.fit=entropy_tree.fit(X_train,y_train)

#PLOTTING FITTED TREE
fig=plt.figure(figsize=(15,10))
tree.plot_tree(entropy_tree.fit, feature_names=['age','gender','member','nmovies'], filled=True)
```



```
#COMPUTING PREDICTED VALUES FOR TESTING DATA
y_pred=entropy_tree.predict_proba(X_test)

#DETERMINING PREDICTED CLASSES
predclass=[]
for i  in range(0, len(y_pred)):
    list=[y_pred[i,0],y_pred[i,1], y_pred[i,2], y_pred[i,3], y_pred[i,4]]
    predclass.append(list.index(max(list))+1)
predclass=numpy.asarray(predclass)

#COMPUTING PERFORMANCE MEASURES FOR FITTED ENTROPY TREE
print()
print('PERFORMANCE MEASURES FOR FITTED ENTROPY TREE')
perf_measures()
```

```
PERFORMANCE MEASURES FOR FITTED ENTROPY TREE
CLASS MEASURES:

CLASS: 1
tp: 8
fp: 15
tn: 127
fn: 2
accuracy: 0.8881578947368421
misclassrate: 0.1118421052631579
sensitivity: 0.8
FNR: 0.2
specificity: 0.8943661971830986
FPR: 0.1056338028169014
precision: 0.34782608695652173
NPV: 0.9844961240310077
F1score: 0.48484848484848486

CLASS: 2
tp: 0
fp: 0
tn: 125
fn: 27
accuracy: 0.8223684210526315
misclassrate: 0.17763157894736842
```

sensitivity: 0.0
FNR: 1.0
specificity: 1.0
FPR: 0.0
precision: 0
NPV: 0.8223684210526315
F1score: 0.0

CLASS: 3
tp: 5
fp: 21
tn: 106
fn: 20
accuracy: 0.7302631578947368
misclassrate: 0.26973684210526316
sensitivity: 0.2
FNR: 0.8
specificity: 0.8346456692913385
FPR: 0.16535433070866143
precision: 0.19230769230769232
NPV: 0.8412698412698413
F1score: 0.19607843137254902

CLASS: 4
tp: 21
fp: 51
tn: 60
fn: 20
accuracy: 0.5328947368421053
misclassrate: 0.46710526315789475
sensitivity: 0.5121951219512195
FNR: 0.4878048780487805
specificity: 0.5405405405405406
FPR: 0.4594594594594595
precision: 0.2916666666666667
NPV: 0.75
F1score: 0.37168141592920356

CLASS: 5
tp: 15
fp: 16
tn: 87

fn: 34
accuracy: 0.6710526315789473
misclassrate: 0.32894736842105265
sensitivity: 0.30612244897959184
FNR: 0.6938775510204082
specificity: 0.8446601941747572
FPR: 0.1553398058252427
precision: 0.4838709677419355
NPV: 0.71900826446281
F1score: 0.375

MICRO MEASURES:
accuracy: 0.7289473684210527
misclassrate: 0.2710526315789474
sensitivity: 0.3223684210526316
FNR: 0.6776315789473685
specificity: 0.8305921052631579
FPR: 0.16940789473684212
precision: 0.3223684210526316
NPV: 0.8305921052631579
F1-score: 0.3223684210526316

MACRO MEASURES:
accuracy: 0.7289473684210527
misclassrate: 0.2710526315789474
sensitivity: 0.3636635141861623
FNR: 0.6363364858138377
specificity: 0.8228425202379469
FPR: 0.17715747976205298
precision: 0.26313428273456324
NPV: 0.8234285301632582
F1-score: 0.28552166643004745

WEIGHTED MACRO MEASURES:
accuracy: 0.6846866343490305
misclassrate: 0.31531336565096957
sensitivity: 0.32236842105263164
FNR: 0.6776315789473685
specificity: 0.7918441801371034
FPR: 0.20815581986289658
precision: 0.2891708153285901
NPV: 0.7833030236786503

```
F1-score: 0.2852919979335258
```

```
#FITTING MULTINOMIAL CLASSIFICATION TREE WITH CHAID SPLITTING CRITERION (ORDINAL CHAID TREE)
from chefboost import Chefboost

movie_data=pandas.read_csv('./movie_data.csv')
code_gender={'M':1,'F':0}
code_member={'yes':1,'no':0}
#Note: rating is used as nominal

movie_data['gender']=movie_data['gender'].map(code_gender)
movie_data['member']=movie_data['member'].map(code_member)

X=movie_data.iloc[:,0:4].values
y=movie_data.iloc[:,4].values

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.20,
random_state=687088)

X_train=pandas.DataFrame(X_train, columns=['age','gender','member','nmovies'])
y_train=pandas.DataFrame(y_train, columns=['rating'])
train_data=pandas.concat([X_train, y_train], axis=1) #one-to-one concatenation

#FITTING BINARY TREE WITH CHAID SPLITTING ALGORITHM
config={'algorithm': 'CHAID', 'max_depth': 3}
tree_chaid=Chefboost.fit(train_data, config, target_label='rating')
```

```
def findDecision(obj): #obj[0]: age, obj[1]: gender, obj[2]: member, obj[3]: nmovies
# {"feature": "age", "instances": 606, "metric_value": 40.4586, "depth": 1}
if obj[0]>24.271368369871805:
# {"feature": "nmovies", "instances": 467, "metric_value": 20.0029, "depth": 2}
if obj[3]<=6:
# {"feature": "member", "instances": 458, "metric_value": 17.7061, "depth": 3}
if obj[2]>0:
# {"feature": "gender", "instances": 290, "metric_value": 15.2447, "depth": 4}
if obj[1]>0:
return 'good'
elif obj[1]<=0:
return 'bad'
else: return 'bad'
elif obj[2]<=0:
# {"feature": "gender", "instances": 168, "metric_value": 12.3326, "depth": 4}
if obj[1]>0:
return 'very good'
elif obj[1]<=0:
return 'okay'
```

```
else: return 'okay'
else: return 'very good'
elif obj[3]>6:
# {"feature": "member", "instances": 9, "metric_value": 2.5345, "depth": 3}
if obj[2]>0:
# {"feature": "gender", "instances": 7, "metric_value": 0.8165, "depth": 4}
if obj[1]<=0:
return 'very bad'
elif obj[1]>0:
return 'bad'
else: return 'bad'
elif obj[2]<=0:
return 'very bad'
else: return 'very bad'
else: return 'very bad'
elif obj[0]<=24.271368369871805:
# {"feature": "nmovies", "instances": 139, "metric_value": 17.1734, "depth": 2}
if obj[3]>3:
# {"feature": "gender", "instances": 70, "metric_value": 12.6878, "depth": 3}
if obj[1]>0:
# {"feature": "member", "instances": 36, "metric_value": 8.7214, "depth": 4}
if obj[2]>0:
return 'very good'
elif obj[2]<=0:
return 'very good'
else: return 'very good'
elif obj[1]<=0:
# {"feature": "member", "instances": 34, "metric_value": 9.7217, "depth": 4}
if obj[2]>0:
return 'very good'
elif obj[2]<=0:
return 'good'
else: return 'good'
else: return 'very good'
elif obj[3]<=3:
# {"feature": "gender", "instances": 69, "metric_value": 11.7781, "depth": 3}
if obj[1]>0:
# {"feature": "member", "instances": 36, "metric_value": 9.6067, "depth": 4}
if obj[2]>0:
return 'very good'
elif obj[2]<=0:
return 'very good'
```

```
else: return 'very good'
elif obj[1]<=0:
# {"feature": "member", "instances": 33, "metric_value": 7.5356, "depth": 4}
if obj[2]>0:
return 'good'
elif obj[2]<=0:
return 'good'
else: return 'good'
else: return 'good'
else: return 'very good'
else: return 'very good'
```

```
#COMPUTING PREDICTION ACCURACY
X_test=pandas.DataFrame(X_test, columns=['age','gender','member','nmovies'])

y_pred=[]
for i in range(len(y_test)):
    y_pred.append(Chefboost.predict(tree_chaid, X_test.iloc[i,:]))
predclass=numpy.asarray(y_pred)

#COMPUTING PERFORMANCE MEASURES FOR FITTED CHAID TREE
#turning rating into ordinal to use perf_measures function
code_rating={'very bad':1,'bad':2,'okay':3,'good':4,'very good':5}
df=pandas.DataFrame({'y_test': y_test,'predclass': predclass})
df['y_test']=df['y_test'].map(code_rating)
df['predclass']=df['predclass'].map(code_rating)
y_test=df['y_test']
predclass=df['predclass']

print()
print('PERFORMANCE MEASURES FOR FITTED CHAID TREE')
perf_measures()
```

```
PERFORMANCE MEASURES FOR FITTED CHAID TREE
CLASS MEASURES:

CLASS: 1
```

tp: 1
fp: 1
tn: 141
fn: 9
accuracy: 0.9342105263157895
misclassrate: 0.06578947368421052
sensitivity: 0.1
FNR: 0.9
specificity: 0.9929577464788732
FPR: 0.007042253521126761
precision: 0.5
NPV: 0.94
F1score: 0.16666666666666666

CLASS: 2
tp: 10
fp: 24
tn: 101
fn: 17
accuracy: 0.7302631578947368
misclassrate: 0.26973684210526316
sensitivity: 0.37037037037037035
FNR: 0.6296296296296297
specificity: 0.808
FPR: 0.192
precision: 0.29411764705882354
NPV: 0.8559322033898306
F1score: 0.32786885245901637

CLASS: 3
tp: 0
fp: 12
tn: 115
fn: 25
accuracy: 0.756578947368421
misclassrate: 0.24342105263157895
sensitivity: 0.0
FNR: 1.0
specificity: 0.905511811023622
FPR: 0.09448818897637795
precision: 0.0
NPV: 0.8214285714285714

```
F1score: 0.0

CLASS: 4
tp: 17
fp: 46
tn: 65
fn: 24
accuracy: 0.5394736842105263
misclassrate: 0.4605263157894737
sensitivity: 0.4146341463414634
FNR: 0.5853658536585366
specificity: 0.5855855855855856
FPR: 0.4144144144144144
precision: 0.2698412698412698
NPV: 0.7303370786516854
F1score: 0.3269230769230769

CLASS: 5
tp: 18
fp: 23
tn: 80
fn: 31
accuracy: 0.6447368421052632
misclassrate: 0.35526315789473684
sensitivity: 0.3673469387755102
FNR: 0.6326530612244898
specificity: 0.7766990291262136
FPR: 0.22330097087378642
precision: 0.43902439024390244
NPV: 0.7207207207207207
F1score: 0.4

MICRO MEASURES:
accuracy: 0.7210526315789474
misclassrate: 0.2789473684210526
sensitivity: 0.3026315789473684
FNR: 0.6973684210526315
specificity: 0.8256578947368421
FPR: 0.17434210526315788
precision: 0.3026315789473684
NPV: 0.8256578947368421
F1-score: 0.3026315789473684
```

```
MACRO MEASURES:
accuracy: 0.7210526315789474
misclassrate: 0.27894736842105267
sensitivity: 0.25047029109746877
FNR: 0.7495297089025312
specificity: 0.8137508344428588
FPR: 0.1862491655571411
precision: 0.30059666142879915
NPV: 0.8136837148381616
F1-score: 0.24429171920975196

WEIGHTED MACRO MEASURES:
accuracy: 0.6689750692520776
misclassrate: 0.3310249307479224
sensitivity: 0.3026315789473685
FNR: 0.6973684210526316
specificity: 0.766122593266926
FPR: 0.23387740673307394
precision: 0.29945305036862846
NPV: 0.7783224955083824
F1-score: 0.28633534103227803
```

□

RANDOM FOREST

Individual decision tree algorithms can be prone to problems, such as bias and over-fitting. A more practical approach is to construct multiple decision trees and combine the results into a single output. This approach is termed **ensemble methods**. The most well-known ensemble method is **bagging** (also known as **bootstrap aggregation**, **bagging=bootstrap+aggregation**). This method was introduced in 1996 by Leo Breiman. In this method, data points in the training set are sampled with replacement (producing a **bootstrap sample**), one-third of which, known as the **out-of-bag (OOB) sample**, is set aside for cross-validation, and the remaining two-thirds of the sample is used to build a decision tree. Decision trees are generated for each bootstrap sample independently from others. The results are then aggregated depending on the type of trees used. If regression trees are trained, the average of predicted values are computed. If classification trees are fitted, the majority of the predictions define the final predicted class. The ensemble method reduces variance and, as the result, yields more accurate predictions than individual decision trees.

The **random forest algorithm** is an extension of the bagging method as it utilizes both bagging and feature randomness to create an uncorrelated forest of decision trees. **Feature randomness** (also known as **feature bagging** or the **random subspace method**) generates a random subset of variables (also called **features**), which ensures low correlation among decision trees. This is a key difference between decision trees and random forests. While decision trees consider all the possible variable splits, random forests only select a subset of those variables.

Random forest algorithms have three main hyper-parameters that need to be set before training. These are node size, the number of trees, and the number of variables sampled.

Variable Importance

Random forest makes it easy to evaluate the **variable importance** (or the **contribution** of each splitting variable to the model). It is sometimes termed the **feature importance**. There are two commonly used ways to evaluate variable importance that are characteristically different from each other. One method is termed the **loss reduction** or **Gini increase** or **Gini importance** or **impurity reduction** or **mean decrease in impurity (MDI)**. It is used to measure how much the model's accuracy decreases when a given variable is excluded. For a random forest with regression trees, the loss functions are the **mean squared error** $MSE = RSS/degree\ of\ freedom\ of\ error$, and the **absolute error** $\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$. The second feature importance method is the **permutation importance** (or the **mean decrease accuracy (MDA)**), which identifies the average decrease in accuracy by randomly permuting the variable values in OOB samples.

Example. Consider the data in the file "housing_data.csv". We construct random forest regression for this data set.

In SAS:

```
proc import out=housing
datafile="./housing_data.csv" dbms=csv replace;
run;

/*SPLITTING DATA INTO 80% TRAINING AND 20% TESTING*/
proc surveyselect data=housing rate=0.8 seed=502305
out=housing outall method=srs;
run;

/*BUILDING RANDOM FOREST REGRESSION*/
```

```
proc hpforest data=housing seed=829743
maxtrees=60 vars_to_try=4 trainfraction=0.7
maxdepth=50;
target median_house_value/level=interval;
input ocean_proximity/level=nominal;
input housing_median_age total_rooms total_bedrooms
population households median_income/level=interval;
partition rolevar=selected(train='1');
save file='C:/Users/000110888/Desktop/random_forest.bin';
run;
```

Loss Reduction Variable Importance

Variable	Number of Rules	MSE	OOB MSE	Absolute Error	OOB Absolute Error
median_income	35554	5.2672E9	4.3027E9	36134	23887
ocean_proximity	336	1.6836E9	1.7055E9	12553	12796
total_rooms	7375	4.5494E8	-5.897E7	4665.515799	-423.229467
housing_median_age	7987	5.285E8	-7.79E7	5698.148519	141.283900
total_bedrooms	8196	3.22E8	-1.709E8	4098.769334	-795.029912
households	20105	5.2346E8	-2.364E8	6723.747118	-1030.371514
population	14896	4.4438E8	-2.694E8	5805.772403	-1414.692176

```
/*COMPUTING PREDICTED VALUES FOR TESTING DATA*/
data test;
set housing;
if(selected='0');
run;

proc hp4score data=test;
id median_house_value;
score file='C:/Users/000110888/Desktop/random_forest.bin'
out=predicted;
run;

/*DETERMINING 10%, 15%, AND 20% ACCURACY*/
data accuracy;
set predicted;
if(abs(median_house_value-P_median_house_value)
<0.10*median_house_value)
then ind10=1; else ind10=0;
if(abs(median_house_value-P_median_house_value)
```

```
<0.15*median_house_value)
then ind15=1; else ind15=0;
if(abs(median_house_value-P_median_house_value)
<0.20*median_house_value)
then ind20=1; else ind20=0;
run;

proc sql;
 select sum(ind10)/count(*) as accuracy10,
sum(ind15)/count(*) as accuracy15,
 sum(ind20)/count(*) as accuracy20
 from accuracy;
 quit;
```

accuracy10	accuracy15	accuracy20
0.352113	0.485915	0.612676

In R:

```
#install.packages("randomForest")
library(randomForest)

housing.data<- read.csv(file="./housing_data.csv", header=TRUE, sep=",")

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
set.seed(902881)
sample <- sample(c(TRUE, FALSE), nrow(housing.data), replace=TRUE, prob=c(0.8,0.2))
train<- housing.data[sample,]
test<- housing.data[!sample,]

#BUILDING RANDOM FOREST REGRESSION
rf.reg<- randomForest(median_house_value ~ housing_median_age + total_rooms + total_bedrooms
+ population + households + median_income + ocean_proximity, data=train, ntree=150, mtry=5,
maxnodes=30)

#DISPLAYING FEATURE IMPORTANCE
print(importance(rf.reg,type=2))
```

```
                   IncNodePurity
housing_median_age  4.241691e+11
total_rooms         4.108330e+11
total_bedrooms      2.028437e+11
population          2.153154e+11
households          2.400010e+11
median_income       1.245747e+13
ocean_proximity     2.914203e+12
```

```
#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
P_median_house_value<- predict(rf.reg, newdata=test)

#accuracy within 10%
accuracy10<-
ifelse(abs(test$median_house_value-P_median_house_value)<0.10*test$median_house_value,1,0)
print(accuracy10<- sum(accuracy10)/length(accuracy10))
```

0.2857143

```
#accuracy within 15%
accuracy15<-
ifelse(abs(test$median_house_value-P_median_house_value)<0.15*test$median_house_value,1,0)
print(accuracy15<- sum(accuracy15)/length(accuracy15))
```

0.4303797

```
#accuracy within 20%
accuracy20<-
ifelse(abs(test$median_house_value-P_median_house_value)<0.20*test$median_house_value,1,0)
print(accuracy20<- sum(accuracy20)/length(accuracy20))
```

0.5334539

In Python:

```
import pandas
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split

housing=pandas.read_csv('C:/Users/000110888/Desktop/housing_data.csv')
coding={'<1H OCEAN': 1, 'INLAND': 2, 'NEAR BAY': 3, 'NEAR OCEAN': 4}
housing['ocean_proximity']=housing['ocean_proximity'].map(coding)
X=housing.iloc[:,0:7].values
y=housing.iloc[:,7].values

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.20,
random_state=348644)

#FITTING RANDOM FOREST REGRESSION TREE
rf_reg=RandomForestRegressor(n_estimators=100, random_state=323445,
max_depth=50, max_features=4)
rf_reg.fit(X_train, y_train)

#DISPLAYING VARIABLE IMPORTANCE
from sklearn.ensemble import ExtraTreesClassifier

var_names=pandas.DataFrame(['housing_median_age','total_rooms','total_bedrooms','population',
'households','median_income','ocean_proximity'], columns=['var_name'])
loss_reduction=pandas.DataFrame(rf_reg.feature_importances_, columns=['loss_reduction'])
var_importance=pandas.concat([var_names, loss_reduction], axis=1)
var_importance=var_importance.sort_values("loss_reduction", axis=0, ascending=False)
print(var_importance)
```

```
             var_name  loss_reduction
5       median_income        0.592797
6     ocean_proximity        0.154105
1         total_rooms        0.061499
0  housing_median_age        0.053247
3          population        0.051611
4          households        0.044158
2      total_bedrooms        0.042583
```

```python
#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
y_pred=rf_reg.predict(X_test)

ind10=[]
ind15=[]
ind20=[]

for sub1, sub2 in zip(y_pred, y_test):
    ind10.append(1) if abs(sub1-sub2)<0.10*sub2 else ind10.append(0)
    ind15.append(1) if abs(sub1-sub2)<0.15*sub2 else ind15.append(0)
    ind20.append(1) if abs(sub1-sub2)<0.20*sub2 else ind20.append(0)

#accuracy within 10%
accuracy10=sum(ind10)/len(ind10)
print(accuracy10)

#accuracy within 15%
accuracy15=sum(ind15)/len(ind15)
print(accuracy15)

#accuracy within 20%
accuracy20=sum(ind20)/len(ind20)
print(accuracy20)
```

```
0.36203866432337434
0.5202108963093146
0.648506151142355
```

□

Example. Consider the data in the file "pneumonia_data.csv". We construct a random forest binary classifier for this data set.

In SAS:

```
proc import out=pneumonia datafile="./pneumonia_data.csv" dbms=csv replace;

/*SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS*/
proc surveyselect data=pneumonia rate=0.8 seed=6132208
out=pneumonia outall method=srs;
run;

/*BUILDING RANDOM FOREST BINARY CLASSIFIER*/
```

```
proc hpforest data=pneumonia seed=115607
maxtrees=60 vars_to_try=4 trainfraction=0.7
maxdepth=50;
target pneumonia/level=binary;
input gender tobacco_use/level=nominal;
input age PM2_5/level=interval;
partition rolevar=selected(train='1');
save file='C:/Users/000110888/Desktop/random_forest.bin';
run;

/*COMPUTING PREDICTED VALUES FOR TESTING DATA*/
data test;
set pneumonia;
if(selected='0');
run;

proc hp4score data=test;
id pneumonia;
score file='C:/Users/000110888/Desktop/random_forest.bin'
out=predicted;
run;
```

Loss Reduction Variable Importance

Variable	Number of Rules	Gini	OOB Gini	Margin	OOB Margin
gender	84	0.056581	0.05607	0.113161	0.112805
tobacco_use	263	0.044094	0.04157	0.088188	0.086383
PM2_5	7647	0.271965	0.00917	0.543929	0.279844
age	3208	0.085176	-0.05001	0.170351	0.035585

```
/*COMPUTING PREDICTION ACCURACY FOR TESTING DATA*/
data predicted;
set predicted;
```

```
match=(pneumonia=lowcase(I_pneumonia));
run;

proc sql;
select sum(match)/count(*) as accuracy
from predicted;
quit;
```

accuracy
0.823188

In R:

```
pneumonia.data<- read.csv(file="./pneumonia_data.csv", header=TRUE, sep=",")

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
set.seed(447558)
sample <- sample(c(TRUE, FALSE), nrow(pneumonia.data), replace=TRUE, prob=c(0.8,0.2))
train<- pneumonia.data[sample,]
test<- pneumonia.data[!sample,]

#BUILDING RANDOM FOREST BINARY CLASSIFIER
library(randomForest)
rf.class<- randomForest(as.factor(pneumonia) ~ age + gender + tobacco_use + PM2_5, data=train,
ntree=150, mtry=4, maxnodes=30)

#DISPLAYING FEATURE IMPORTANCE
print(importance(rf.class,type=2))
```

```
            MeanDecreaseGini
age                 36.39492
gender              67.32295
tobacco_use         56.93959
PM2_5              151.83923
```

```
#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
predclass<- predict(rf.class, newdata=test)
test<- cbind(test,predclass)
```

```
accuracy<- c()
n<- nrow(test)
for (i in 1:n)
    accuracy[i]<- ifelse(test$pneumonia[i]==test$predclass[i],1,0)

print(accuracy<- sum(accuracy)/length(accuracy))
```

0.815864

In Python:

```
import pandas
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

pneumonia_data=pandas.read_csv('C:/Users/000110888/Desktop/pneumonia_data.csv')
code_gender={'M':1,'F':0}
code_tobacco_use={'yes':1,'no':0}
code_pneumonia={'yes':1,'no':0}

pneumonia_data['gender']=pneumonia_data['gender'].map(code_gender)
pneumonia_data['tobacco_use']=pneumonia_data['tobacco_use'].map(code_tobacco_use)
pneumonia_data['pneumonia']=pneumonia_data['pneumonia'].map(code_pneumonia)

X=pneumonia_data.iloc[:,0:4].values
y=pneumonia_data.iloc[:,4].values

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.20,
random_state=786756)

#FITTING RANDOM FOREST BINARY CLASSIFIER
rf_class=RandomForestClassifier(n_estimators=150, criterion='entropy',
random_state=778554, max_depth=50, max_features=4)
rf_class.fit(X_train, y_train)

#DISPLAYING VARIABLE IMPORTANCE
from sklearn.ensemble import ExtraTreesClassifier

var_names=pandas.DataFrame(['gender','age','tobacco_use','PM2_5'], columns=['var_name'])
loss_reduction=pandas.DataFrame(rf_class.feature_importances_, columns=['loss_reduction'])
var_importance=pandas.concat([var_names, loss_reduction], axis=1)
var_importance=var_importance.sort_values("loss_reduction", axis=0, ascending=False)
print(var_importance)
```

```
      var_name  loss_reduction
3        PM2_5        0.581401
1          age        0.233343
0       gender        0.095434
```

```
2  tobacco_use        0.089823
```

```
#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
y_pred=rf_class.predict(X_test)
y_test=pandas.DataFrame(y_test,columns=['pneumonia'])
y_pred=pandas.DataFrame(y_pred,columns=['predicted'])
df=pandas.concat([y_test,y_pred],axis=1)

match=[]
for i in range(len(df)):
    if df['pneumonia'][i]==df['predicted'][i]:
        match.append(1)
    else:
        match.append(0)

accuracy=sum(match)/len(match)

print(accuracy)
```

```
0.8121387283236994
```

□

Example. Consider the data in the file "movie_data.csv". We construct random forest multinomial classifier for this data set.

In SAS:

```
proc import out=movie
datafile="./movie_data.csv" dbms=csv replace;
run;

/*SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS*/
proc surveyselect data=movie rate=0.8 seed=550040
out=movie outall method=srs;
run;

/*BUILDING RANDOM FOREST MULTINOMIAL CLASSIFIER*/
proc hpforest data=movie seed=454545
```

```
maxtrees=150 vars_to_try=4 trainfraction=0.7
maxdepth=10;
target rating/level=ordinal;
input age member/level=nominal;
input age nmovies/level=interval;
partition rolevar=selected(train='1');
save file='C:/Users/000110888/Desktop/random_forest.bin';
run;
```

Loss Reduction Variable Importance

Variable	Number of Rules	Gini	OOB Gini	Margin	OOB Margin
member	378	0.004665	-0.00439	0.002648	-0.00475
age	1234	0.049053	-0.02003	0.038625	-0.01943
nmovies	3134	0.051944	-0.02365	0.059274	-0.00833

```
/*COMPUTING PREDICTED VALUES FOR TESTING DATA*/
data test;
set movie;
if(selected='0');
run;

proc hp4score data=test;
id rating;
score file='C:/Users/000110888/Desktop/random_forest.bin'
out=predicted;
run;

/*COMPUTING PREDICTION ACCURACY FOR TESTING DATA*/
data predicted;
set predicted;
match=(rating=lowcase(I_rating));
run;
```

```
proc sql;
select sum(match)/count(*) as accuracy
from predicted;
quit;
```

accuracy
0.271523

In R:

```
movie.data<- read.csv(file="./movie_data.csv", header=TRUE, sep=",")

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
set.seed(566222)
sample <- sample(c(TRUE, FALSE), nrow(movie.data),replace=TRUE, prob=c(0.8,0.2))
train<- movie.data[sample,]
test<- movie.data[!sample,]

#BUILDING RANDOM FOREST MULTINOMIAL CLASSIFIER
library(randomForest)
rf.class<- randomForest(as.factor(rating) ~ age + gender + member + nmovies, data=train, ntree=150,
mtry=4, maxnodes=30)

#DISPLAYING FEATURE IMPORTANCE
print(importance(rf.class,type=2))
```

```
        MeanDecreaseGini
age             64.973483
gender          10.542498
member           5.531971
nmovies         25.470534
```

```
#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
predclass<- predict(rf.class, newdata=test)
test<- cbind(test,predclass)

accuracy<- c()
n<- nrow(test)
```

```
for (i in 1:n)
    accuracy[i]<- ifelse(test$rating[i]==test$predclass[i],1,0)

print(accuracy<- sum(accuracy)/length(accuracy))
```
0.3212435

In Python:

```
import pandas
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

movie_data=pandas.read_csv('C:/Users/000110888/Desktop/movie_data.csv')
code_gender={'M':1,'F':0}
code_member={'yes':1,'no':0}
code_rating={'very bad':1,'bad':2,'okay':3,'good':4,'very good':5}

movie_data['gender']=movie_data['gender'].map(code_gender)
movie_data['member']=movie_data['member'].map(code_member)
movie_data['rating']=movie_data['rating'].map(code_rating)

X=movie_data.iloc[:,0:4].values
y=movie_data.iloc[:,4].values

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.20, random_state=599555)

#FITTING RANDOM FOREST FOR MULTINOMIAL CLASSIFIER
rf_class=RandomForestClassifier(n_estimators=150, random_state=663474, max_depth=50, max_features=4)
rf_class.fit(X_train, y_train)

#DISPLAYING VARIABLE IMPORTANCE
from sklearn.ensemble import ExtraTreesClassifier

var_names=pandas.DataFrame(['age','gender','member','nmovies'], columns=['var_name'])
loss_reduction=pandas.DataFrame(rf_class.feature_importances_, columns=['loss_reduction'])
var_importance=pandas.concat([var_names, loss_reduction], axis=1)
var_importance=var_importance.sort_values("loss_reduction", axis=0, ascending=False)
print(var_importance)
```

```
  var_name  loss_reduction
0      age        0.572954
3  nmovies        0.269890
2   member        0.084660
1   gender        0.072495
```

```
#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
y_pred=rf_class.predict(X_test)
y_test=pandas.DataFrame(y_test,columns=['rating'])
y_pred=pandas.DataFrame(y_pred,columns=['predicted'])
df=pandas.concat([y_test,y_pred],axis=1)

match=[]
for i in range(len(df)):
    if df['rating'][i]==df['predicted'][i]:
        match.append(1)
    else:
        match.append(0)

accuracy=sum(match)/len(match)

print(accuracy)
```

0.3026315789473684

□

BOOSTING METHOD

Another ensemble method is known as **boosting**. Boosting as opposed to bagging doesn't involve bootstrap sampling. Instead, models are generated sequentially and iteratively, meaning that it is necessary to have information from iteration i before conducting iteration $i+1$. Note that the boosting process cannot be parallelized (modeling cannot be done on several trees simultaneously), unlike bagging, which is straightforwardly parallelizable.

The method of boosting was introduced by Michael Kearns and Leslie Valiant in 1989. The question posed asked whether it was possible to combine, in some fashion, a selection of weak machine learning models (termed **weak learners**) to produce a single strong machine learning model (a **strong learner**). Weak, in this instance means a model that is only slightly better than chance at predicting a response. Correspondingly, a strong learner is well-correlated to the true response.

This motivated the concept of boosting. The idea is to build iteratively weak machine learning models on a continually-updated response variable in the training data set and then add them together to produce a final, strong learning model. This differs from bagging, which simply averages the models on separate bootstrapped samples.

A basic boosting algorithm proceeds as follows:

1. The initial estimator is set to zero, that is, $\hat{f}(x) = 0$, and the residuals are set to current responses $r = y$, for all elements in the training set.
2. The number of boosting trees B is specified and then the loop over $b = 1, ..., B$ is run:
   Step 1. A weak-learning tree $\hat{f}^b$ with k splits is grown on the training data (x, r).
   Step 2. Estimator $\hat{f}$ is updated as $\hat{f}_{new}(x) = \hat{f}_{old}(x) + \lambda \hat{f}^b(x)$ for some scale parameter $\lambda,\ 0 < \lambda < 1$, called the **shrinkage rate** (or **learning rate**).
   Step 3. Residuals are updated as $r_{new} = r_{old} - \lambda \hat{f}^b(x)$.
3. The final boosted model is computed as the sum of individual weak learners, $\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x)$.

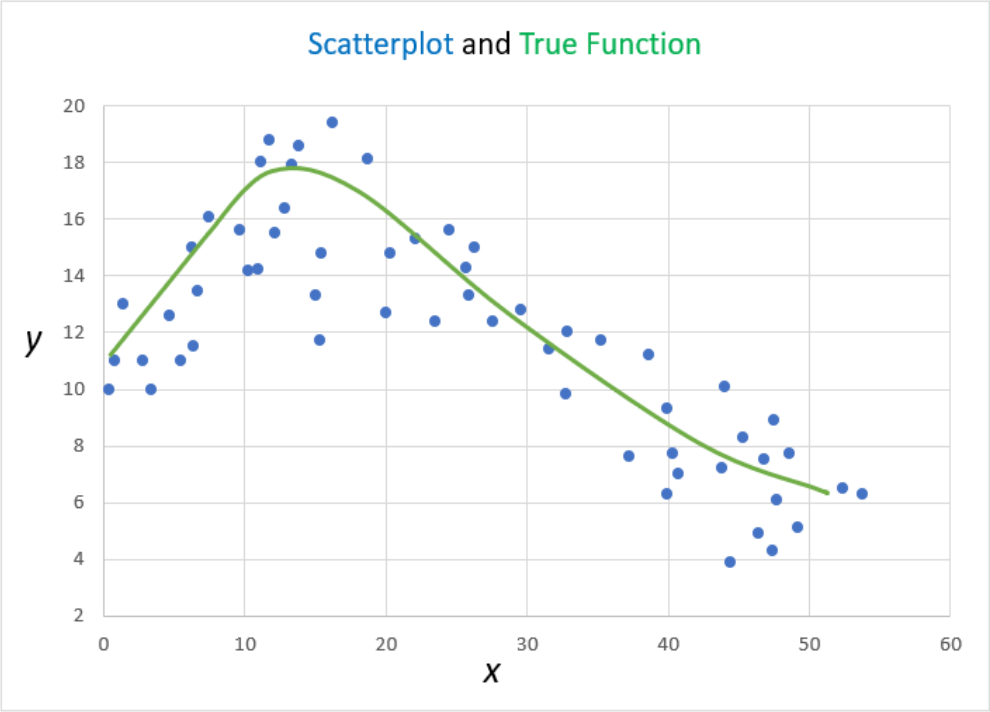
Notice that each subsequent tree is fitted to the residuals of the data. Hence each subsequent iteration is slowly improving the overall strong learner by improving its performance in poorly-performing regions of the feature space. It can be seen that this procedure is heavily dependent on the order in which the trees are grown. This process is said to **learn slowly**. Such **slow learning procedures** tend to produce well-performing machine learning models.

In the boosting algorithm, there are three hyperparameters: the number of boosted trees B, the number of splits k, and the shrinkage rate λ.

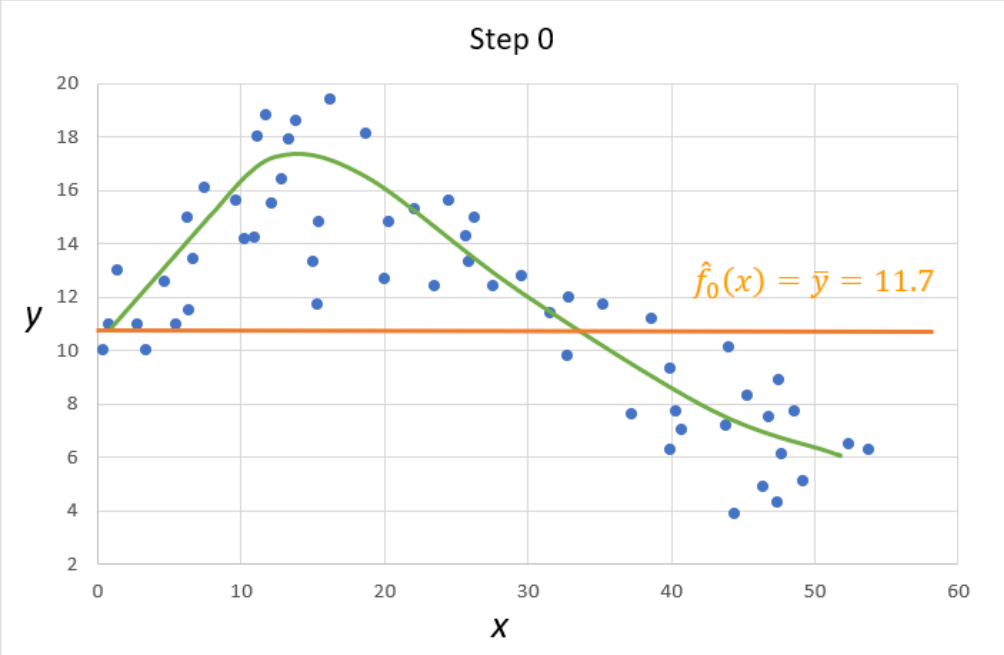
GRADIENT BOOSTING METHOD

The **gradient boosting** method combines the **method of gradient descent** (or **steepest descent**) and the boosting algorithm. It was first introduced by Jerome Friedman in 1999.

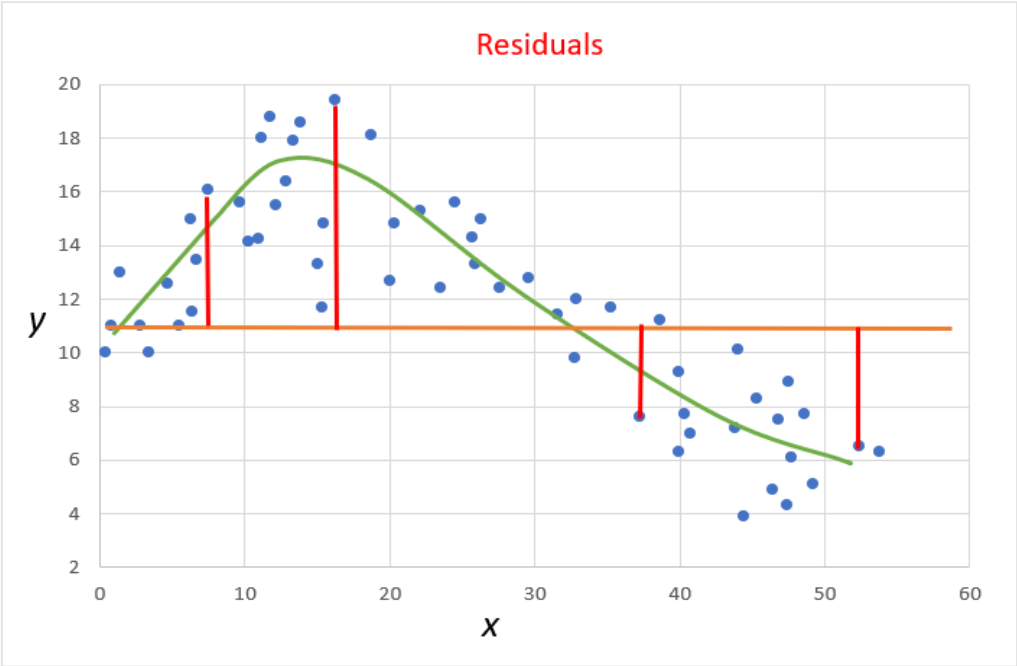
We present a simple example to explain how gradient boosting works. Suppose y depends on x through a non-linear relationship $y = f(x)$ depicted in the scatterplot below.



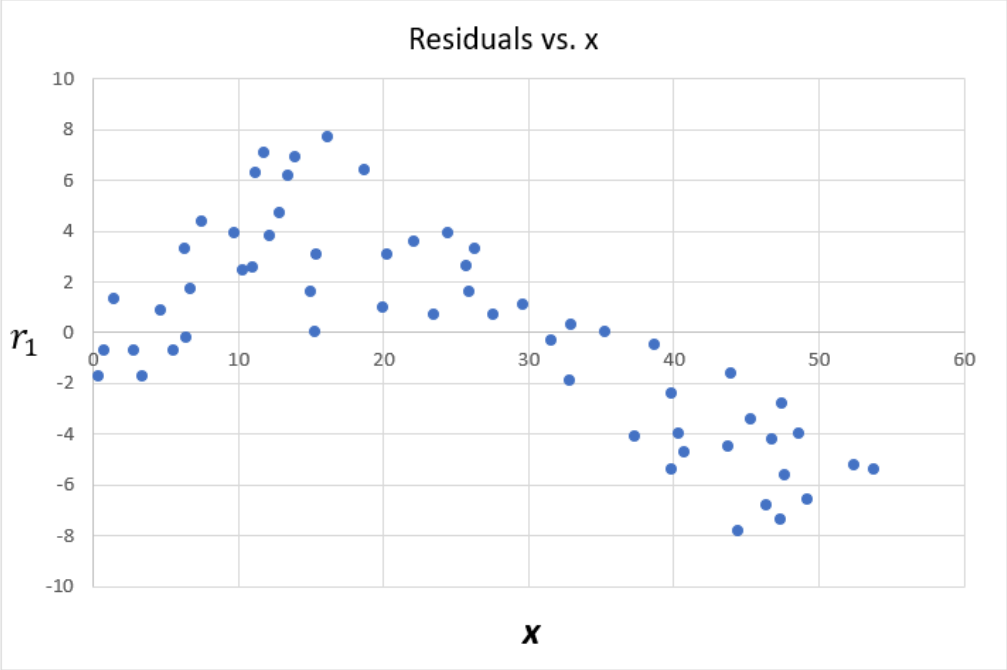
We initially predict the response y by the sample mean $\bar{y}$, that is, we let $\hat{f}_0(x) = \bar{y}$.



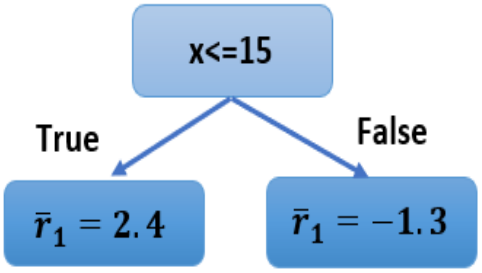
To improve our prediction, we will focus on the residuals (i.e., the vertical distances between the observed y's and the prediction $\bar{y}$). The residuals $r_1 = y - \bar{y}$ are shown as the vertical red lines in the figure below.



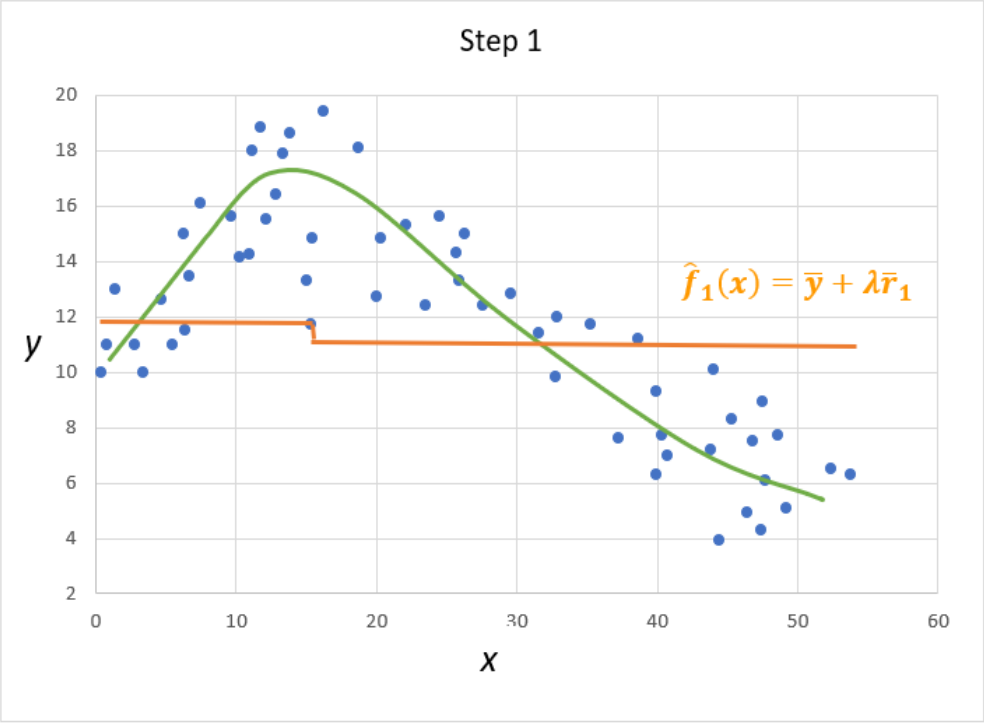
Next, we plot the residuals against x.



In the next step, we use the residuals r_1 as the target variable. Suppose for simplicity, we build a very simple regression tree, with one split and two terminal nodes (trees like this are called **decision stumps**). Suppose we split at 15, and so the decision stump looks this:

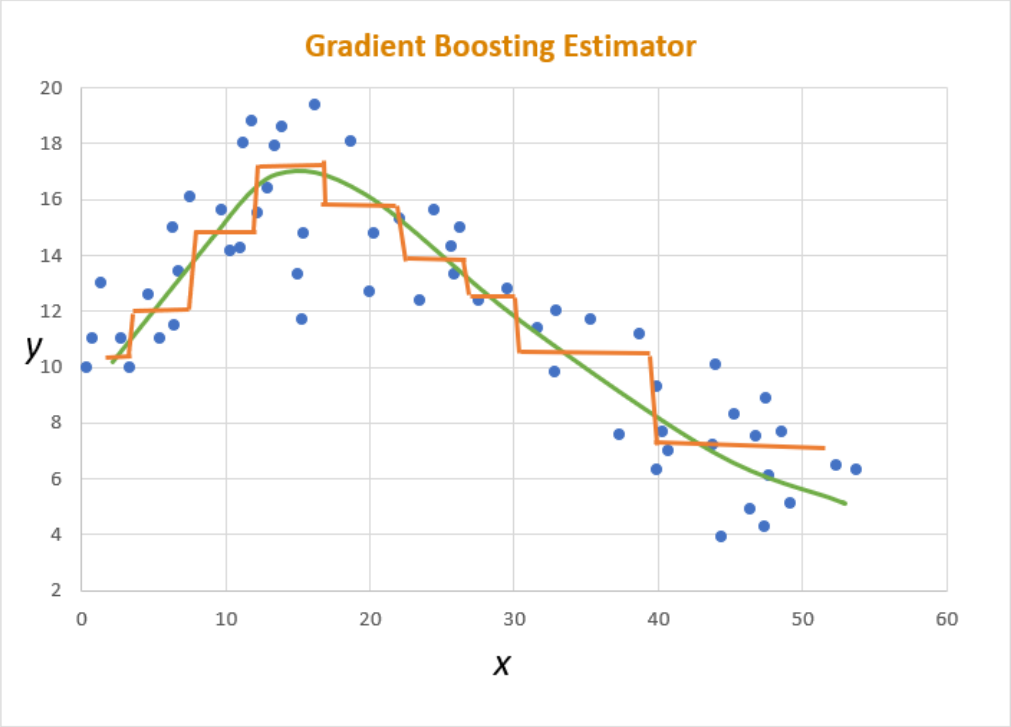


We then add the predicted $\bar{r}_1$ to the initial prediction $\bar{y}$ to reduce residuals. However, the gradient boosting algorithm does not simply add $\bar{r}_1$ to $\bar{y}$ as it overfits the model to the training data. Instead, $\bar{r}_1$ is scaled down by the shrinkage rate (or learning rate) λ, and then added to $\bar{y}$. For instance, we can take $\lambda = 0.2$. Then for $x \leq 15$, $\hat{f}_1(x) = \hat{f}_0(x) + \lambda \bar{r}_1 = \bar{y} + \lambda \bar{r}_1 = 11.7 + (0.2)(2.4) = 11.8$, and for $x > 15$, $\hat{f}_1(x) = 11.7 + (0.2)(-1.3) = 11.0$. We plot this fitted function in the figure below.



Now, in the next step, we update the residuals to $r_2 = y - \hat{f}_1(x)$ and build a regression tree, which will give us another split and another pair of estimates $\bar{r}_2$. We then update the fitted function $\hat{f}_2(x) = \hat{f}_1(x) + \lambda \bar{r}_2$.

We iterate these steps until the model prediction stops improving. The figures below schematically show the boosting process for some number of iterations.



Further, putting the gradient boosting algorithm into rigorous mathematical terms, we can write:

1. We initialize the model with a constant value $\hat{f}_0(x) = \arg\min_\gamma \sum_{i=1}^n L(y_i, \gamma)$ where by L we denote a pre-specified loss function.

Example. For the squared loss function $L = (y_i - \gamma)^2$, the value of γ that minimizes $\sum_{i=1}^n L(y_i, \gamma)$ solves

$$\frac{\partial}{\partial \gamma} \sum_{i=1}^n (y_i - \gamma)^2 = -2 \sum_{i=1}^n (y_i - \gamma) = -2 \sum_{i=1}^n y_i + 2n\gamma = 0.$$

Solving we get $\gamma = \bar{y}$. Thus, for the squared loss function, we initialize the model with $\hat{f}_0(x) = \bar{y}$. □

2. We define b as our iteration counter that will range between 1 and B. For each iteration b, we conduct the following steps:

Step 1. We compute **residuals** according to the formula

$$r_{ib} \;=\; -\Big[\frac{\partial L\big(y_i, f(x_i)\big)}{\partial f(x_i)}\Big]_{f(x_i)=\hat{f}_{b-1}(x_i)}, \quad i = 1, \ldots, n.$$

Example. For the squared loss function, we compute

$$r_{ib} \;=\; -\Big[\frac{\partial L\big(y_i, f(x_i)\big)}{\partial f(x_i)}\Big]_{f(x_i)=\hat{f}_{b-1}(x_i)} \;=\; -\Big[\frac{\partial \big(y_i - f(x_i)\big)^2}{\partial f(x_i)}\Big]_{f(x_i)=\hat{f}_{b-1}(x_i)} = 2\big(y_i - \hat{f}_{b-1}(x_i)\big).$$

Ignoring the multiplicative constant 2, we see that the residuals r_{ib} are the distances between the observed y_i and fitted $\hat{f}_{b-1}(x_i)$ (so, these are residuals in the regular sense). $\square$

Step 2. We train a regression tree with target r_{ib} and feature x. This tree defines terminal node regions $R_{jb},\ j = 1, \ldots, J_b$.

Step 3. For each region, we compute optimal estimators

$$\gamma_{jb} = \arg\min_{\gamma} \sum_{x_i \in R_{jb}} L\big(y_i, \hat{f}_{b-1}(x_i) + \gamma\big), \quad j = 1, \ldots, J_b.$$

Example. For the squared loss function, we compute

$$\gamma_{jb} = \arg\min_{\gamma} \sum_{x_i \in R_{jb}} L\big(y_i, \hat{f}_{b-1}(x_i) + \gamma\big) = \arg\min_{\gamma} \sum_{x_i \in R_{jb}} \big(y_i - \hat{f}_{b-1}(x_i) - \gamma\big)^2.$$

The value of γ that minimizes this sum is the solution of the equation:

$$\frac{\partial}{\partial \gamma} \sum_{x_i \in R_{jb}} \big(y_i - \hat{f}_{b-1}(x_i) - \gamma\big)^2 = 0,$$

$$-2 \sum_{x_i \in R_{jb}} \big(y_i - \hat{f}_{b-1}(x_i) - \gamma\big) = 0,$$

$$\sum_{x_i \in R_{jb}} \big(y_i - \hat{f}_{b-1}(x_i)\big) = \sum_{x_i \in R_{jb}} \gamma \;=\; \gamma\, n_{jb},$$

where n_{jb} is the number of data points in the region R_{jb}. Finally, we get

$$\gamma \;=\; \frac{1}{n_{jb}} \sum_{x_i \in R_{jb}} \big(y_i - \hat{f}_{b-1}(x_i)\big) = \frac{r_{jb}}{2n_{jb}} = \bar{r}_{jb}/2.$$

Ignoring the 2 in the denominator, we see that the estimator γ is the average of the residuals. $\square$

Step 4. We update the model as

$$\hat{f}_b(x) = \hat{f}_{b-1}(x) + \lambda \sum_{j=1}^{J_b} \gamma_{jb} \mathbb{I}(x \in R_{jb})$$

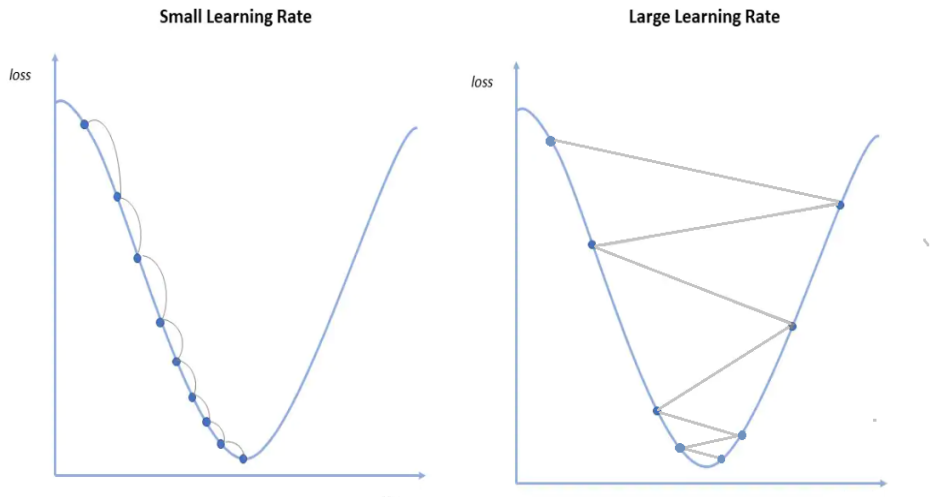
where the shrinkage (or learning) rate λ is a pre-defined constant between 0 and 1 (typically, 0.1 or smaller), and $\mathbb{I}(\cdot)$ is the indicator function (1 if true, and 0, otherwise).

Example. Consider the case of the squared loss function. Suppose x_i belongs to the region $R_j b$. We estimate $f_b(x_i)$ by

$$\hat{f}_b(x_i) = \hat{f}_{b-1}(x_i) + \lambda \sum_{j=1}^{J_b} \gamma_{jb} \mathbb{I}(x \in R_{jb}) = \hat{f}_{b-1}(x_i) + \lambda\, \bar{r}_{jb}/2 = f_{b-1}(x_i) + \lambda_1\, \bar{r}_{jb}.$$

Here λ_1 absorbed the constant 2, but still is a constant between 0 and 1, and can be considered the learning rate. □

Remark. The method of **gradient boosting** is closely related to the method of **gradient descent** (or **steepest descent**), which is a first-order iterative optimization algorithm for finding a local minimum of a differentiable function. The idea is to take repeated steps in the opposite direction of the gradient of the function at the current point because this is the direction of the steepest descent. The descent is depicted in the figure below. Note that if the learning rate (length of step) is small, one would descend slowly along one slope. However, one can choose to take large learning steps. Convergence is still guaranteed but it will take more time and computations to reach the minimum.



Remark. In the literature, the method of gradient boosting of a regression tree goes by a variety of names: gradient boosting machine (GBM), functional gradient boosting, multiple additive regression trees (MART), boosted regression trees (BRT), generalized boosting model, or tree net. In R, we will fit the extreme gradient boosting (XGBoost) method which is a popular modern implementation of the gradient boosting method with some extensions, like second-order optimization.

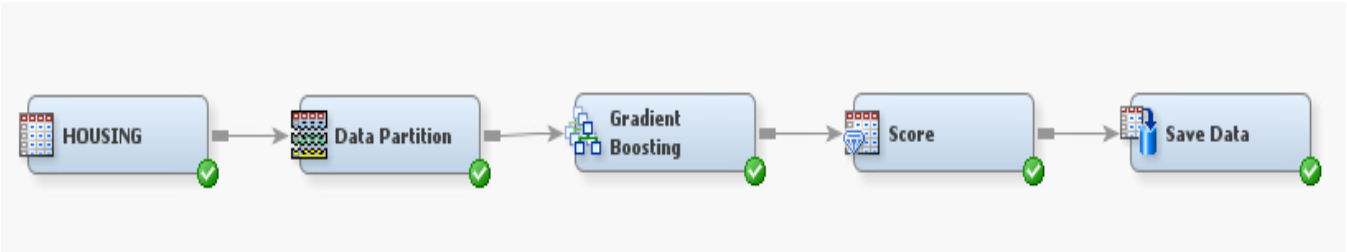
Example. We apply the gradient boosting algorithm to the data in the file "housing_data.csv". The codes below run the algorithm and output the list of features in the order of their importance, and also compute the proportion of correctly predicted median house prices within 10%, 15%, and 20% of the true values.

In SAS: Due to the large memory required to run the model, we have to resort to SAS Enterprise Miner Workstation 14.2 (in Student Virtual Lab). It runs on the SAS Viya platform that utilizes Cloud Analytics Service (CAS).

First, we go to Student Virtual Lab (SVL), open SAS, import the data set into SAS, and store it in the `sasuser` library. The code is:

```
proc import out=sasuser.housing datafile="./housing_data.csv" dbms=csv replace;
run;
```

Then we open SAS Enterprise Miner Workstation (EM), create a new project named, say, "XGBoostReg", and specify SAS Server Directory as the **desktop in SVL**. Then right-click "Data Sources" and extract the housing data set from the sasuser library. We also change the role of "median_house_value" to "target". Next, we right-click "Diagrams" and create a new process flow diagram depicted here:



We can set specifications for data partition (click on the node "Data Partition") to 80% of training data, 0% of validation data, and 20% of testing data. See the snippet below.

Data Set Allocations	
Training	80.0
Validation	0.0
Test	20.0

Further, it is recommended to set the specifications for the "Gradient Boosting" node to the ones displayed here:

Property	Value
General	
Node ID	Boost
Imported Data	
Exported Data	
Notes	
Train	
Variables	
Series Options	
N Iterations	50
Seed	786554
Shrinkage	0.01
Train Proportion	60
Splitting Rule	
Huber M-Regression	No
Maximum Branch	2
Maximum Depth	4
Minimum Categorical Size	5
Reuse Variable	2
Categorical Bins	30
Interval Bins	100
Missing Values	Use in search
Performance	Disk
Node	
Leaf Fraction	0.1
Number of Surrogate Rules	0
Split Size	.
Split Search	
Exhaustive	5000
Node Sample	20000
Subtree	
Assessment Measure	Average Square Error
Score	
Subseries	N Iterations
Number of Iterations	100

The next step is to right-click on the "Save Data" node and run the path. The output is the scored testing data set that is located in the SAS data file that can be retrieved from the folder "XGBooostReg" on the desktop. The path to the file is

"XGBoostReg/Workspaces/EMWS1/EMSave/em_save_test.sas7bdat".

Once we locate this file, we open it in SAS (in the library "Tmp1") and run the following lines of code to compute the proportion of predictions with 10%, 15%, and 20% of the true values.

```
data accuracy;
set tmp1.em_save_test;
ind10=(abs(R_median_house_value)<0.10*median_house_value);
ind15=(abs(R_median_house_value)<0.15*median_house_value);
ind20=(abs(R_median_house_value)<0.20*median_house_value);
run;

proc sql;
select sum(ind10)/count(*) as accuracy10,
sum(ind15)/count(*) as accuracy15,
sum(ind20)/count(*) as accuracy20
from accuracy;
quit;
```

accuracy10	accuracy15	accuracy20
0.172535	0.257042	0.304577

Remark. We can save the SAS code created in EM and run it in SAS 9.4, but SAS keeps crashing because the code requires a lot of memory. Alternatively to EM, one can create a session in CAS and run SAS code in that session. □.

In R:

```
#install.packages("xgboost")
library(xgboost)

housing.data<- read.csv(file="./housing_data.csv", header=TRUE, sep=",")

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
set.seed(203945)
sample <- sample(c(TRUE, FALSE), nrow(housing.data), replace=TRUE, prob=c(0.8,0.2))
train<- housing.data[sample,]
test<- housing.data[!sample,]

train.x<- data.matrix(train[-8])
train.y<- data.matrix(train[8])
```

```
test.x<- data.matrix(test[-8])
test.y<- data.matrix(test[8])

#FITTING EXTREME GRADIENT BOOSTED REGRESSION TREE
xgb.reg<- xgboost(data=train.x, label=train.y, max.depth=6, eta=0.01,
subsample=0.8, colsample_bytree=0.5, nrounds=1000, objective="reg:linear")
#eta=learning rate, colsample_bytree defines what percentage of features (columns)
# will be used for building each tree

#DISPLAYING FEATURE IMPORTANCE
print(xgb.importance(colnames(train.x), model=xgb.reg))
```

```
              Feature       Gain      Cover  Frequency
1:      median_income 0.46805085 0.23565339 0.16615392
2:    ocean_proximity 0.18991669 0.07413562 0.03699257
3:        total_rooms 0.10167336 0.15539794 0.19189115
4:         population 0.06185199 0.16370695 0.16587879
5:         households 0.06151389 0.14717595 0.14897076
6: housing_median_age 0.06136185 0.09601853 0.12748556
7:     total_bedrooms 0.05563136 0.12791161 0.16262725
```

```
#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
pred.y<- predict(xgb.reg, test.x)

#accuracy within 10%
accuracy10<- ifelse(abs(test.y-pred.y)<0.10*test.y,1,0)
print(sum(accuracy10)/length(accuracy10))
```

0.3344768

```
#accuracy within 15%
accuracy15<- ifelse(abs(test.y-pred.y)<0.15*test.y,1,0)
print(sum(accuracy15)/length(accuracy15))
```

0.4716981

```
#accuracy within 20%
accuracy20<- ifelse(abs(test.y-pred.y)<0.20*test.y,1,0)
print(sum(accuracy20)/length(accuracy20))
```

0.5763293

In Python:

```
import pandas
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingRegressor

housing=pandas.read_csv('C:/Users/000110888/OneDrive - CSULB/Desktop/housing_data.csv')
coding={'<1H OCEAN': 1, 'INLAND': 2, 'NEAR BAY': 3, 'NEAR OCEAN': 4}
housing['ocean_proximity']=housing['ocean_proximity'].map(coding)
X=housing.iloc[:,0:7].values
y=housing.iloc[:,7].values

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.20, random_state=115407)

#FITTING GRADIENT BOOSTED REGRESSION TREE
gbreg_params = {'n_estimators': 1000, 'max_depth': 6, 'learning_rate': 0.01,
'loss': 'squared_error'}
gb_reg=GradientBoostingRegressor(**gbreg_params)
gb_reg.fit(X_train, y_train)

#DISPLAYING VARIABLE IMPORTANCE
var_names=pandas.DataFrame(['housing_median_age','total_rooms','total_bedrooms','population',
'households','median_income','ocean_proximity'], columns=['var_name'])
loss_reduction=pandas.DataFrame(gb_reg.feature_importances_, columns=['loss_reduction'])
var_importance=pandas.concat([var_names, loss_reduction], axis=1)
print(var_importance.sort_values("loss_reduction", axis=0, ascending=False))
```

```
             var_name  loss_reduction
5       median_income        0.663030
6     ocean_proximity        0.132614
3          population        0.046976
0  housing_median_age        0.045639
2      total_bedrooms        0.040047
1         total_rooms        0.039412
4          households        0.032283
```

```
#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
y_pred=gb_reg.predict(X_test)

ind10=[]
ind15=[]
ind20=[]

for sub1, sub2 in zip(y_pred, y_test):
    ind10.append(1) if abs(sub1-sub2)<0.10*sub2 else ind10.append(0)
    ind15.append(1) if abs(sub1-sub2)<0.15*sub2 else ind15.append(0)
    ind20.append(1) if abs(sub1-sub2)<0.20*sub2 else ind20.append(0)

#accuracy within 10%
accuracy10=sum(ind10)/len(ind10)
print(accuracy10)

#accuracy within 15%
accuracy15=sum(ind15)/len(ind15)
print(accuracy15)

#accuracy within 20%
accuracy20=sum(ind20)/len(ind20)
print(accuracy20)
```

```
0.3602811950790861
0.47451669595782076
0.5905096660808435
```

□

Example. Here we apply the gradient boosting algorithm to the data in the file "pneumonia_data.csv". The codes below run the binary classifier algorithm and output the list of features in the order of their importance, and also compute prediction accuracy for a cut-off of 0.5.

Remark. With binary classifiers, a binary cross entropy loss function is used. It is defined as $L(y_i, \gamma) = y_i \ln \gamma + (1 - y_i) \ln(1 - \gamma)$. To minimize $\sum_{i=1}^{n} L(y_i, \gamma)$ with respect to γ, we solve

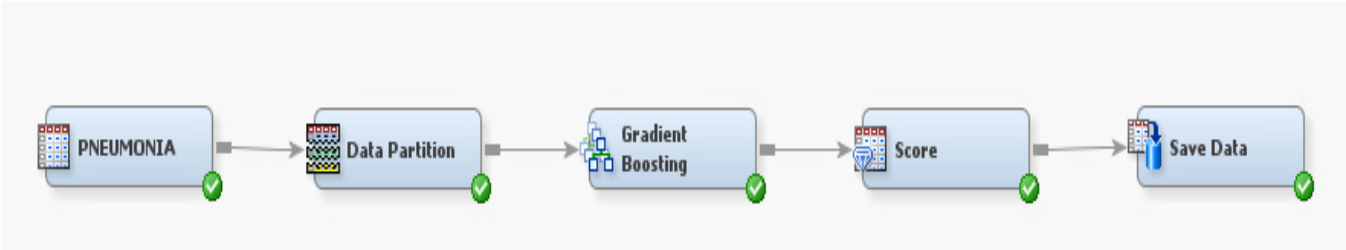
$$\frac{\partial}{\partial \gamma} \sum_{i=1}^{n} \left[y_i \ln \gamma + (1 - y_i) \ln(1 - \gamma) \right] = \sum_{i=1}^{n} \left[\frac{y_i}{\gamma} - \frac{1 - y_i}{1 - \gamma} \right] = \frac{n\bar{y}}{\gamma} - \frac{n - n\bar{y}}{1 - \gamma} = 0.$$

From here, $\gamma = \bar{y}$. □

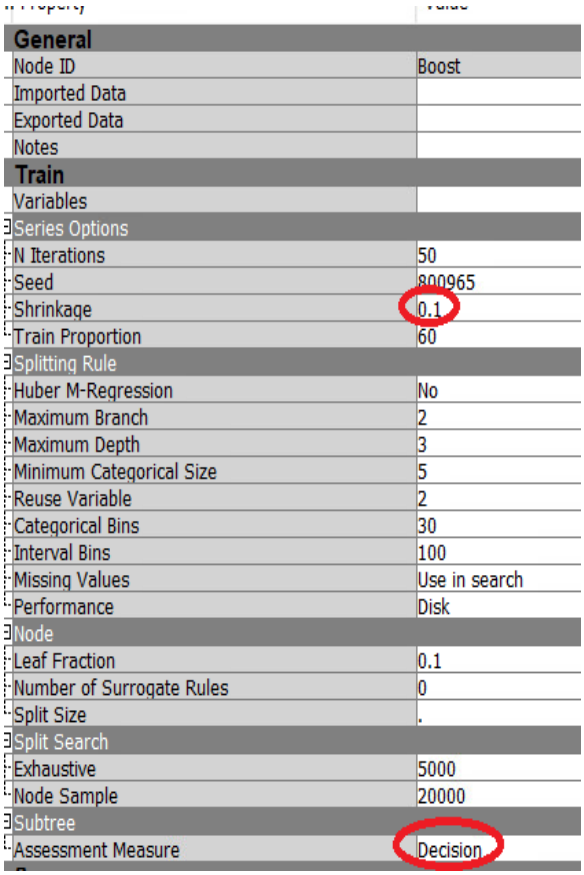
In SAS: First we save the data in "pneumonia_data.csv" in the sasuser library by running `proc import`:

```
proc import out=sasuser.pneumonia datafile="./pneumonia_data.csv"
dbms=csv replace;
run;
```

Then create a new project in the Enterprise Miner Workstation. The process flow diagram is:



For the "Data Partition" node, we specified 80% training, 0% validating, and 20% testing sets. The specifications for "Gradient Boosting" node are set at:



General	
Node ID	Boost
Imported Data	
Exported Data	
Notes	
Train	
Variables	
Series Options	
N Iterations	50
Seed	800965
Shrinkage	0.1
Train Proportion	60
Splitting Rule	
Huber M-Regression	No
Maximum Branch	2
Maximum Depth	3
Minimum Categorical Size	5
Reuse Variable	2
Categorical Bins	30
Interval Bins	100
Missing Values	Use in search
Performance	Disk
Node	
Leaf Fraction	0.1
Number of Surrogate Rules	0
Split Size	.
Split Search	
Exhaustive	5000
Node Sample	20000
Subtree	
Assessment Measure	Decision

Note: Shrinkage rate should be specified as 0.1 (0.01 is too small), and the assessment measure should be "Decision" to run a binary classification.

Once we run the path, the output is written into a SAS data file "em_save_test.sas7bdat". We open it in tmp1 folder and run these lines of code:

```
data tmp1.em_save_test;
set tmp1.em_save_test;
match=(EM_CLASSIFICATION=EM_CLASSTARGET);
run;

proc sql;
select sum(match)/count(*) as accuracy
from tmp1.em_save_test;
quit;
```

accuracy
0.771676

In R:

```
library(xgboost)

pneumonia.data<- read.csv(file="./pneumonia_data.csv", header=TRUE, sep=",")

pneumonia.data$pneumonia<- ifelse(pneumonia.data$pneumonia=="yes",1,0)

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
set.seed(447558)
sample <- sample(c(TRUE, FALSE), nrow(pneumonia.data), replace=TRUE, prob=c(0.8,0.2))
train<- pneumonia.data[sample,]
test<- pneumonia.data[!sample,]

train.x<- data.matrix(train[-5])
train.y<- data.matrix(train[5])
test.x<- data.matrix(test[-5])
test.y<- data.matrix(test[5])

#FITTING EXTREME GRADIENT BOOSTED BINARY CLASSIFIER
xgb.class<- xgboost(data=train.x, label=train.y, max.depth=6, eta=0.1, subsample=0.8,
colsample_bytree=0.5, nrounds=1000, objective="binary:logistic")

#DISPLAYING FEATURE IMPORTANCE
print(xgb.importance(colnames(train.x), model=xgb.class))
```

```
       Feature       Gain      Cover  Frequency
1:       PM2_5 0.62919417 0.51274361 0.50292653
2:         age 0.20259705 0.35043859 0.39769614
3:      gender 0.09556518 0.07007141 0.05678705
4: tobacco_use 0.07264360 0.06674638 0.04259029
```

```
#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
pred.prob<- predict(xgb.class, test.x)

len<- length(pred.prob)
pred.pneumonia<- c()
match<- c()
```

```
for (i in 1:len){
    pred.pneumonia[i]<- ifelse(pred.prob[i]>=0.5, 1,0)
    match[i]<- ifelse(test.y[i]==pred.pneumonia[i], 1,0)
}

print(prop<- sum(match)/len)
```

0.878187

In Python:

```
import pandas
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingClassifier

pneumonia_data=pandas.read_csv('C:/Users/000110888/OneDrive - CSULB/Desktop/pneumonia_data.csv')
code_gender={'M':1,'F':0}
code_tobacco_use={'yes':1,'no':0}
code_pneumonia={'yes':1,'no':0}

pneumonia_data['gender']=pneumonia_data['gender'].map(code_gender)
pneumonia_data['tobacco_use']=pneumonia_data['tobacco_use'].map(code_tobacco_use)
pneumonia_data['pneumonia']=pneumonia_data['pneumonia'].map(code_pneumonia)

X=pneumonia_data.iloc[:,0:4].values
y=pneumonia_data.iloc[:,4].values

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.20,
random_state=550078)

#FITTING GRADIENT BOOSTED BINARY CLASSIFIER
gbclass_params = {'n_estimators': 1000, 'max_depth': 6, 'learning_rate': 0.1}
gb_class=GradientBoostingClassifier(**gbclass_params)
gb_class.fit(X_train, y_train)

#DISPLAYING VARIABLE IMPORTANCE
var_names=pandas.DataFrame(['gender','age','tobacco_use','PM2_5'], columns=['var_name'])
loss_reduction=pandas.DataFrame(gb_class.feature_importances_, columns=['loss_reduction'])
var_importance=pandas.concat([var_names, loss_reduction], axis=1)
print(var_importance.sort_values("loss_reduction", axis=0, ascending=False))
```

```
      var_name  loss_reduction
3        PM2_5        0.623300
1          age        0.159204
0       gender        0.111911
2  tobacco_use        0.105585
```

```
#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
y_pred=gb_class.predict(X_test)
y_test=pandas.DataFrame(y_test,columns=['pneumonia'])
y_pred=pandas.DataFrame(y_pred,columns=['predicted'])
df=pandas.concat([y_test,y_pred],axis=1)

match=[]
for i in range(len(df)):
    if df['pneumonia'][i]==df['predicted'][i]:
        match.append(1)
    else:
        match.append(0)

accuracy=sum(match)/len(match)

print(accuracy)
```

```
0.8786127167630058
```

□

Example. We use the gradient boosting algorithm to solve the multinomial classification prediction problem on the data in the file "movie_data.csv".

Remark. For multinomial classifiers, a multi-class cross-entropy loss function is applied. Suppose there are c classes, and n_j observations belong to class $j, j = 1, \ldots, c$, where $n_1 + \ldots, n_c = n$. Denote by t_{ij} an indicator of y_i belonging to class j. Note that $\sum_{i=1}^{n} t_{ij} = n_j, j = 1, \ldots, c$. The loss function is given by $L(y_i, \gamma_1, \ldots, \gamma_c) = t_{i1} \ln \gamma_1 + t_{i2} \ln \gamma_2 + \cdots + t_{ic} \ln \gamma_c = t_{i1} \ln \gamma_1 + t_{i2} \ln \gamma_2 + \cdots + t_{ic} \ln(1 - \gamma_1 - \gamma_2 - \cdots - \gamma_{c-1})$. To minimize $\sum_{i=1}^{n} L(y_i, \gamma_1, \ldots, \gamma_c)$ with respect to $\gamma_1, \ldots, \gamma_c$, we solve for $j = 1, \ldots, c - 1$,

$$\frac{\partial}{\partial \gamma_j} \sum_{i=1}^{n} \left[t_{i1} \ln \gamma_1 + t_{i2} \ln \gamma_2 + \cdots + t_{ic} \ln(1 - \gamma_1 - \gamma_2 - \cdots - \gamma_{c-1}) \right] = \sum_{i=1}^{n} \left[\frac{t_{ij}}{\gamma_j} - \frac{t_{jc}}{1 - \gamma_1 - \gamma_2 - \cdots - \gamma_{c-1}} \right]$$

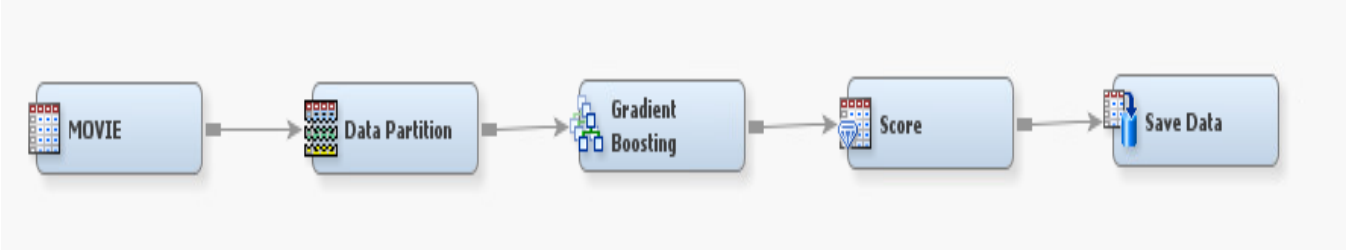
$$= \frac{n_j}{\gamma_j} - \frac{n_c}{1 - \gamma_1 - \gamma_2 - \cdots - \gamma_{c-1}} = \frac{n_j}{\gamma_j} - \frac{n_c}{\gamma_c} = 0, \quad \text{or}, \quad \gamma_j = \frac{n_j}{n_c} \gamma_c.$$

Since $\gamma_1 + \gamma_2 + \cdots + \gamma_c = 1$, we have that $\gamma_c\left(\frac{n_1}{n_c} + \cdots + \frac{n_{c-1}}{n_c} + 1\right) = 1$, or $\gamma_c = \frac{n_c}{n_1 + \cdots + n_c} = \frac{n_c}{n}$, and $\gamma_j = \frac{n_j}{n_c} \cdot \frac{n_c}{n} = \frac{n_j}{n}$, $j = 1, \ldots, c$. $\square$.

In SAS: We run proc import to create a SAS data file sasuser.movie:

```
proc import out=sasuser.movie datafile="./movie_data.csv"
dmbs=csv replace;
run;
```

Then in EM we create the following flow diagram:



For the "Data Partition" node, we specified 80% training, 0% validating, and 20% testing sets. The specifications for "Gradient Boosting" nodes are set at:

Series Options	
N Iterations	50
Seed	400113
Shrinkage	0.1
Train Proportion	60
Splitting Rule	
Huber M-Regression	No
Maximum Branch	2
Maximum Depth	3
Minimum Categorical Size	5
Reuse Variable	2
Categorical Bins	30
Interval Bins	100
Missing Values	Use in search
Performance	Disk
Node	
Leaf Fraction	0.1
Number of Surrogate Rules	0
Split Size	.
Split Search	
Exhaustive	5000
Node Sample	20000
Subtree	
Assessment Measure	Decision

Next we run the path and open the resulting data file "em_save_test.sas7bdat" in the tmp1 library. The accuracy of prediction is evaluated by submitting this code:

```
data tmp1.em_save_test;
set tmp1.em_save_test;
match=(EM_CLASSIFICATION=EM_CLASSTARGET);
run;

proc sql;
select sum(match)/count(*) as accuracy
from tmp1.em_save_test;
quit;
```

accuracy
0.312102

In R:

```
library(xgboost)

movie.data<- read.csv(file="./movie_data.csv", header=TRUE, sep=",")

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
set.seed(103321)
sample <- sample(c(TRUE, FALSE), nrow(movie.data), replace=TRUE, prob=c(0.8,0.2))
train<- movie.data[sample,]
test<- movie.data[!sample,]

train.x<- data.matrix(train[-5])
train.y<- data.matrix(train[5])
train.y<- train.y-1 #must range between 0 and 4 for prediction
test.x<- data.matrix(test[-5])
test.y<- data.matrix(test[5])
test.y<- test.y-1

#FITTING GRADIENT BOOSTED MULTINOMIAL CLASSIFIER
xgb.mclass<- xgboost(data=train.x, label=train.y, max.depth=6, eta=0.1, subsample=0.8, colsam-
ple_bytree=0.5, nrounds=1000, num_class=5, objective="multi:softprob")

#DISPLAYING FEATURE IMPORTANCE
print(xgb.importance(colnames(train.x), model=xgb.mclass))
```

```
   Feature       Gain      Cover  Frequency
1:     age 0.66514614 0.51438911 0.56634783
2: nmovies 0.21685109 0.29686671 0.26099577
3:  gender 0.06530198 0.09410162 0.08994277
4:  member 0.05270080 0.09464255 0.08271363
```

```
#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
pred.prob <- predict(xgb.mclass, test.x, reshape=TRUE)
pred.prob<- as.data.frame(pred.prob)
colnames(pred.prob)<- 0:4

pred.class<- apply(pred.prob, 1, function(x) colnames(pred.prob)[which.max(x)])

match<- c()
n<- length(test.y)
for (i in 1:n) {
    match[i]<- ifelse(pred.class[i]==as.character(test.y[i]),1,0)
```

}

print(accuracy<- sum(match)/n)

0.2435897

```
import pandas
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingClassifier

movie_data=pandas.read_csv('C:/Users/000110888/OneDrive - CSULB/Desktop/movie_data.csv')
code_gender={'M':1,'F':0}
code_member={'yes':1,'no':0}
code_rating={'very bad':1,'bad':2,'okay':3,'good':4,'very good':5}

movie_data['gender']=movie_data['gender'].map(code_gender)
movie_data['member']=movie_data['member'].map(code_member)
movie_data['rating']=movie_data['rating'].map(code_rating)

X=movie_data.iloc[:,0:4].values
y=movie_data.iloc[:,4].values

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.20,
random_state=566033)

#FITTING GRADIENT BOOSTED MULTINOMIAL CLASSIFIER
gbmclass_params = {'n_estimators': 1000, 'max_depth': 6, 'learning_rate': 0.1}
gb_mclass=GradientBoostingClassifier(**gbmclass_params)
gb_mclass.fit(X_train, y_train)

#DISPLAYING VARIABLE IMPORTANCE
var_names=pandas.DataFrame(['age','gender','member','nmovies'], columns=['var_name'])
loss_reduction=pandas.DataFrame(gb_mclass.feature_importances_, columns=['loss_reduction'])
var_importance=pandas.concat([var_names, loss_reduction], axis=1)
print(var_importance.sort_values("loss_reduction", axis=0, ascending=False))
```

```
  var_name  loss_reduction
0      age        0.510458
3  nmovies        0.323769
1   gender        0.092721
2   member        0.073052
```

```
#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
y_pred=gb_mclass.predict(X_test)
y_test=pandas.DataFrame(y_test,columns=['rating'])
y_pred=pandas.DataFrame(y_pred,columns=['predicted'])
df=pandas.concat([y_test,y_pred],axis=1)

match=[]
for i in range(len(df)):
    if df['rating'][i]==df['predicted'][i]:
        match.append(1)
    else:
        match.append(0)

accuracy=sum(match)/len(match)

print(accuracy)
```

```
0.3223684210526316
```

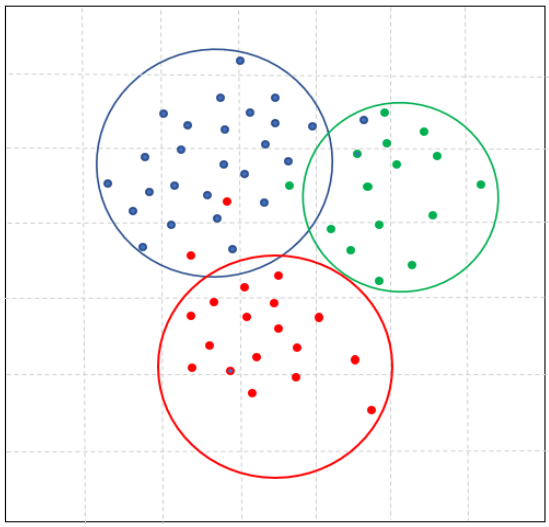
□

K-NEAREST NEIGHBOR REGRESSION AND CLASSIFICATION

The **k Nearest-neighbor (kNN) algorithm** can be used for regression as well as binary multinomial classifications. The space is divided into classes with k nearest neighbors in each class. Euclidean distance is used to measure the distance between neighbors. The **Euclidean distance** between two d-dimensional vectors $\mathbf{x} = (x_1, \ldots, x_d)$ and $\mathbf{y} = (y_1, \ldots, y_d)$ in our regular Euclidean geometry is defined as $distance(\mathbf{x}, \mathbf{y}) = \sqrt{(x_1 - y_1)^2 + \cdots + (x_d - y_d)^2}$. For regression, the mean value of the class is used for prediction. In classification, the proportion of observations in each

category is computed as predicted probabilities within each class.

As an illustration, the figure below depicts the partitioning of the space into three classes (red, blue, and green). Surely, some outlying observations are misclassified as they are closer to the observations in some other class.



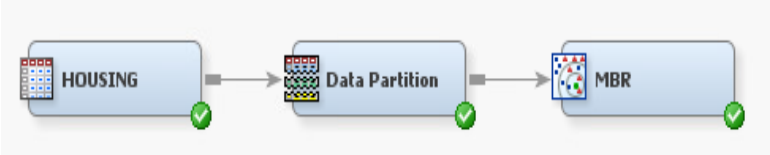
Historical Note: The kNN algorithm was first described in "Discriminatory Analysis. Nonparametric Discrimination: Consistency Properties", by Evelyn Fix and Joseph Hodges, report, UC Berkeley, 1951.

Example. We apply the kNN algorithm to build a regression for the data set in the file "housing_data.csv".

In SAS: We save the data file in the sasuser library using the following code.

```
proc import out=sasuser.housing datafile="./housing_data.csv" dbms=csv replace;
run;
```

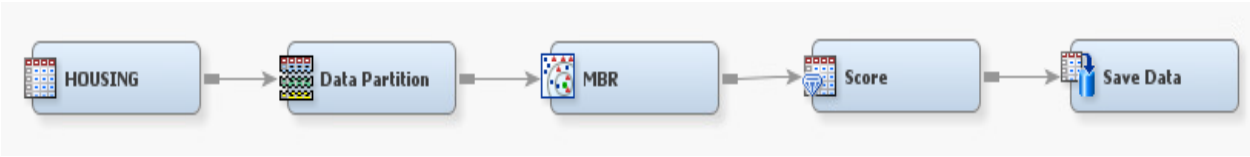
Then we use SAS Enterprise miner to fit a k nearest-neighbor regression (termed **Memory-Based Reasoning (MBR)**, using the path diagram:



For the "Data Partition" node, we specify to split the data into 70% training, 10% validation, and 20% testing sets. The testing set contains 569 observations, the validation test contains 284 observations, and the training set contains 1989 observations. In the MBR node, we specify the number of neighbors: $569/5 = 113.8 \approx 114$ for $k = 5$ classes, $569/6 = 94.8 \approx= 95$ for $k = 6$ classes, etc. We run the paths and note the value for the MSE for the testing set (summarized in the table below).

Number of classes	Number of neighbors	MSE
5	114	6918425992
6	95	6830858760
7	81	6719387249
8	71	6646008646
9	**63**	**6605133365**
10	57	6586164839

We can see that MSE starts leveling out at $k = 9$, so we pick that number of classes and run the full path that includes scoring of the testing set. The diagram is given in the following figure:



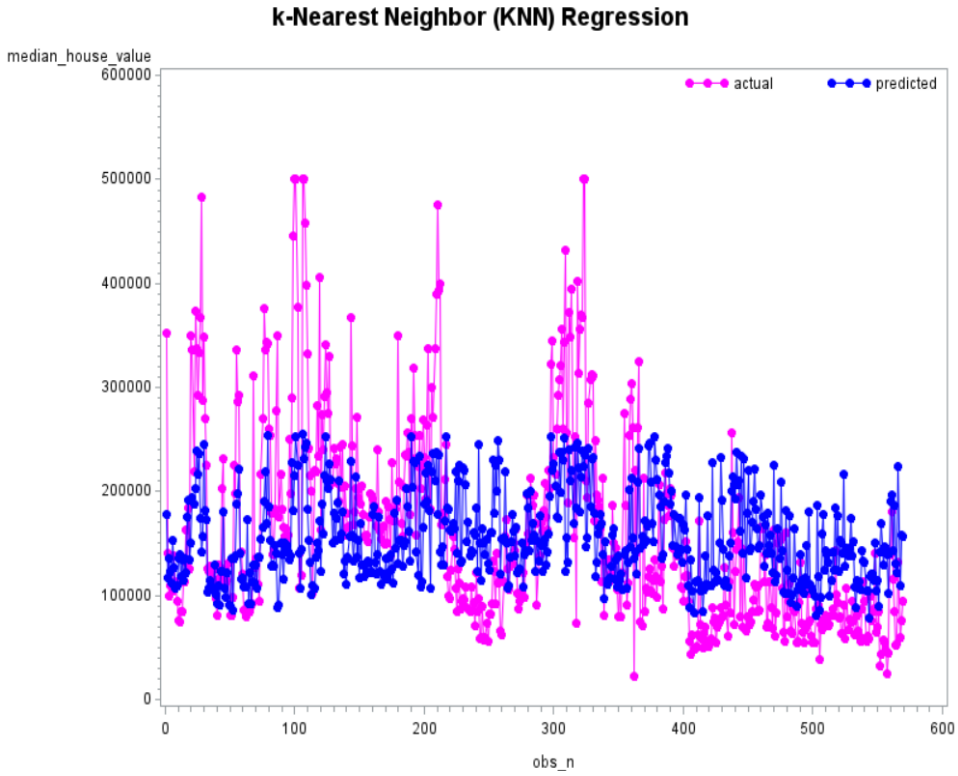
Next, we locate the file with predictions "./Workspaces/EMWS1/EMSave/em_save_test.sas7bdat", open it in SAS (in the library "Tmp1") and run the code below to compute the proportion of predictions with 10%, 15%, and 20% of the actual values, and plot the actual and predicted values.

```
/*COMPUTING ACCURACY WITHIN 10%, 15%, AND 20%*/
data accuracy;
set tmp1.em_save_test;
ind10=(abs(R_median_house_value)<0.10*median_house_value);
ind15=(abs(R_median_house_value)<0.15*median_house_value);
ind20=(abs(R_median_house_value)<0.20*median_house_value);
```

```
obs_n=_N_;
run;

proc sql;
select mean(ind10) as accuracy10,
mean(ind15) as accuracy15, mean(ind20) as
accuracy20
from accuracy;
quit;
```

accuracy10	accuracy15	accuracy20
0.142355	0.214411	0.265378

```
/*PLOTTING ACTUAL AND PREDICTED VALUES FOR TESTING DATA*/;
goptions reset=all border;
title1 "k-Nearest Neighbor (KNN) Regression";
symbol1 interpol=join value=dot color=magenta;
symbol2 interpol=join value=dot color=blue;
legend1 value=("actual" "predicted")
position=(top right inside) label=none;
proc gplot data=accuracy;
plot median_house_value*obs_n
EM_PREDICTION*obs_n/ overlay legend=legend1;
run;
```



In R:

```
housing.data<- read.csv(file="./housing_data.csv", header=TRUE, sep=",")

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
set.seed(880352)
sample <- sample(c(TRUE, FALSE), nrow(housing.data), replace=TRUE, prob=c(0.8,0.2))
train<- housing.data[sample,]
test<- housing.data[!sample,]

train.x<- data.matrix(train[-8])
train.y<- data.matrix(train[8])
test.x<- data.matrix(test[-8])
test.y<- data.matrix(test[8])

#TRAINING K-NEAREST NEIGHBOR REGRESSION
install.packages("caret") #Classification and Regression Training
library(caret)
print(train(median_house_value~ ., data=train, method="knn"))
```

```
  k  RMSE
  5  81040.04
  7  79199.11
  9  78165.81

RMSE was used to select the optimal model using the smallest value.
The final value used for the model was k = 9.
```

```
#FITTING OPTIMAL KNN REGRESSION (K=9)
knn.reg<- knnreg(train.x, train.y, k=9)

#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
pred.y<- predict(knn.reg, test.x)

#accuracy within 10%
accuracy10<- ifelse(abs(test.y-pred.y)<0.10*test.y,1,0)
print(mean(accuracy10))
```

```
0.1372881
```

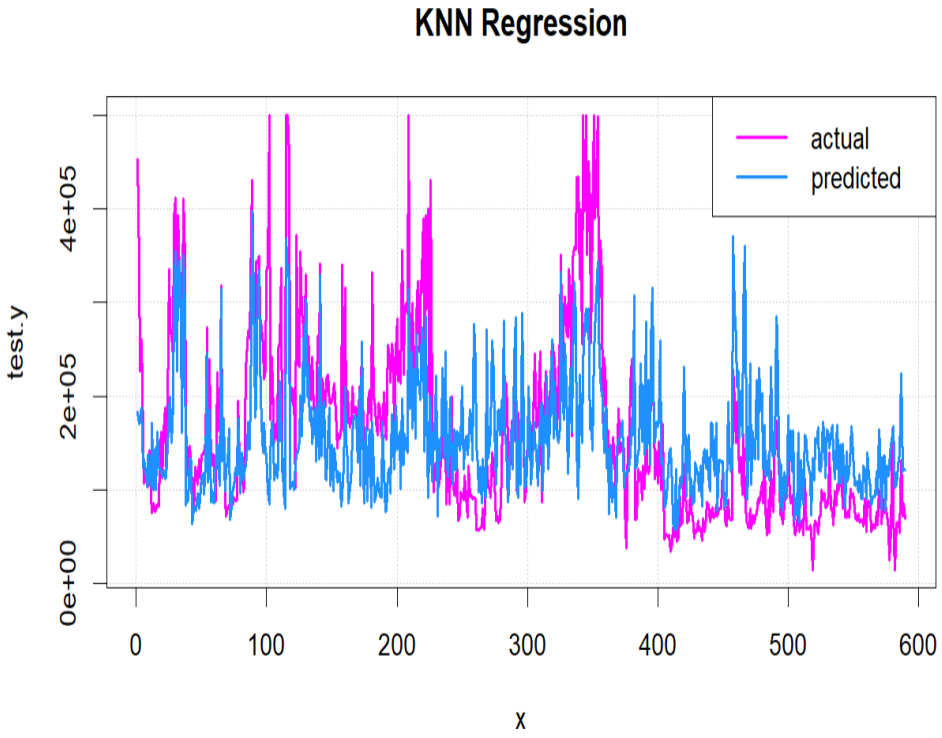
```
#accuracy within 15%
accuracy15<- ifelse(abs(test.y-pred.y)<0.15*test.y,1,0)
print(mean(accuracy15))
```

```
0.1830508
```

```
#accuracy within 20%
accuracy20<- ifelse(abs(test.y-pred.y)<0.20*test.y,1,0)
print(mean(accuracy20))
```

```
0.2610169
```

```
#PLOTTING ACTUAL AND PREDICTED VALUES FOR TESTING DATA
x<- 1:length(test.y)
plot(x, test.y, type="l", lwd=2, col="magenta", main="KNN Regression", panel.first=grid())
lines(x, pred.y, lwd=2, col="dodgerblue")
legend("topright", c("actual", "predicted"), lty=1, lwd=2,col=c("magenta","dodgerblue"))
```

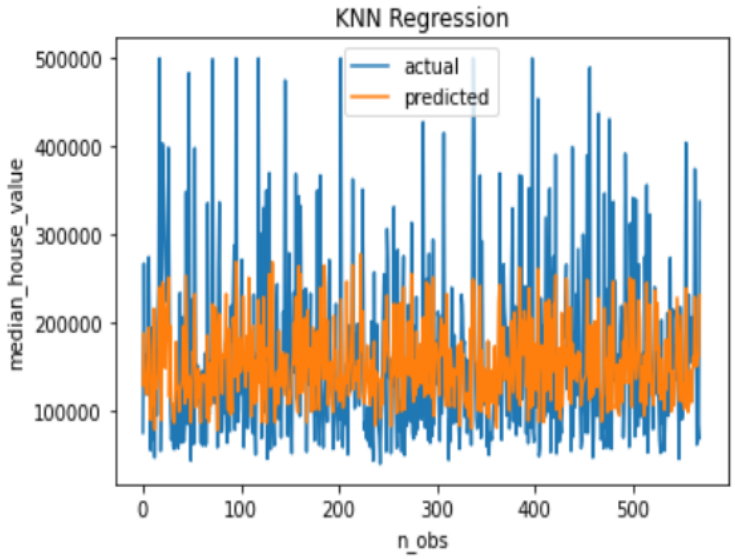

KNN Regression
actual
predicted
test.y
0e+00
2e+05
4e+05
0
100
200
300
400
500
600
x

In Python:

```
import pandas
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from statistics import mean

housing=pandas.read_csv('./housing_data.csv')
coding={'<1H OCEAN': 1, 'INLAND': 2, 'NEAR BAY': 3, 'NEAR OCEAN': 4}
housing['ocean_proximity']=housing['ocean_proximity'].map(coding)
X=housing.iloc[:,0:7].values
y=housing.iloc[:,7].values

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.20,
random_state=833567)

#FITTING kNN REGRESSION
reg=KNeighborsRegressor(n_neighbors=63)
kNN_reg=reg.fit(X_train, y_train)

#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
y_pred=kNN_reg.predict(X_test)
ind10=[]
ind15=[]
ind20=[]

for sub1, sub2 in zip(y_pred, y_test):
    ind10.append(1) if abs(sub1-sub2)<0.10*sub2 else ind10.append(0)
    ind15.append(1) if abs(sub1-sub2)<0.15*sub2 else ind15.append(0)
    ind20.append(1) if abs(sub1-sub2)<0.20*sub2 else ind20.append(0)

#accuracy within 10%
accuracy10=mean(ind10)
print('accuracy within 10% =', round(accuracy10,4))

#accuracy within 15%
accuracy15=mean(ind15)
print('accuracy within 15% =', round(accuracy15,4))

#accuracy within 20%
accuracy20=mean(ind20)
print('accuracy within 20% =', round(accuracy20,4))
```

```
accuracy within 10% = 0.1213
accuracy within 15% = 0.1898
accuracy within 20% = 0.2689
```

```
#plotting actual and predicted obsevations vs. observation number
import matplotlib.pyplot as plt

n_obs=list(range(0,len(y_test)))
plt.plot(n_obs, y_test, label="actual")
plt.plot(n_obs, y_pred, label="predicted")
plt.xlabel('n_obs')
plt.ylabel('median_house_value')
plt.title('KNN Regression')
plt.legend()
plt.show()
```



□

Example. For the data set "pneumonia_data.csv", we build the kNN binary classifier.

In SAS:

In SAS Enterprise Miner, we partition the data into 70% training, 10% validation, and 20% testing sets (346 rows), and run the MBR node, varying the number of neighbors ($= 346/k$) and record the misclassification rate for the testing set. The results are summarized here:

Number of classes	Number of neighbors	Misclassification Rate
3	115	0.321
4	**87**	**0.301**
5	69	0.321
6	58	0.318
7	49	0.321
8	43	0.335

We choose to utilize $k = 4$ classes because it results in the smallest misclassification rate, and run the full path:

Now we open the SAS file with scored data "em_save_test.sas7dat" in the tmp1 folder and compute the accuracy of prediction by running the following SAS code:

```
/*COMPUTING PREDICTION ACCURACY*/
data accuracy;
set tmp1.em_save_test;
match=(em_classification=em_classtarget);
run;

proc sql;
select mean(match) as accuracy
from accuracy;
quit;
```

accuracy
0.699422

In R:
pneumonia.data<- read.csv(file="./pneumonia_data.csv", header=TRUE, sep=",")

```
pneumonia.data$pneumonia<- ifelse(pneumonia.data$pneumonia=="yes",1,0)

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
set.seed(704467)
sample <- sample(c(TRUE, FALSE), nrow(pneumonia.data), replace=TRUE, prob=c(0.8,0.2))
train<- pneumonia.data[sample,]
test<- pneumonia.data[!sample,]

train.x<- data.matrix(train[-5])
train.y<- data.matrix(train[5])
test.x<- data.matrix(test[-5])
test.y<- data.matrix(test[5])

#TRAINING K-NEAREST NEIGHBOR BINARY CLASSIFIER
library(caret)
print(train(as.factor(pneumonia)~ ., data=train, method="knn"))


  k  Accuracy
  5  0.6704615
  7  0.6781109
  9  0.6807456

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was k = 9.

#FITTING OPTIMAL KNN BINARY CLASSIFIER (K=9)
knn.class<- knnreg(train.x, train.y, k=9)

#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
pred.prob<- predict(knn.class, test.x)

len<- length(pred.prob)
pred.y<- c()
match<- c()
for (i in 1:len){
    pred.y[i]<- ifelse(pred.prob[i]>=0.5, 1,0)
    match[i]<- ifelse(test.y[i]==pred.y[i], 1,0)
}
print(paste("accuracy=",round(mean(match),digits=4)))


"accuracy= 0.7072"
```

```
#alternative (frugal) way
pred.y1<- floor(0.5+predict(knn.class, test.x))
print(paste("accuracy=", round(1-mean(test.y!=pred.y1),digits=4)))
```

```
"accuracy= 0.7072"
```

In Python:

```
import pandas
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from statistics import mean

pneumonia_data=pandas.read_csv('./pneumonia_data.csv')
code_gender={'M':1,'F':0}
code_tobacco_use={'yes':1,'no':0}
code_pneumonia={'yes':1,'no':0}

pneumonia_data['gender']=pneumonia_data['gender'].map(code_gender)
pneumonia_data['tobacco_use']=pneumonia_data['tobacco_use'].map(code_tobacco_use)
pneumonia_data['pneumonia']=pneumonia_data['pneumonia'].map(code_pneumonia)

X=pneumonia_data.iloc[:,0:4].values
y=pneumonia_data.iloc[:,4].values

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.20,
random_state=459147)

#FITTING kNN BINARY CLASSIFIER WITH k=4
biclass=KNeighborsClassifier(n_neighbors=87)
kNN_biclass=biclass.fit(X_train, y_train)

#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
y_pred=kNN_biclass.predict(X_test)
y_test=pandas.DataFrame(y_test,columns=['pneumonia'])
y_pred=pandas.DataFrame(y_pred,columns=['predicted'])
df=pandas.concat([y_test,y_pred],axis=1)
```

```
match=[]
for i in range(len(df)):
    if df['pneumonia'][i]==df['predicted'][i]:
        match.append(1)
    else:
        match.append(0)

print('accuracy=', mean(match))
```

```
0.6705202312138728
```

□

Example. Consider the data in the file "movie_data.csv". We fit a multinomial classifier using the kNN algorithm.

In SAS:

In SAS Enterprise Miner, we run the path ending in the MBR node for various numbers of classes (see the table below). For the analysis, we choose $k = 5$ as it corresponds to the minimal misclassification rate for the testing set (157 rows).

Number of classes	Number of neighbors	Misclassification Rate
3	52	0.720
4	39	0.739
5	**31**	**0.707**
6	26	0.707
7	22	0.726

We use $k = 5$ and run the full path depicted in the figure below.

We open the scored testing set in the tmp1 folder in SAS and compute the accuracy of prediction.

```
/*COMPUTING PREDICTION ACCURACY*/
data accuracy;
set tmp1.em_save_test;
match=(em_classification=em_classtarget);
run;

proc sql;
select mean(match) as accuracy
from accuracy;
quit;
```

accuracy
0.292994

In R:

```
movie.data<- read.csv(file="./movie_data.csv", header=TRUE, sep=",")

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
set.seed(123857)
sample <- sample(c(TRUE, FALSE), nrow(movie.data), replace=TRUE, prob=c(0.8,0.2))
train<- movie.data[sample,]
test<- movie.data[!sample,]

train.x<- data.matrix(train[-5])
train.y<- data.matrix(train[5])
test.x<- data.matrix(test[-5])
test.y<- data.matrix(test[5])

#FITTING K-NEAREST NEIGHBOR MULTINOMIAL CLASSIFIER
#k=3 reasonably maximizes prediction accuracy for testing set
library(caret)
knn.mclass<- knnreg(train.x, train.y, k=3)

#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
pred.y<- round(predict(knn.mclass, test.x), digits=0)
print(paste("accuracy=", round(1-mean(test.y!=pred.y),digits=4)))
```

```
accuracy= 0.2133
```

In Python:

```
import pandas
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from statistics import mean

movie_data=pandas.read_csv('./movie_data.csv')
code_gender={'M':1,'F':0}
code_member={'yes':1,'no':0}
code_rating={'very bad':1,'bad':2,'okay':3,'good':4,'very good':5}

movie_data['gender']=movie_data['gender'].map(code_gender)
movie_data['member']=movie_data['member'].map(code_member)
movie_data['rating']=movie_data['rating'].map(code_rating)

X=movie_data.iloc[:,0:4].values
y=movie_data.iloc[:,4].values

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.20,
random_state=844632)

#FITTING kNN MULTINOMIAL CLASSIFIER
multiclass=KNeighborsClassifier(n_neighbors=31)
kNN_multiclass=multiclass.fit(X_train, y_train)

#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
y_pred=kNN_multiclass.predict(X_test)
y_test=pandas.DataFrame(y_test,columns=['rating'])
y_pred=pandas.DataFrame(y_pred,columns=['predicted'])
df=pandas.concat([y_test,y_pred],axis=1)

match=[]
for i in range(len(df)):
    if df['rating'][i]==df['predicted'][i]:
        match.append(1)
    else:
        match.append(0)

print('accuracy=', mean(match))
```

```
accuracy= 0.3157894736842105
```

□

SUPPORT VECTOR MACHINE REGRESSION AND CLASSIFICATION

Historical Note. Support vector machine (SVM) analysis is a machine learning tool for regression and classification. It was first proposed by Vladimir Vapnik in his book "The Nature of Statistical Learning Theory", Springer-Verlag, New York, 1995.

Support Vector Regression

The goal of Support Vector Regression is to find a function $f(x_1, ..., x_k)$ that deviates from the observed response y by a value not greater than a pre-specified ε for each training point, and at the same time is as flat as possible.

Let $x = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1k} \\ & & \cdots & \\ x_{n1} & x_{n2} & \dots & x_{nk} \end{bmatrix}$ be the $n \times k$ matrix of predictor values in the training set. Let $\beta = (\beta_1, \dots, \beta_k)'$ be the column-vector of slopes, and $b = (b_1, \dots, b_n)'$ be the column-vector of intercepts. To find a linear function $f(x) = x\beta + b$ and ensure that it is as flat as possible, we need to find $f(x)$ with the minimal norm $J(\beta) = \frac{1}{2}\beta'\beta$. We also need to observe the constraint that all residuals do not exceed ε, that is, $\left|y_i - \mathrm{x}_i\beta - b\right| \le \varepsilon, \;\; i = 1, ..., n$, where $\mathrm{x}_i = (x_{i1}, ..., x_{ik})$.

It is possible that no such function f exists. To deal with the infeasible constraints, two non-negative **slack variables** ξ_i and ξ_i^* are introduced for each data point. The objective now is to minimize (the expression is termed the **primal formula**) $J(\beta) = \frac{1}{2}\beta'\beta + C\sum_{i=1}^{n}(\xi_i + \xi_i^*)$ such that $y_i - \mathrm{x}_i\beta - b \le \varepsilon + \xi_i$, and $\mathrm{x}_i\beta + b - y_i \le \varepsilon + \xi_i^*$, for all $i = 1, \dots, n$. Here the constant C is the **box constraint**, a positive numeric value that controls the penalty imposed on observations that lie outside the ε-margin and helps to prevent overfitting (it is also known as **regularization parameter**). This value determines the trade-off between the flatness of f and the amount up to which deviations larger than ε are tolerated.

The optimization problem is computationally simpler if formulated in terms of **Lagrange multipliers** α_i and α_i^* for the ith individual, $i = 1, \dots, n$. This leads to minimization of **Lagrangian** (the expression is termed the **dual formula**):

$$L(\alpha) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)\mathrm{x}_i\mathrm{x}_j' \; + \; \varepsilon\sum_{i=1}^{n}\left(\alpha_i + \alpha_i^*\right) \; + \; \sum_{i=1}^{n}\left(\alpha_i^* - \alpha_i\right)y_i,$$

subject to the constraints:

$$\sum_{i=1}^{n}\left(\alpha_i - \alpha_i^*\right) = 0, \;\; \text{and} \;\; 0 \le \alpha_i, \; \alpha_i^* \le C.$$

The β parameter is found as

$$\beta = \sum_{i=1}^{n} \big(\alpha_i - \alpha_i^*\big)\mathrm{x}_i'.$$

The function f is calculated according to the formula:

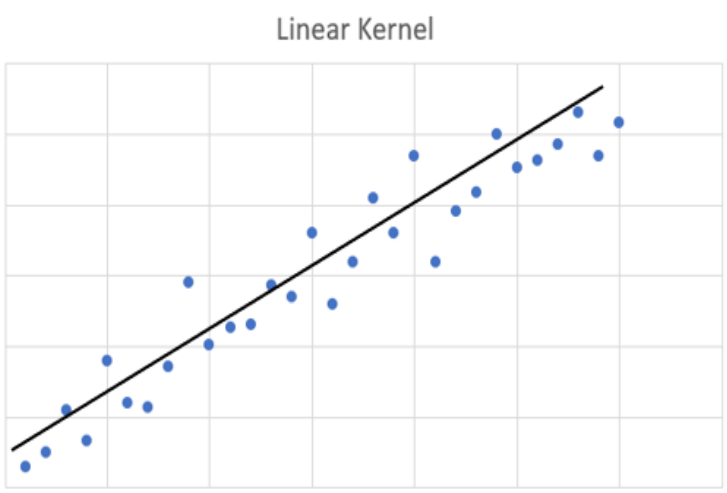
$$f(x) = \sum_{i=1}^{n} \big(\alpha_i - \alpha_i^*\big)\mathrm{x}_i\, x' + b.$$

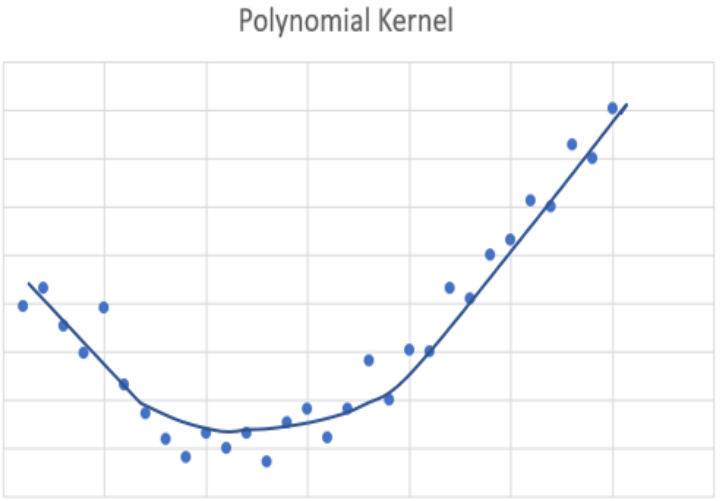
To obtain the optimal solution, the **Karush-Kuhn-Tucker (KKT) complementarity conditions** are used as optimization constraints. For linear support vector regression they are as follows: $\alpha_i(\varepsilon + \xi_i - y_i + \mathrm{x}_i\beta + b) = 0$, $\alpha_i^*(\varepsilon + \xi_i^* + y_i - \mathrm{x}_i\beta - b) = 0$, $\xi_i(C - \alpha_i) = 0$, and $\xi_i^*(C - \alpha_i^*) = 0$, for any $i = 1, \ldots, n$. These conditions indicate that all observations strictly inside the epsilon tube have Lagrange multipliers $\alpha_i = \alpha_i^* = 0$. Those observations for which Lagrange multipliers are non-zero (observations on the boundary of the epsilon tube) are called **support vectors**.

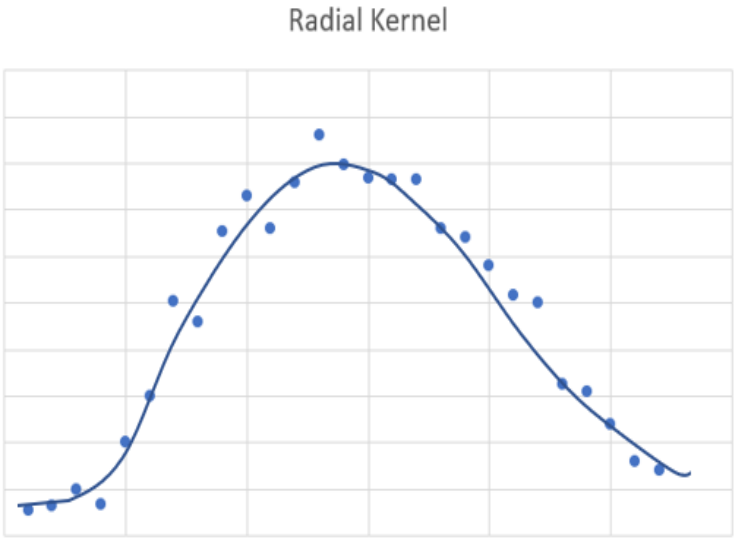
Some regression problems, however, cannot be adequately described using a linear model. In such a case, the Lagrange dual formulation allows it to be extended to nonlinear functions, using kernels. Nonlinear support vector regression finds the coefficients that minimize the Lagrangian

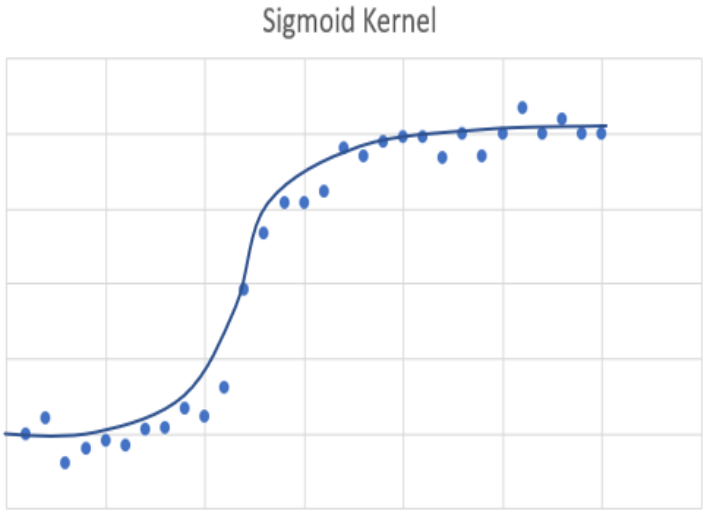
$$L(\alpha) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)G\big(\mathrm{x}_i, \mathrm{x}_j\big) \;+\; \varepsilon\sum_{i=1}^{n}\big(\alpha_i + \alpha_i^*\big) \;+\; \sum_{i=1}^{n}\big(\alpha_i^* - \alpha_i\big)y_i,$$

where $G\big(\mathrm{x}_i, \mathrm{x}_j\big)$ is the kernel function. Several types of kernels are used: $G\big(\mathrm{x}_i, \mathrm{x}_j\big) = \mathrm{x}_i\mathrm{x}_j'$ is a **linear kernel**, $G\big(\mathrm{x}_i, \mathrm{x}_j\big) = \big(1 + \mathrm{x}_i\mathrm{x}_j'\big)^d$ is a **polynomial** kernel of degree d, $G\big(\mathrm{x}_i, \mathrm{x}_j\big) = \exp\big(-||\mathrm{x}_i - \mathrm{x}_j||^2\big)$ is a **radial basis function (RBF)** or **radial** or **Gaussian** kernel, and $G\big(\mathrm{x}_i, \mathrm{x}_j\big) = tanh(\mathrm{x}_i\mathrm{x}_j')$ is a **sigmoid** kernel, where $tanh(x) = (e^x - e^{-x})/(e^x + e^{-x})$ is the hyperbolic function (see the illustrations below).









Example. We apply the support vector regression to the data in the file "housing_data.csv". The codes below run the analysis and compute the prediction accuracy within 10%, 15%, and 20% of the true values. SAS Enterprise Miner doesn't handle this method, so we use R and Python only.

In R:

```
housing.data<- read.csv(file="./housing_data.csv", header=TRUE, sep=",")

housing.data$ocean_proximity<- ifelse(housing.data$ocean_proximity=='<1H OCEAN', 1, ifelse(housing.data$
ifelse(housing.data$ocean_proximity=='NEAR BAY',3,4)))
```

```
#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
set.seed(234564)
sample <- sample(c(TRUE, FALSE), nrow(housing.data), replace=TRUE, prob=c(0.8,0.2))
train<- housing.data[sample,]
test<- housing.data[!sample,]
test.x<- data.matrix(test[-8])
test.y<- data.matrix(test[8])

install.packages("e1071")
library(e1071)

#FITTING SVR WITH LINEAR KERNEL
svm.reg<- svm(median_house_value ~ housing_median_age+total_rooms+total_bedrooms
+population+households+median_income+ocean_proximity, data=train, kernel="linear")

#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
pred.y<- predict(svm.reg, test.x)

#accuracy within 10%
accuracy10<- ifelse(abs(test.y-pred.y)<0.10*test.y,1,0)

#accuracy within 15%
accuracy15<- ifelse(abs(test.y-pred.y)<0.15*test.y,1,0)

#accuracy within 20%
accuracy20<- ifelse(abs(test.y-pred.y)<0.20*test.y,1,0)

print('Linear Kernel')


"Linear Kernel"

print(paste('within 10%:', round(mean(accuracy10),4)))


"within 10%: 0.2212"

print(paste('within 15%:', round(mean(accuracy15),4)))


"within 15%: 0.3583"

print(paste('within 20%:', round(mean(accuracy20),4)))
```

```
"within 20%: 0.5031"
```

```
#FITTING SVR WITH POLYNOMIAL KERNEL
svm.reg<- svm(median_house_value ~ housing_median_age+total_rooms+total_bedrooms
+population+households+median_income+ocean_proximity, + data=train, kernel="poly")

#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
pred.y<- predict(svm.reg, test.x)

#accuracy within 10%
accuracy10<- ifelse(abs(test.y-pred.y)<0.10*test.y,1,0)

#accuracy within 15%
accuracy15<- ifelse(abs(test.y-pred.y)<0.15*test.y,1,0)

#accuracy within 20%
accuracy20<- ifelse(abs(test.y-pred.y)<0.20*test.y,1,0)

print('Polynomial Kernel')
```

```
"Polynomial Kernel"
```

```
print(paste('within 10%:', round(mean(accuracy10),4)))
```

```
"within 10%: 0.2414"
```

```
print(paste('within 15%:', round(mean(accuracy15),4)))
```

```
"within 15%: 0.3536"
```

```
print(paste('within 20%:', round(mean(accuracy20),4)))
```

```
"within 20%: 0.4408"
```

```
#FITTING SVR WITH RADIAL KERNEL
svm.reg<- svm(median_house_value ~ housing_median_age+total_rooms+total_bedrooms
+population+households+median_income+ocean_proximity,+ data=train, kernel="radial")

#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
pred.y<- predict(svm.reg, test.x)
```

```
#accuracy within 10%
accuracy10<- ifelse(abs(test.y-pred.y)<0.10*test.y,1,0)

#accuracy within 15%
accuracy15<- ifelse(abs(test.y-pred.y)<0.15*test.y,1,0)

#accuracy within 20%
accuracy20<- ifelse(abs(test.y-pred.y)<0.20*test.y,1,0)

print('Radial Kernel')
```

```
"Radial Kernel"
```

```
print(paste('within 10%:', round(mean(accuracy10),4)))
```

```
"within 10%: 0.3676"
```

```
print(paste('within 15%:', round(mean(accuracy15),4)))
```

```
"within 15%: 0.5249"
```

```
print(paste('within 20%:', round(mean(accuracy20),4)))
```

```
"within 20%: 0.6526"
```

```
#FITTING SVR WITH SIGMOID KERNEL
svm.reg<- svm(median_house_value ~ housing_median_age+total_rooms+total_bedrooms
+population+households+median_income+ocean_proximity, + data=train, kernel="sigmoid")

#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
pred.y<- predict(svm.reg, test.x)

#accuracy within 10%
accuracy10<- ifelse(abs(test.y-pred.y)<0.10*test.y,1,0)

#accuracy within 15%
accuracy15<- ifelse(abs(test.y-pred.y)<0.15*test.y,1,0)

#accuracy within 20%
accuracy20<- ifelse(abs(test.y-pred.y)<0.20*test.y,1,0)

print('Sigmoid Kernel')
```

```
"Sigmoid Kernel"
```

```
print(paste('within 10%:', round(mean(accuracy10),4)))
```

```
"within 10%: 0.0047"
```

```
print(paste('within 15%:', round(mean(accuracy15),4)))
```

```
"within 15%: 0.0093"
```

```
print(paste('within 20%:', round(mean(accuracy20),4)))
```

```
"within 20%: 0.014"
```

We see that the support vector regression with the radial kernel has the best prediction accuracy, thus, it's the best-fitted model.

In Python:

```
import pandas
from sklearn.model_selection import train_test_split
from sklearn.svm import SVR
from statistics import mean

housing=pandas.read_csv('C:/Users/000110888/OneDrive - CSULB/Desktop/housing_data.csv')
coding={'<1H OCEAN': 1, 'INLAND': 2, 'NEAR BAY': 3, 'NEAR OCEAN': 4}
housing['ocean_proximity']=housing['ocean_proximity'].map(coding)
X=housing.iloc[:,0:7].values
y=housing.iloc[:,7].values

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.20,
random_state=445021)

##############################################################################################
#FITTING SUPPORT VECTOR REGRESSION WITH LINEAR KERNEL
svreg_linear=SVR(kernel='linear').fit(X_train, y_train)

#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
y_pred=svreg_linear.predict(X_test)

ind10=[]
ind15=[]
ind20=[]

for sub1, sub2 in zip(y_pred, y_test):
    ind10.append(1) if abs(sub1-sub2)<0.10*sub2 else ind10.append(0)
    ind15.append(1) if abs(sub1-sub2)<0.15*sub2 else ind15.append(0)
    ind20.append(1) if abs(sub1-sub2)<0.20*sub2 else ind20.append(0)

print('Linear Kernel Prediction Accuracy')
#accuracy within 10%
print('within 10%:', round(mean(ind10),4))

#accuracy within 15%
print('within 15%:', round(mean(ind15),4))
```

```
#accuracy within 20%
print('within 20%:', round(mean(ind20),4))

#########################################################################################
#FITTING SUPPORT VECTOR REGRESSION WITH POLYNOMIAL KERNEL
svreg_poly=SVR(kernel='poly').fit(X_train, y_train)

#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
y_pred=svreg_poly.predict(X_test)

ind10=[]
ind15=[]
ind20=[]

for sub1, sub2 in zip(y_pred, y_test):
    ind10.append(1) if abs(sub1-sub2)<0.10*sub2 else ind10.append(0)
    ind15.append(1) if abs(sub1-sub2)<0.15*sub2 else ind15.append(0)
    ind20.append(1) if abs(sub1-sub2)<0.20*sub2 else ind20.append(0)

print('')
print('Polynomial Kernel Prediction Accuracy')
#accuracy within 10%
print('within 10%:', round(mean(ind10),4))

#accuracy within 15%
print('within 15%:', round(mean(ind15),4))

#accuracy within 20%
print('within 20%:', round(mean(ind20),4))


#########################################################################################
#FITTING SUPPORT VECTOR REGRESSION WITH RADIAL KERNEL
svreg_radial=SVR(kernel='rbf').fit(X_train, y_train)

#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
y_pred=svreg_radial.predict(X_test)

ind10=[]
ind15=[]
ind20=[]
```

```
for sub1, sub2 in zip(y_pred, y_test):
    ind10.append(1) if abs(sub1-sub2)<0.10*sub2 else ind10.append(0)
    ind15.append(1) if abs(sub1-sub2)<0.15*sub2 else ind15.append(0)
    ind20.append(1) if abs(sub1-sub2)<0.20*sub2 else ind20.append(0)

print('')
print('Radial Kernel Prediction Accuracy')
#accuracy within 10%
print('within 10%:', round(mean(ind10),4))

#accuracy within 15%
print('within 15%:', round(mean(ind15),4))

#accuracy within 20%
print('within 20%:', round(mean(ind20),4))

#############################################################################################
#FITTING SUPPORT VECTOR REGRESSION WITH SIGMOID KERNEL
svreg_sigmoid=SVR(kernel='sigmoid').fit(X_train, y_train)

#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
y_pred=svreg_sigmoid.predict(X_test)

ind10=[]
ind15=[]
ind20=[]

for sub1, sub2 in zip(y_pred, y_test):
    ind10.append(1) if abs(sub1-sub2)<0.10*sub2 else ind10.append(0)
    ind15.append(1) if abs(sub1-sub2)<0.15*sub2 else ind15.append(0)
    ind20.append(1) if abs(sub1-sub2)<0.20*sub2 else ind20.append(0)

print('')
print('Sigmoid Kernel Prediction Accuracy')
#accuracy within 10%
print('within 10%:', round(mean(ind10),4))

#accuracy within 15%
print('within 15%:', round(mean(ind15),4))

#accuracy within 20%
print('within 20%:', round(mean(ind20),4))
```

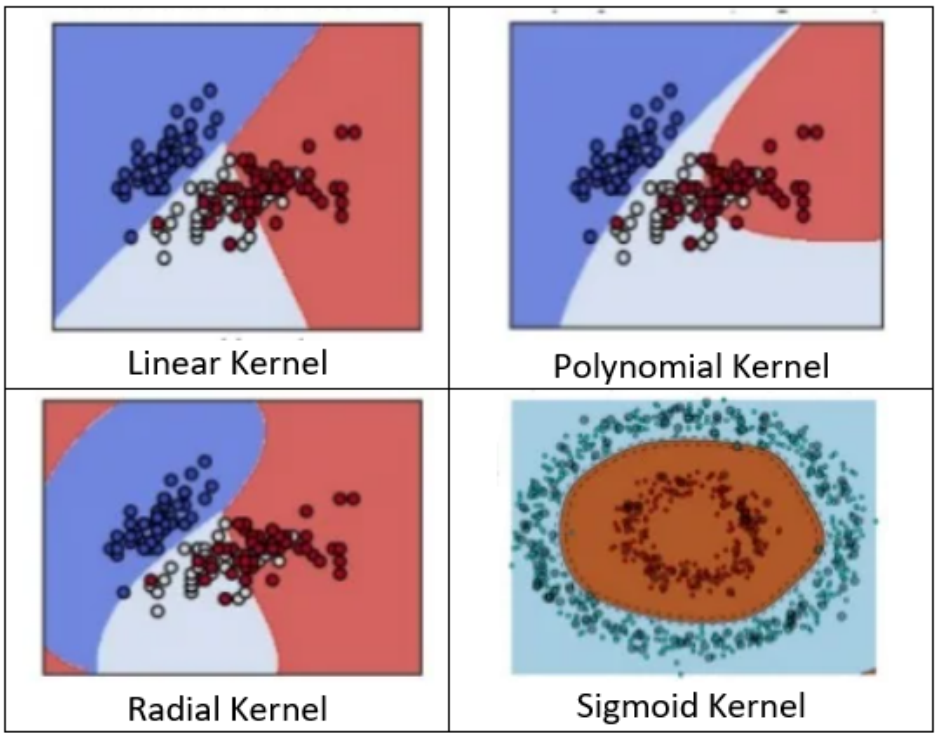
```
Linear Kernel Prediction Accuracy
within 10%: 0.1494
within 15%: 0.2267
within 20%: 0.2953
```

```
Polynomial Kernel Prediction Accuracy
within 10%: 0.1019
within 15%: 0.1634
within 20%: 0.2179

Radial Kernel Prediction Accuracy
within 10%: 0.1002
within 15%: 0.1634
within 20%: 0.2197

Sigmoid Kernel Prediction Accuracy
within 10%: 0.0967
within 15%: 0.1617
within 20%: 0.2179
```

Comparing prediction accuracies, we see that the SVR with the linear kernel fits the data the best. □

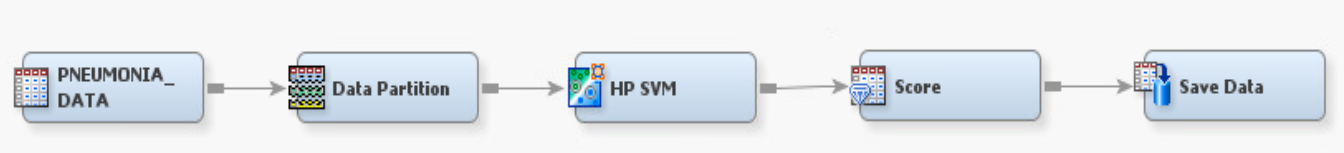
Support Vector Machine for Binary Classifier

For binary response, a support vector machine classifies data by finding the best hyperplane that separates data points of one class from those of the other class. The best hyperplane for an SVM is the one with the largest margin between the two classes. Margin means the maximal width of the slab parallel to the hyperplane that has no interior data points. The **support vectors** are the data points that are closest to the separating hyperplane; these points are on the boundary of the slab. In mathematical terms, the response variable $y_i = \pm 1$ represents the category that the ith individual belongs to. The hyperplane has the equation $f(x) = x\beta + b = 0$. To find the best separating hyperplane, we find β and b that minimize $\beta'\beta$ such that for all $i = 1, \ldots, n$, $y_i f(x_i) \geq 1$. The **support vectors** are the points on the boundary, that is, those for which $y_i f(x_i) = 1$. The dual formulation in this setting is to maximize with respect to α $\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j \, y_i \, y_j x_i x_j'$, subject to the constraints $\sum_{i=1}^{n} y_i \alpha_i = 0$, and $0 \leq \alpha_i \leq C, \quad i = 1, \ldots, n$. Some binary classification problems can't be solved with a simple hyperplane. In this case, kernels (polynomial, radial, or sigmoid) are used. The resulting separating borders are schematically depicted in the figure below. Note different shapes for different kernels.



Example. We use the support vector machine binary classifier for the data "pneumonia_data.csv".

In SAS: In Enterprise Miner, we use the following path diagram that includes the HP SVM node. The only choices for kernels are polynomial (degree 2 or higher), radial, and sigmoid, which are specified by changing "Optimization Method" to "Active Set" and choosing a kernel under "Active Set Options".



We run the SAS code given below to compute prediction accuracy for the three models.

```
data polynomial_kernel;
set './polynomial_kernel.sas7bdat';
match=(pneumonia=lowcase(EM_CLASSIFICATION));
run;

proc sql;
select mean(match) as accuracy
from polynomial_kernel;
run;
```

accuracy
0.725434

```
data radial_kernel;
set './radial_kernel.sas7bdat';
match=(pneumonia=lowcase(EM_CLASSIFICATION));
run;

proc sql;
select mean(match) as accuracy
from radial_kernel;
run;
```

accuracy
0.725434

```
data sigmoid_kernel;
set './sigmoid_kernel.sas7bdat';
match=(pneumonia=lowcase(EM_CLASSIFICATION));
run;

proc sql;
select mean(match) as accuracy
from sigmoid_kernel;
run;
```

accuracy
0.66763

Models with polynomial (quadratic) and radial kernels have the largest prediction accuracy.

In R:

```
pneumonia.data<- read.csv(file="./pneumonia_data.csv", header=TRUE, sep=",")

pneumonia.data$pneumonia<- ifelse(pneumonia.data$pneumonia=="yes",1,0)
pneumonia.data$gender<- ifelse(pneumonia.data$gender=='M',1,0)
pneumonia.data$tobacco_use<- ifelse(pneumonia.data$tobacco_use=='yes',1,0)

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
set.seed(966452)
sample <- sample(c(TRUE, FALSE), nrow(pneumonia.data), replace=TRUE, prob=c(0.8,0.2))
train<- pneumonia.data[sample,]
test<- pneumonia.data[!sample,]

train.x<- data.matrix(train[-5])
train.y<- data.matrix(train[5])
test.x<- data.matrix(test[-5])
test.y<- data.matrix(test[5])
```

```
library(e1071)

#FITTING SVM WITH LINEAR KERNEL
svm.class<- svm(as.factor(pneumonia) ~ gender + age + tobacco_use + PM2_5, data=train, kernel="linear")

#computing prediction accuracy for testing data
pred.y<- as.numeric(predict(svm.class, test.x))-1

for (i in 1:length(pred.y))
    match[i]<- ifelse(test.y[i]==pred.y[i], 1,0)
print(paste("accuracy=", round(mean(match), digits=4)))


"accuracy= 0.7216"

#FITTING SVM WITH POLYNOMIAL KERNEL
svm.class<- svm(as.factor(pneumonia) ~ gender + age + tobacco_use + PM2_5, data=train, kernel="polynomial")

#computing prediction accuracy for testing data
pred.y<- as.numeric(predict(svm.class, test.x))-1

for (i in 1:length(pred.y))
    match[i]<- ifelse(test.y[i]==pred.y[i], 1,0)
print(paste("accuracy=", round(mean(match), digits=4)))


"accuracy= 0.7335"

#FITTING SVM WITH RADIAL KERNEL
svm.class<- svm(as.factor(pneumonia) ~ gender + age + tobacco_use + PM2_5, data=train, kernel="radial")

#computing prediction accuracy for testing data
pred.y<- as.numeric(predict(svm.class, test.x))-1

for (i in 1:length(pred.y))
    match[i]<- ifelse(test.y[i]==pred.y[i], 1,0)
print(paste("accuracy=", round(mean(match), digits=4)))


"accuracy= 0.7425"
```

```
#FITTING SVM WITH SIGMOID KERNEL
svm.class<- svm(as.factor(pneumonia) ~ gender + age + tobacco_use + PM2_5, data=train, kernel="sigmoid")

#computing prediction accuracy for testing data
pred.y<- as.numeric(predict(svm.class, test.x))-1

for (i in 1:length(pred.y))
    match[i]<- ifelse(test.y[i]==pred.y[i], 1,0)
print(paste("accuracy=", round(mean(match), digits=4)))
```

```
"accuracy= 0.6257"
```

The model with the radial kernel has the largest accuracy of prediction.

In Python:

```
import pandas
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from statistics import mean

pneumonia_data=pandas.read_csv('./pneumonia_data.csv')
code_gender={'M':1,'F':0}
code_tobacco_use={'yes':1,'no':0}
code_pneumonia={'yes':1,'no':0}

pneumonia_data['gender']=pneumonia_data['gender'].map(code_gender)
pneumonia_data['tobacco_use']=pneumonia_data['tobacco_use'].map(code_tobacco_use)
pneumonia_data['pneumonia']=pneumonia_data['pneumonia'].map(code_pneumonia)

X=pneumonia_data.iloc[:,0:4].values
y=pneumonia_data.iloc[:,4].values

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.20,
random_state=2346678)

#############################################################################################
#FITTING SUPPORT VECTOR BINARY CLASSIFIER WITH LINEAR KERNEL
svc_linear=SVC(kernel='linear').fit(X_train, y_train)

#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
y_pred=svc_linear.predict(X_test)

y_test=pandas.DataFrame(y_test,columns=['pneumonia'])
y_pred=pandas.DataFrame(y_pred,columns=['predicted'])
df=pandas.concat([y_test,y_pred],axis=1)

match=[]
for i in range(len(df)):
    if df['pneumonia'][i]==df['predicted'][i]:
        match.append(1)
    else:
        match.append(0)

print('Linear Kernel')
print('accuracy=', round(mean(match),4))
```

```
#########################################################################################
#FITTING SUPPORT VECTOR BINARY CLASSIFIER WITH POLYNOMIAL KERNEL
svc_poly=SVC(kernel='poly').fit(X_train, y_train)

#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
y_pred=svc_poly.predict(X_test)

y_test=pandas.DataFrame(y_test,columns=['pneumonia'])
y_pred=pandas.DataFrame(y_pred,columns=['predicted'])
df=pandas.concat([y_test,y_pred],axis=1)

match=[]
for i in range(len(df)):
    if df['pneumonia'][i]==df['predicted'][i]:
        match.append(1)
    else:
        match.append(0)

print('')
print('Polynomial Kernel')
print('accuracy=', round(mean(match),4))

#########################################################################################
#FITTING SUPPORT VECTOR BINARY CLASSIFIER WITH RADIAL KERNEL
svc_radial=SVC(kernel='rbf').fit(X_train, y_train)

#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
y_pred=svc_radial.predict(X_test)

y_test=pandas.DataFrame(y_test,columns=['pneumonia'])
y_pred=pandas.DataFrame(y_pred,columns=['predicted'])
df=pandas.concat([y_test,y_pred],axis=1)

match=[]
for i in range(len(df)):
    if df['pneumonia'][i]==df['predicted'][i]:
        match.append(1)
    else:
        match.append(0)

print('')
print('Radial Kernel')
print('accuracy=', round(mean(match),4))
```

```
#########################################################################################
#FITTING SUPPORT VECTOR BINARY CLASSIFIER WITH SIGMOID KERNEL
svc_sigmoid=SVC(kernel='sigmoid').fit(X_train, y_train)

#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
y_pred=svc_sigmoid.predict(X_test)

y_test=pandas.DataFrame(y_test,columns=['pneumonia'])
y_pred=pandas.DataFrame(y_pred,columns=['predicted'])
df=pandas.concat([y_test,y_pred],axis=1)

match=[]
for i in range(len(df)):
    if df['pneumonia'][i]==df['predicted'][i]:
        match.append(1)
    else:
        match.append(0)

print('')
print('Sigmoid Kernel')
print('accuracy=', round(mean(match),4))
```

```
Linear Kernel
accuracy= 0.6792

Polynomial Kernel
accuracy= 0.659

Radial Kernel
accuracy= 0.6416

Sigmoid Kernel
accuracy= 0.5491
```

Comparing the accuracies, we conclude that the model with linear kernel has the best fit. □

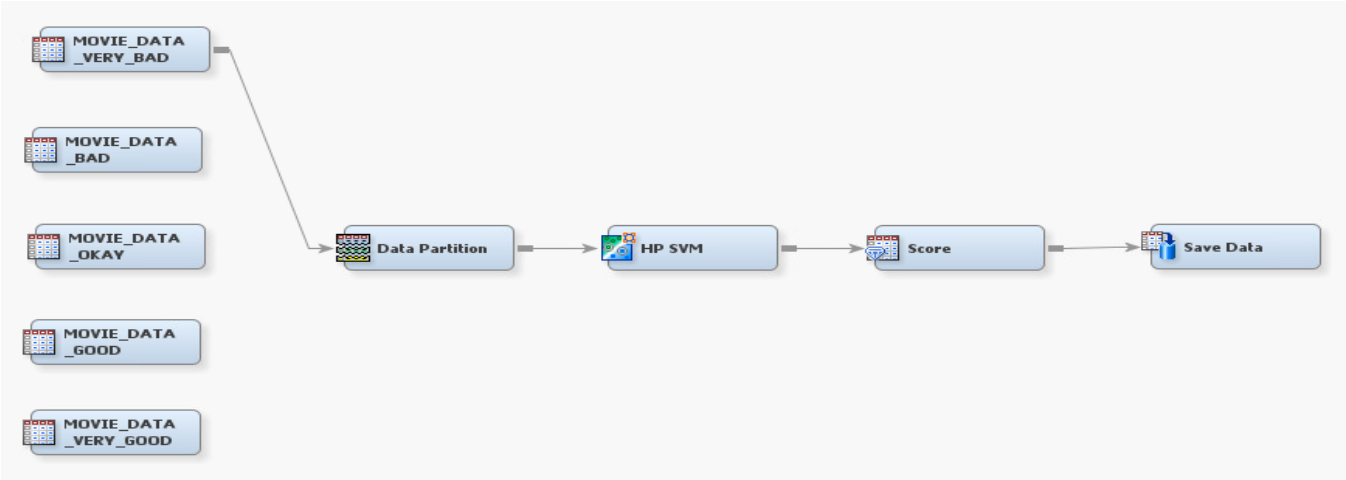
Support Vector Machine for Multinomial Classifier

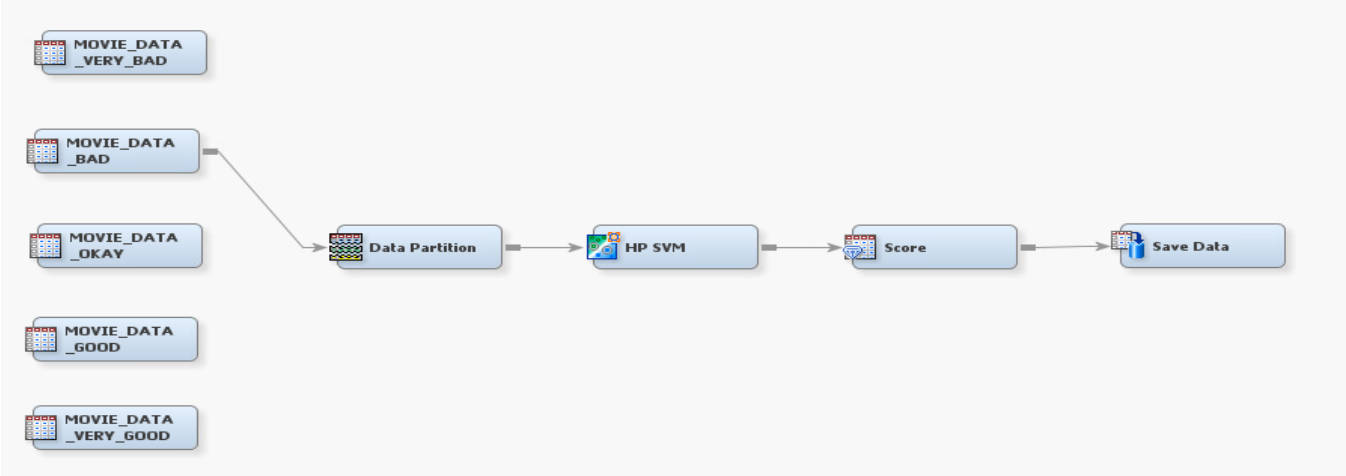
One-versus-all is the most practical approach in the case of multinomial classification problems. Suppose there are c classes. The approach dictates the creation of c indicator variables for the classes and running c separate support vector machines for binary classification. For each class,

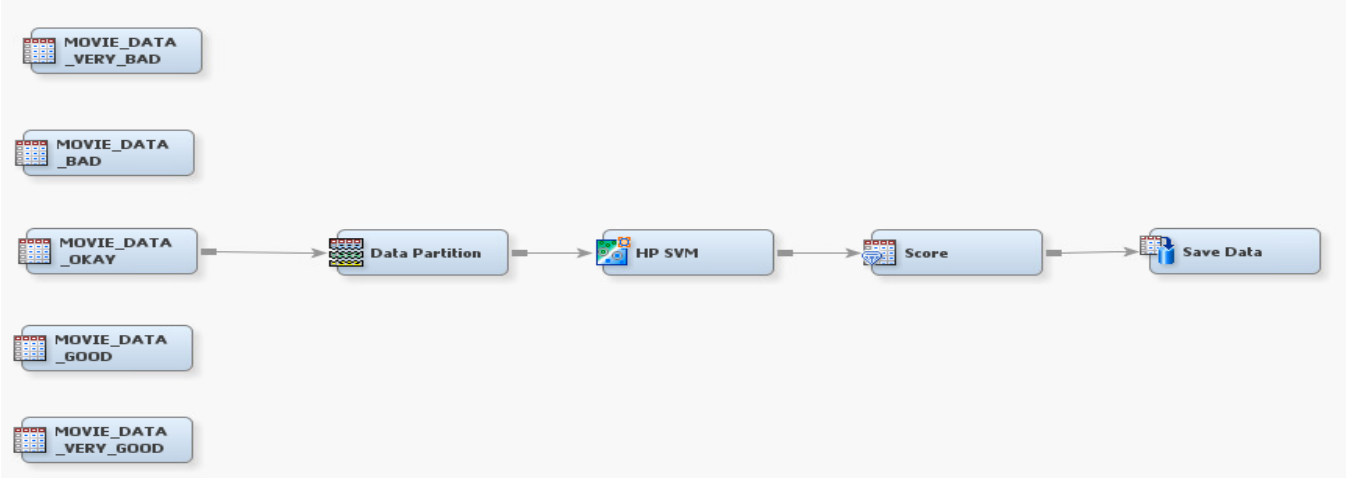
output the predicted probability and choose the class with the highest value as the predicted class.

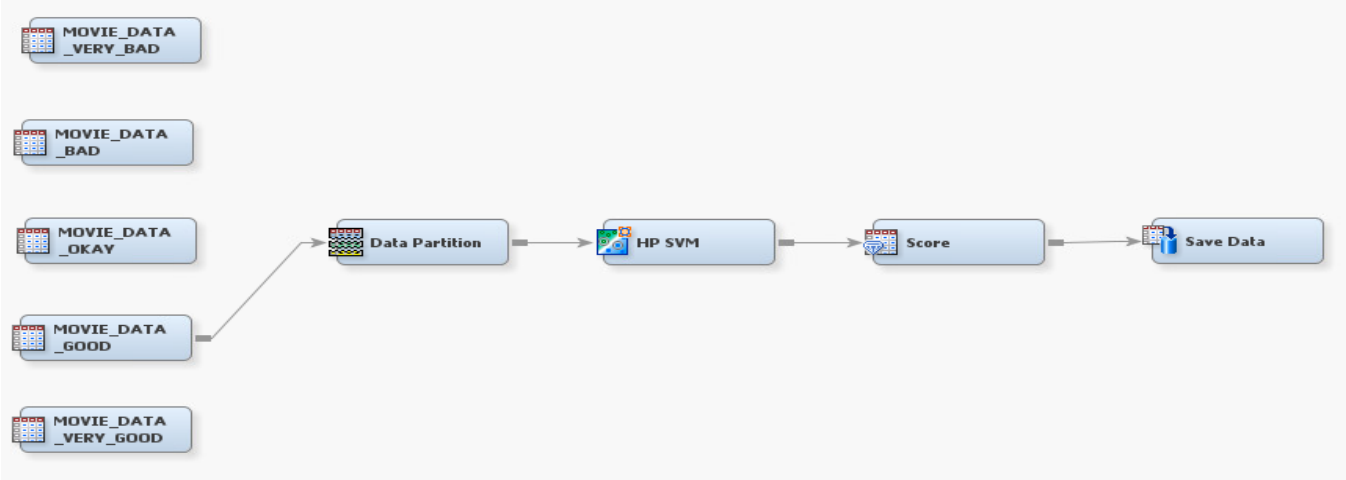
Example. Returning to the data in the file "movie_data.csv", we use SAS Enterprise Miner, R, and Python to fit a support vector multinomial classifier.

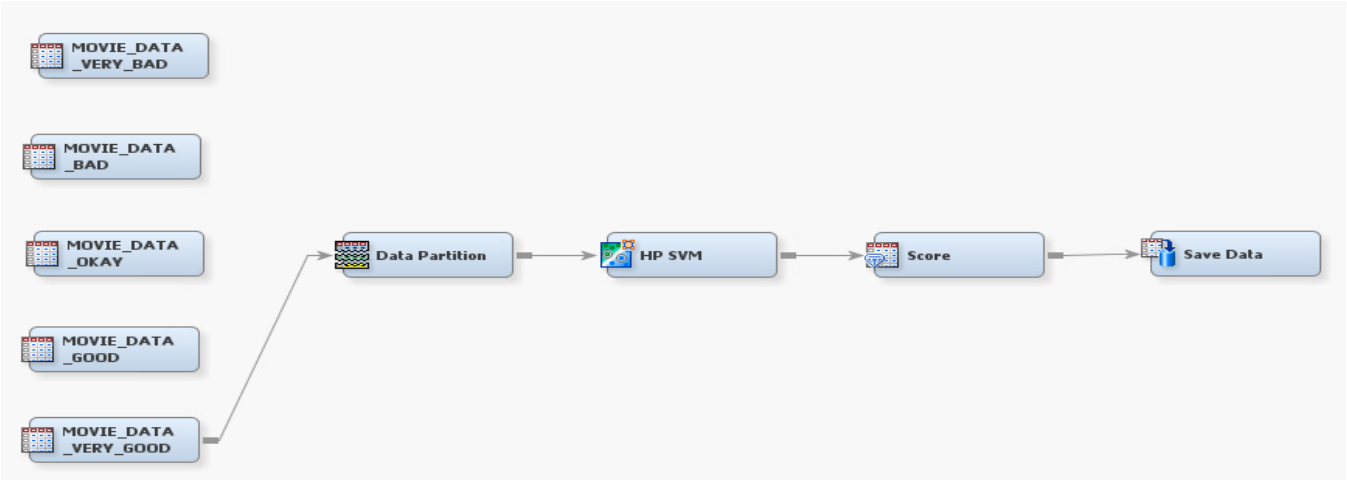
In SAS Enterprise Miner:
We first create indicator variables for the classes "very bad", "bad", "okay", "good", and "very good" and save the data into the file "movie_data_ind.csv", removing the rating variable. Then in Enterprise Miner, we run the given path with polynomial (quadratic), radial, and sigmoid kernels for each of the five classes separately. We specify each indicator variable as the binary target variable and rename the data set to reflect what class is being modeled. We run the five path diagrams depicted below.






MOVIE_DATA _VERY_BAD
MOVIE_DATA _BAD
MOVIE_DATA _OKAY
Data Partition
HP SVM
Score
Save Data
MOVIE_DATA _GOOD
MOVIE_DATA _VERY_GOOD


MOVIE_DATA _VERY_BAD
MOVIE_DATA _BAD
MOVIE_DATA _OKAY
Data Partition
HP SVM
Score
Save Data
MOVIE_DATA _GOOD
MOVIE_DATA _VERY_GOOD



Then we collected all the scored data by type of kernel and identified the class with the highest predicted probability. We then computed and outputted the prediction accuracy for each kernel. The code and output follow.

```
proc import out=sasuser.movies
datafile="./movie_data_ind.csv" dbms=csv replace;
run;

data movies;
set sasuser.movies;
_dataobs_=_N_;
run;

proc sort data=movies;
by _dataobs_;
run;


/*computing prediction accuracy for SVM with quadratic kernel*/
data quadratic_very_bad;
set './quadratic_verybad.sas7bdat';
predprob_very_bad=em_eventprobability;
keep _dataobs_ predprob_very_bad;
run;

proc sort;
by _dataobs_;
```

```
run;

data quadratic_bad;
set './quadratic_bad.sas7bdat';
predprob_bad=em_eventprobability;
keep _dataobs_ predprob_bad;
run;

proc sort;
by _dataobs_;
run;

data quadratic_okay;
set './quadratic_okay.sas7bdat';
predprob_okay=em_eventprobability;
keep _dataobs_ predprob_okay;
run;

proc sort;
by _dataobs_;
run;

data quadratic_good;
set './quadratic_good.sas7bdat';
predprob_good=em_eventprobability;
keep _dataobs_ predprob_good;
run;

proc sort;
by _dataobs_;
run;

data quadratic_very_good;
set './quadratic_verygood.sas7bdat';
predprob_very_good=em_eventprobability;
keep _dataobs_ predprob_very_good;
run;

proc sort;
by _dataobs_;
run;
```

```
data quadratic_kernel;
merge movies quadratic_very_bad quadratic_bad
quadratic_okay quadratic_good quadratic_very_good;
by _dataobs_;
if cmiss(predprob_very_bad, predprob_bad,
predprob_okay, predprob_good, predprob_very_good)=0;
run;

data quadratic_kernel;
set quadratic_kernel;
predprob_max=max(predprob_very_bad, predprob_bad,
predprob_okay, predprob_good, predprob_very_good);
if (predprob_very_good=predprob_max) then pred_class='very good';
if (predprob_very_bad=predprob_max) then pred_class='very bad';
if (predprob_bad=predprob_max) then pred_class='bad';
if (predprob_okay=predprob_max) then pred_class='okay';
if (predprob_good=predprob_max) then pred_class='good';
keep rating pred_class;
run;

data quadratic_kernel;
set quadratic_kernel;
match=(rating=pred_class);
run;

proc sql;
select mean(match) as accuracy
from quadratic_kernel;
quit;
```

accuracy
0.147887

```
/*************************************************************/
/*computing prediction accuracy for SVM with radial kernel*/
data radial_very_bad;
set './radial_verybad.sas7bdat';
predprob_very_bad=em_eventprobability;
```

```
keep _dataobs_ predprob_very_bad;
run;

proc sort;
by _dataobs_;
run;

data radial_bad;
set './radial_bad.sas7bdat';
predprob_bad=em_eventprobability;
keep _dataobs_ predprob_bad;
run;

proc sort;
by _dataobs_;
run;

data radial_okay;
set './radial_okay.sas7bdat';
predprob_okay=em_eventprobability;
keep _dataobs_ predprob_okay;
run;

proc sort;
by _dataobs_;
run;

data radial_good;
set './radial_good.sas7bdat';
predprob_good=em_eventprobability;
keep _dataobs_ predprob_good;
run;

proc sort;
by _dataobs_;
run;

data radial_very_good;
set './radial_verygood.sas7bdat';
predprob_very_good=em_eventprobability;
keep _dataobs_ predprob_very_good;
run;
```

```
proc sort;
by _dataobs_;
run;

data radial_kernel;
merge movies radial_very_bad radial_bad
radial_okay radial_good radial_very_good;
by _dataobs_;
if cmiss(predprob_very_bad, predprob_bad,
predprob_okay, predprob_good, predprob_very_good)=0;
run;

data radial_kernel;
set radial_kernel;
predprob_max=max(predprob_very_bad, predprob_bad,
predprob_okay, predprob_good, predprob_very_good);
if (predprob_very_good=predprob_max) then pred_class='very good';
if (predprob_very_bad=predprob_max) then pred_class='very bad';
if (predprob_bad=predprob_max) then pred_class='bad';
if (predprob_okay=predprob_max) then pred_class='okay';
if (predprob_good=predprob_max) then pred_class='good';
keep rating pred_class;
run;

data radial_kernel;
set radial_kernel;
match=(rating=pred_class);
run;

proc sql;
select mean(match) as accuracy
from radial_kernel;
quit;
```

accuracy
0.309859

```
/*********************************************************/
/*computing prediction accuracy for SVM with sigmoid kernel*/
data sigmoid_very_bad;
set './sigmoid_verybad.sas7bdat';
predprob_very_bad=em_eventprobability;
keep _dataobs_ predprob_very_bad;
run;

proc sort;
by _dataobs_;
run;

data sigmoid_bad;
set './sigmoid_bad.sas7bdat';
predprob_bad=em_eventprobability;
keep _dataobs_ predprob_bad;
run;

proc sort;
by _dataobs_;
run;

data sigmoid_okay;
set './sigmoid_okay.sas7bdat';
predprob_okay=em_eventprobability;
*keep _dataobs_ predprob_okay;
run;

proc sort;
by _dataobs_;
run;

data sigmoid_good;
set './sigmoid_good.sas7bdat';
predprob_good=em_eventprobability;
keep _dataobs_ predprob_good;
run;

proc sort;
by _dataobs_;
run;
```

```
data sigmoid_very_good;
set './sigmoid_verygood.sas7bdat';
predprob_very_good=em_eventprobability;
keep _dataobs_ predprob_very_good;
run;

proc sort;
by _dataobs_;
run;

data sigmoid_kernel;
merge movies sigmoid_very_bad sigmoid_bad
sigmoid_okay sigmoid_good sigmoid_very_good;
by _dataobs_;
if cmiss(predprob_very_bad, predprob_bad,
predprob_okay, predprob_good, predprob_very_good)=0;
run;

data sigmoid_kernel;
set sigmoid_kernel;
predprob_max=max(predprob_very_bad, predprob_bad,
predprob_okay, predprob_good, predprob_very_good);
if (predprob_very_good=predprob_max) then pred_class='very good';
if (predprob_very_bad=predprob_max) then pred_class='very bad';
if (predprob_bad=predprob_max) then pred_class='bad';
if (predprob_okay=predprob_max) then pred_class='okay';
if (predprob_good=predprob_max) then pred_class='good';
keep rating pred_class;
run;

data sigmoid_kernel;
set sigmoid_kernel;
match=(rating=pred_class);
run;

proc sql;
select mean(match) as accuracy
from sigmoid_kernel;
quit;
```

accuracy
0.260563

We see that the largest accuracy is for the model with the radial kernel.

In R:

```
movie.data<- read.csv(file="./movie_data.csv", header=TRUE, sep=",")

movie.data$gender<- ifelse(movie.data$gender=='M',1,0)
movie.data$member<- ifelse(movie.data$member=='yes',1,0)
#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
set.seed(444625)
sample <- sample(c(TRUE, FALSE), nrow(movie.data), replace=TRUE, prob=c(0.8,0.2))
train<- movie.data[sample,]
test<- movie.data[!sample,]

train.x<- data.matrix(train[-5])
train.y<- data.matrix(train[5])
test.x<- data.matrix(test[-5])
test.y<- data.matrix(test[5])

library(e1071)

#FITTING SVM WITH LINEAR KERNEL
svm.multiclass<- svm(as.factor(rating) ~ age + gender + member + nmovies,
data=train, kernel="linear")

#computing prediction accuracy for testing data
pred.y<- as.numeric(predict(svm.multiclass, test.x))

print(paste("accuracy=", round(1-mean(test.y!=pred.y),digits=4)))
```

```
"accuracy= 0.2892"
```

```
#FITTING SVM WITH POLYNOMIAL KERNEL
svm.multiclass<- svm(as.factor(rating) ~ age + gender + member + nmovies,
```

```
data=train, kernel="polynomial")

#computing prediction accuracy for testing data
pred.y<- as.numeric(predict(svm.multiclass, test.x))

print(paste("accuracy=", round(1-mean(test.y!=pred.y),digits=4)))
```

```
"accuracy= 0.3133"
```

```
#FITTING SVM WITH RADIAL KERNEL
svm.multiclass<- svm(as.factor(rating) ~ age + gender + member + nmovies,
data=train, kernel="radial")

#computing prediction accuracy for testing data
pred.y<- as.numeric(predict(svm.multiclass, test.x))

print(paste("accuracy=", round(1-mean(test.y!=pred.y),digits=4)))
```

```
"accuracy= 0.3133"
```

```
#FITTING SVM WITH SIGMOID KERNEL
svm.multiclass<- svm(as.factor(rating) ~ age + gender + member + nmovies,
data=train, kernel="sigmoid")

#computing prediction accuracy for testing data
pred.y<- as.numeric(predict(svm.multiclass, test.x))

print(paste("accuracy=", round(1-mean(test.y!=pred.y),digits=4)))
```

```
"accuracy= 0.2651"
```

The models with polynomial and radial kernels have the best fit.

In Python:

```
import pandas
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from statistics import mean

movie_data=pandas.read_csv('./movie_data.csv')
code_gender={'M':1,'F':0}
code_member={'yes':1,'no':0}
code_rating={'very bad':1,'bad':2,'okay':3,'good':4,'very good':5}

movie_data['gender']=movie_data['gender'].map(code_gender)
movie_data['member']=movie_data['member'].map(code_member)
movie_data['rating']=movie_data['rating'].map(code_rating)

X=movie_data.iloc[:,0:4].values
y=movie_data.iloc[:,4].values

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.20,
random_state=457752)

#############################################################################################
#FITTING SUPPORT VECTOR MULTINOMIAL CLASSIFIER WITH LINEAR KERNEL
svmc_linear=SVC(kernel='linear').fit(X_train, y_train)

#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
y_pred=svmc_linear.predict(X_test)
y_test=pandas.DataFrame(y_test,columns=['rating'])
y_pred=pandas.DataFrame(y_pred,columns=['predicted'])
df=pandas.concat([y_test,y_pred],axis=1)

match=[]
for i in range(len(df)):
    if df['rating'][i]==df['predicted'][i]:
        match.append(1)
    else:
        match.append(0)

print('Linear Kernel')
print('accuracy=', round(mean(match),4))
```

```
#############################################################################################
#FITTING SUPPORT VECTOR MULTINOMIAL CLASSIFIER WITH POLYNOMIAL KERNEL
svmc_poly=SVC(kernel='poly').fit(X_train, y_train)

#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
y_pred=svmc_poly.predict(X_test)
y_test=pandas.DataFrame(y_test,columns=['rating'])
y_pred=pandas.DataFrame(y_pred,columns=['predicted'])
df=pandas.concat([y_test,y_pred],axis=1)

match=[]
for i in range(len(df)):
    if df['rating'][i]==df['predicted'][i]:
        match.append(1)
    else:
        match.append(0)

print('')
print('Polynomial Kernel')
print('accuracy=', round(mean(match),4))

#############################################################################################
#FITTING SUPPORT VECTOR MULTINOMIAL CLASSIFIER WITH RADIAL KERNEL
svmc_radial=SVC(kernel='rbf').fit(X_train, y_train)

#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
y_pred=svmc_radial.predict(X_test)
y_test=pandas.DataFrame(y_test,columns=['rating'])
y_pred=pandas.DataFrame(y_pred,columns=['predicted'])
df=pandas.concat([y_test,y_pred],axis=1)

match=[]
for i in range(len(df)):
    if df['rating'][i]==df['predicted'][i]:
        match.append(1)
    else:
        match.append(0)

print('')
print('Radial Kernel')
print('accuracy=', round(mean(match),4))
```

```
#########################################################################################
#FITTING SUPPORT VECTOR MULTINOMIAL CLASSIFIER WITH SIGMOID KERNEL
svmc_sigmoid=SVC(kernel='sigmoid').fit(X_train, y_train)

#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
y_pred=svmc_sigmoid.predict(X_test)
y_test=pandas.DataFrame(y_test,columns=['rating'])
y_pred=pandas.DataFrame(y_pred,columns=['predicted'])
df=pandas.concat([y_test,y_pred],axis=1)

match=[]
for i in range(len(df)):
    if df['rating'][i]==df['predicted'][i]:
        match.append(1)
    else:
        match.append(0)

print('')
print('Sigmoid Kernel')
print('accuracy=', round(mean(match),4))
```

```
Linear Kernel
accuracy= 0.3421

Polynomial Kernel
accuracy= 0.3224

Radial Kernel
accuracy= 0.3553

Sigmoid Kernel
accuracy= 0.3289
```

The model with radial kernel has the best fit. □.

NAIVE BAYES CLASSIFICATION

Naive Bayes Classification is a method used for binary or multinomial classification (but not a regression) that utilizes the Bayes' formula. Suppose there are k predictors $\mathbf{X} = (X_1, \ldots, X_k)$

which are binary, categorical, or continuous. And let Y denote the response variable. By the Bayes' formula

$$\mathbb{P}(Y|\mathbf{X}) = \frac{\mathbb{P}(\mathbf{X}|Y)\mathbb{P}(Y)}{\mathbb{P}(\mathbf{X})}.$$

The Naive Bayes classification method assumes that the predictors are conditionally independent, given Y, that is,

$$\mathbb{P}(Y|\mathbf{X}) = \frac{\mathbb{P}(Y)\prod_{i=1}^{k}\mathbb{P}(X_i|Y)}{\mathbb{P}(\mathbf{X})}.$$

This conditional independence assumption is rather naive, hence the name of the technique.

In classification problem (binary or multinomial), we compute the conditional (posterior) probability $\mathbb{P}(Y|\mathbf{X})$ of each class, and classify the record into the class with the highest probability. Since we compare the posterior probabilities and the denominator $\mathbb{P}(\mathbf{X})$ is present in each expression, it can be ignored. That is, $\mathbb{P}(Y|\mathbf{X})$ is proportional to $\mathbb{P}(Y)\prod_{i=1}^{k}\mathbb{P}(X_i|Y)$ up to a multiplicative constant.

To estimate the prior probability $\mathbb{P}(Y = y)$ of each class y, we compute the proportion of observations in each class in the training set. To compute the empirical conditional probabilities $\mathbb{P}(X_i = x|Y = y)$ for categorical predictors, we calculate the fraction of observations in the class $Y = y$ in the training set for which $X_i = x$. If a predictor is continuous, we assume that the underlying distribution is normal (Gaussian) with estimated mean $\hat{\mu} = \bar{x}$ and estimated variance $\hat{\sigma}^2 = s^2$.

Characteristics of Naive Bayes Classifiers

1. Robust to outliers because they average out when computing posterior probabilities.
2. Handles missing values by ignoring the missing data points in calculations.
3. Robust to irrelevant predictors since $\mathbb{P}(X_i|Y)$ is almost uniformly distributed and factors out in comparisons of posterior probabilities.
4. Correlated predictors can degrade the performance of the technique. The conditional independence assumption is the key.

Example. Suppose the training data are as given in the table below.

ID	Home Owner	Marital Status	Annual Income ($K)	Defaulted Borrower
1	yes	single	125	no
2	no	married	100	no
3	no	single	70	no
4	yes	married	120	no
5	no	divorced	95	yes
6	no	married	60	no
7	yes	divorced	220	no
8	no	single	85	yes
9	no	married	75	no
10	no	single	90	yes

The prior probabilities are $\mathbb{P}(default = no) = 7/10 = 0.7$, $\mathbb{P}(defaul = yes) = 3/10 = 0.3$. The conditional probabilities are:

$$\mathbb{P}(homeowner = yes \,|\, default = no) = 3/7,$$
$$\mathbb{P}(homeowner = yes \,|\, default = yes) = 0,$$
$$\mathbb{P}(homeowner = no \,|\, default = no) = 4/7,$$
$$\mathbb{P}(homeowner = no \,|\, default = yes) = 1,$$
$$\mathbb{P}(maritalstatus = single \,|\, default = no) = 2/7,$$
$$\mathbb{P}(maritalstatus = single \,|, default = yes) = 2/3,$$
$$\mathbb{P}(maritalstatus = married \,|\, default = no) = 4/7,$$
$$\mathbb{P}(maritalstatus = married \,|\, default = yes) = 0,$$
$$\mathbb{P}(maritalstatus = divorced \,|\, default = no) = 1/7,$$
$$\mathbb{P}(maritalstatus = divorced \,|\, default = yes) = 1/3.$$

The posterior density for annual income is normal with the estimated parameters for *default=no*, $\hat{\mu}$ =sample mean= $(125 + 100 + 70 + 120 + 60 + 220 + 75)/7 = 110$, $\hat{\sigma}^2 = s^2 = 2975$, and for *default=yes*, $\hat{\mu} = (95 + 85 + 90)/3 = 90$, and $\hat{\sigma}^2 = s^2 = 25$.

Suppose we would like to predict the default status for a person who is not a home owner, who is single, and whose annual income is $\$120K$. We write

$$\mathbb{P}(\mathbf{X} \,|\, default = no) = \mathbb{P}(homeowner = no \,|\, default = no) \times \mathbb{P}(maritalstatus = single \,|\, default = no) \times$$
$$\mathbb{P}(annualincome = \$120K \,|\, defaul = no) = (4/7)(2/7)\frac{1}{\sqrt{(2\pi)(2975)}}\, e^{-\frac{(120-110)^2}{(2)(2975)}} = 0.001215,$$

and

$$\mathbb{P}(\mathbf{X} \,|\, default = yes) = \mathbb{P}(homeowner = no \,|\, default = no) \times \mathbb{P}(maritalstatus = single \,|\, default = no) \times$$

$$\mathbb{P}(annualincome = \$120K \,|\, defaul = no) = (1)(2/3)\frac{1}{\sqrt{(2\pi)(25)}}\, e^{-\frac{(120-90)^2}{(2)(25)}} = (8.1)(10)^{-10}.$$

Hence,

$$\begin{aligned}\mathbb{P}(default = no \,|\, \mathbf{X}) &= \mathbb{P}(default = no)\mathbb{P}(\mathbf{X} \,|\, default = no)/\mathbb{P}(\mathbf{X}) \\ &= (0.7)(0.001215)/\mathbb{P}(\mathbf{X}) = 0.000851/\mathbb{P}(\mathbf{X}),\end{aligned}$$

and

$$\begin{aligned}\mathbb{P}(default = yes \,|\, \mathbf{X}) &= \mathbb{P}(default = yes)\mathbb{P}(\mathbf{X} \,|\, default = yes)/\mathbb{P}(\mathbf{X}) \\ &= (0.3)(8.1)(10)^{-10}/\mathbb{P}(\mathbf{X}) = (2.43)(10)^{-10}/\mathbb{P}(\mathbf{X}).\end{aligned}$$

We can see that $\mathbb{P}(default = no \,|\, \mathbf{X}) > \mathbb{P}(default = yes \,|\, \mathbf{X})$ and so we predict *default=no* for this person. □

Example. We use the naive Bayes binary classifier for the data "pneumonia_data.csv".

In SAS:

```
proc import out=pneumonia datafile="./pneumonia_data.csv" dbms=csv replace;

/*SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS*/
proc surveyselect data=pneumonia rate=0.8 seed=999454
out=pneumonia outall method=srs;
run;

data train (drop=selected);
set pneumonia;
if selected=1;
run;

data test (drop=selected);
set pneumonia;
if selected=0;
run;
```

```
/*COMPUTING PRIOR PROBABILITIES*/
proc freq data=train noprint;
 table pneumonia/out=priors;
run;

data priors;
set priors;
 percent=percent/100;
 if pneumonia='no' then call symput('prior_no', percent);
 if pneumonia='yes' then call symput('prior_yes', percent);
run;

/*COMPUTING POSTERIOR PROBABILITIES FOR CATEGORICAL PREDICTORS*/
proc freq data=train noprint;
 table pneumonia*gender/out=gender_perc nocum list;
run;

data gender_perc;
set gender_perc;
 percent=percent/100;
if pneumonia='no' and gender='F' then call symput('female_no', percent);
if pneumonia='no' and gender='M' then call symput('male_no', percent);
if pneumonia='yes' and gender='F' then call symput('female_yes', percent);
if pneumonia='yes' and gender='M' then call symput('male_yes', percent);
 run;

proc freq data=train noprint;
 table pneumonia*tobacco_use/out=tobacco_use_perc nocum list;
run;

data tobacco_use_perc;
set tobacco_use_perc;
 percent=percent/100;
if pneumonia='no' and tobacco_use='no' then call symput('tobacco_no_no', percent);
if pneumonia='no' and tobacco_use='yes' then call symput('tobacco_yes_no', percent);
if pneumonia='yes' and tobacco_use='no' then call symput('tobacco_no_yes', percent);
if pneumonia='yes' and tobacco_use='yes' then call symput('tobacco_yes_yes', percent);
 run;

/*COMPUTING MEAN AND STANDARD DEVIATION FOR NUMERICAL PREDICTORS*/
proc means data=train mean std noprint;
 class pneumonia;
```

```
  var age PM2_5;
output out=stats;
run;

data stats;
 set stats;
if pneumonia='no' and _stat_='MEAN' then
  do;
   call symput('age_mean_no',age);
   call symput('PM2_5_mean_no',PM2_5);
  end;
if pneumonia='no' and _stat_='STD' then
  do;
   call symput('age_std_no',age);
   call symput('PM2_5_std_no',PM2_5);
  end;
if pneumonia='yes' and _stat_='MEAN' then
  do;
   call symput('age_mean_yes',age);
   call symput('PM2_5_mean_yes',PM2_5);
  end;
if pneumonia='yes' and _stat_='STD' then
  do;
   call symput('age_std_yes',age);
   call symput('PM2_5_std_yes',PM2_5);
  end;
run;

/*COMPUTING POSTERIOR PROBABILITIES FOR TESTING DATA*/
data test;
set test;
if (gender='F' and tobacco_use='no') then
do;
pred_prob_no=&prior_no*&female_no*&tobacco_no_no*1/(2*3.14)*1/(&age_std_no*&PM2_5_std_no)
*exp(-(age-&age_mean_no)**2/(2*&age_std_no**2)-(PM2_5-&PM2_5_mean_no)**2/(2*&PM2_5_std_no**2))
pred_prob_yes=&prior_yes*&female_yes*&tobacco_no_yes*1/(2*3.14)*1/(&age_std_yes*&PM2_5_std_yes
*exp(-(age-&age_mean_yes)**2/(2*&age_std_yes**2)-(PM2_5-&PM2_5_mean_yes)**2/
(2*&PM2_5_std_yes**2));
end;
if (gender='M' and tobacco_use='no') then
do;
pred_prob_no=&prior_no*&male_no*&tobacco_no_no*1/(2*3.14)*1/(&age_std_no*&PM2_5_std_no)
```

```
*exp(-(age-&age_mean_no)**2/(2*&age_std_no**2)-(PM2_5-&PM2_5_mean_no)**2/(2*&PM2_5_std_no**2))
pred_prob_yes=&prior_yes*&male_yes*&tobacco_no_yes*1/(2*3.14)*1/(&age_std_yes*&PM2_5_std_yes)
*exp(-(age-&age_mean_yes)**2/(2*&age_std_yes**2)-(PM2_5-&PM2_5_mean_yes)**2/
(2*&PM2_5_std_yes**2));
end;
if (gender='F' and tobacco_use='yes') then
do;
pred_prob_no=&prior_no*&female_no*&tobacco_yes_no*1/(2*3.14)*1/(&age_std_no*&PM2_5_std_no)
*exp(-(age-&age_mean_no)**2/(2*&age_std_no**2)-(PM2_5-&PM2_5_mean_no)**2/(2*&PM2_5_std_no**2))
pred_prob_yes=&prior_yes*&female_yes*&tobacco_yes_yes*1/(2*3.14)*1/(&age_std_yes*&PM2_5_std_y
*exp(-(age-&age_mean_yes)**2/(2*&age_std_yes**2)-(PM2_5-&PM2_5_mean_yes)**2/
(2*&PM2_5_std_yes**2));
end;
if (gender='M' and tobacco_use='yes') then
do;
pred_prob_no=&prior_no*&male_no*&tobacco_yes_no*1/(2*3.14)*1/(&age_std_no*&PM2_5_std_no)
*exp(-(age-&age_mean_no)**2/(2*&age_std_no**2)-(PM2_5-&PM2_5_mean_no)**2/(2*&PM2_5_std_no**2))
pred_prob_yes=&prior_yes*&male_yes*&tobacco_yes_yes*1/(2*3.14)*1/(&age_std_yes*&PM2_5_std_yes
*exp(-(age-&age_mean_yes)**2/(2*&age_std_yes**2)-(PM2_5-&PM2_5_mean_yes)**2/
(2*&PM2_5_std_yes**2));
end;
run;

/*COMPUTING PREDICTION ACCURACY*/
data test;
 set test;
  if pred_prob_no < pred_prob_yes then pred_class='yes';
  else pred_class='no';
  if pneumonia=pred_class then pred=1; else pred=0;
 run;

 proc sql;
  select mean(pred) as accuracy
   from test;
quit;
```

accuracy
0.733333

In R: All the values for predictors have to be numeric, so we need to replace all string values with numeric values before running the technique.

```
pneumonia.data<- read.csv(file="./pneumonia_data.csv", header=TRUE, sep=",")

pneumonia.data$pneumonia<- ifelse(pneumonia.data$pneumonia=="yes",1,0)
pneumonia.data$gender<- ifelse(pneumonia.data$gender=='M',1,0)
pneumonia.data$tobacco_use<- ifelse(pneumonia.data$tobacco_use=='yes',1,0)

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
set.seed(1012312)
sample <- sample(c(TRUE, FALSE), nrow(pneumonia.data), replace=TRUE, prob=c(0.8,0.2))
train<- pneumonia.data[sample,]
test<- pneumonia.data[!sample,]

test.x<- data.matrix(test[-5])
test.y<- data.matrix(test[5])

#FITTING NAIVE BAYES BINARY CLASSIFIER
library(e1071)
nb.class<- naiveBayes(as.factor(pneumonia) ~ gender + age + tobacco_use + PM2_5, data=train)

#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
pred.y<- as.numeric(predict(nb.class, test.x))-1

match<- c()
for (i in 1:length(pred.y))
    match[i]<- ifelse(test.y[i]==pred.y[i], 1, 0)
print(paste('accuracy=', round(mean(match)*100, digits=2),'%'))
```

```
"accuracy= 74.28 %"
```

In Python:

```
import pandas
from sklearn.model_selection import train_test_split
from sklearn import metrics

pneumonia_data=pandas.read_csv('./pneumonia_data.csv')
code_gender={'M':1,'F':0}
code_tobacco_use={'yes':1,'no':0}
code_pneumonia={'yes':1,'no':0}

pneumonia_data['gender']=pneumonia_data['gender'].map(code_gender)
pneumonia_data['tobacco_use']=pneumonia_data['tobacco_use'].map(code_tobacco_use)
pneumonia_data['pneumonia']=pneumonia_data['pneumonia'].map(code_pneumonia)

X=pneumonia_data.iloc[:,0:4].values
y=pneumonia_data.iloc[:,4].values

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.20,
random_state=9994445)

#FITTING NAIVE BAYES BINARY CLASSIFIER
from sklearn.naive_bayes import GaussianNB
gauss_nb=GaussianNB()
gauss_nb.fit(X_train, y_train)

#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
y_pred = gauss_nb.predict(X_test)
print('Accuracy:', round(metrics.accuracy_score(y_test, y_pred)*100, 2),'%')
```

```
Accuracy: 71.68 %
```

□

Example. For the data "movie_data.csv" we fit a naive Bayes multinomial classifier.

In SAS:

```
proc import out=movie datafile="./movie_data.csv" dbms=csv replace;

/*SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS*/
```

```
proc surveyselect data=movie rate=0.8 seed=121800
out=movie outall method=srs;
run;

data train (drop=selected);
set movie;
if selected=1;
run;

data test (drop=selected);
set movie;
if selected=0;
run;

/*COMPUTING PRIOR PROBABILITIES*/
proc freq data=train noprint;
 table rating/out=priors;
run;

data priors;
set priors;
 percent=percent/100;
 if rating='very bad' then call symput('prior_very_bad', percent);
 if rating='bad' then call symput('prior_bad', percent);
 if rating='okay' then call symput('prior_okay', percent);
 if rating='good' then call symput('prior_good', percent);
 if rating='very good' then call symput('prior_very_good', percent);
run;

/*COMPUTING POSTERIOR PROBABILITIES FOR CATEGORICAL PREDICTORS*/
proc freq data=train noprint;
 table rating*gender/out=gender_perc
   nocum list;
run;

data gender_perc;
set gender_perc;
 percent=percent/100;
if rating='very bad' and gender='F' then call symput('female_very_bad', percent);
if rating='very bad' and gender='M' then call symput('male_very_bad', percent);
if rating='bad' and gender='F' then call symput('female_bad', percent);
if rating='bad' and gender='M' then call symput('male_bad', percent);
```

```
if rating='okay' and gender='F' then call symput('female_okay', percent);
if rating='okay' and gender='M' then call symput('male_okay', percent);
if rating='good' and gender='F' then call symput('female_good', percent);
if rating='good' and gender='M' then call symput('male_good', percent);
if rating='very good' and gender='F' then call symput('female_very_good', percent);
if rating='very good' and gender='M' then call symput('male_very_good', percent);
run;

proc freq data=train noprint;
 table rating*member/out=member_perc nocum list;
run;

data member_perc;
set member_perc;
 percent=percent/100;
if rating='very bad' and member='no' then call symput('member_no_very_bad', percent);
if rating='very bad' and member='yes' then call symput('member_yes_very_bad', percent);
if rating='bad' and member='no' then call symput('member_no_bad', percent);
if rating='bad' and member='yes' then call symput('member_yes_bad', percent);
if rating='okay' and member='no' then call symput('member_no_okay', percent);
if rating='okay' and member='yes' then call symput('member_yes_okay', percent);
if rating='good' and member='no' then call symput('member_no_good', percent);
if rating='good' and member='yes' then call symput('member_yes_good', percent);
if rating='very good' and member='no' then call symput('member_no_very_good', percent);
if rating='very good' and member='yes' then call symput('member_yes_very_good', percent);
 run;

/*COMPUTING MEAN AND STANDARD DEVIATION FOR NUMERICAL PREDICTORS*/
proc means data=train mean std noprint;
 class rating;
  var age nmovies;
output out=stats;
run;

data stats;
 set stats;
if rating='very bad' and _stat_='MEAN' then
  do;
   call symput('age_mean_very_bad',age);
   call symput('nmovies_mean_very_bad',nmovies);
  end;
if rating='very bad' and _stat_='STD' then
```

```
  do;
   call symput('age_std_very_bad',age);
   call symput('nmovies_std_very_bad',nmovies);
  end;
if rating='bad' and _stat_='MEAN' then
  do;
   call symput('age_mean_bad',age);
   call symput('nmovies_mean_bad',nmovies);
  end;
if rating='bad' and _stat_='STD' then
  do;
   call symput('age_std_bad',age);
   call symput('nmovies_std_bad',nmovies);
  end;
if rating='okay' and _stat_='MEAN' then
  do;
   call symput('age_mean_okay',age);
   call symput('nmovies_mean_okay',nmovies);
  end;
if rating='okay' and _stat_='STD' then
  do;
   call symput('age_std_okay',age);
   call symput('nmovies_std_okay',nmovies);
  end;
if rating='good' and _stat_='MEAN' then
  do;
   call symput('age_mean_good',age);
   call symput('nmovies_mean_good',nmovies);
  end;
if rating='good' and _stat_='STD' then
  do;
   call symput('age_std_good',age);
   call symput('nmovies_std_good',nmovies);
  end;
if rating='very good' and _stat_='MEAN' then
  do;
   call symput('age_mean_very_good',age);
   call symput('nmovies_mean_very_good',nmovies);
  end;
if rating='very good' and _stat_='STD' then
  do;
   call symput('age_std_very_good',age);
```

```
   call symput('nmovies_std_very_good',nmovies);
  end;
run;

/*COMPUTING POSTERIOR PROBABILITIES FOR TESTING DATA*/
data test;
set test;
if (gender='F' and member='no') then
do;
pred_prob_very_bad=&prior_very_bad*&female_very_bad*&member_no_very_bad*1/(2*3.14)*1/(&age_std
&nmovies_std_very_bad)*exp(-(age-&age_mean_very_bad)**2/(2*&age_std_very_bad**2)
-(nmovies-&nmovies_mean_very_bad)**2/(2*&nmovies_std_very_bad**2));
pred_prob_bad=&prior_bad*&female_bad*&member_no_bad*1/(2*3.14)*1/(&age_std_bad*&nmovies_std_ba
*exp(-(age-&age_mean_bad)**2/(2*&age_std_bad**2)-(nmovies-&nmovies_mean_bad)**2/(2*&nmovies_s
pred_prob_okay=&prior_okay*&female_okay*&member_no_okay*1/(2*3.14)*1/(&age_std_okay*&nmovies_s
*exp(-(age-&age_mean_okay)**2/(2*&age_std_okay**2)-(nmovies-&nmovies_mean_okay)**2/(2*&nmovies
pred_prob_good=&prior_good*&female_good*&member_no_good*1/(2*3.14)*1/(&age_std_good*&nmovies_s
*exp(-(age-&age_mean_good)**2/(2*&age_std_good**2)-(nmovies-&nmovies_mean_good)**2/(2*&nmovies
pred_prob_very_good=&prior_very_good*&female_very_good*&member_no_very_good*1/(2*3.14)*1/(&age
*&nmovies_std_very_good)*exp(-(age-&age_mean_very_good)**2/(2*&age_std_very_good**2)
-(nmovies-&nmovies_mean_very_good)**2/(2*&nmovies_std_very_good**2));
end;

if (gender='M' and member='no') then
do;
pred_prob_very_bad=&prior_very_bad*&male_very_bad*&member_no_very_bad*1/(2*3.14)*1/(&age_std_v
*&nmovies_std_very_bad)*exp(-(age-&age_mean_very_bad)**2/(2*&age_std_very_bad**2)-(nmovies-&nm
**2/(2*&nmovies_std_very_bad**2));
pred_prob_bad=&prior_bad*&male_bad*&member_no_bad*1/(2*3.14)*1/(&age_std_bad*&nmovies_std_bad)
*exp(-(age-&age_mean_bad)**2/(2*&age_std_bad**2)-(nmovies-&nmovies_mean_bad)**2/(2*&nmovies_s
pred_prob_okay=&prior_okay*&male_okay*&member_no_okay*1/(2*3.14)*1/(&age_std_okay*&nmovies_std
*exp(-(age-&age_mean_okay)**2/(2*&age_std_okay**2)-(nmovies-&nmovies_mean_okay)**2/(2*&nmovies
pred_prob_good=&prior_good*&male_good*&member_no_good*1/(2*3.14)*1/(&age_std_good*&nmovies_std
*exp(-(age-&age_mean_good)**2/(2*&age_std_good**2)-(nmovies-&nmovies_mean_good)**2/(2*&nmovies
pred_prob_very_good=&prior_very_good*&male_very_good*&member_no_very_good*1/(2*3.14)*1/(&age_s
*&nmovies_std_very_good)*exp(-(age-&age_mean_very_good)**2/(2*&age_std_very_good**2)-(nmovies
-&nmovies_mean_very_good)**2/(2*&nmovies_std_very_good**2));
end;

if (gender='F' and member='yes') then
do;
pred_prob_very_bad=&prior_very_bad*&female_very_bad*&member_yes_very_bad*1/(2*3.14)*1/(&age_s
```

```
*&nmovies_std_very_bad)*exp(-(age-&age_mean_very_bad)**2/(2*&age_std_very_bad**2)-(nmovies-&nm
**2/(2*&nmovies_std_very_bad**2));
pred_prob_bad=&prior_bad*&female_bad*&member_yes_bad*1/(2*3.14)*1/(&age_std_bad*&nmovies_std_b
*exp(-(age-&age_mean_bad)**2/(2*&age_std_bad**2)-(nmovies-&nmovies_mean_bad)**2/(2*&nmovies_s
pred_prob_okay=&prior_okay*&female_okay*&member_yes_okay*1/(2*3.14)*1/(&age_std_okay*&nmovies
*exp(-(age-&age_mean_okay)**2/(2*&age_std_okay**2)-(nmovies-&nmovies_mean_okay)**2/(2*&nmovie
pred_prob_good=&prior_good*&female_good*&member_yes_good*1/(2*3.14)*1/(&age_std_good*&nmovies
*exp(-(age-&age_mean_good)**2/(2*&age_std_good**2)-(nmovies-&nmovies_mean_good)**2/(2*&nmovie
pred_prob_very_good=&prior_very_good*&female_very_good*&member_yes_very_good*1/(2*3.14)*1/(&a
*&nmovies_std_very_good)*exp(-(age-&age_mean_very_good)**2/(2*&age_std_very_good**2)-(nmovies
-&nmovies_mean_very_good)**2/(2*&nmovies_std_very_good**2));
end;

if (gender='M' and member='yes') then
do;
pred_prob_very_bad=&prior_very_bad*&male_very_bad*&member_yes_very_bad*1/(2*3.14)*1/(&age_std
*&nmovies_std_very_bad)*exp(-(age-&age_mean_very_bad)**2/(2*&age_std_very_bad**2)-(nmovies
-&nmovies_mean_very_bad)**2/(2*&nmovies_std_very_bad**2));
pred_prob_bad=&prior_bad*&male_bad*&member_yes_bad*1/(2*3.14)*1/(&age_std_bad*&nmovies_std_ba
*exp(-(age-&age_mean_bad)**2/(2*&age_std_bad**2)-(nmovies-&nmovies_mean_bad)**2/(2*&nmovies_s
pred_prob_okay=&prior_okay*&male_okay*&member_yes_okay*1/(2*3.14)*1/(&age_std_okay*&nmovies_s
*exp(-(age-&age_mean_okay)**2/(2*&age_std_okay**2)-(nmovies-&nmovies_mean_okay)**2/(2*&nmovie
pred_prob_good=&prior_good*&male_good*&member_yes_good*1/(2*3.14)*1/(&age_std_good*&nmovies_s
*exp(-(age-&age_mean_good)**2/(2*&age_std_good**2)-(nmovies-&nmovies_mean_good)**2/(2*&nmovie
pred_prob_very_good=&prior_very_good*&male_very_good*&member_yes_very_good*1/(2*3.14)*1/(&age
*&nmovies_std_very_good)*exp(-(age-&age_mean_very_good)**2/(2*&age_std_very_good**2)-(nmovies
-&nmovies_mean_very_good)**2/(2*&nmovies_std_very_good**2));
end;
run;

/*COMPUTING PREDICTION ACCURACY*/
data test;
 set test;
 max_prob=max(pred_prob_very_bad, pred_prob_bad,
 pred_prob_okay, pred_prob_good, pred_prob_very_good);
 if max_prob=pred_prob_very_good then pred_class='very good';
 if max_prob=pred_prob_very_bad then pred_class='very bad';
 if max_prob=pred_prob_bad then pred_class='bad';
 if max_prob=pred_prob_okay then pred_class='okay';
 if max_prob=pred_prob_good then pred_class='good';
 if pred_class=rating then pred=1; else pred=0;
run;
```

```
proc sql;
  select mean(pred) as accuracy
   from test;
quit;
```

accuracy
0.278146

In R:

```
movie.data<- read.csv(file="./movie_data.csv", header=TRUE, sep=",")

movie.data$gender<- ifelse(movie.data$gender=='M',1,0)
movie.data$member<- ifelse(movie.data$member=='yes',1,0)

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
set.seed(444625)
sample <- sample(c(TRUE, FALSE), nrow(movie.data), replace=TRUE, prob=c(0.8,0.2))
train<- movie.data[sample,]
test<- movie.data[!sample,]

test.x<- data.matrix(test[-5])
test.y<- data.matrix(test[5])

#FITTING NAIVE BAYES BINARY CLASSIFIER
library(e1071)
nb.multiclass<- naiveBayes(as.factor(rating) ~ age + gender + member + nmovies, data=train)

#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
pred.y<- as.numeric(predict(nb.multiclass, test.x))

print(paste('accuracy=', round((1-mean(test.y!=pred.y))*100, digits=2), '%'))
```

```
"accuracy= 32.53 %"
```

In Python:

```
import pandas
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from statistics import mean

movie_data=pandas.read_csv('./movie_data.csv')
code_gender={'M':1,'F':0}
code_member={'yes':1,'no':0}
code_rating={'very bad':1,'bad':2,'okay':3,'good':4,'very good':5}

movie_data['gender']=movie_data['gender'].map(code_gender)
movie_data['member']=movie_data['member'].map(code_member)
movie_data['rating']=movie_data['rating'].map(code_rating)

X=movie_data.iloc[:,0:4].values
y=movie_data.iloc[:,4].values

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.20,
random_state=457752)

#FITTING NAIVE BAYES BINARY CLASSIFIER
gnb=GaussianNB()
gnb.fit(X_train, y_train)

#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
y_pred = gnb.predict(X_test)
y_test=pandas.DataFrame(y_test,columns=['rating'])
y_pred=pandas.DataFrame(y_pred,columns=['predicted'])
df=pandas.concat([y_test,y_pred],axis=1)

match=[]
for i in range(len(df)):
    if df['rating'][i]==df['predicted'][i]:
        match.append(1)
    else:
        match.append(0)

print('accuracy=', round(mean(match)*100,2),'%')
```

```
accuracy= 36.84 %
```

□

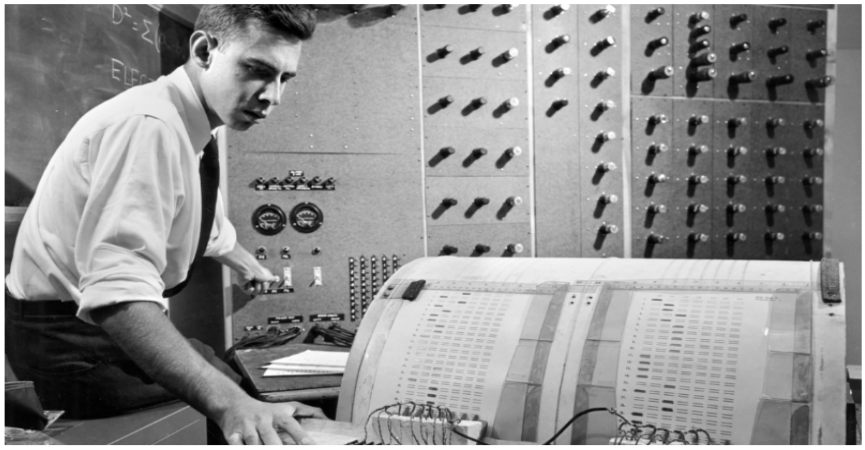
ARTIFICIAL NEURAL NETWORK

An **artificial neural network (ANN)** is a subfield of Artificial Intelligence where it attempts to mimic the network of neurons that makes up a human brain so that computers will have the option to understand things and make decisions in a human-like manner. The ANN is designed by programming computers to behave simply like interconnected brain cells.

An ANN consists of an **input layer**, **hidden layers** of **nodes** (or **neurons**, or **perceptrons**), and an **output layer**. The first layer receives raw input, it is processed by multiple hidden layers, and the last layer produces the result.

Historical Note: The oldest type of neural network, known as **Perceptron**, was introduced by Frank Rosenblatt in 1958.

CORNELL CHRONICLE

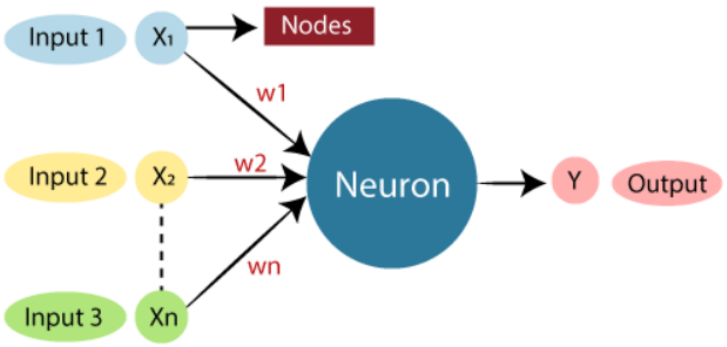


Division of Rare and Manuscript Collections

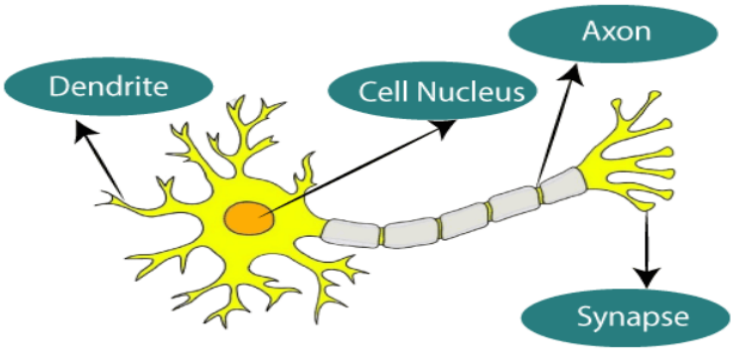
Frank Rosenblatt '50, Ph.D. '56, works on the "perceptron" – what he described as the first machine "capable of having an original idea."

Professor's perceptron paved the way for AI – 60 years too soon

A typical ANN looks something like this:



A typical diagram of a biological neural network in the brain looks like this:



Dendrites from biological neural networks represent inputs in ANN, cell nucleus represents nodes, synapse represents weights, and axon represents output.

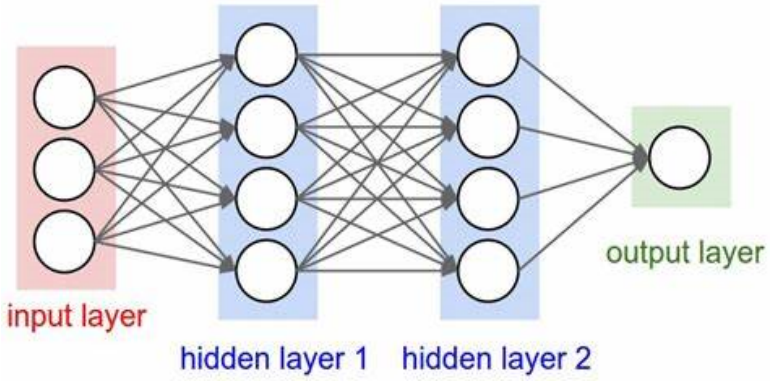
Glossary

• **Dendrite** is a short-branched extension of a nerve cell, along which impulses received from other cells at synapses are transmitted to the cell body.

• **Synapse** is a junction between two nerve cells, consisting of a minute gap across which impulses pass by diffusion of a neurotransmitter.

• **Axon** is a long threadlike part of a nerve cell along which impulses are conducted from the cell body to other cells.

To understand the concept of the architecture of an ANN, we have to understand what a neural network consists of. In order to define a neural network that consists of a large number of artificial neurons, which are nodes arranged in a sequence of layers. Let us look at three types of layers available in an ANN: input layer, hidden layer, and output layer.



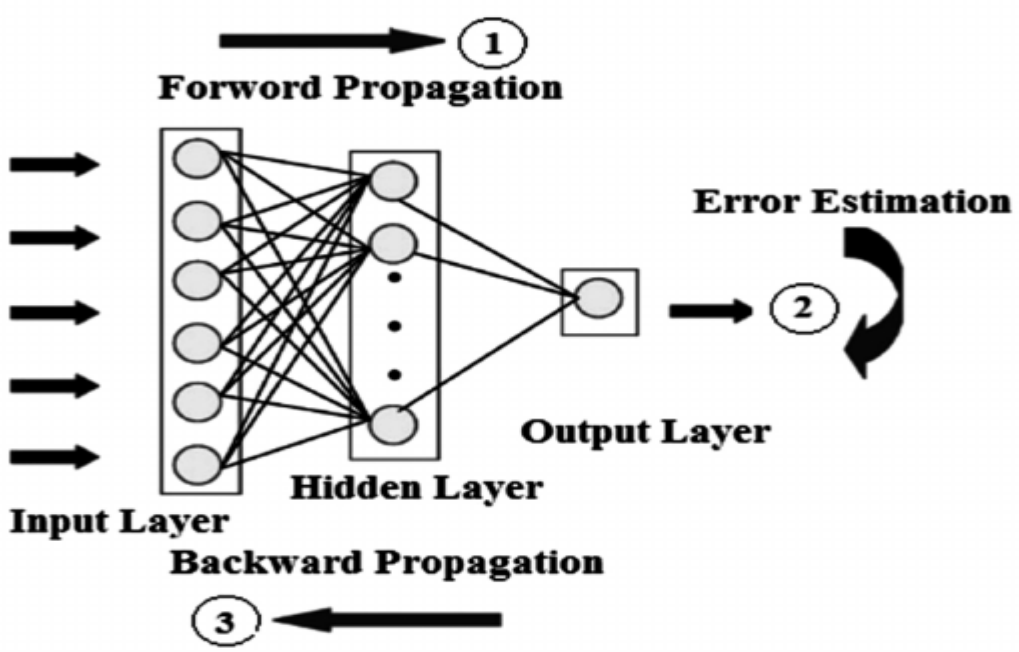
Input Layer accepts inputs provided by a programmer. The **input features** (or the predictor variables) can be categorical or numeric.

Hidden Layer performs all the calculations to find hidden features and patterns.

Output Layer consists of the output variable (or response variable). For regression ANNs, the output variable is numeric; for binary ANN, the output variable is binary, and for multinomial ANN, the output variable assumes multi-class values.

In an ANN, the input goes through a series of transformations using the hidden layer, which finally results in the output expressed as a linear combination of weighted input features with a bias term included.

It determines the weighted total that is passed as an input to an **activation function** to produce the output. Activation functions choose whether a node should fire or not. Only those who are fired make it to the output layer. This process is called **feed forward** (or **forward propagation**). After producing the output, an error (or loss) is calculated and a correction is sent back to the network. This process is known as **back propagation** (or **backward propagation**).



Historical Note. ANN with back propagation was introduced in Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). "Learning representations by back-propagating errors". *Nature*, 323(6088), 533–536. Most of the ANN applications in the literature utilize multi-layer feed-forward with a back propagation learning algorithm.

An **epoch** is a complete cycle through the full training set when building an ANN. An **iteration** is the number of steps through partitioned packets of the training data, needed to complete one epoch.

Learning Algorithm

An ANN starts with a set of initial weights and then gradually modifies the weights during the training cycle to settle down to a set of weights capable of realizing the input-output mapping with a minimum error.

Denote by $\mathbf{x_i} = (x_{i1}, \ldots, x_{ik})'$, $i = 1, \ldots, n$, the set of vectors of input variables (predictor variables), and let $\mathbf{y} = (y_1, \ldots, y_n)$ be the output vector. Also, suppose there is one hidden layer with m neurons $h_1, \ldots, h_m$. The response of the hidden layer for the ith individual is the vector $\mathbf{h_i} = (h_{i1}, \ldots, h_{im})'$. An ANN produces outputs governed by the relations:

$$\mathbf{h_i} = f\big(\mathbf{W_h}\,\mathbf{x_i} + \mathbf{b_i}\big), \quad \text{and} \quad \hat{y}_i = f(\mathbf{W_i^*}\,\mathbf{h_i} + b_i^*),$$

where f is the activation function,

$$\mathbf{W_h} = \begin{bmatrix} w_{11} & \dots & w_{1k} \\ \dots & \dots & \dots \\ w_{m1} & \dots & w_{mk} \end{bmatrix}$$

is the hidden layer weight matrix, $\mathbf{W_i^*} = (w_{i1}^*, \dots, w_{im}^*)$ is the vector of output weights for individual i, $\mathbf{b_i} = (b_{i1}, \dots, b_{im})'$ is the hidden layer bias vector for individual i, and b_i^* is the output layer bias for individual i.

The activation functions that are used in SAS, R, and Python are (defined for $x \in \mathbb{R}$) **logistic** (or **sigmoid**) $f(x) = \dfrac{\exp(x)}{1+\exp(x)}$, and **hyperbolic tangent** (or **tanh**) $f(x) = \tanh(x) = \dfrac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$.

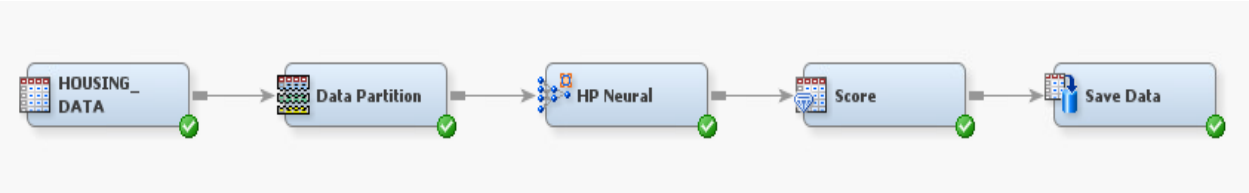
The loss functions used to compute errors in the back propagation algorithm are: mean squared error for regression and cross-entropy for classification.

The method of **steepest descent** is used to update the weights. For example, for the mean squared error loss function, the loss function is $L = \dfrac{1}{2}\sum_{i=1}^{n}\left(y_i - f(\mathbf{W_i^*}\,\mathbf{h_i} + b_i^*)\right)^2$. The weights are updated according to the recursive relation $w_{ij}^*(new) = w_{ij}^*(old) - \lambda\dfrac{\partial L}{\partial w_{ij}^*}, j = 1, \dots, m,$, where λ is referred to as **learning rate**. The same algorithm applies to the weights in the hidden layers W_h.

Example. We fit an ANN to the housing data.

In SAS:

We run the following path diagram in Enterprise Miner, choosing logistic as the activation function.

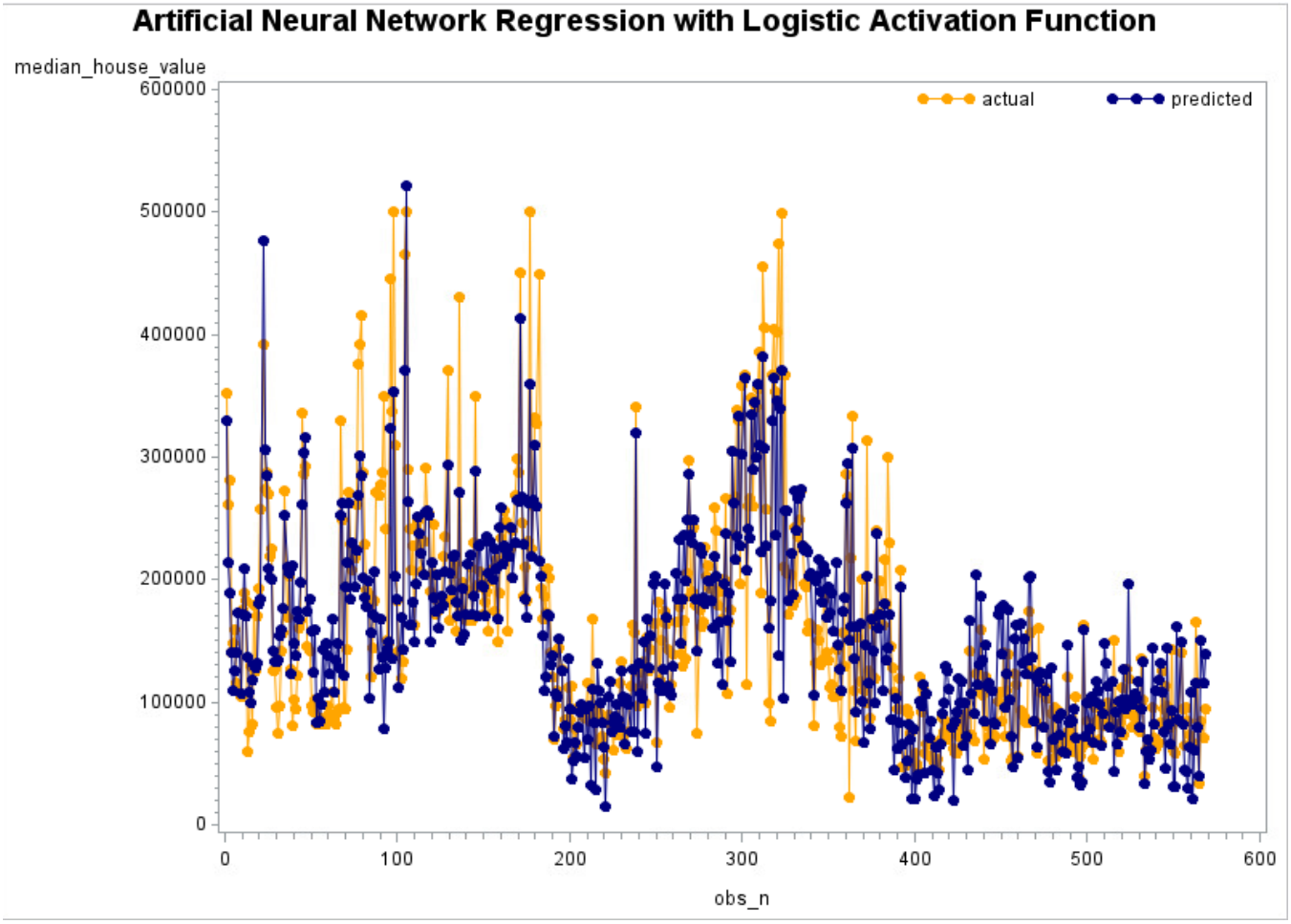


Then we run the following code to compute the accuracy.

```
data accuracy;
set tmp1.em_save_test;
ind10=(abs(R_median_house_value)<0.10*median_house_value);
ind15=(abs(R_median_house_value)<0.15*median_house_value);
ind20=(abs(R_median_house_value)<0.20*median_house_value);
obs_n=_N_;
run;

proc sql;
select mean(ind10) as accuracy10,
mean(ind15) as accuracy15, mean(ind20) as
accuracy20
from accuracy;
quit;
```

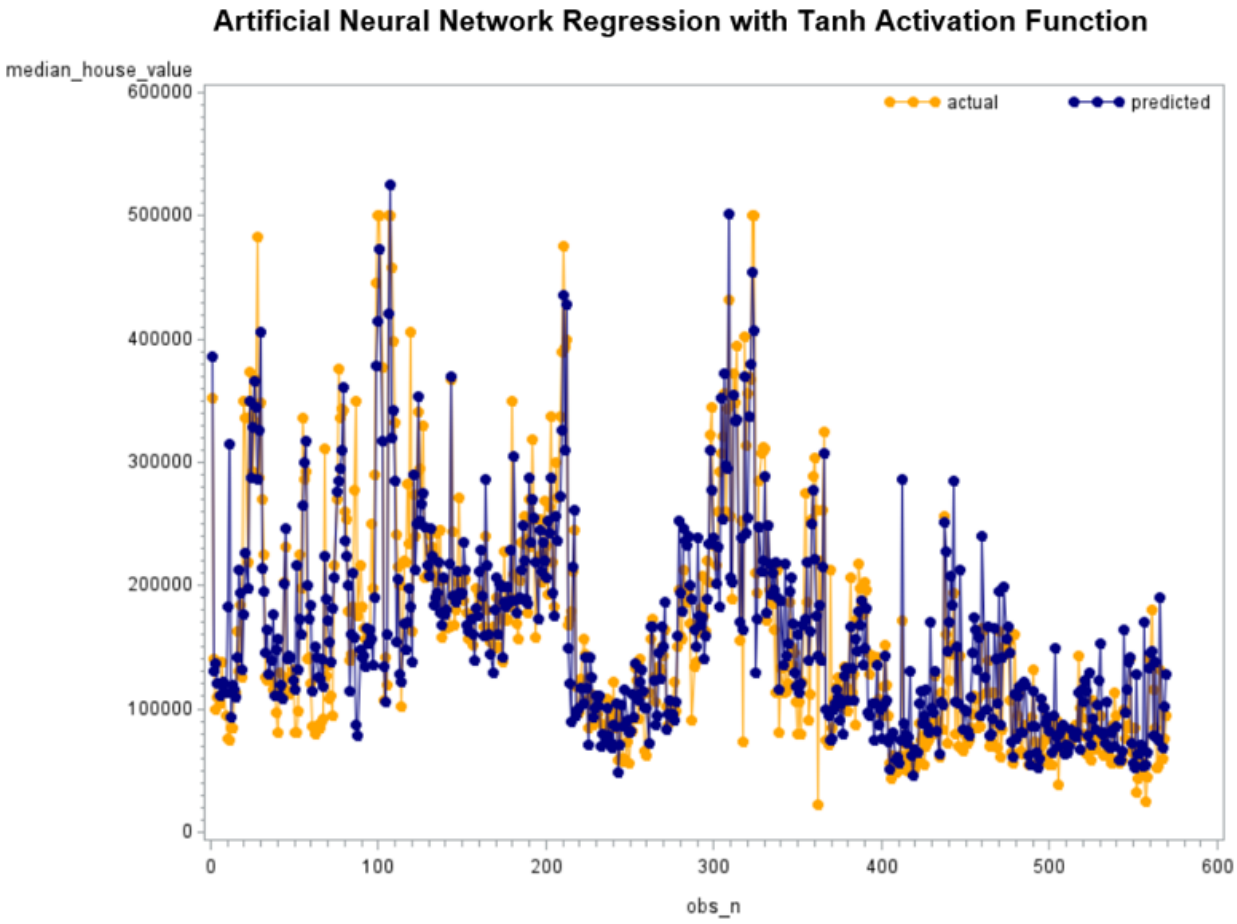
accuracy10	accuracy15	accuracy20
0.274648	0.40493	0.507042

```
/*PLOTTING ACTUAL AND PREDICTED VALUES FOR TESTING DATA*/;
goptions reset=all border;
title1 "Artificial Neural Network Regression with Logistic Activation Function";
symbol1 interpol=join value=dot color=orange;
symbol2 interpol=join value=dot color=navy;
legend1 value=("actual" "predicted")
position=(top right inside) label=none;
proc gplot data=accuracy;
plot median_house_value*obs_n
EM_PREDICTION*obs_n/ overlay legend=legend1;
run;
```



Next, we run the same diagram, changing the activation function to the default function tanh. The accuracy and plot are given below.

accuracy10	accuracy15	accuracy20
0.328647	0.472759	0.595782

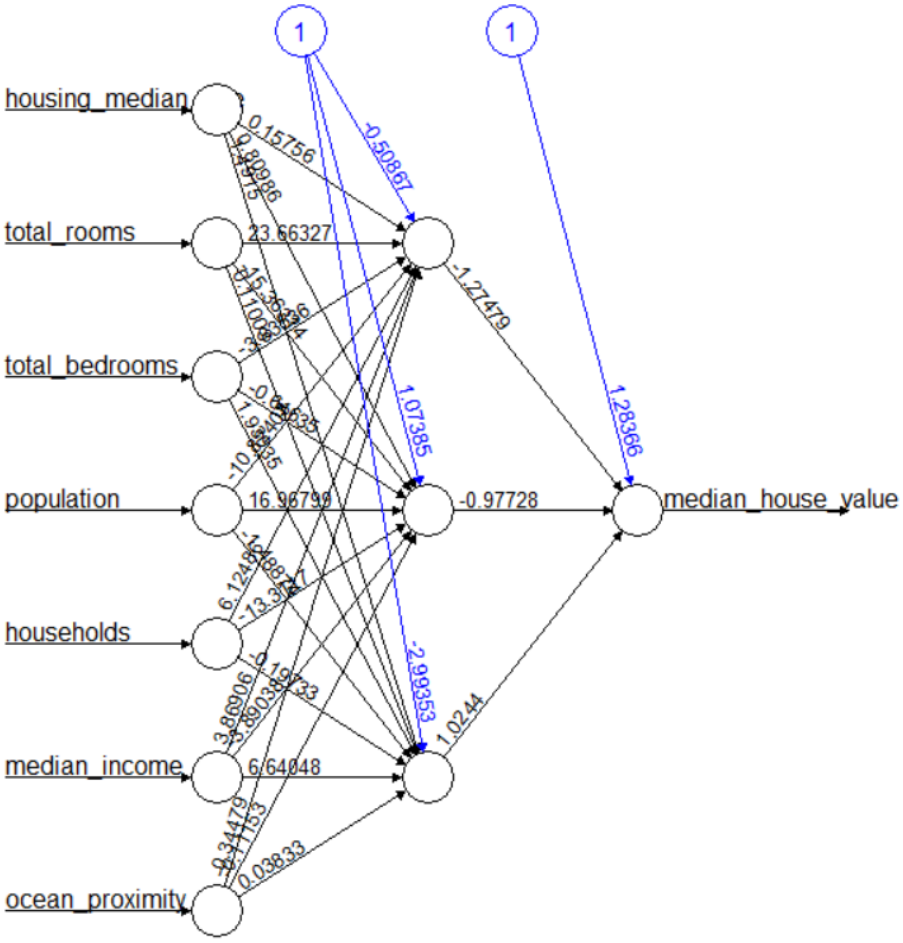


The fitted ANN model with the tanh activation function has a higher accuracy than that with the logistic activation function.

In R:

```
housing.data<- read.csv(file="./housing_data.csv", header=TRUE, sep=",")

housing.data$ocean_proximity<- ifelse(housing.data$ocean_proximity=='<1H OCEAN', 1,
ifelse(housing.data$ocean_proximity=='INLAND',2, ifelse(housing.data$ocean_proximity=='NEAR
BAY',3,4)))

#SCALING VARIABLES TO FALL IN [0,1]
library(dplyr)
```

```
scale01 <- function(x){
    (x-min(x))/(max(x)-min(x))
}

housing.data<- housing.data %>% mutate_all(scale01)

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
set.seed(346634)
sample <- sample(c(TRUE, FALSE), nrow(housing.data), replace=TRUE, prob=c(0.8,0.2))
train<- housing.data[sample,]
test<- housing.data[!sample,]
test.x<- data.matrix(test[-8])
test.y<- data.matrix(test[8])

#FITTING ANN WITH LOGISTIC ACTIVATION FUNCTION
#install.packages("neuralnet")
library(neuralnet)
ann.reg<- neuralnet(median_house_value ~ housing_median_age+total_rooms+total_bedrooms
+population+households+median_income +ocean_proximity, data=train, hidden=3, act.fct="logistic")

#PLOTTING THE DIAGRAM
plot(ann.reg)
```




```
#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
pred.y<- predict(ann.reg, test.x)

#accuracy within 10%
accuracy10<- ifelse(abs(test.y-pred.y)<0.10*test.y,1,0)

#accuracy within 15%
accuracy15<- ifelse(abs(test.y-pred.y)<0.15*test.y,1,0)

#accuracy within 20%
accuracy20<- ifelse(abs(test.y-pred.y)<0.20*test.y,1,0)

print('Prediction Accuracy')
print(paste('within 10%:', round(mean(accuracy10),4)))
```

```
 "within 10%: 0.2702"
```

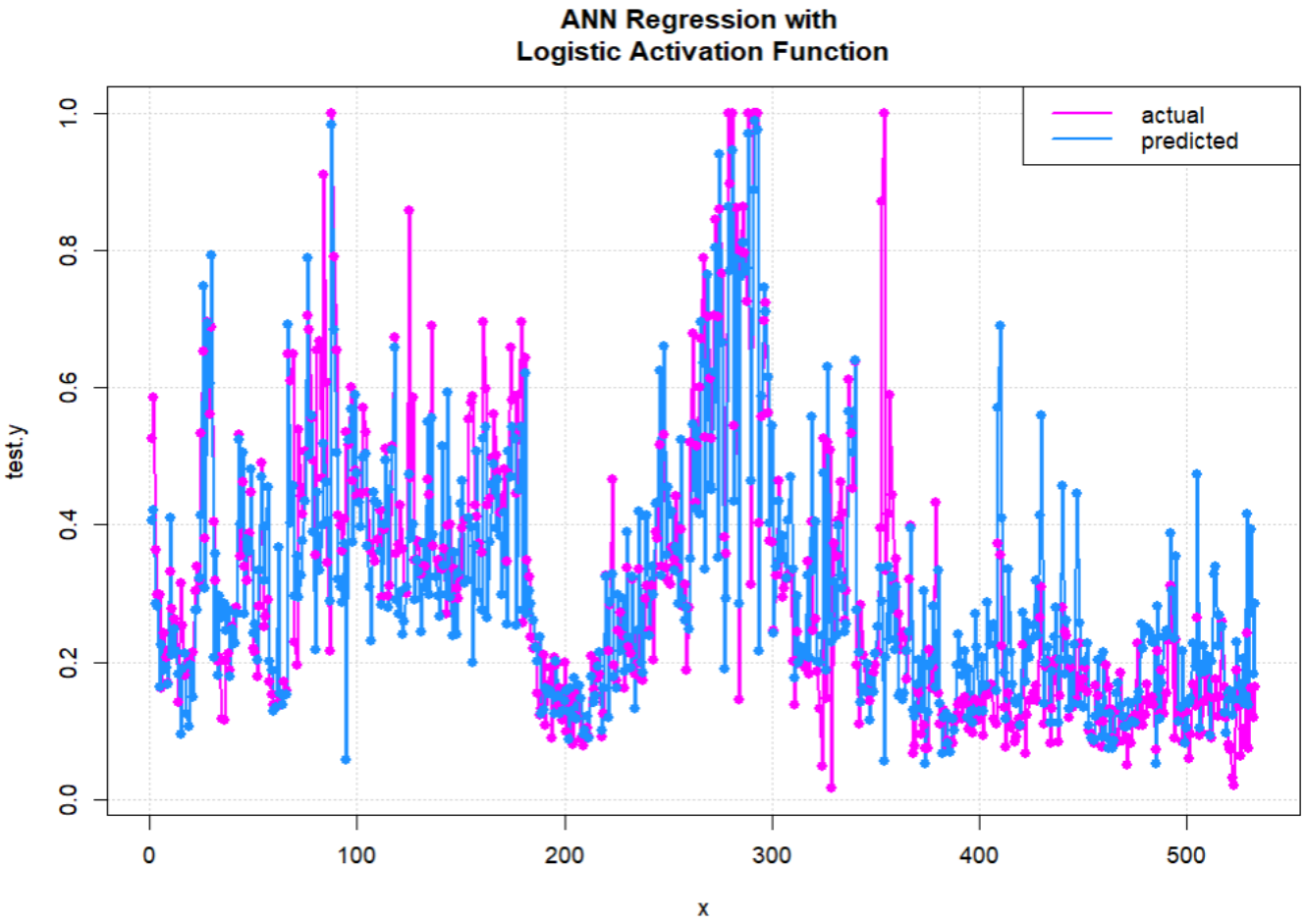
```
print(paste('within 15%:', round(mean(accuracy15),4)))
```

```
"within 15%: 0.3809"
```

```
print(paste('within 20%:', round(mean(accuracy20),4)))
```

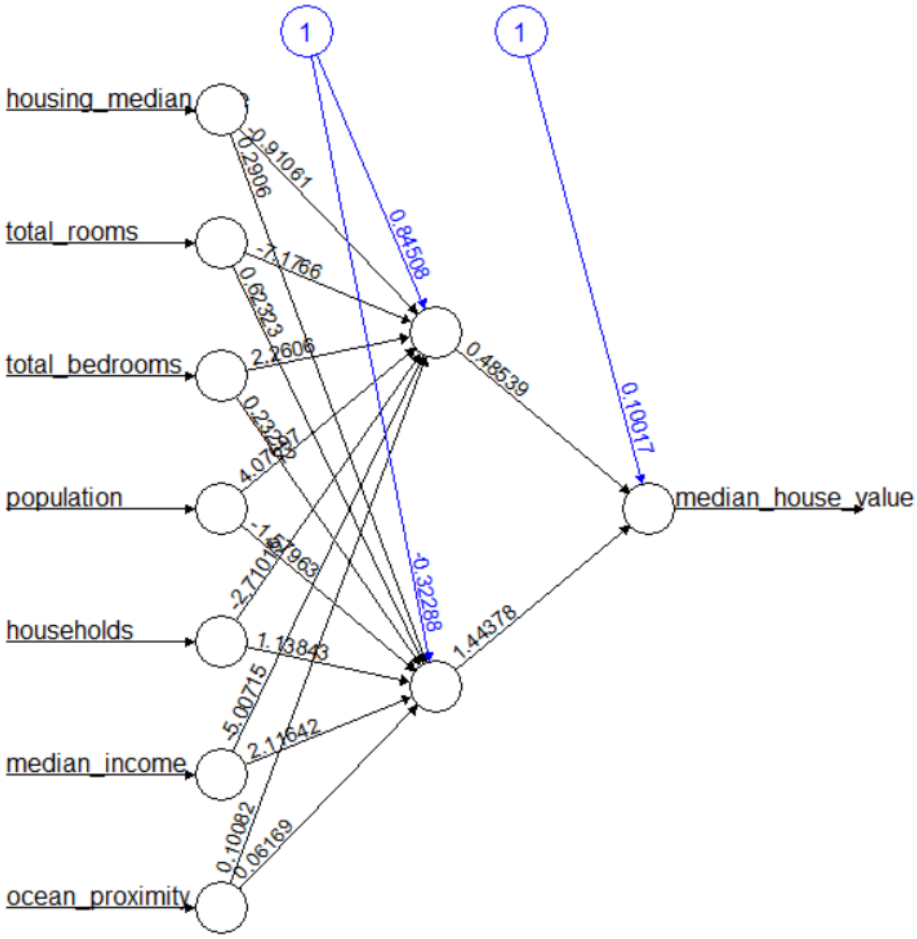
```
"within 20%: 0.4709"
```

```
#PLOTTING ACTUAL AND PREDICTED VALUES FOR TESTING DATA
x<- 1:length(test.y)
plot(x, test.y, type="l", lwd=2, col="magenta", main="ANN Regression with Logistic Activation
Function", panel.first=grid())
lines(x, pred.y, lwd=2, col="dodgerblue")
points(x,test.y, pch=16, col="magenta")
points(x, pred.y, pch=16, col="dodgerblue")
legend("topright", c("actual", "predicted"), lty=1, lwd=2, col=c("magenta","dodgerblue"))
```



#FITTING ANN WITH TANH ACTIVATION FUNCTION ann.reg<- neuralnet(median_house_value ~ housing_median_age+total_rooms+total_bedrooms+population +households+median_income +ocean_proximity, data=train, hidden=2, act.fct="tanh")

#PLOTTING THE DIAGRAM
plot(ann.reg)




```
#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
pred.y<- predict(ann.reg, test.x)

#accuracy within 10%
accuracy10<- ifelse(abs(test.y-pred.y)<0.10*test.y,1,0)

#accuracy within 15%
accuracy15<- ifelse(abs(test.y-pred.y)<0.15*test.y,1,0)

#accuracy within 20%
accuracy20<- ifelse(abs(test.y-pred.y)<0.20*test.y,1,0)

print('Prediction Accuracy')
print(paste('within 10%:', round(mean(accuracy10),4)))
```

```
 "within 10%: 0.2383 "
```

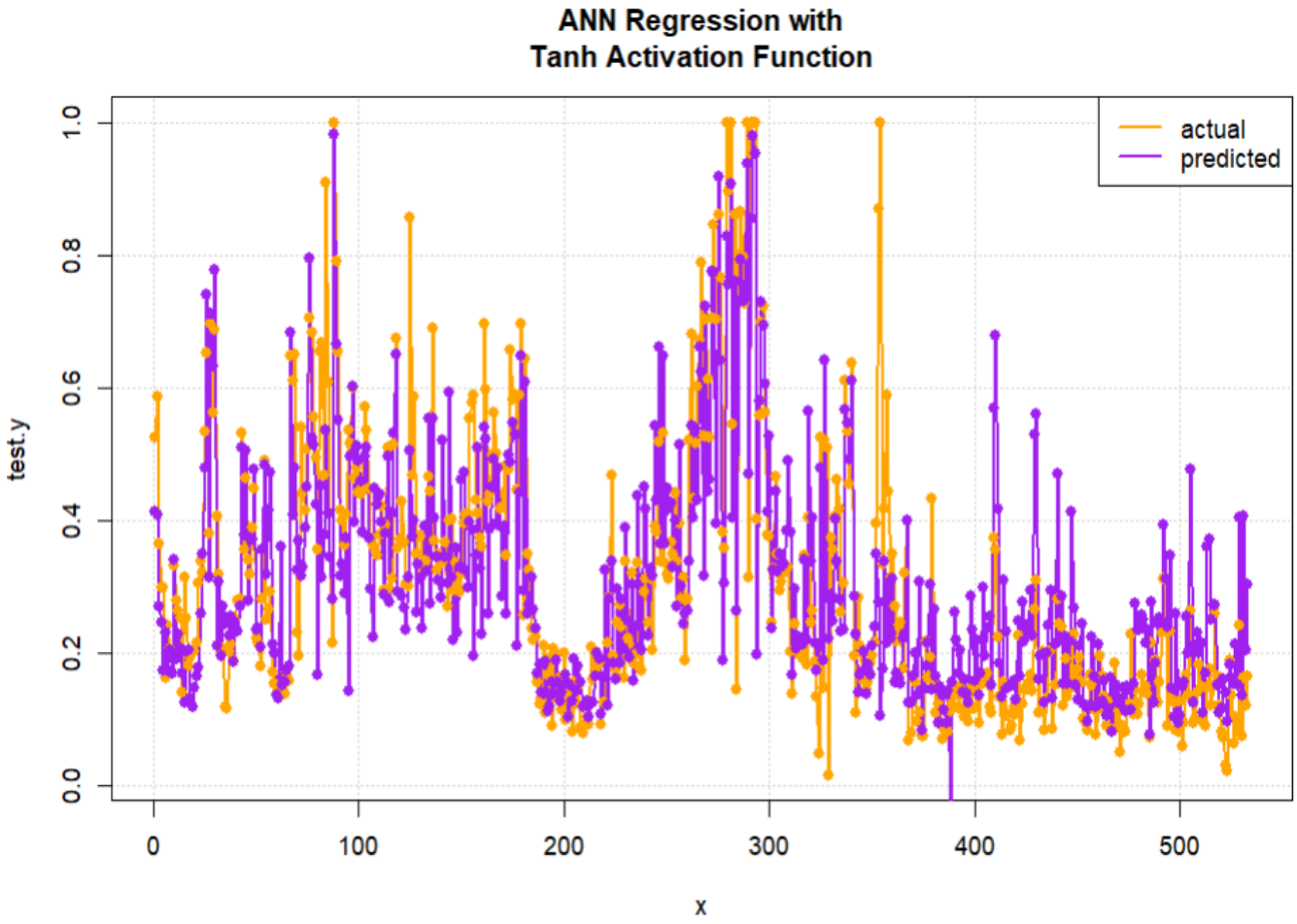
```
print(paste('within 15%:', round(mean(accuracy15),4)))
```

```
"within 15%: 0.3265"
```

```
print(paste('within 20%:', round(mean(accuracy20),4)))
```

```
"within 20%: 0.454"
```

```
#PLOTTING ACTUAL AND PREDICTED VALUES FOR TESTING DATA
x<- 1:length(test.y)
plot(x, test.y, type="l", lwd=2, col="orange", main="ANN Regression with Tanh Activation Function", panel.first=grid())
lines(x, pred.y, lwd=2, col="purple")
points(x,test.y, pch=16, col="orange")
points(x, pred.y, pch=16, col="purple")
legend("topright", c("actual", "predicted"), lty=1, lwd=2, col=c("orange","purple"))
```



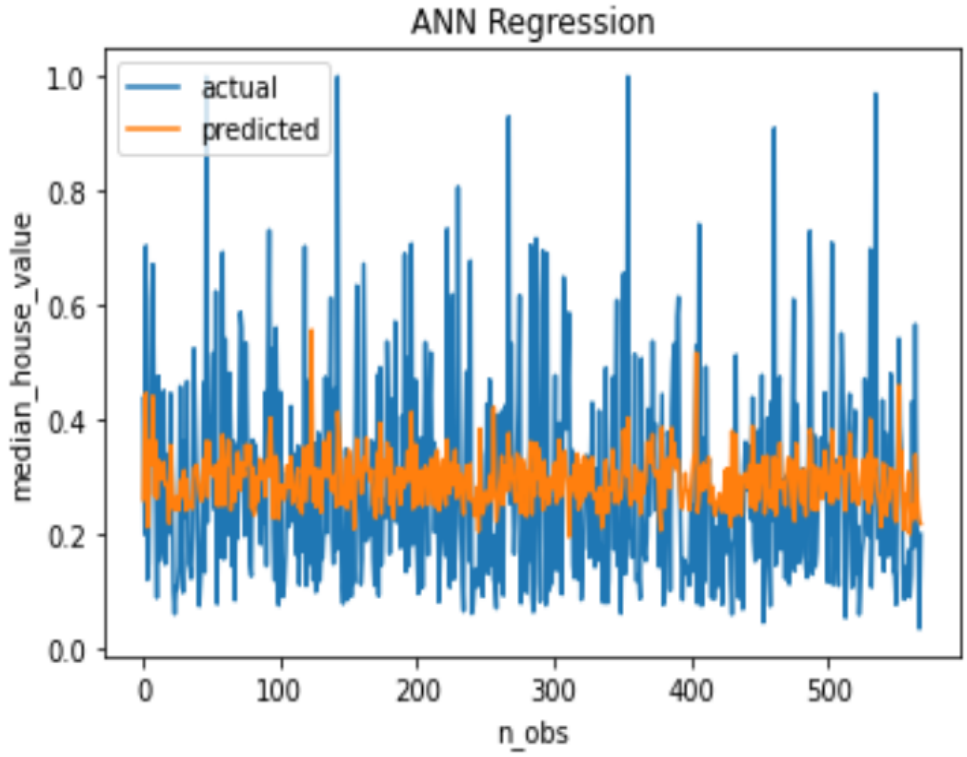
In Python: We will an ANN with tanh and sigmoid (logistic) activation functions.

```
import numpy
import pandas
from statistics import mean
housing_data=pandas.read_csv('./housing_data.csv')

coding={'<1H OCEAN': 1, 'INLAND': 2, 'NEAR BAY': 3, 'NEAR OCEAN': 4}
housing_data['ocean_proximity']=housing_data['ocean_proximity'].map(coding)

#SCALING VARIABLES TO FALL IN [0,1]
from sklearn import preprocessing
scaler=preprocessing.MinMaxScaler()
scaler_fit=scaler.fit_transform(housing_data)
scaled_housing_data=pandas.DataFrame(scaler_fit, columns=housing_data.columns)

X=scaled_housing_data.iloc[:,0:7].values
y=scaled_housing_data.iloc[:,7].values

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.20,
random_state=449626)

#FITTING AN ARTIFICIAL NEURAL NETWORK
# Installing required libraries
!pip install tensorflow
!pip install keras
```

```
from keras.models import Sequential
from keras.layers import Dense
model = Sequential()

#Defining the input layer and one hidden layer
model.add(Dense(units=3, input_dim=7, kernel_initializer='uniform',
activation='tanh'))

#Defining the output neuron
model.add(Dense(1))

#Compiling the model
model.compile(loss='mean_squared_error')

#Fitting the ANN to the training set
model.fit(X_train, y_train)

#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
y_pred=model.predict(X_test)

ind10=[]
ind15=[]
ind20=[]

for sub1, sub2 in zip(y_pred, y_test):
    ind10.append(1) if abs(sub1-sub2)<0.10*sub2 else ind10.append(0)
    ind15.append(1) if abs(sub1-sub2)<0.15*sub2 else ind15.append(0)
    ind20.append(1) if abs(sub1-sub2)<0.20*sub2 else ind20.append(0)
```

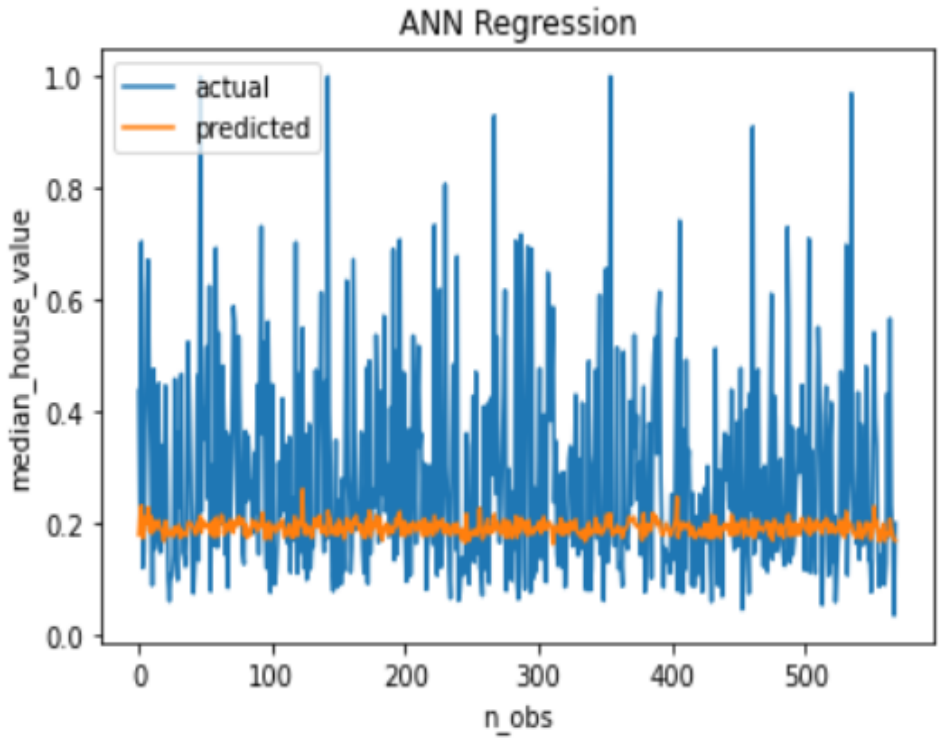
```
#accuracy within 10%
accuracy10=mean(ind10)
print('accuracy within 10% =', round(accuracy10,4))

#accuracy within 15%
accuracy15=mean(ind15)
print('accuracy within 15% =', round(accuracy15,4))

#accuracy within 20%
accuracy20=mean(ind20)
print('accuracy within 20% =', round(accuracy20,4))


#plotting actual and predicted obsevations vs. observation number
import matplotlib.pyplot as plt

n_obs=list(range(0,len(y_test)))
plt.plot(n_obs, y_test, label="actual")
plt.plot(n_obs, y_pred, label="predicted")
plt.xlabel('n_obs')
plt.ylabel('median_house_value')
plt.title('ANN Regression')
plt.legend()
plt.show()
```

```
accuracy within 10% = 0.1301
accuracy within 15% = 0.2144
accuracy within 20% = 0.2865
```




```
model.add(Dense(units=3, input_dim=7, kernel_initializer='uniform',
activation='sigmoid'))
```

```
accuracy within 10% = 0.0879
accuracy within 15% = 0.1424
accuracy within 20% = 0.2162
```

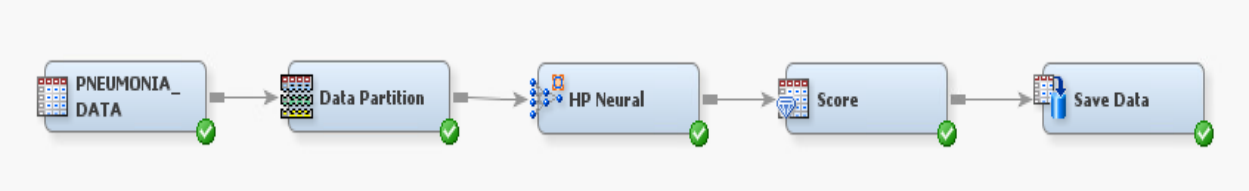


□

Example. For the pneumonia data, we fit an ANN.

In SAS:

In Enterprise Miner we run the following diagram:



Note that the scale for the target variable must be specified as "nominal". Then we run the following code to compute the accuracy for the model with the logistic activation function:

```
data accuracy;
set tmp1.em_save_test;
match=(em_classification=em_classtarget);
run;

proc sql;
select mean(match) as accuracy
from accuracy;
quit;
```

accuracy
0.706052

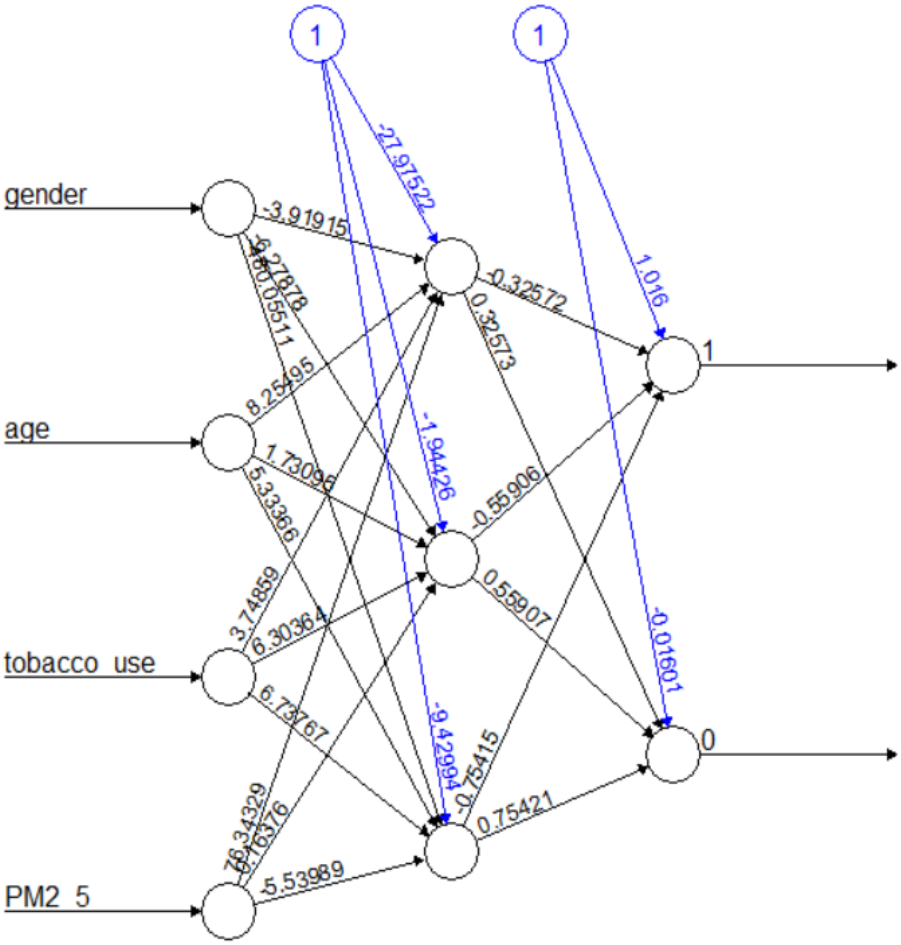
and that for the model with the tanh activation function:

accuracy
0.760116

Note that the model with the tanh activation function has a higher prediction accuracy.

In R:
pneumonia.data<- read.csv(file="./pneumonia_data.csv", header=TRUE, sep=",")

```
pneumonia.data$pneumonia<- ifelse(pneumonia.data$pneumonia=="yes",1,0)
pneumonia.data$gender<- ifelse(pneumonia.data$gender=='M',1,0)
pneumonia.data$tobacco_use<- ifelse(pneumonia.data$tobacco_use=='yes',1,0)

#SCALING VARIABLES TO FALL IN [0,1]
library(dplyr)

scale01 <- function(x){
    (x-min(x))/(max(x)-min(x))
}

pneumonia.data<- pneumonia.data %>% mutate_all(scale01)

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
set.seed(503548)
sample <- sample(c(TRUE, FALSE), nrow(pneumonia.data), replace=TRUE, prob=c(0.8,0.2))
train<- pneumonia.data[sample,]
test<- pneumonia.data[!sample,]

train.x<- data.matrix(train[-5])
train.y<- data.matrix(train[5])
test.x<- data.matrix(test[-5])
test.y<- data.matrix(test[5])

library(neuralnet)

#FITTING ANN WITH LOGISTIC ACTIVATION FUNCTION AND ONE LAYER WITH THREE
NEURONS
ann.class<- neuralnet(as.factor(pneumonia) ~ gender + age + tobacco_use + PM2_5, data=train,
hidden=3, act.fct="logistic")

#PLOTTING THE DIAGRAM
plot(ann.class)
```

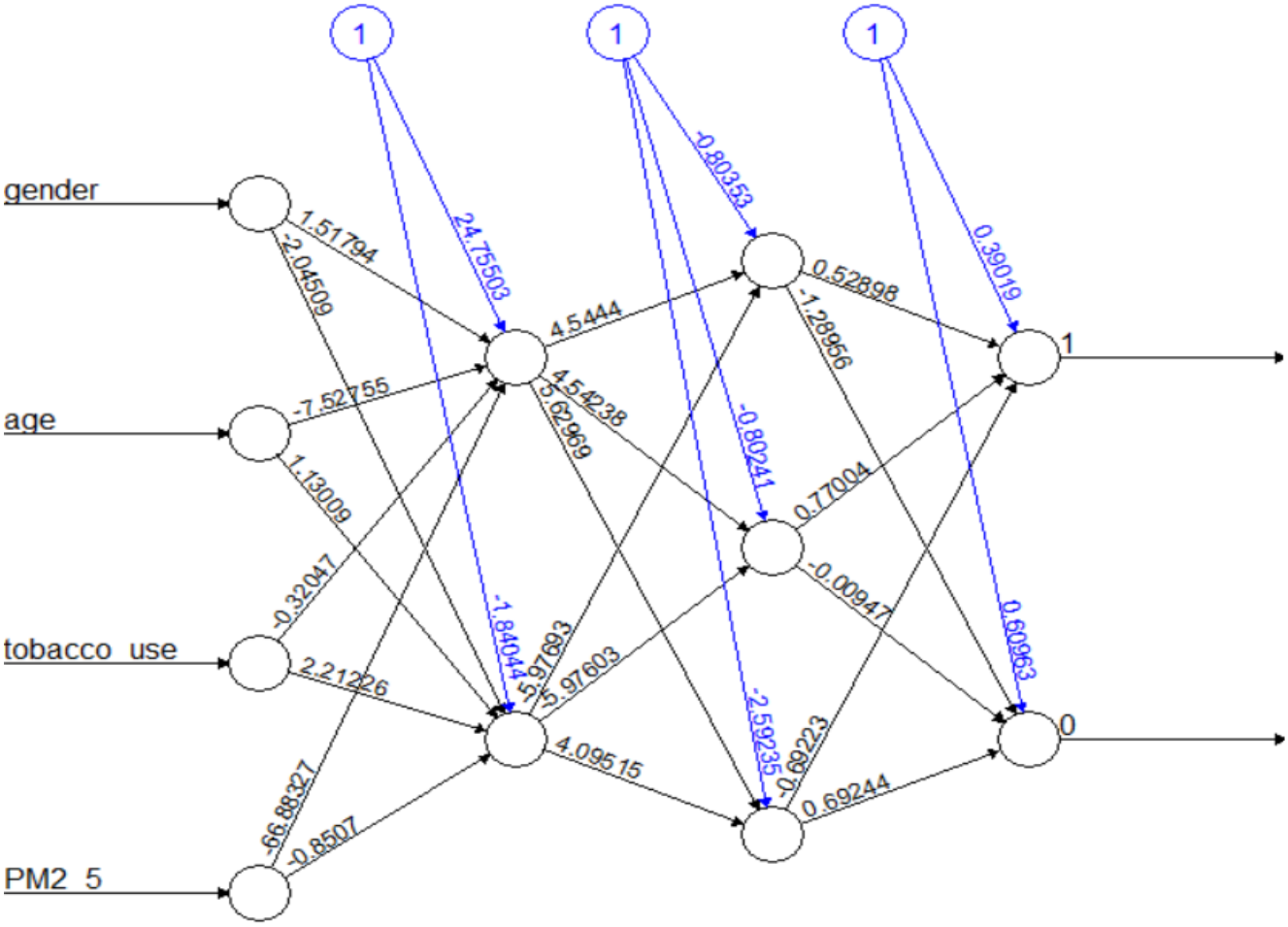



```
#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
pred.prob<- predict(ann.class, test.x)[,1]

match<- c()
pred.y<- c()
for (i in 1:length(test.y)){
    pred.y[i]<- ifelse(pred.prob[i]>0.5,1,0)
    match[i]<- ifelse(test.y[i]==pred.y[i],1,0)
}

print(paste("accuracy=", round(mean(match), digits=4)))
```

```
"accuracy= 0.2747"
```

```
#FITTING ANN WITH LOGISTIC ACTIVATION FUNCTION AND C(2,3) LAYERS
ann.class<- neuralnet(as.factor(pneumonia) ~ gender + age + tobacco_use + PM2_5, data=train,
```

```
hidden=c(2,3), act.fct="logistic")

#PLOTTING THE DIAGRAM
plot(ann.class)
```

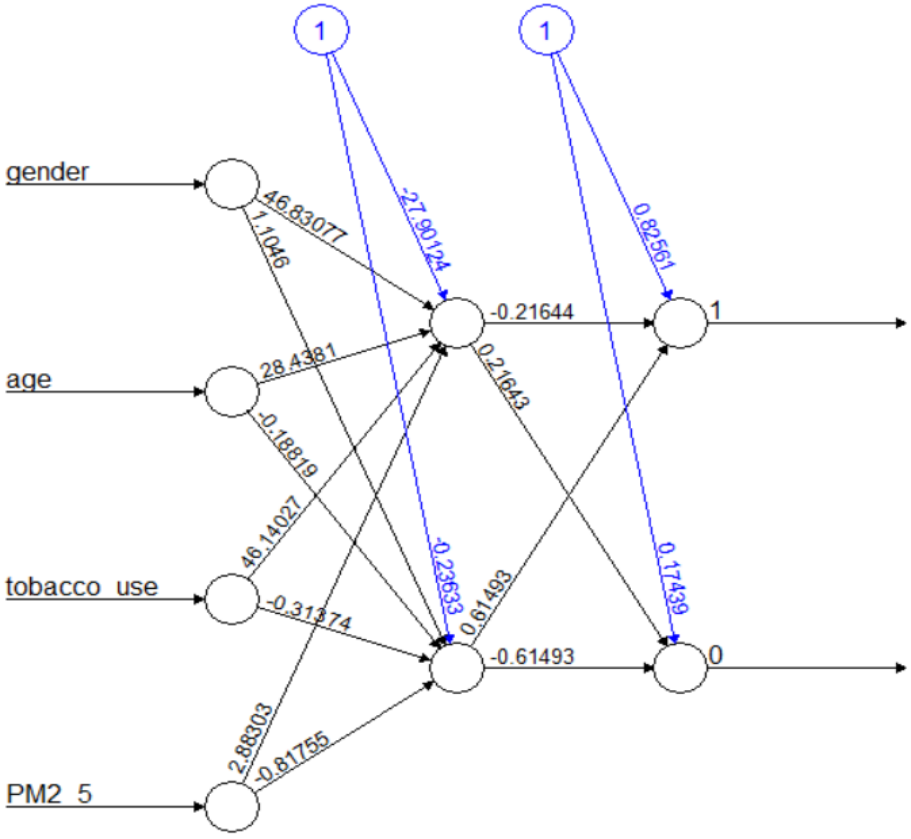



```
#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
pred.prob<- predict(ann.class, test.x)[,1]

match<- c()
pred.y<- c()
for (i in 1:length(test.y)){
    pred.y[i]<- ifelse(pred.prob[i]>0.5,1,0)
    match[i]<- ifelse(test.y[i]==pred.y[i],1,0)
}
```

```
print(paste("accuracy=", round(mean(match), digits=4)))
```

```
"accuracy= 0.2623"
```

```
#FITTING ANN WITH TANH ACTIVATION FUNCTION
ann.class<- neuralnet(as.factor(pneumonia) ~ gender + age + tobacco_use + PM2_5, data=train,
hidden=2, act.fct="tanh")

#PLOTTING THE DIAGRAM
plot(ann.class)
```




```
#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
pred.prob<- predict(ann.class, test.x)[,1]

match<- c()
pred.y<- c()
for (i in 1:length(test.y)){
```

```
    pred.y[i]<- ifelse(pred.prob[i]>0.5,1,0)
    match[i]<- ifelse(test.y[i]==pred.y[i],1,0)
}

print(paste("accuracy=", round(mean(match), digits=4)))
```

```
"accuracy= 0.3025"
```

In Python:

```
import numpy
import pandas
from sklearn.model_selection import train_test_split

pneumonia_data=pandas.read_csv('./pneumonia_data.csv')
code_gender={'M':1,'F':0}
code_tobacco_use={'yes':1,'no':0}
code_pneumonia={'yes':1,'no':0}

pneumonia_data['gender']=pneumonia_data['gender'].map(code_gender)
pneumonia_data['tobacco_use']=pneumonia_data['tobacco_use'].map(code_tobacco_use)
pneumonia_data['pneumonia']=pneumonia_data['pneumonia'].map(code_pneumonia)
y=pneumonia_data['pneumonia'].values

#SCALING VARIABLES TO FALL IN [0,1]
from sklearn import preprocessing
scaler=preprocessing.MinMaxScaler()
scaler_fit=scaler.fit_transform(pneumonia_data)
scaled_pneumonia_data=pandas.DataFrame(scaler_fit, columns=pneumonia_data.columns)

X=scaled_pneumonia_data.iloc[:,0:4].values
y=scaled_pneumonia_data.iloc[:,4].values

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.20,
random_state=505606)

#FITTING AN ARTIFICIAL NEURAL NETWORK
import keras
from keras.models import Sequential
from keras.layers import Dense
```

```
biclassifier=Sequential()

#Defining the input layer and first hidden layer
biclassifier.add(Dense(units=3, activation='sigmoid'))

#Defining the output neuron
biclassifier.add(Dense(1))

#Compiling the model
biclassifier.compile(loss='binary_crossentropy')

#Fitting the ANN to the training set
biclassifier.fit(X_train, y_train)

#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
y_pred=numpy.round(biclassifier.predict(X_test),0) #predicted probability of 1

from sklearn import metrics
print('Accuracy:', round(metrics.accuracy_score(y_test, y_pred)*100, 2),'%')
```

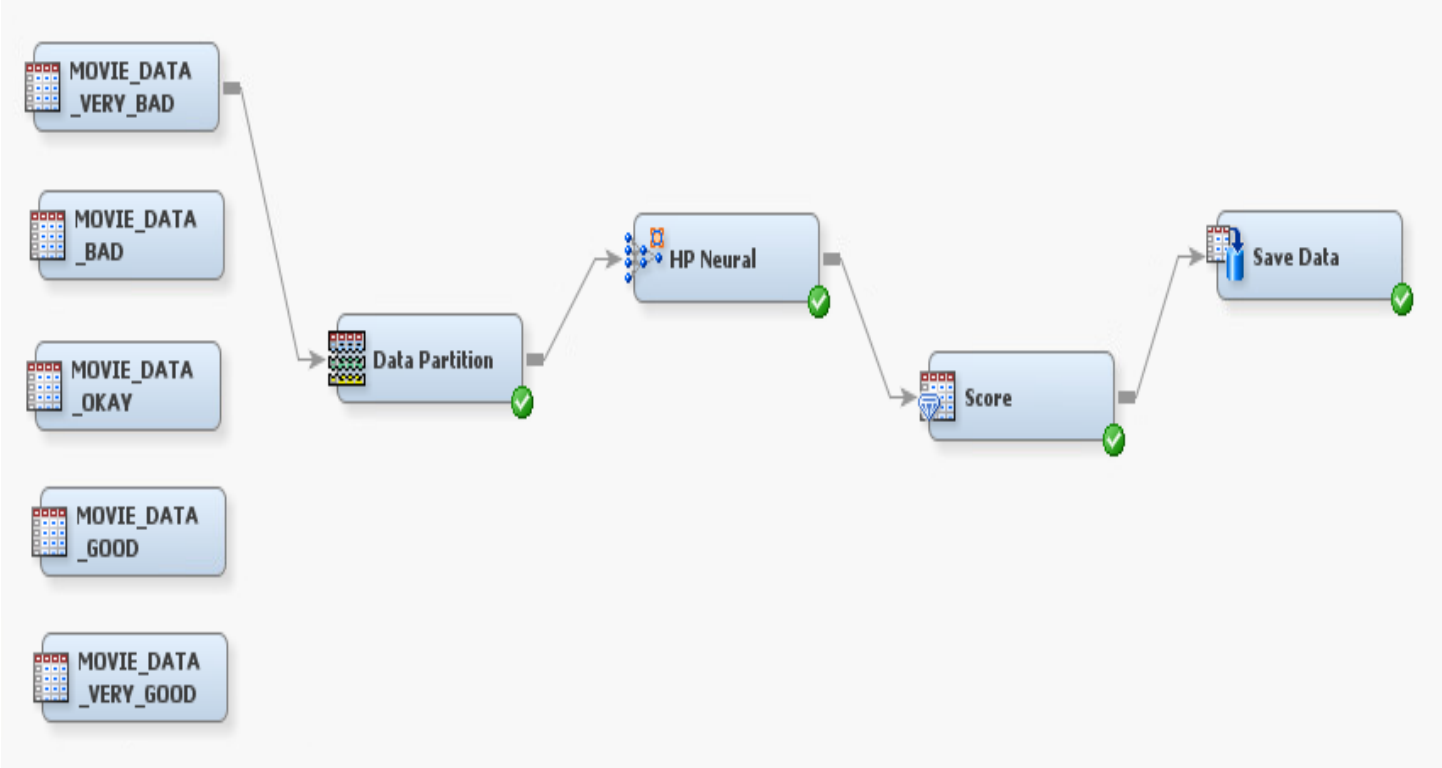
```
Accuracy: 67.92 %
```

□

Example. We fit an ANN to the movies data.

In SAS:

In Enterprise Miner we run the following diagram, with the logistic activation function for the model. We set the indicator variables to "Target" with "Nominal" scale, and to "Rejected" all the other indicators.



Then we run the following code to compute the accuracy.

```
data rating;
set sasuser.movies;
_dataobs_=_n_;
keep _dataobs_ rating;
run;

proc sort;
by _dataobs_;
run;

data very_bad;
set './em_save_test_verybad.sas7bdat';
predprob_verybad=em_eventprobability;
class_verybad=em_classtarget;
keep _dataobs_ class_verybad predprob_verybad;
run;

proc sort;
```

```
by _dataobs_;
run;

data bad;
set './em_save_test_bad.sas7bdat';
predprob_bad=em_eventprobability;
class_bad=em_classtarget;
keep _dataobs_ class_bad predprob_bad;
run;
proc sort;
by _dataobs_;
run;

data okay;
set './em_save_test_okay.sas7bdat';
predprob_okay=em_eventprobability;
class_okay=em_classtarget;
keep _dataobs_ class_okay predprob_okay;
run;

proc sort;
by _dataobs_;
run;

data good;
set './em_save_test_good.sas7bdat';
predprob_good=em_eventprobability;
class_good=em_classtarget;
keep _dataobs_ class_good predprob_good;
run;

proc sort;
by _dataobs_;
run;

data very_good;
set './em_save_test_verygood.sas7bdat';
predprob_verygood=em_eventprobability;
class_verygood=em_classtarget;
keep _dataobs_ class_verygood predprob_verygood;
run;
```

```
proc sort;
by _dataobs_;
run;

data all_data;
merge rating very_bad bad okay good very_good;
by _dataobs_;
if cmiss(predprob_verybad, predprob_bad,
predprob_okay, predprob_good, predprob_verygood)=0;
run;

data all_data;
set all_data;
predprob_max=max(predprob_very_bad, predprob_bad,
predprob_okay, predprob_good, predprob_very_good);
if (predprob_very_good=predprob_max) then pred_class='very good';
if (predprob_very_bad=predprob_max) then pred_class='very bad';
if (predprob_bad=predprob_max) then pred_class='bad';
if (predprob_okay=predprob_max) then pred_class='okay';
if (predprob_good=predprob_max) then pred_class='good';
keep rating pred_class;
run;

data all_data;
set all_data;
match=(rating=pred_class);
run;

proc sql;
select mean(match) as accuracy
from all_data;
quit;
```

accuracy
0.302817

Setting the activation function to the default value of tanh, we run the diagram and the SAS code again to output the accuracy of

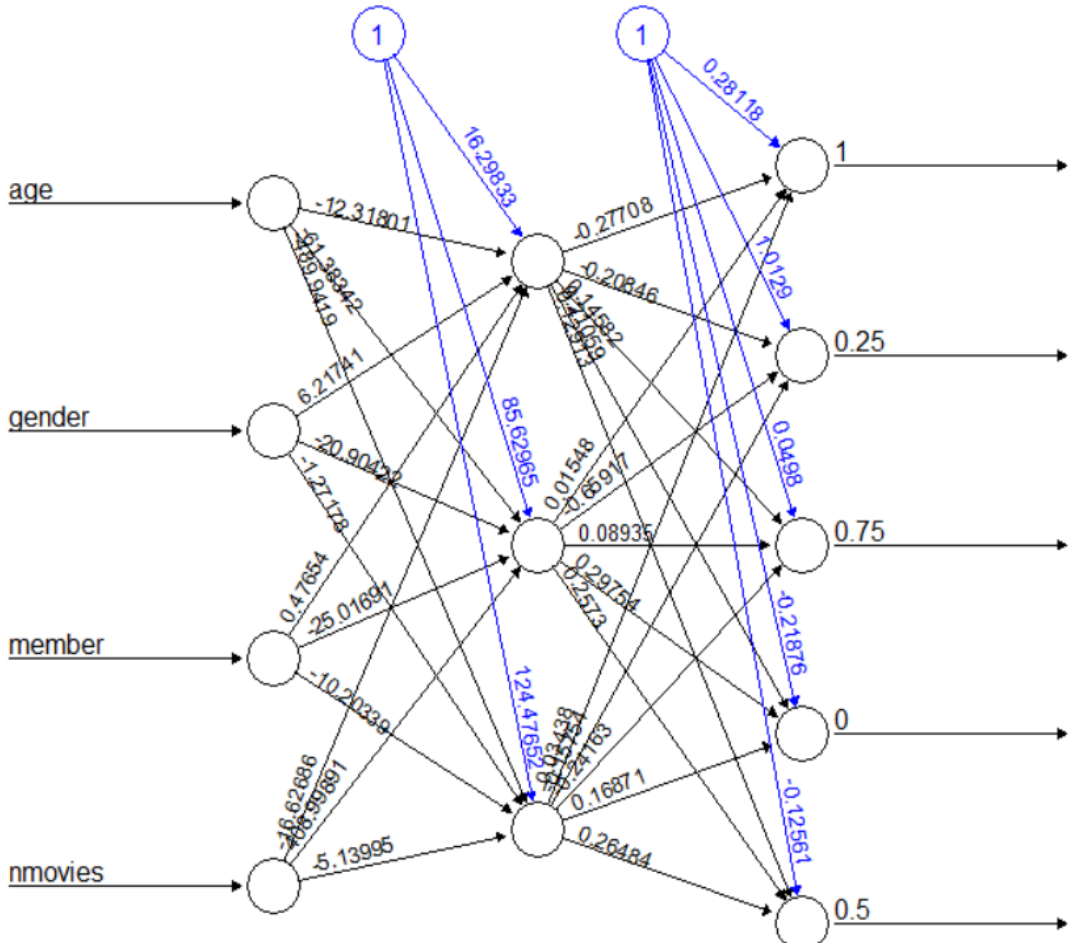
accuracy
0.246479

The model with the logistic activation function has a higher accuracy.

In R:

```
movie.data<- read.csv(file="./movie_data.csv", header=TRUE, sep=",")

movie.data$gender<- ifelse(movie.data$gender=='M',1,0)
movie.data$member<- ifelse(movie.data$member=='yes',1,0)

movie.data$rating<- ifelse(movie.data$rating=='very bad',1, ifelse(movie.data$rating=='bad',2,ifelse(movie.data
ifelse(movie.data$rating=='good',4,5))))

#SCALING VARIABLES TO FALL IN [0,1]
library(dplyr)

scale01 <- function(x){
    (x-min(x))/(max(x)-min(x))
}

movie.data<- movie.data %>% mutate_all(scale01)

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
set.seed(100001)
sample <- sample(c(TRUE, FALSE), nrow(movie.data), replace=TRUE, prob=c(0.8,0.2))
train<- movie.data[sample,]
test<- movie.data[!sample,]

train.x<- data.matrix(train[-5])
train.y<- data.matrix(train[5])
test.x<- data.matrix(test[-5])
test.y<- data.matrix(test[5])
```

```
#FITTING ANN WITH LOGISTIC ACTIVATION FUNCTION
library(neuralnet)
set.seed(4544446)
ann.mclass<- neuralnet(as.factor(rating) ~ age + gender + member + nmovies, data=train, hidden=3, act.fct="logistic")

#PLOTTING THE DIAGRAM
plot(ann.mclass)
```




```
#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
pred.prob <- predict(ann.mclass, test.x)
pred.prob<- as.data.frame(pred.prob)

colnames(pred.prob)<- c(0, 0.25, 0.5, 0.75, 1)
```

```
pred.class<- apply(pred.prob, 1, function(x) colnames(pred.prob)[which.max(x)])

match<- c()
for (i in 1:length(test.y)) {
    match[i]<- ifelse(pred.class[i]==as.character(test.y[i]),1,0)
}

print(accuracy<- mean(match))
```
0.3611111

In Python:

```
import numpy
import pandas
from sklearn.model_selection import train_test_split
from statistics import mean

movie_data=pandas.read_csv('./movie_data_ind.csv')

code_gender={'M':1,'F':0}
code_member={'yes':1,'no':0}
code_rating={'very bad':1,'bad':2,'okay':3,'good':4,'very good':5}

movie_data['gender']=movie_data['gender'].map(code_gender)
movie_data['member']=movie_data['member'].map(code_member)
movie_data['rating']=movie_data['rating'].map(code_rating)

#SCALING VARIABLES TO FALL IN [0,1]
from sklearn import preprocessing
scaler=preprocessing.MinMaxScaler()
scaler_fit=scaler.fit_transform(movie_data)
scaled_movie_data=pandas.DataFrame(scaler_fit, columns=movie_data.columns)

X=scaled_movie_data.iloc[:,0:4].values
y=scaled_movie_data.iloc[:,4:10].values

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.20, random_state=116008)

y_train=y_train[:,1:6]
y_true=y_test[:,0]
y_test=y_test[:,1:6]

#FITTING AN ARTIFICIAL NEURAL NETWORK
import keras
from keras.models import Sequential
from keras.layers import Dense
import tensorflow
tensorflow.random.set_seed(454545)
```

```
multiclassifier=Sequential()

#Defining one hidden layer
multiclassifier.add(Dense(units=3, activation='sigmoid'))

#Defining the output neuron
multiclassifier.add(Dense(units=5, activation='tanh'))

#Compiling the model
multiclassifier.compile(loss='categorical_crossentropy')

#Fitting the ANN to the training set
multiclassifier.fit(X_train, y_train)

#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
pred_prob=pandas.DataFrame(multiclassifier.predict(X_test))

y_pred=0.25*pred_prob.idxmax(axis=1)

match=[]
for i in range(len(y_pred)):
    if y_pred[i]==y_true[i]:
        match.append(1)
    else:
        match.append(0)

print('accuracy=', round(mean(match),4))
```

```
accuracy= 0.3158
```

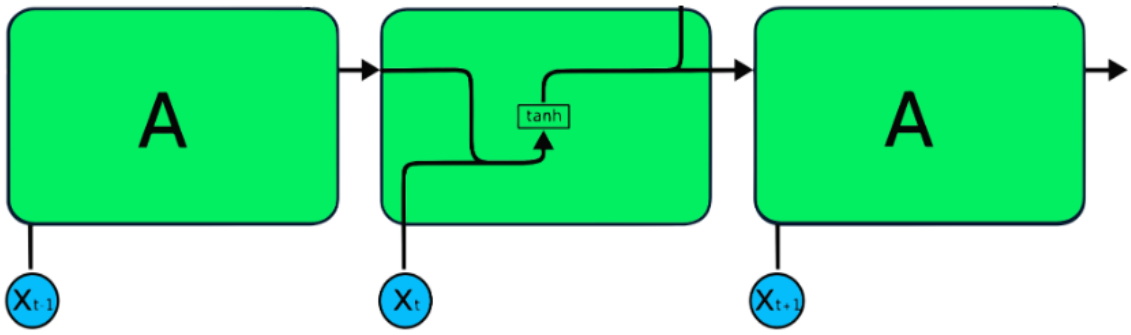
□

RECURRENT NEURAL NETWORK

A **recurrent neural network** (RNN) is a type of neural network in which the output is determined by the current input and previously received inputs. RNN is used to model time series data. The simple RNN repeating modules have a basic structure with a single layer, and it remembers only one previous time step information. Simple RNN models usually run into two major issues:

• **Vanishing Gradient** problem occurs when the gradient becomes so small that updating parameters becomes insignificant; eventually, the algorithm stops learning.

• **Exploding Gradient** problem occurs when the gradient becomes too large, which makes the model unstable. In this case, larger error gradients accumulate, and the model weights become too large. This issue can cause longer training times and poor model performance.



More advanced RNN architectures that easily solve these problems are **long short term memory** (LSTM) and **gated recurrent unit** (GRU), as they are capable of remembering long periods of information. The LSTM has four interacting layers that communicate with each other. This four-layered structure helps LSTM retain long-term memory. The gated recurrent unit (GRU) is a variation of LSTM. It uses an update gate and reset gate to solve the vanishing gradient problem. These gates decide what information is important and pass it to the output. The gates can be trained to store information from long ago.

Example. We fit LSTM and GRU recurrent neural networks to the data in the file "TSLA.csv". This file contains daily closing Tesla stock price between 6/29/2010 and 3/13/2023, a total of 3198 observations. These historical data were downloaded from *https://yahoofinance.com*. We use R and Python.

In R:

```
tsla.data<- read.csv(file="./TSLA.csv", header=TRUE, sep=",")

#splitting data into testing and training sets
tsla.data$Year<- as.numeric(format(as.Date(tsla.data$Date, format="%Y-%m-%d"),"%Y"))

train.data<- tsla.data[which(tsla.data$Year<2022),1:2]
test.data<- tsla.data[which(tsla.data$Year>=2022),1:2]

#plotting training and testing data
plot(as.POSIXct(tsla.data$Date), tsla.data$Close, main="Daily Tesla Stock Closing Prices", xlab="Time",
ylab="Stock Price", pch="", panel.first=grid())
```

```
lines(as.POSIXct(train.data$Date), train.data$Close, lwd=2, col="blue")
lines(as.POSIXct(test.data$Date), test.data$Close, lwd=2, col="green")
legend("topleft", c("training", "testing"), lty=1, col=c("blue","green"))
```


```
#scaling prices to fall in [0,1]
price<- tsla.data$Close
price.sc<- (price-min(price))/(max(price)-min(price))

#creating train.x and train.y
nsteps<- 60 #width of sliding window
train.matrix <- matrix(nrow=nrow(train.data)-nsteps, ncol=nsteps+1)
for (i in 1:(nrow(train.data)-nsteps))
    train.matrix[i,]<- price.sc[i:(i+nsteps)]
```

```
train.x<- array(train.matrix[,-ncol(train.matrix)],dim=c(nrow(train.matrix),nsteps,1))
train.y<- train.matrix[,ncol(train.matrix)]

#creating test.x and test.y
test.matrix<- matrix(nrow=nrow(test.data), ncol=nsteps+1)
for (i in 1:nrow(test.data))
    test.matrix[i,]<- price.sc[(i+nrow(train.matrix)):(i+nsteps+nrow(train.matrix))]

test.x<- array(test.matrix[,-ncol(test.matrix)],dim=c(nrow(test.matrix),nsteps,1))
test.y<- test.matrix[,ncol(test.matrix)]

#FITTING LSTM MODEL
library(keras)
LSTM.model <- keras_model_sequential()

#specifying model structure
LSTM.model %>% layer_lstm(input_shape=dim(train.x)[2:3], units=nsteps)
LSTM.model %>% layer_dense(units=1, activation="tanh")
LSTM.model %>% compile(loss="mean_squared_error")

#training model
epochs<- 5
for(i in 1:epochs){
    LSTM.model %>% fit(train.x, train.y, batch_size=32, epochs=1)
    LSTM.model %>% reset_states() #clears the hidden states in network after every batch
}

#predicting for testing data
pred.y<- LSTM.model %>% predict(test.x, batch_size=32)

#rescaling test.y and pred.y back to the original scale
test.y.re<- test.y*(max(price)-min(price))+min(price)
pred.y.re<- pred.y*(max(price)-min(price))+min(price)

#computing prediction accuracy
accuracy10<- ifelse(abs(test.y.re-pred.y.re)<0.10*test.y.re,1,0)
accuracy15<- ifelse(abs(test.y.re-pred.y.re)<0.15*test.y.re,1,0)
accuracy20<- ifelse(abs(test.y.re-pred.y.re)<0.20*test.y.re,1,0)

print(paste("accuracy within 10%:", round(mean(accuracy10),4)))
```

```
"accuracy within 10%: 0.8494"
```

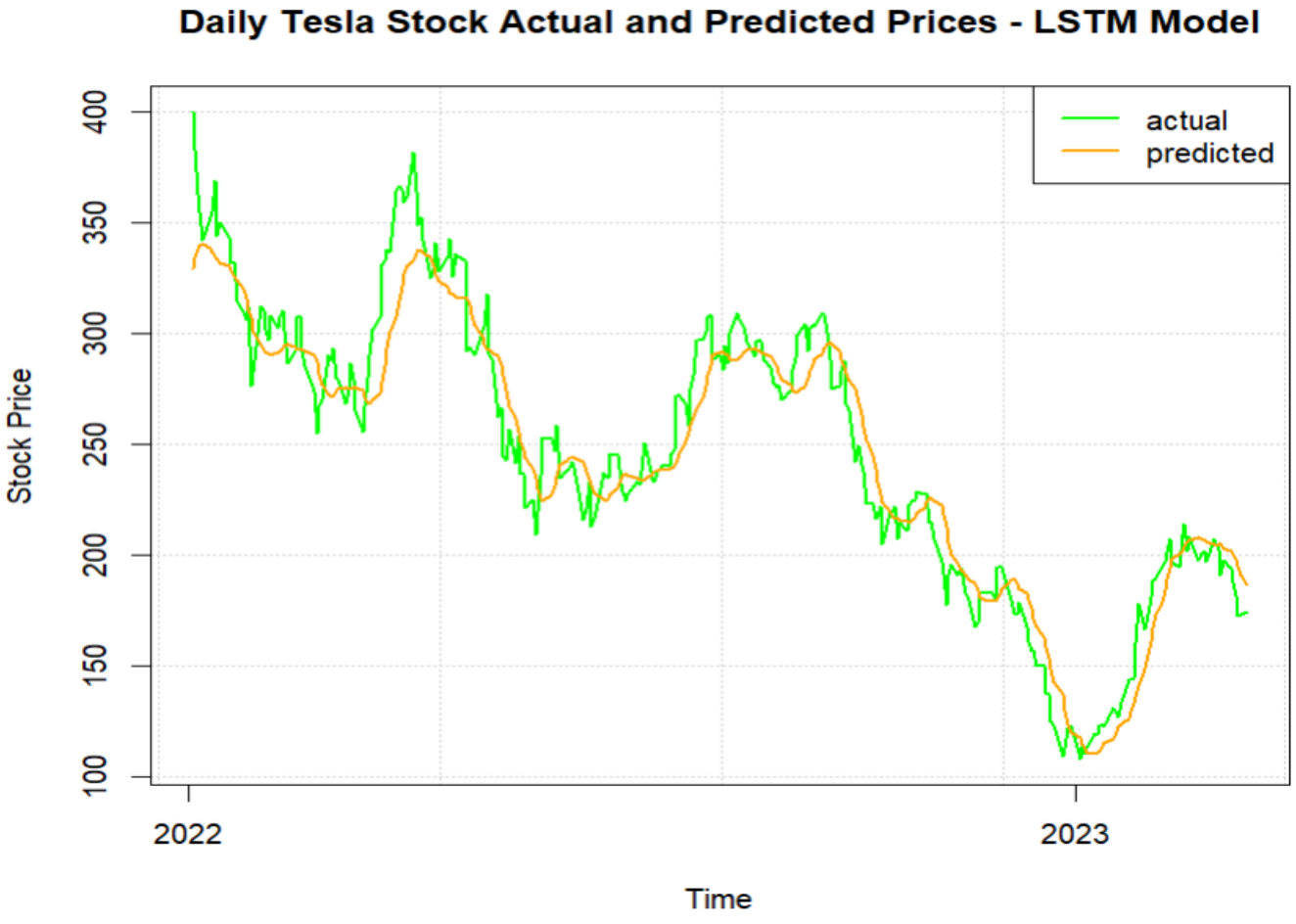
```
print(paste("accuracy within 15%:", round(mean(accuracy15),4)))
```

```
"accuracy within 15%: 0.9632"
```

```
print(paste("accuracy within 20%:", round(mean(accuracy20),4)))
```

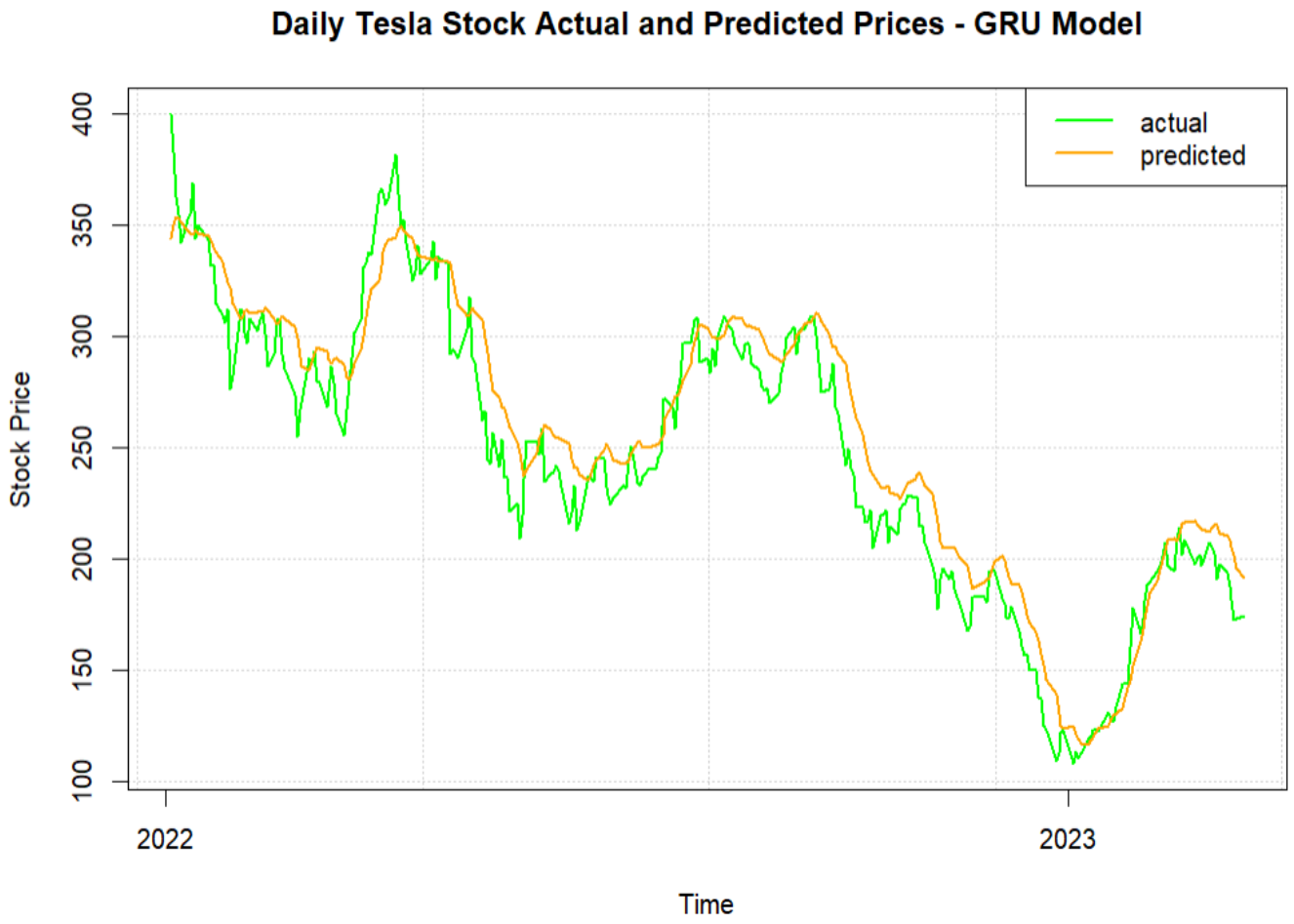
```
"accuracy within 20%: 0.9933"
```

```
#plotting actual and predicted values for testing data
plot(as.POSIXct(test.data$Date), test.y.re, type="l", lwd=2, col="green", main="Daily Tesla Stock Actual and Predicted Prices - LSTM Model", xlab="Time", ylab="Stock Price", panel.first=grid())
lines(as.POSIXct(test.data$Date), pred.y.re, lwd=2, col="orange")
legend("topright", c("actual", "predicted"), lty=1, lwd=2, col=c("green","orange"))
```



```
#FITTING GRU MODEL
GRU.model <- keras_model_sequential()

#specifying model structure
GRU.model %>% layer_gru(input_shape=dim(train.x)[2:3], units=nsteps)
GRU.model %>% layer_dense(units=1, activation="tanh")
GRU.model %>% compile(loss="mean_squared_error")

#training model
epochs<- 5
for(i in 1:epochs){
    GRU.model %>% fit(train.x, train.y, batch_size=32, epochs=1)
    GRU.model %>% reset_states()
}

#predicting for testing data
pred.y<- GRU.model %>% predict(test.x, batch_size=32)

#rescaling pred.y back to the original scale
pred.y<- pred.y*(max(price)-min(price))+min(price)

#computing prediction accuracy
accuracy10<- ifelse(abs(test.y.re-pred.y.re)<0.10*test.y.re,1,0)
accuracy15<- ifelse(abs(test.y.re-pred.y.re)<0.15*test.y.re,1,0)
accuracy20<- ifelse(abs(test.y.re-pred.y.re)<0.20*test.y.re,1,0)

print(paste("accuracy within 10%:", round(mean(accuracy10),4)))


"accuracy within 10%: 0.7659"

print(paste("accuracy within 15%:", round(mean(accuracy15),4)))


"accuracy within 15%: 0.9164"

print(paste("accuracy within 20%:", round(mean(accuracy20),4)))


"accuracy within 20%: 0.99"

#plotting actual and predicted values for testing data
plot(as.POSIXct(test.data$Date), test.y.re, type="l", lwd=2, col="green", main="Daily Tesla Stock
Actual and Predicted Prices - GRU Model", xlab="Time", ylab="Stock Price", panel.first=grid())
```

```
lines(as.POSIXct(test.data$Date), pred.y.re, lwd=2, col="orange")
legend("topright", c("actual", "predicted"), lty=1, lwd=2, col=c("green","orange"))
```



In Python:

```
import numpy
import pandas
import matplotlib.pyplot as plt
from statistics import mean

from sklearn.metrics import mean_squared_error

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, GRU
from tensorflow.random import set_seed

tsla_data=pandas.read_csv('./TSLA.csv', index_col="Date", parse_dates=["Date"])

#plotting daily Tesla stock closing prices
time_start = 2010
time_end = 2021
tsla_data.loc[f"{time_start}":f"{time_end}", "Close"].plot(figsize=(16, 4), color="blue", legend=True)
tsla_data.loc[f"{time_end+1}":, "Close"].plot(figsize=(16, 4), color="green", legend=True)
plt.legend([f"Training set (Before {time_end+1})", f"Testing set ({time_end+1} and after)"])
plt.title("Daily Tesla Stock Closing Prices")
plt.ylabel("Stock Price")
plt.show()

#rescaling data
tsla_data["Close_sc"]=(tsla_data["Close"]-min(tsla_data["Close"]))/(max(tsla_data["Close"].values)-min(tsla_data["Close"]))
train_set=tsla_data.loc[f"{time_start}":f"{time_end}", "Close_sc"].values
test_set=tsla_data.loc[f"{time_end+1}":, "Close_sc"].values
```

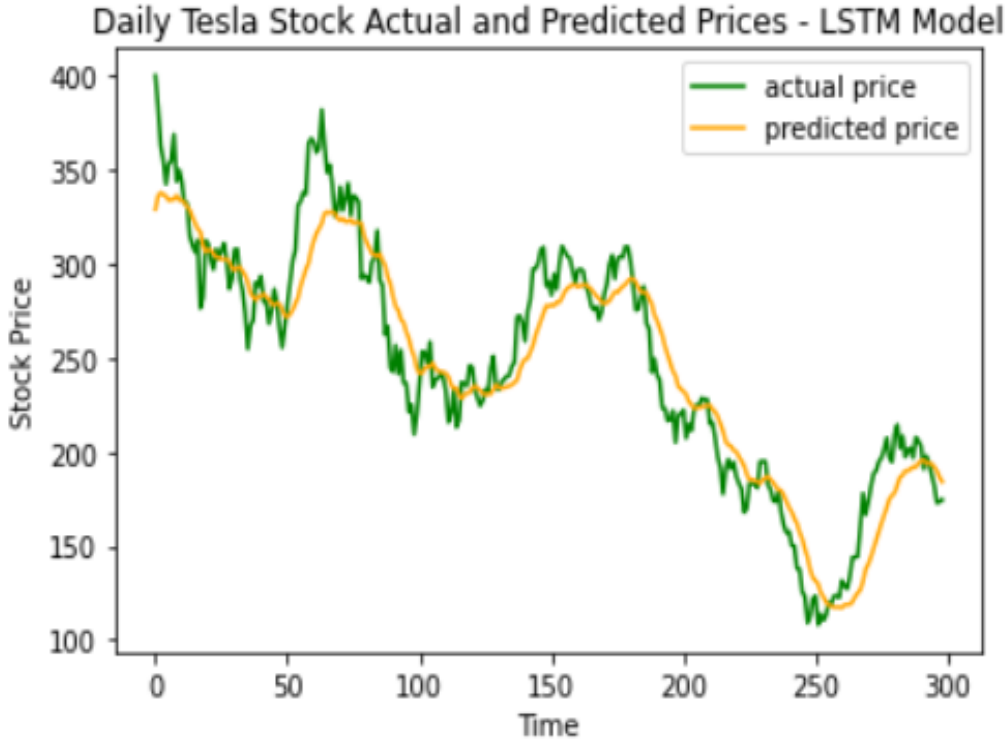
```
#splitting training data into samples
nsteps=60  #width of sliding window

def split_sequence(sequence, n_steps):
    x, y = list(), list()
    for i in range(len(sequence)):
        end_ix = i + n_steps
        if end_ix > len(sequence) - 1:
            break
        seq_x, seq_y = sequence[i:end_ix], sequence[end_ix]
        x.append(seq_x)
        y.append(seq_y)
    return numpy.array(x), numpy.array(y)

train_x, train_y=split_sequence(train_set, nsteps)

#################################################################
#FITTING LSTM MODEL
#################################################################
features=1 #predictors and response are the same variable
#reshaping train_x
train_x=train_x.reshape(train_x.shape[0], train_x.shape[1],features)

#specifying LSTM model architecture
model_lstm = Sequential()
model_lstm.add(LSTM(units=6, activation="tanh", input_shape=(nsteps, features)))
model_lstm.add(Dense(units=1))

#compiling the model
model_lstm.compile(loss="mse")
model_lstm.fit(train_x, train_y, epochs=5, batch_size=32)
```

```
#creating testing set by adding nsteps observations from training set to testing set
inputs=tsla_data.loc[:,"Close_sc"][len(tsla_data)-len(test_set)-nsteps:].values
inputs=inputs.reshape(-1, 1)

#splitting into samples
test_x, test_y=split_sequence(inputs, nsteps)

#reshaping
test_x=test_x.reshape(test_x.shape[0], test_x.shape[1], features)

#predicting for testing data
pred_y=model_lstm.predict(test_x)

#inverse transforming the values
pred_y=pred_y*(max(tsla_data["Close"].values)-min(tsla_data["Close"]))+min(tsla_data["Close"])
test_y=test_y*(max(tsla_data["Close"].values)-min(tsla_data["Close"]))+min(tsla_data["Close"])

#computing prediction accuracy
ind10=[]
ind15=[]
ind20=[]

for sub1, sub2 in zip(pred_y, test_y):
    ind10.append(1) if abs(sub1-sub2)<0.10*sub2 else ind10.append(0)
    ind15.append(1) if abs(sub1-sub2)<0.15*sub2 else ind15.append(0)
    ind20.append(1) if abs(sub1-sub2)<0.20*sub2 else ind20.append(0)

print('accuracy within 10%:', round(mean(ind10),4))
print('accuracy within 15%:', round(mean(ind15),4))
print('accuracy within 20%:', round(mean(ind20),4))
```

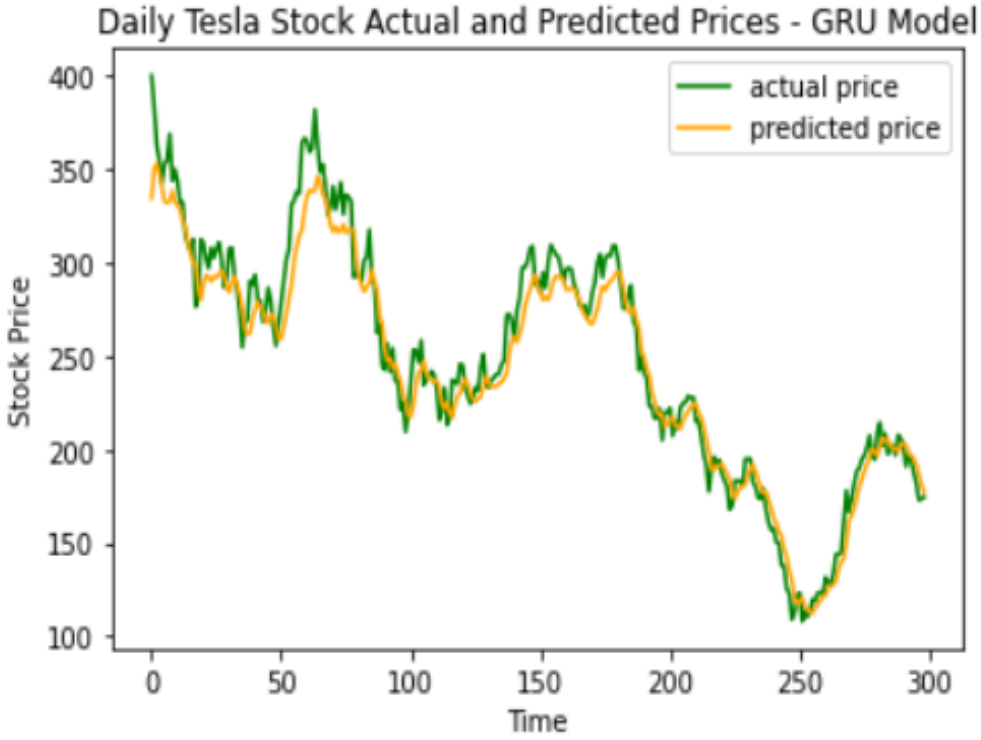
```
#plotting actual and predicted values for testing data
plt.plot(test_y, color="green", label="actual price")
plt.plot(pred_y, color="orange", label="predicted price")
plt.title("Daily Tesla Stock Actual and Predicted Prices - LSTM Model")
plt.xlabel("Time")
plt.ylabel("Stock Price")
plt.legend()
plt.show()

######################################################################
#FITTING GRU MODEL
######################################################################
#specifying GRU model architecture
model_gru = Sequential()
model_gru.add(GRU(units=6, activation="tanh", input_shape=(nsteps, features)))
model_gru.add(Dense(units=1))

# Compiling the model
model_gru.compile(loss="mse")
model_gru.fit(train_x, train_y, epochs=5, batch_size=32)

#predicting for testing data
pred_y=model_gru.predict(test_x)

#inverse transforming the values
pred_y=pred_y*(max(tsla_data["Close"].values)-min(tsla_data["Close"]))+min(tsla_data["Close"])

#computing prediction accuracy
ind10=[]
ind15=[]
ind20=[]
```

```
for sub1, sub2 in zip(pred_y, test_y):
    ind10.append(1) if abs(sub1-sub2)<0.10*sub2 else ind10.append(0)
    ind15.append(1) if abs(sub1-sub2)<0.15*sub2 else ind15.append(0)
    ind20.append(1) if abs(sub1-sub2)<0.20*sub2 else ind20.append(0)

print('accuracy within 10%:', round(mean(ind10),4))
print('accuracy within 15%:', round(mean(ind15),4))
print('accuracy within 20%:', round(mean(ind20),4))

#plotting actual and predicted values for testing data
plt.plot(test_y, color="green", label="actual price")
plt.plot(pred_y, color="orange", label="predicted price")
plt.title("Daily Tesla Stock Actual and Predicted Prices - GRU Model")
plt.xlabel("Time")
plt.ylabel("Stock Price")
plt.legend()
plt.show()
```

```
accuracy within 10%: 0.7391
accuracy within 15%: 0.8997
accuracy within 20%: 0.9732
```



```
accuracy within 10%: 0.9465
accuracy within 15%: 0.99
accuracy within 20%: 1
```



□

Example. We consider the data in file "TSLA_Shocks.csv". For the variable of interest, we took the daily stock volume and computed an indicator variable for **shock**, which we defined as the situation when the traded volume increased or decreased by more than 15% in one business day. The binary variable *Shock* contains 44.85% of ones. We fit LSTM and GRU networks for the binary response variable.

In R:

```
tsla.data<- read.csv(file="./TSLA_Shocks.csv", header=TRUE, sep=",")

#splitting data into testing and training sets
tsla.data$Year<- as.numeric(format(as.Date(tsla.data$Date, format="%Y-%m-%d"),"%Y"))
train.data<- tsla.data[which(tsla.data$Year<2022),1:2]
```

```
test.data<- tsla.data[which(tsla.data$Year>=2022),1:2]

nsteps<- 60 #width of sliding window
train.matrix <- matrix(nrow=nrow(train.data)-nsteps, ncol=nsteps+1)
for (i in 1:(nrow(train.data)-nsteps))
    train.matrix[i,]<- tsla.data$Shock[i:(i+nsteps)]

train.x<- array(train.matrix[,-ncol(train.matrix)],dim=c(nrow(train.matrix),nsteps,1))
train.y<- train.matrix[,ncol(train.matrix)]

#creating test.x and test.y
test.matrix<- matrix(nrow=nrow(test.data), ncol=nsteps+1)
for (i in 1:nrow(test.data))
test.matrix[i,]<- tsla.data$Shock[(i+nrow(train.matrix)):(i+nsteps+nrow(train.matrix))]

test.x<- array(test.matrix[,-ncol(test.matrix)],dim=c(nrow(test.matrix),nsteps,1))
test.y<- test.matrix[,ncol(test.matrix)]

#FITTING LSTM MODEL
library(keras)
#defining model architecture
LSTM.biclass<- keras_model_sequential()
LSTM.biclass %>% layer_dense(input_shape=dim(train.x)[2:3], units=nsteps)
LSTM.biclass %>% layer_lstm(units=25)
LSTM.biclass %>% layer_dense(units=1, activation="sigmoid")
LSTM.biclass %>% compile(loss="binary_crossentropy")

#training model
LSTM.biclass %>% fit(train.x, train.y, batch_size=32, epochs=5)

#computing prediction accuracy for testing data
pred.prob<- LSTM.biclass %>% predict(test.x)
match<- cbind(test.y, pred.prob)
tp<- matrix(NA, nrow=nrow(match), ncol=99)
tn<- matrix(NA, nrow=nrow(match), ncol=99)

for (i in 1:99) {
    tp[,i]<- ifelse(match[,1]==1 & match[,2]>0.01*i,1,0)
    tn[,i]<- ifelse(match[,1]==0 & match[,2]<=0.01*i,1,0)
}

trueclassrate<- matrix(NA, nrow=99, ncol=2)
```

```
for (i in 1:99){
    trueclassrate[i,1]<- 0.01*i
    trueclassrate[i,2]<- sum(tp[,i]+tn[,i])/nrow(match)
}

print(trueclassrate[which(trueclassrate[,2]==max(trueclassrate[,2])),])
```

```
   [,1]     [,2]
 [1,] 0.53 0.795302
 [2,] 0.54 0.795302
 [3,] 0.55 0.795302
      .   .   .
[45,] 0.97 0.795302
[46,] 0.98 0.795302
[47,] 0.99 0.795302
```

The prediction accuracy is 79.53% for any cut-off between 0.53 and 0.99.

```
#FITTING GRU MODEL
#defining model architecture
GRU.biclass<- keras_model_sequential()
GRU.biclass %>% layer_dense(input_shape=dim(train.x)[2:3], units=nsteps)
GRU.biclass %>% layer_lstm(units=25)
GRU.biclass %>% layer_dense(units=1, activation="sigmoid")
GRU.biclass %>% compile(loss="binary_crossentropy")

#training model
GRU.biclass %>% fit(train.x, train.y, batch_size=32, epochs=5)

#computing prediction accuracy for testing data
pred.prob<- GRU.biclass %>% predict(test.x)
match<- cbind(test.y, pred.prob)
tp<- matrix(NA, nrow=nrow(match), ncol=99)
tn<- matrix(NA, nrow=nrow(match), ncol=99)

for (i in 1:99) {
    tp[,i]<- ifelse(match[,1]==1 & match[,2]>0.01*i,1,0)
    tn[,i]<- ifelse(match[,1]==0 & match[,2]<=0.01*i,1,0)
}

trueclassrate<- matrix(NA, nrow=99, ncol=2)
for (i in 1:99) {
```

```
    trueclassrate[i,1]<- 0.01*i
    trueclassrate[i,2]<- sum(tp[,i]+tn[,i])/nrow(match)
}

print(trueclassrate[which(trueclassrate[,2]==max(trueclassrate[,2])),])
```

```
      [,1]     [,2]
 [1,] 0.52 0.795302
 [2,] 0.53 0.795302
 [3,] 0.54 0.795302
       .   .   .
[46,] 0.97 0.795302
[47,] 0.98 0.795302
[48,] 0.99 0.795302
```

The prediction accuracy is 79.53% for any cut-off between 0.52 and 0.99.

In Python:

```
import numpy
import pandas
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, GRU
from tensorflow.random import set_seed

tsla_data=pandas.read_csv('./TSLA_Shocks.csv', index_col="Date", parse_dates=["Date"])

#splitting into training and testing sets
time_start = 2010
time_end = 2021
def train_test_split(time_start, time_end):
    train=tsla_data.loc[f"{time_start}":f"{time_end}", "Shock"].values
    test=tsla_data.loc[f"{time_end+1}":, "Shock"].values
    return train, test

train_set, test_set = train_test_split(time_start, time_end)

#splitting training data into samples
nsteps = 60

def split_sequence(sequence):
    x, y = list(), list()
    for i in range(len(sequence)):
        end_i = i + nsteps
        if end_i > len(sequence)-1:
            break
        seq_x, seq_y=sequence[i:end_i], sequence[end_i]
        x.append(seq_x)
        y.append(seq_y)
    return numpy.array(x), numpy.array(y)

train_x, train_y = split_sequence(train_set)
```

```
#################################################################
#FITTING LSTM MODEL
#################################################################
#reshaping train_x
features = 1
train_x=train_x.reshape(train_x.shape[0],train_x.shape[1],features)

#specifying model architecture
model_lstm = Sequential()
model_lstm.add(LSTM(units=6, activation="sigmoid", input_shape=(nsteps, features)))
model_lstm.add(Dense(units=1))

# Compiling the model
model_lstm.compile(loss="binary_crossentropy")
model_lstm.fit(train_x, train_y, epochs=5, batch_size=32)
inputs = tsla_data.loc[:,"Shock"][len(tsla_data.loc[:,"Shock"])-len(test_set)-nsteps :].values

#splitting into samples
test_x, test_y = split_sequence(inputs)

#reshaping
test_x = test_x.reshape(test_x.shape[0], test_x.shape[1], features)

#predicting for testing data
pred_prob=model_lstm.predict(test_x)
```

```
cutoff=[]
accuracy=[]
for i in range(99):
    tp=0
    tn=0
    cutoff.append(0.01*(i+1))
    for sub1, sub2 in zip(pred_prob, test_y):
        tp_ind=1 if (sub1>0.01*(i+1) and sub2==1) else 0
        tn_ind=1 if (sub1<0.01*(i+1) and sub2==0) else 0
        tp+=tp_ind
        tn+=tn_ind
    accuracy_i=(tp+tn)/len(pred_prob)
    accuracy.append(accuracy_i)

df=pandas.DataFrame({'accuracy': accuracy,'cut-off': cutoff})
max_accuracy=max(accuracy)
optimal=df[df['accuracy']==max_accuracy]
print(optimal)
```

```
   accuracy  cut-off
47  0.795302     0.48
48  0.795302     0.49
49  0.795302     0.50
     .    .    .
96  0.795302     0.97
97  0.795302     0.98
98  0.795302     0.99
```

The prediction accuracy is 79.53% for any cut-off between 0.48 and 0.99.

```
#################################################################
#FITTING GRU MODEL
#################################################################
#specifying model architecture
model_gru = Sequential()
model_gru.add(GRU(units=6, activation="sigmoid", input_shape=(nsteps, features)))
model_gru.add(Dense(units=1))

# Compiling the model
model_gru.compile(loss="binary_crossentropy")
model_gru.fit(train_x, train_y, epochs=5, batch_size=32)

#predicting for testing data
pred_prob=model_gru.predict(test_x)

cutoff=[]
accuracy=[]
for i in range(99):
    tp=0
    tn=0
    cutoff.append(0.01*(i+1))
    for sub1, sub2 in zip(pred_prob, test_y):
        tp_ind=1 if (sub1>0.01*(i+1) and sub2==1) else 0
        tn_ind=1 if (sub1<0.01*(i+1) and sub2==0) else 0
        tp+=tp_ind
        tn+=tn_ind

    accuracy_i=(tp+tn)/len(pred_prob)
    accuracy.append(accuracy_i)

df=pandas.DataFrame({'accuracy': accuracy,'cut-off': cutoff})
max_accuracy=max(accuracy)
optimal=df[df['accuracy']==max_accuracy]
print(optimal)
```

```
    accuracy  cut-off
49  0.795302     0.50
50  0.795302     0.51
51  0.795302     0.52
     .    .    .
```

```
96  0.795302     0.97
97  0.795302     0.98
98  0.795302     0.99
```

The prediction accuracy is 79.53% for any cut-off between 0.48 and 0.99. □

Example. The file "LA_weather.csv" contains hourly weather conditions in LA (rain/fog/clear/cloudy) between 10/1/2012 12 PM and 11/30//2017 12 AM. There data were downloaded from kaggle.com and cleaned. We use this data set to fit RNN for multinomial classification in R and Python.

In R:

```
LA.weather<- read.csv(file="./LA_weather.csv", header=TRUE, sep=",")

LA.weather$Rain<- ifelse(LA.weather$Condition=="rain",1, 0)
LA.weather$Fog<- ifelse(LA.weather$Condition=="fog",1, 0)
LA.weather$Clear<- ifelse(LA.weather$Condition=="clear",1,0)
LA.weather$Cloudy<- ifelse(LA.weather$Condition=="cloudy",1,0)
LA.weather$Year<- format(as.Date(LA.weather$Date, format="%Y-%m-%d"),"%Y")

#DEFINING FUNCTION THAT FITS RNN MODEL

rnn.model<- function(modelname, varname) {

#creating train.x, train.y, test.x, and test.y sets
train.data<- LA.weather[which(LA.weather$Year<2017),varname]
test.data<- LA.weather[which(LA.weather$Year==2017),varname]

nsteps<- 60
train.matrix <- matrix(nrow=length(train.data)-nsteps, ncol=nsteps+1)
for (i in 1:(length(train.data)-nsteps))
    train.matrix[i,]<- LA.weather[i:(i+nsteps),varname]

train.x<- array(train.matrix[,-ncol(train.matrix)],dim=c(nrow(train.matrix),nsteps,1))
train.y<- train.matrix[,ncol(train.matrix)]

test.matrix<- matrix(nrow=length(test.data), ncol=nsteps+1)
for (i in 1:length(test.data))
    test.matrix[i,]<- LA.weather[(i+nrow(train.matrix)):(i+nsteps+nrow(train.matrix)),varname]
```

```
test.x<- array(test.matrix[,-ncol(test.matrix)],dim=c(nrow(test.matrix),nsteps,1))
test.y<- test.matrix[,ncol(test.matrix)]

#defining model architecture
library(keras)
fitted.model<- keras_model_sequential()
fitted.model %>% layer_dense(input_shape=dim(train.x)[2:3], units=nsteps)
if (modelname=='lstm') {
    fitted.model %>% layer_lstm(units=6)
} else fitted.model %>% layer_gru(units=6)
fitted.model %>% layer_dense(units=1, activation='sigmoid')
fitted.model %>% compile(loss='binary_crossentropy')

#training model
fitted.model %>% fit(train.x, train.y, batch_size=32, epochs=5)

#computing predicted probability of rain for testing data
pred.prob<- fitted.model %>% predict(test.x)
return(list(test.y, pred.prob))
}

#DEFINING FUNCTION THAT COMPUTES PREDICTION ACCURACY
library(dplyr)

accuracy<- function() {

test.y<- bind_cols(test.rain, test.fog, test.clear, test.cloudy)
colnames(test.y)<- 1:4
true.class<- as.numeric(apply(test.y, 1, function(x) colnames(test.y)[which.max(x)]))

pred.prob<- bind_cols(pred.prob.rain, pred.prob.fog, pred.prob.clear, pred.prob.cloudy)
colnames(pred.prob)<- 1:4
pred.class<- as.numeric(apply(pred.prob, 1, function(x) colnames(pred.prob)[which.max(x)]))

match<- c()
for (i in 1:length(pred.class)) {
    match[i]<- ifelse(pred.class[i]==true.class[i],1,0)
}
return(round(mean(match),4))
}

#RUNNING LSTM BINARY CLASSIFICATION MODELS
```

```
list.rain<- (rnn.model('lstm', 'Rain'))
test.rain<- list.rain[1]
pred.prob.rain<- list.rain[2]

list.fog<- rnn.model('lstm', 'Fog')
test.fog<- list.fog[1]
pred.prob.fog<- list.fog[2]

list.clear<- rnn.model('lstm', 'Clear')
test.clear<- list.clear[1]
pred.prob.clear<- list.clear[2]

list.cloudy<- rnn.model('lstm', 'Cloudy')
test.cloudy<- list.cloudy[1]
pred.prob.cloudy<- list.cloudy[2]
print(accuracy())
```

```
0.7772
```

```
#RUNNING GRU BINARY CLASSIFICATION MODELS
list.rain<- (rnn.model('gru', 'Rain'))
test.rain<- list.rain[1]
pred.prob.rain<- list.rain[2]

list.fog<- rnn.model('gru', 'Fog')
test.fog<- list.fog[1]
pred.prob.fog<- list.fog[2]

list.clear<- rnn.model('gru', 'Clear')
test.clear<- list.clear[1]
pred.prob.clear<- list.clear[2]

list.cloudy<- rnn.model('gru', 'Cloudy')
test.cloudy<- list.cloudy[1]
pred.prob.cloudy<- list.cloudy[2]
print(accuracy())
```

```
0.7802
```

In Python:

```
import numpy
import pandas
import matplotlib.pyplot as plt
from statistics import mean
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, GRU

LA_weather=pandas.read_csv('./LA_weather.csv', index_col="Date", parse_dates=["Date"])

#creating dummy variables
LA_weather=pandas.get_dummies(LA_weather['Condition'])


#FITTING LSTM MODEL

#########################################################
#Building Model for Rain
#########################################################
#defining training and testing sets
time_start = 2012
time_end = 2016
def train_test_split(time_start, time_end):
    train=LA_weather.loc[f"{time_start}":f"{time_end}", "rain"].values
    test=LA_weather.loc[f"{time_end+1}":, "rain"].values
    return train, test
train_set, test_set = train_test_split(time_start, time_end)

#splitting training data into samples
nsteps = 60
def split_sequence(sequence):
    x, y = list(), list()
    for i in range(len(sequence)):
        end_i = i + nsteps
        if end_i > len(sequence) - 1:
            break
        seq_x, seq_y = sequence[i:end_i], sequence[end_i]
        x.append(seq_x)
```

```
        y.append(seq_y)
    return numpy.array(x), numpy.array(y)

train_x, train_y = split_sequence(train_set)

#reshaping train_x
features = 1
train_x = train_x.reshape(train_x.shape[0],train_x.shape[1],features)

#specifying LSTM model architecture
fitted_model = Sequential()
fitted_model.add(LSTM(units=6, activation="sigmoid", input_shape=(nsteps, features)))
fitted_model.add(Dense(units=1, activation="sigmoid"))

#compiling model
fitted_model.compile(loss="binary_crossentropy")
fitted_model.fit(train_x, train_y, epochs=5, batch_size=32)
inputs=LA_weather.loc[:,"rain"][len(LA_weather.loc[:,"rain"]) - len(test_set) - nsteps : ].values

#splitting into samples
test_x, test_rain = split_sequence(inputs)

#reshaping
test_x = test_x.reshape(test_x.shape[0], test_x.shape[1], features)

#predicting for testing data
pred_prob_rain = fitted_model.predict(test_x)


#########################################################
#Building Model for Fog
#########################################################
#defining training and testing sets
def train_test_split(time_start, time_end):
    train=LA_weather.loc[f"{time_start}":f"{time_end}", "fog"].values
    test=LA_weather.loc[f"{time_end+1}":, "fog"].values
    return train, test
train_set, test_set = train_test_split(time_start, time_end)
```

```
#splitting training data into samples
def split_sequence(sequence):
    x, y = list(), list()
    for i in range(len(sequence)):
        end_i = i + nsteps
        if end_i > len(sequence) - 1:
            break
        seq_x, seq_y = sequence[i:end_i], sequence[end_i]
        x.append(seq_x)
        y.append(seq_y)
    return numpy.array(x), numpy.array(y)

train_x, train_y = split_sequence(train_set)

#reshaping train_x
train_x = train_x.reshape(train_x.shape[0],train_x.shape[1],features)

#specifying LSTM model architecture
fitted_model = Sequential()
fitted_model.add(LSTM(units=6,  activation="sigmoid", input_shape=(nsteps, features)))
fitted_model.add(Dense(units=1, activation="sigmoid"))

#compiling model
fitted_model.compile(loss="binary_crossentropy")
fitted_model.fit(train_x, train_y, epochs=5, batch_size=32)
inputs=LA_weather.loc[:,"fog"][len(LA_weather.loc[:,"fog"]) - len(test_set) - nsteps : ].values

#splitting into samples
test_x, test_fog = split_sequence(inputs)

#reshaping
test_x = test_x.reshape(test_x.shape[0], test_x.shape[1], features)

#predicting for testing data
pred_prob_fog = fitted_model.predict(test_x)
```

```
#########################################################
#Building Model for Clear
#########################################################
#defining training and testing sets
def train_test_split(time_start, time_end):
    train=LA_weather.loc[f"{time_start}":f"{time_end}", "clear"].values
    test=LA_weather.loc[f"{time_end+1}":, "clear"].values
    return train, test
train_set, test_set = train_test_split(time_start, time_end)


#splitting training data into samples
def split_sequence(sequence):
    x, y = list(), list()
    for i in range(len(sequence)):
        end_i = i + nsteps
        if end_i > len(sequence) - 1:
            break
        seq_x, seq_y = sequence[i:end_i], sequence[end_i]
        x.append(seq_x)
        y.append(seq_y)
    return numpy.array(x), numpy.array(y)

train_x, train_y = split_sequence(train_set)

#reshaping train_x
train_x = train_x.reshape(train_x.shape[0],train_x.shape[1],features)

#specifying LSTM model architecture
fitted_model = Sequential()
fitted_model.add(LSTM(units=6,  activation="sigmoid", input_shape=(nsteps, features)))
fitted_model.add(Dense(units=1, activation="sigmoid"))

#compiling model
fitted_model.compile(loss="binary_crossentropy")
fitted_model.fit(train_x, train_y, epochs=5, batch_size=32)
```

```
inputs=LA_weather.loc[:,"clear"][len(LA_weather.loc[:,"clear"]) - len(test_set) - nsteps : ].values

#splitting into samples
test_x, test_clear = split_sequence(inputs)

#reshaping
test_x = test_x.reshape(test_x.shape[0], test_x.shape[1], features)

#predicting for testing data
pred_prob_clear = fitted_model.predict(test_x)

#########################################################
#Building Model for Cloudy
#########################################################
#defining training and testing sets
def train_test_split(time_start, time_end):
    train=LA_weather.loc[f"{time_start}":f"{time_end}", "cloudy"].values
    test=LA_weather.loc[f"{time_end+1}":, "cloudy"].values
    return train, test
train_set, test_set = train_test_split(time_start, time_end)

#splitting training data into samples
def split_sequence(sequence):
    x, y = list(), list()
    for i in range(len(sequence)):
        end_i = i + nsteps
        if end_i > len(sequence) - 1:
            break
        seq_x, seq_y = sequence[i:end_i], sequence[end_i]
        x.append(seq_x)
        y.append(seq_y)
    return numpy.array(x), numpy.array(y)

train_x, train_y = split_sequence(train_set)
```

```
#reshaping train_x
train_x = train_x.reshape(train_x.shape[0],train_x.shape[1],features)

#specifying LSTM model architecture
fitted_model = Sequential()
fitted_model.add(LSTM(units=6,  activation="sigmoid", input_shape=(nsteps, features)))
fitted_model.add(Dense(units=1, activation="sigmoid"))

#compiling model
fitted_model.compile(loss="binary_crossentropy")
fitted_model.fit(train_x, train_y, epochs=5, batch_size=32)
inputs=LA_weather.loc[:,"cloudy"][len(LA_weather.loc[:,"cloudy"]) - len(test_set) - nsteps : ].values

#splitting into samples
test_x, test_cloudy = split_sequence(inputs)

#reshaping
test_x = test_x.reshape(test_x.shape[0], test_x.shape[1], features)

#predicting for testing data
pred_prob_cloudy = fitted_model.predict(test_x)

#####################################################
#Computing Prediction Accuracy
#####################################################
pred_prob_all=numpy.concatenate((pred_prob_rain, pred_prob_fog,pred_prob_clear,pred_prob_cloudy), axis=1)
pred_prob_all=pandas.DataFrame(pred_prob_all)
pred_class=pred_prob_all.idxmax(axis=1)

test_all=numpy.c_[test_rain,test_fog, test_clear, test_cloudy]
test_all=pandas.DataFrame(test_all)
true_class=test_all.idxmax(axis=1)
```

```
match=[]
for i in range(len(pred_class)):
    if pred_class[i]==true_class[i]:
            match.append(1)
    else:
            match.append(0)

print(round(mean(match),4))
```

```
0.7533
```

We fit a GRU model by running the code identical to the one for LSTM above, but with 'LSTM' replaced by 'GRU'. The prediction accuracy for this model is

```
0.7801
```

□

CHANGE-POINT DETECTION

Consider a time series data set consisting of n normally distributed observations. And suppose that the first segment of k observations has a $N(\mu_1, \sigma^2)$ distribution whereas the remaining segment of $n-k$ observations has a $N(\mu_2, \sigma^2)$ distribution where $\mu_1 \neq \mu_2$. That is, a change in mean occurs at some unknown step k. The **change-point detection** is a collection of methods to identify the value of k.

The change-point detection problem is also applicable to finding k when the mean doesn't change but the variance does, or when both mean and variance change. It also extends to the case of several segments with different means, variances, or both.

There are many well-developed methods to identify the point(s) of change. We will present the theory for the most basic approach.

The method of binary segmentation is often used to detect the change points. First, one change point is detected in the complete set of observations, then the series is split around this change point, and the algorithm is applied to the two resulting segments. The process continues until a pre-specified number of splits is detected.

To identify the value of k where the change occurs, the method of maximum likelihood estimation is employed. We assume that $y_1, \ldots, y_k \sim N(\mu_1, \sigma^2)$, and $y_{k+1}, \ldots, y_n \sim N(\mu_2, \sigma^2)$. The maximum likelihood estimators of μ_1, μ_2, and σ^2 are

$$\hat{\mu}_1 = \bar{y}_1 = \frac{1}{k}\sum_{i=1}^{k} y_i, \quad \hat{\mu}_2 = \bar{y}_2 = \frac{1}{n-k}\sum_{i=k+1}^{n} y_i,$$

and

$$\hat{\sigma}^2 = \frac{1}{n-1}\sum_{i=1}^{n} (y_i - \bar{y})^2, \text{ where } \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i.$$

The likelihood function for these data has the form:

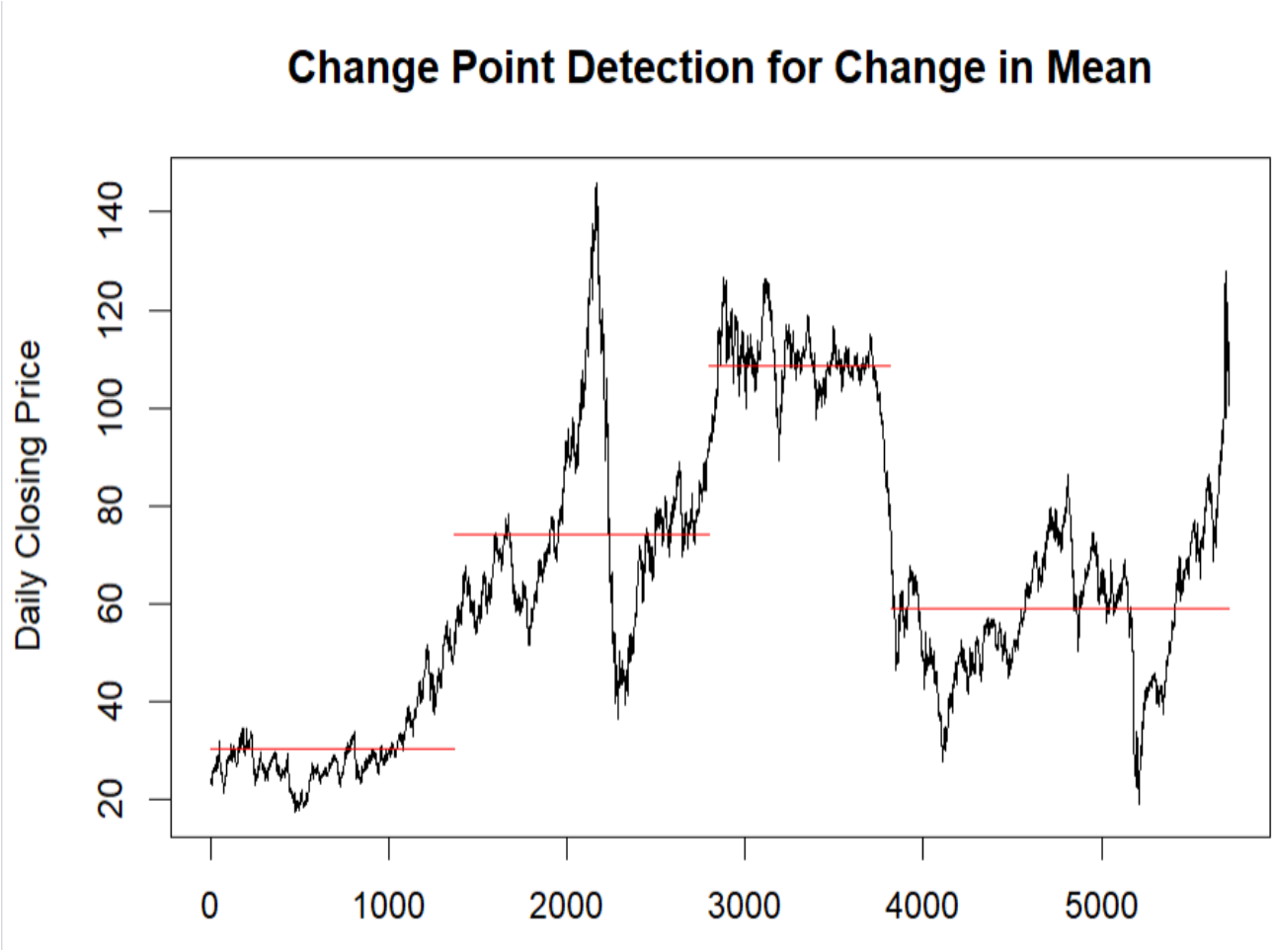
$$L = L(\hat{\mu}_1, \hat{\mu}_2, \hat{\sigma}^2 \,|\, y_1, \ldots, y_n) = (2\pi\,\hat{\sigma}^2)^{-n/2} \exp\Big\{ - \frac{\sum_{i=1}^{k} (y_i - \bar{y}_1)^2 + \sum_{i=k+1}^{n} (y_i - \bar{y}_2)^2}{2\,\hat{\sigma}^2} \Big\}.$$

The value of k that maximizes the likelihood function is the optimal one.

To apply the change-point detection to real-life data, we will use the library "changepoint" in R, and utilize functions cpt.mean(), cpt.var(), and cpt.meanvar() with options `method="BinSeg"` (binary segmentation), `Q=` (the number of splits k), and `penalty="AIC"`. Here AIC stands for Akaike Information Criterion which dictates choosing k that minimizes $AIC = -2\ln L + p\ln n$ where p is the number of parameters that have to be estimated from the data (in our example, we estimated μ_1, μ_2 and σ^2, so $p = 3$).

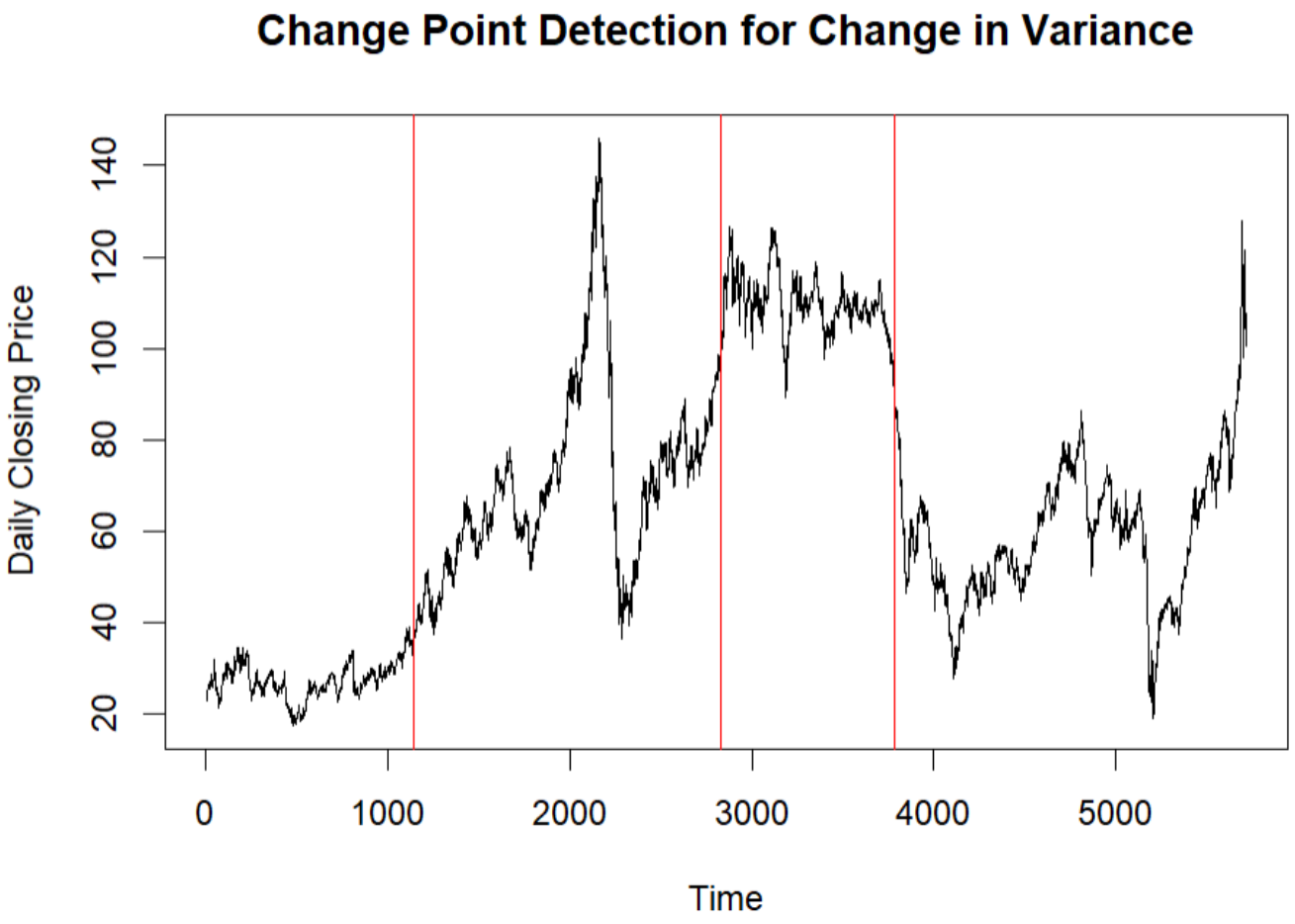
Example. The file "crudeoil_data.csv" contains daily closing prices of (Brent) crude oil between 2000 and 2022. These data were extracted from the file "commodity 2000-2022.csv" downloaded from kaggle.com. We apply change-point methods to identify changes in mean, changes in variance, and simultaneous changes in mean and variance.

```
#application to crude oil's closing price data
crudeoil.data<- read.csv(file="./crudeoil_data.csv", header=TRUE, sep=",")

library(changepoint)
ansmean=cpt.mean(crudeoil.data$Close, penalty="AIC", method="BinSeg", Q=3)
plot(ansmean, cpt.col="red", ylab="Daily Closing Price", main="Change Point Detection for
Change in Mean")
print(ansmean)
```



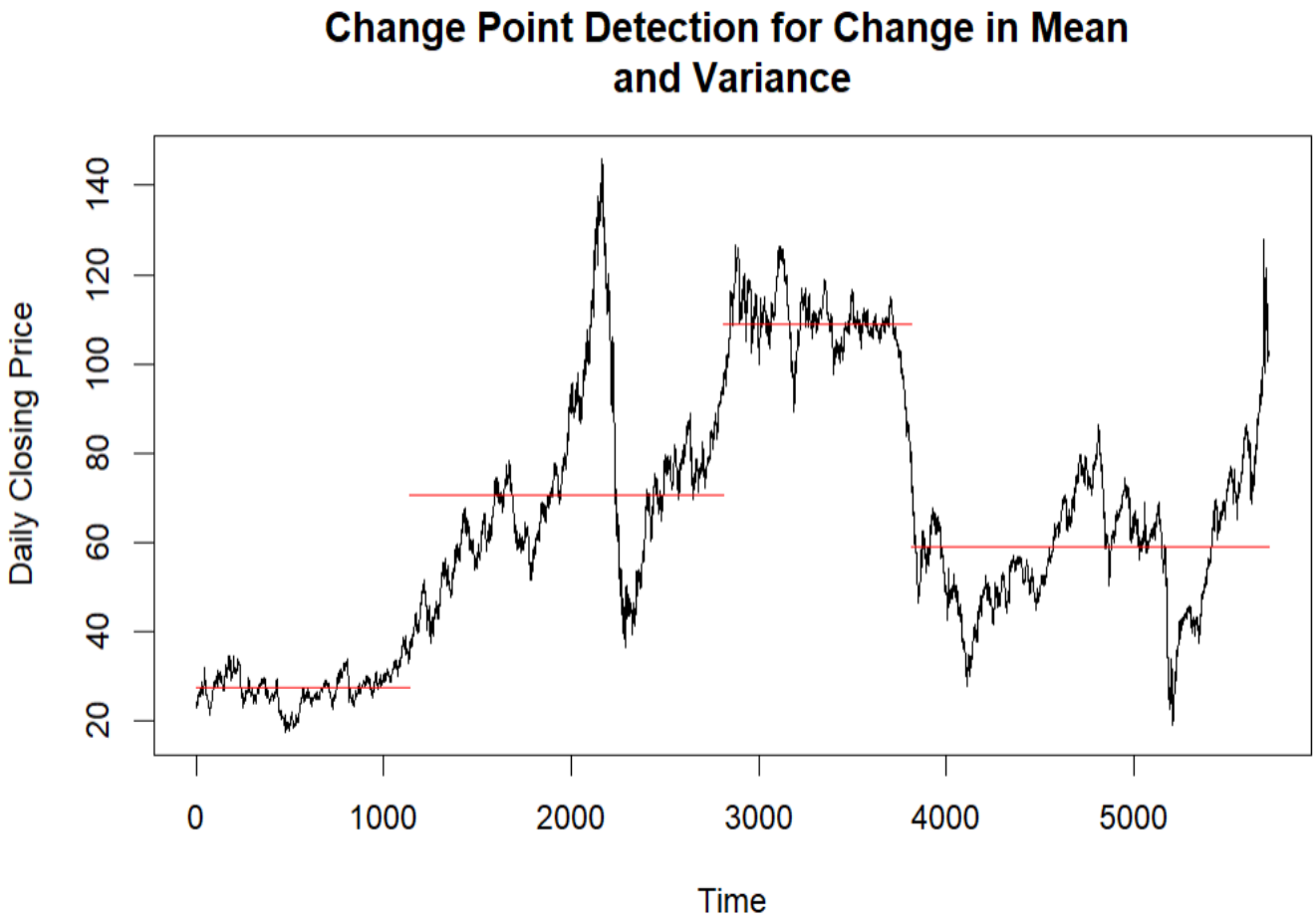

```
Changepoint Locations : 1368 2794 3816
```

```
ansvar=cpt.var(crudeoil.data$Close, penalty="AIC", method="BinSeg", Q=3)
plot(ansvar, cpt.col="red", ylab="Daily Closing Price", main="Change Point Detection for Change
in Variance")
print(ansvar)
```




```
Changepoint Locations : 1144 2827 3784
```

```
ansmeanvar=cpt.meanvar(crudeoil.data$Close, penalty="AIC", method="BinSeq", Q=3)
plot(ansmeanvar, cpt.col="red", ylab="Daily Closing Price", main="Change Point Detection for
Change in Mean and Variance")
print(ansmeanvar)
```




```
Changepoint Locations : 1140 2812 3816
```

□

ANOMALY DETECTION

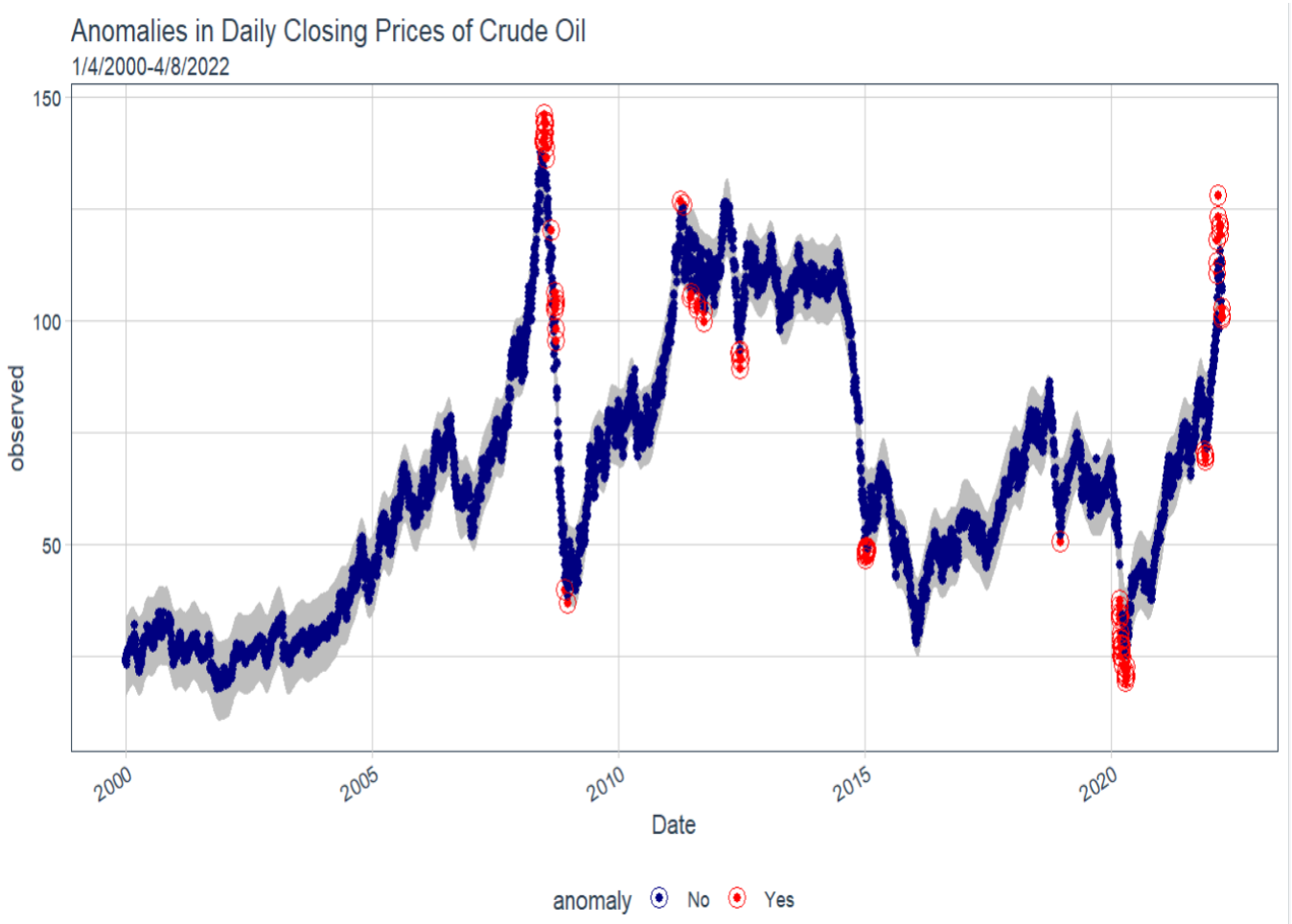
Anomalies of a time series data can be defined as outliers of the **remainders** once the linear trend and seasonal periodicity are taken into account.

We use library "anomalize" in R to identify anomalies. We call the function `time_decompose()` with the option `method="stl"` (factoring the linear and seasonal components), and the function `anomalize()` with the option `method="iqr"`. By default, this method defines outliers as observations lying below $Q1-3\cdot IQR$ or above $Q3+3\cdot IQR$, where $Q1$ is the first quartile (25th percentile),

$Q3$ is the third quartile (75th percentile), and the interquartile range is $IQR = Q3 - Q1$.
The default setting can be changed by specifying a value for alpha other than 0.05 (option "alpha="). Outliers are defined as values that lie $0.15/\alpha \times IQR$ distance away from the quartiles. The default value of $\alpha = 0.05$, thus resulting in the multiplicative constant of 3. If alpha is increased, more observations become outliers. If alpha is decreased, fewer observations are labeled as outliers.

Example. We use the crude oil data from the previous example to detect and plot anomalies in the daily closing prices.

```
crudeoil.data<- read.csv(file="./crudeoil_data.csv", header=TRUE, sep=",")

crudeoil.data$Date<- as.Date(crudeoil.data$Date, format="%Y-%m-%d")

library(tibbletime) #creates indices for date in time series data
crudeoil.data_tbl <- as_tbl_time(crudeoil.data, Date)

library(anomalize)
library(tidyverse)
crudeoil.data_tbl %>% time_decompose(Close, method="stl") %>% anomalize(remainder, method="iqr")
%>% time_recompose() %>% plot_anomalies(time_recomposed=TRUE, color_no='navy', color_yes='red',
fill_ribbon='gray', size_circles=4) + labs(title="Anomalies in Daily Closing Prices of Crude Oil",
subtitle="1/4/2000-4/8/2022")
```



NATURAL LANGUAGE PROCESSING (TEXT MINING)

Natural Language Processing (NLP) is a collection of techniques that allows working with and analyzing strings of words. Some most common applications are summarizing large volumes of text (e.g., computing word frequencies for different authors) and categorizing sentences (e.g., classifying news headlines as negative/neutral/positive, or classifying customer complaints by issues addressed). Below we consider two examples.

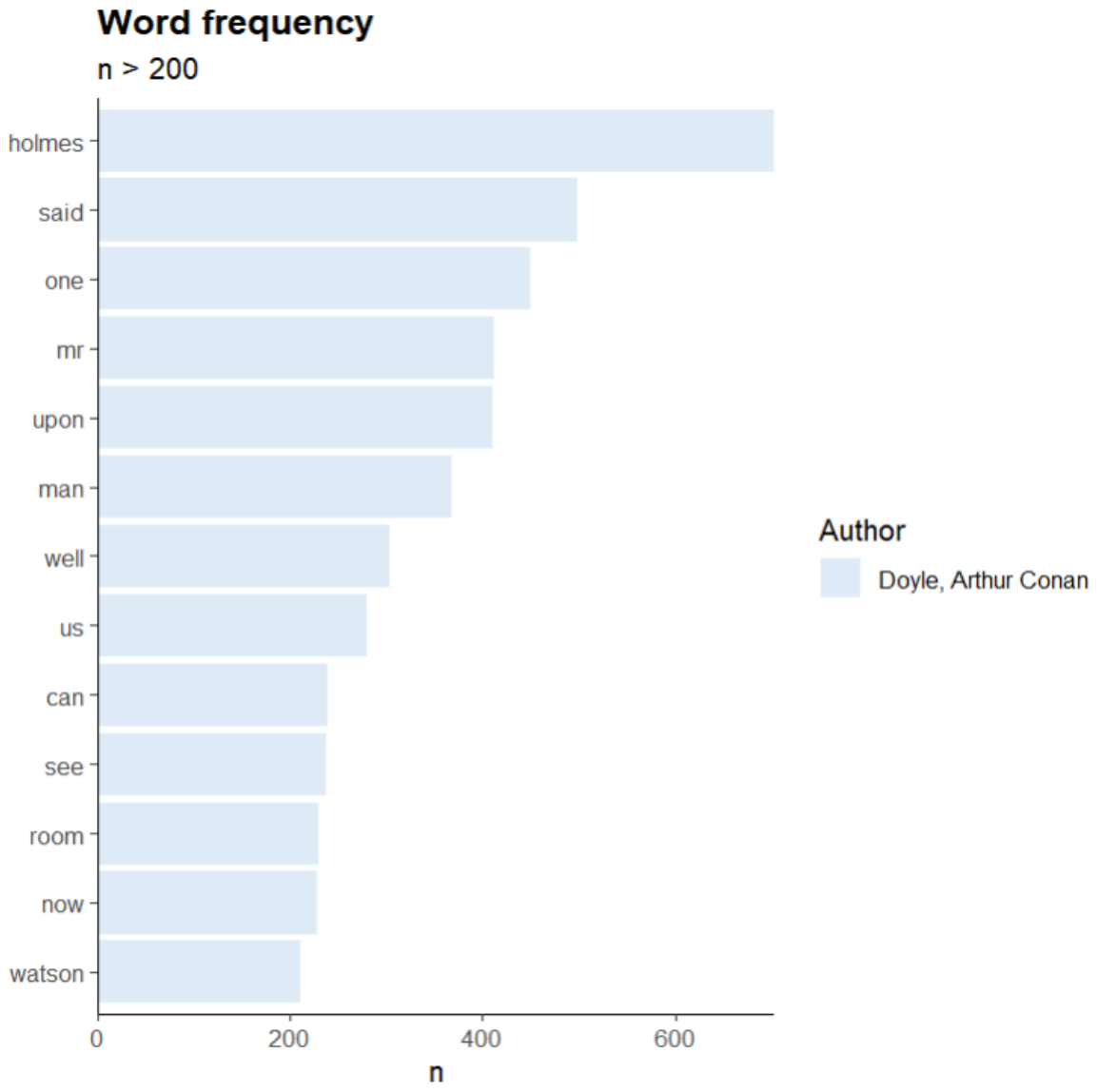
Example. The R code below downloads a digital book from the Project Gutenberg collection, divides the text into words, removing all articles, prepositions, etc. (called "stopwords"), computes frequency distribution of words, and visualizes the results by plotting bar graph and word cloud.

```
#install.packages(c("gutenbergr", "stringr", "dplyr", "tidytext", "stopwords", "tibble", "ggraph",
"wordcloud"))

library(gutenbergr)
library(stringr)
library(dplyr)
library(tidytext)
library(stopwords)
library(tibble)
library(ggplot2)
library(wordcloud)

book<- gutenberg_download(108, meta_fields="author")

#puts text into tibble format
book<- as_tibble(book) %>% mutate(document=row_number())
%>% select(-gutenberg_id)

#creates tokens (words)
#tokenization is the process of splitting text into tokens
tidy_book <- book %>% unnest_tokens(word, text) %>%
group_by(word) %>% filter(n() > 10) %>% ungroup()

#identifying and removing stopwords (prepositions, articles)
stopword<- as_tibble(stopwords::stopwords("en"))
stopword<- rename(stopword, word=value)
tb <- anti_join(tidy_book, stopword, by="word")

#calculating word frequency
word_count<- count(tb, word, sort=TRUE)
print(word_count, n=15)
```

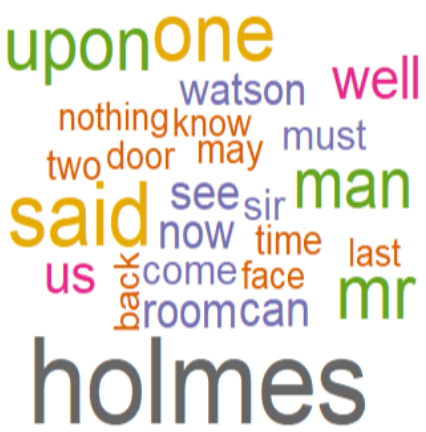
```
  word        n
1 holmes    703
2 said      499
3 one       449
4 mr        412
5 upon      411
6 man       367
7 well      303
8 us        279
9 can       238
```

```
10 see       237
11 room      229
12 now       227
13 watson    210
14 come      189
15 sir       188
```

```
#plotting bar graph
tb %>% count(author, word, sort=TRUE) %>%
filter(n > 200) %>% mutate(word=reorder(word, n)) %>%
ggplot(aes(word, n)) + geom_col(aes(fill=author)) + xlab(NULL)
+ scale_y_continuous(expand=c(0, 0)) + coord_flip() +
theme_classic(base_size = 12) + labs(fill="Author", title="Word frequency",
subtitle="n > 200") + theme(plot.title=element_text(lineheight=.8, face="bold")) + scale_fill_brewer()
```




```
#plotting word cloud
tb %>% count(word) %>% with(wordcloud(word, n, max.words=25, colors=brewer.pal(8, "Dark2")))
#brewer.pal(n,name) = color palette, n=# of colors, name=c("Accent", "Dark2", "Paired"
#"Pastel1", "Pastel2", "Set1", "Set2", "Set3")
```



Example. The data set "FinancialNewsHeadlines.csv" contains the sentiments for financial news headlines from the perspective of an investor. It was downloaded from Kaggle (https://www.kaggle.com/datasets/ankurzing/sentiment-analysis-for-financial-news). The data set contains two columns, "Sentiment" and "News Headline". The sentiment can be negative, neutral, or positive. We conduct a **sentiment analysis** on these data by training a **Bidirectional Encoder Representations from Transformers (BERT)** model. This methodology was introduced in 2018 by researchers at Google. BERT learns information from a text from the left and right side of each word during training and consequently gains a deeper understanding of the context. We compute the accuracy of prediction and test the model with a few sentences of our own.

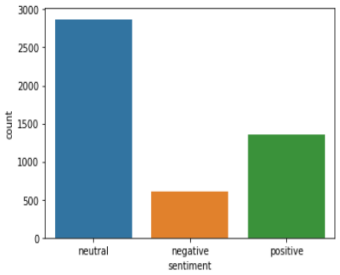
```
#!pip install wordcloud
import pandas
import numpy
import seaborn
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from wordcloud import WordCloud

data=pandas.read_csv("./FinancialNewsHeadlines.csv",encoding='ISO-8859-1')
data=data.rename(columns={'neutral':'sentiment','According to Gran , the company has no plans to move all production to Russ
data.drop_duplicates(subset=['statement'],keep='first',inplace=True)
text = " ".join([x for x in data.statement])

#plotting wordclouds for all news
wordcloud = WordCloud(background_color='white').generate(text)

plt.figure(figsize=(8,6))
plt.imshow(wordcloud,interpolation='bilinear')
plt.axis('off')
plt.show()
```

```
#plotting wordclouds for neutral news

text = " ".join([x for x in data.statement[data.sentiment=='neutral']])

wordcloud = WordCloud(background_color='white').generate(text)

plt.figure(figsize=(8,6))
plt.imshow(wordcloud,interpolation='bilinear')
plt.axis('off')
plt.show()
```


```
#plotting wordclouds for positive news

text = " ".join([x for x in data.statement[data.sentiment=='positive']])

wordcloud = WordCloud(background_color='white').generate(text)

plt.figure(figsize=(8,6))
plt.imshow(wordcloud,interpolation='bilinear')
plt.axis('off')
plt.show()
```


```
#plotting wordclouds for negative news

text = " ".join([x for x in data.statement[data.sentiment=='negative']])

wordcloud = WordCloud(background_color='white').generate(text)

plt.figure(figsize=(8,6))
plt.imshow(wordcloud,interpolation='bilinear')
plt.axis('off')
plt.show()
```


```
#plotting bar graph for sentiments
seaborn.countplot(data.sentiment)
```



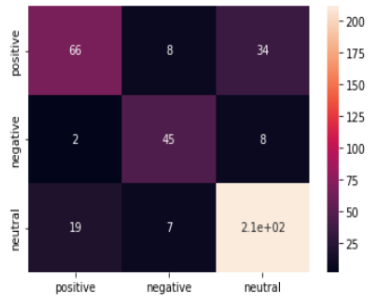

```
#displaying frequency by sentiment
data['sentiment'].value_counts()
```

```
neutral     2871
positive    1362
negative     604
```

```
#training model
numpy.random.seed(5677934)
train, test = train_test_split(data,test_size = 0.2)

#!pip install simpletransformers
#!pip install torch

from simpletransformers.classification import ClassificationModel

# Create a TransformerModel
model = ClassificationModel('bert', 'bert-base-cased', num_labels=3,
args={'reprocess_input_data': True, 'overwrite_output_dir': True},use_cuda=False)

def making_label(st):
    if(st=='positive'):
        return 0
    elif(st=='neutral'):
        return 2
    else:
        return 1

train['label']=train['sentiment'].apply(making_label)
test['label']=test['sentiment'].apply(making_label)

train_df = pandas.DataFrame({
    'text': train['statement'][:1500].replace(r'\n', ' ', regex=True),
    'label': train['label'][:1500]
})

eval_df = pandas.DataFrame({
    'text': test['statement'][-400:].replace(r'\n', ' ', regex=True),
    'label': test['label'][-400:]
})

model.train_model(train_df)
```

```
#computing predicted sentiments for testing set
result, model_outputs, wrong_predictions = model.eval_model(eval_df)

lst = []
for arr in model_outputs:
    lst.append(numpy.argmax(arr))

true = eval_df['label'].tolist()
predicted = lst

#displaying confusion matrix (positive/negative/neutral)
import sklearn
confmatrix = sklearn.metrics.confusion_matrix(true, predicted)
print(confmatrix)

#displaying heatmap for confusion matrix
df_cm = pandas.DataFrame(confmatrix, ['positive','negative','neutral'], ['positive','negative','neutral'])

seaborn.heatmap(df_cm, annot=True)
plt.show()
```

```
[[ 66   8  34]
 [  2  45   8]
 [ 19   7 211]]
```




```
#displaying performance metrics
sklearn.metrics.classification_report(true,predicted,target_names=['positive','negative','neutral'])
```

```
'              precision    recall  f1-score   support\n\n    positive       0.76      0.61      0.68       108\n    negative
0.75      0.82      0.78        55\n     neutral       0.83      0.89      0.86       237\n\n    accuracy
0.81       400\n   macro avg       0.78      0.77      0.77       400\nweighted avg       0.80      0.81      0.80       400\n'
```

```
#computing predicted accuracy
sklearn.metrics.accuracy_score(true,predicted)
```

```
0.805
```

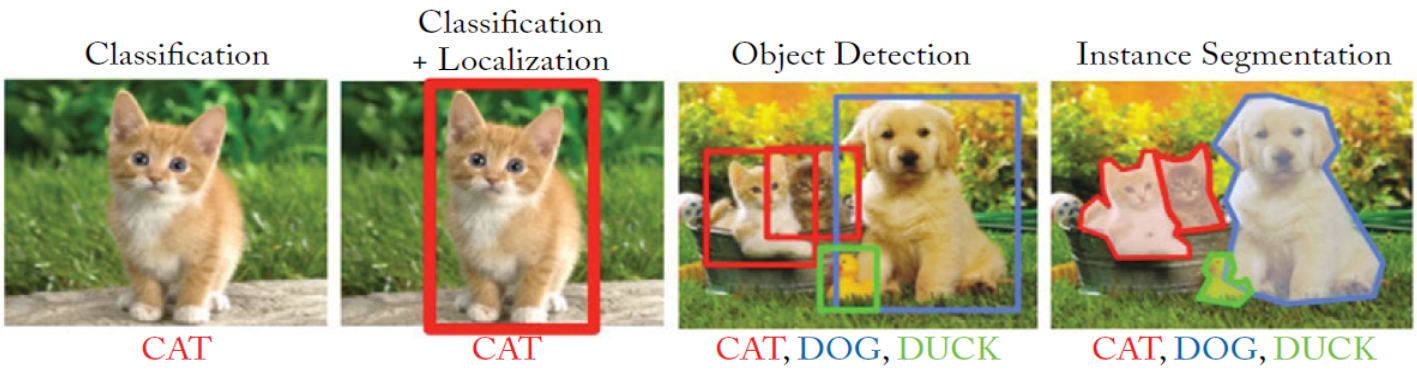
```
#using the trained model to classify user-defined sentences
def classify(statement):
        result = model.predict([statement])
        pred_class = numpy.where(result[1][0] == numpy.amax(result[1][0]))
        pred_class = int(pred_class[0])
        sentiment_dict = {0:'positive',1:'negative',2:'neutral'}
        print(sentiment_dict[pred_class])
        return

classify('People keep money in a bank.')
classify('S&P rose 1000 points in one day.')
classify('Inflation is going down now.')
```

```
neutral
positive
negative
```

Note that the trained BERT model has 80.5% accuracy and that last statement is misclassified. □

CONVOLUTIONAL NEURAL NETWORK

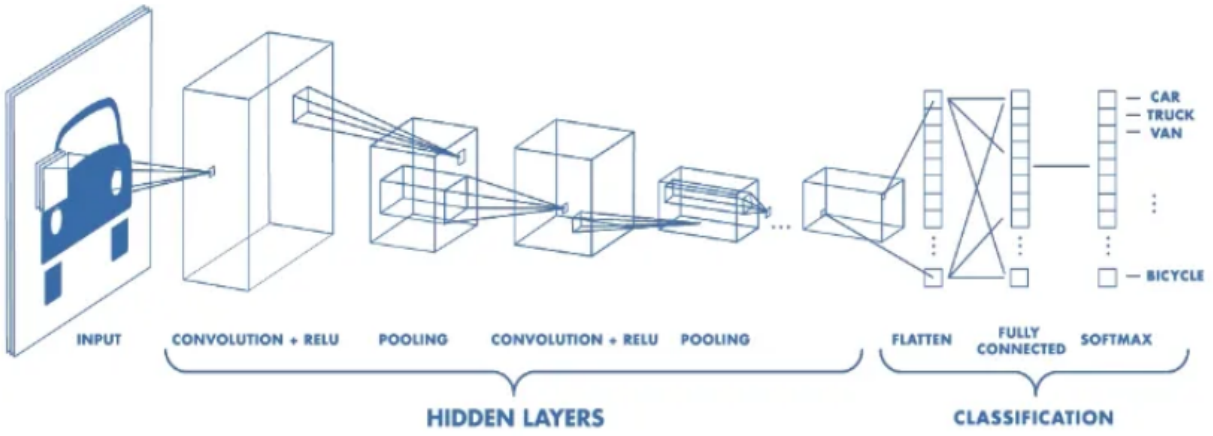
A **convolutional neural network** (CNN) is one of the most popular types of deep learning algorithms. It works well on images.



A CNN is an excellent tool for (i) image classification (categorization of image: cat/dog/horse), (ii) classification and localization of an object in the image (drawing a **bounding box** around an

object and naming the class); (iii) object detection (localization and labeling of all objects present in the image); and (iv) instance segmentation (segmenting individual objects present in the picture).

Historical Note. The earliest form of CNN was the Neocognitron model proposed by Kunihiko Fukushima (Fukushima and Miyake, 1982). The Neocognitron was motivated by the seminal work by David Hubel and Torsten Wiesel (1959) which demonstrated that the neurons in the brain are organized in the form of layers. These layers learn to recognize visual patterns by first extracting local features and subsequently combining them to obtain a higher-level representations.



The architecture of CNN comprises several layers: a **convolution** layer, a **pooling** layer, and a **fully connected** layer.

Convolution Layer

An image of size $r \times c$ pixels is represented by three matrices of size $r \times c$ each, containing the intensity values of red, green, and blue primary colors (numbers between 0 and 255) on the RGB scale. The convolution layer performs a **dot product** between two matrices, where one matrix is the set of **learnable parameters** (otherwise known as a **kernel** or **filter** or **feature detector**), and the other matrix is the restricted portion of the image.

Example. Mathematically, the kernel is a matrix of weights. For example, the following 3×3 kernel detects vertical lines.

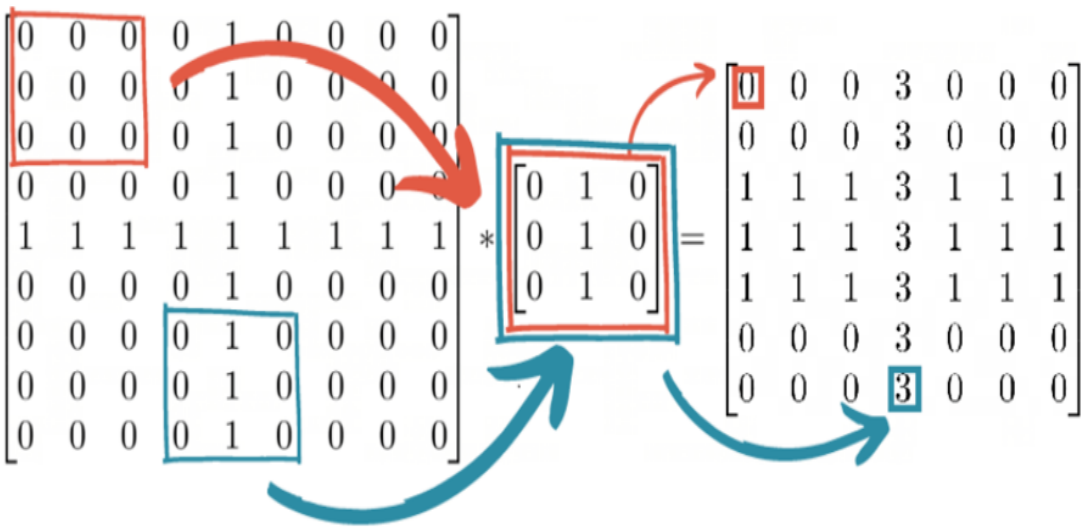
$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Suppose we have an 9×9 input image of a plus sign. This has two kinds of lines, horizontal and vertical, and a crossover. In matrix format, the image would look as follows:

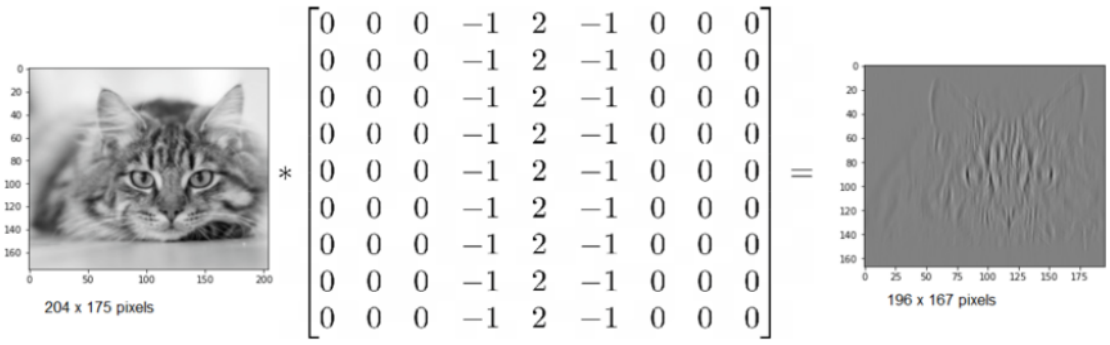
$$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Suppose we want to test the vertical line detector kernel on the plus sign image. To perform the convolution, we slide the convolution kernel over the image. At each position, we multiply each element of the convolution kernel by the element of the image that it covers and sum the results.

Since the kernel has width 3, it can only be positioned at 7 different positions horizontally in an image of width 9. So the end result of the convolution operation on an image of size 9×9 with a 3×3 convolution kernel is a new image of size 7×7.



To see how it works on an image, we consider a grayscaled image of a tabby cat with dimensions 204×175 pixels, which we can be represented by a matrix with values in the range between 0 and 1, where 1 is white and 0 is black.



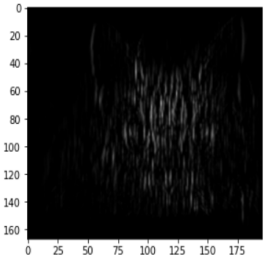
Applying the convolution with a sophisticated vertical line detector, a 9×9 convolution kernel, we see that the filter has performed a kind of vertical line detection. The vertical stripes on the tabby cat's head are highlighted in the output. The output image is 8 pixels smaller in both dimensions due to the size of the kernel (9×9).

A convolution layer is made up of a series of convolution kernels: a vertical line detector, a horizontal line detector, and various diagonal, border, curve, and corner detectors. These feature detector kernels serve as the first stage of the image recognition process.

Later layers in the neural network can build on the features detected by earlier layers and identify ever more complex shapes.

Activation Function

After passing an image through a convolutional layer, the output is normally passed through an activation function. A common activation function is called **rectified linear unit** (ReLU) defined as $f(x) = \max(0, x)$. The activation function has the effect of adding non-linearity into the convolutional neural network. If the activation function was not present, all the layers of the neural network could be condensed down to a single matrix multiplication. In the case of the cat image above, applying a ReLU function to the first layer output results in a stronger contrast highlighting the vertical lines, and removing the noise originating from other non-vertical features.



Repeating Structure of a CNN

A CNN can be viewed as a series of convolutional layers, followed by an activation function, followed by a **pooling** (**downscaling**) layer, repeated many times.

With the repeated combination of these operations, the first layer detects simple features such as edges in an image, and the second layer begins to detect higher-level features. By the tenth layer, a convolutional neural network can detect more complex shapes such as eyes. By the twentieth layer, it is often able to differentiate faces from one another.

This power comes from the repeated layering of operations, each of which can detect slightly higher-order features than its predecessor.

Once the image is processed through the convolution and pooling layers, it goes into the classification stage consisting of a flatten layer, a connected layer, and a softmax layer. The **flatten** layer collapses the spatial dimensions of the input into a single channel dimension. **Fully connected** layers connect every neuron in one layer to every neuron in another layer. The flattened matrix goes through a fully connected layer to classify the images.

Softmax Function

The **softmax function** is a normalized exponential function that takes as input a vector $(z_1, \ldots, z_K)$ of K real numbers, and normalizes it into a probability distribution consisting of K probabilities proportional to the exponentials of the input numbers. That is, before applying softmax, some vector components could be negative, or greater than one, and might not sum to 1, but after applying softmax, each component will be in the interval $(0, 1)$, and the components will add up to 1, so that they can be interpreted as probabilities. We have

$$softmax(z_1, \ldots, z_K) = \Big(\frac{e^{z_1}}{\sum_{j=1}^{K} e^{z_j}}, \ldots, \frac{e^{z_K}}{\sum_{j=1}^{K} e^{z_j}}\Big).$$

Applications of Convolutional Neural Networks

Several companies, such as Tesla and Uber, are using convolutional neural networks as the computer vision component of a self-driving car. A self-driving car's computer vision system must be capable of localization, obstacle avoidance, and path planning.

Say, consider the case of pedestrian detection. A pedestrian is a kind of obstacle that moves. A convolutional neural network must be able to identify the location of the pedestrian and extrapolate their current motion to calculate if a collision is imminent.

A convolutional neural network for object detection is slightly more complex than a classification model, in that it must not only classify an object but also return the four coordinates of its bounding box.

Furthermore, the convolutional neural network designer must avoid unnecessary false alarms for irrelevant objects, such as litter, but also take into account the high cost of miscategorizing a true pedestrian and causing a fatal accident.

A major challenge for this kind of use is collecting labeled training data. Google's Captcha system is used for authenticating websites, where a user is asked to categorize images as fire hydrants, traffic lights, cars, etc. This is actually a useful way to collect labeled training images for purposes such as self-driving cars and Google StreetView.

Another famous application of CNN is that of drug discovery. The first stage of a drug development program is drug discovery, where a pharmaceutical company identifies candidate compounds that are more likely to interact with the body in a certain way. Testing candidate molecules in pre-clinical or clinical trials is expensive, so it is advantageous to be able to screen molecules as early as possible.

Proteins that play an important role in disease are known as 'targets'. Some targets can cause inflammation or help tumors grow. The goal of drug discovery is to identify molecules that will interact with the target for a particular disease. The drug molecule must have the appropriate

shape to interact with the target and bind to it, like a key fitting in a lock.

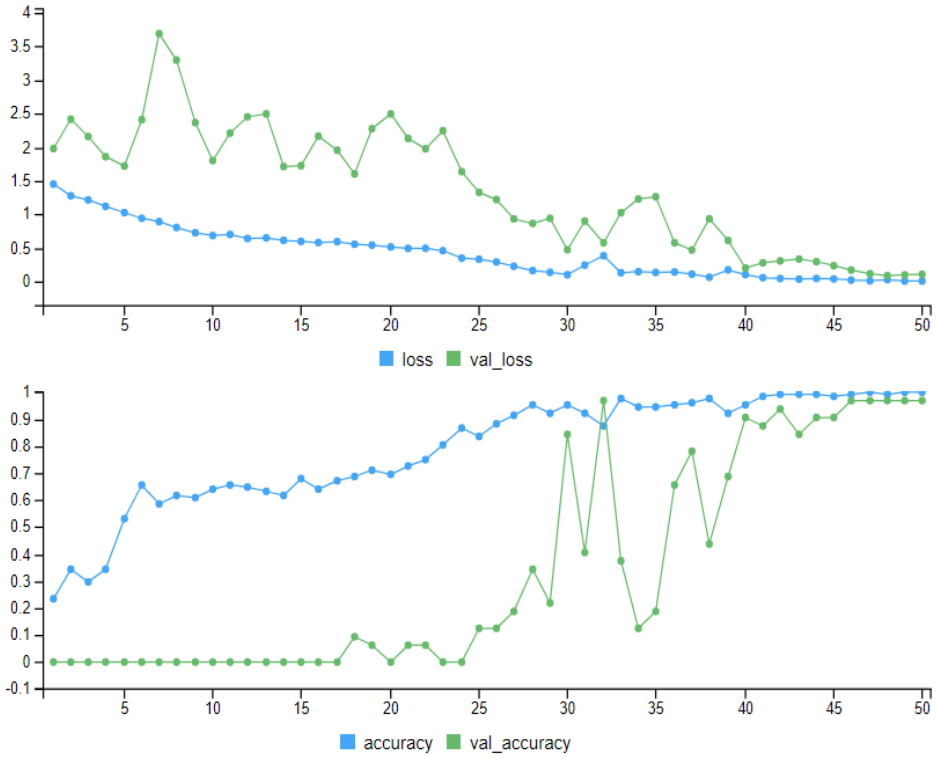
The San Francisco-based startup Atomwise developed an algorithm called AtomNet, based on a convolutional neural network, which was able to analyze and predict interactions between molecules.

Atomwise was able to use AtomNet to identify lead candidates for drug research programs. AtomNet successfully identified a candidate treatment for the Ebola virus, which had previously not been known to have any antiviral activity. The molecule later went on to pre-clinical trials.

Example. The folder "PlayingCardsImages" contains .jpg images of 43 cards of each suit (clubs, diamonds, hearts, and spades). The following R code splits the data into a training set (40 cards of each suit) and a testing set (3 cards of each suit), then a CNN model is trained on the training set and used to predict the suit for the images in the testing set.

```
library(keras)
library(EBImage)

setwd("./PlayingCardImages/Club")
img.card<- sample(dir());
cards<- list(NULL);
for(i in 1:length(img.card)) {
    cards[[i]]<- readImage(img.card[i])
      cards[[i]]<- resize(cards[[i]], 100, 100)
}
club<- cards

setwd("./PlayingCardImages/Heart")
img.card<- sample(dir());
cards<- list(NULL);
for(i in 1:length(img.card)) {
    cards[[i]]<- readImage(img.card[i])
      cards[[i]]<- resize(cards[[i]], 100, 100)
}
heart<- cards

setwd("./PlayingCardImages/Spade")
img.card<- sample(dir());
cards<- list(NULL);
for(i in 1:length(img.card)) {
    cards[[i]]<- readImage(img.card[i])
      cards[[i]]<- resize(cards[[i]], 100, 100)
}
```

```
spade<- cards

setwd("./PlayingCardImages/Diamond")
img.card<- sample(dir());
cards<- list(NULL);
for(i in 1:length(img.card)) {
    cards[[i]]<- readImage(img.card[i])
      cards[[i]]<- resize(cards[[i]], 100, 100)
}
diamond<- cards

#splitting into training and testing sets and permuting dimensions
train.pool<- c(club[1:40], heart[1:40], spade[1:40], diamond[1:40])
train<- aperm(combine(train.pool), c(4,1,2,3))
test.pool<- c(club[41:43], heart[41:43], spade[41:43], diamond[41:43])
test<- aperm(combine(test.pool), c(4,1,2,3))

#creating image labels
train.y<- c(rep(0,40),rep(1,40),rep(2,40),rep(3,40))
test.y<- c(rep(0,3),rep(1,3),rep(2,3),rep(3,3))
train.lab<- to_categorical(train.y)
test.lab<- to_categorical(test.y)

#building model
model.card<- keras_model_sequential()
model.card %>% layer_conv_2d(filters=40, kernel_size=c(4,4),
activation='relu', input_shape=c(100,100,4)) %>%
layer_conv_2d(filters=40, kernel_size=c(4,4), activation='relu') %>%
layer_max_pooling_2d(pool_size=c(4,4))%>% layer_dropout(rate=0.25) %>%
layer_conv_2d(filters=80, kernel_size=c(4,4), activation='relu') %>%
layer_conv_2d(filters=80, kernel_size=c(4,4), activation='relu') %>%
layer_max_pooling_2d(pool_size=c(4,4)) %>% layer_dropout(rate=0.35) %>%
layer_flatten() %>% layer_dense(units=256, activation='relu') %>%
layer_dropout(rate=0.25) %>% layer_dense(units=4, activation="softmax") %>%

compile(loss='categorical_crossentropy', optimizer=optimizer_adam(), metrics=c("accuracy"))

history<- model.card %>% fit(train, train.lab, epochs=50, batch_size=40, validation_split=0.2)
```




```
#computing prediction accuracy for testing set
model.card %>% evaluate(test, test.lab)
pred.class<- as.array(model.card %>% predict(test) %>% k_argmax())
print(pred.class)
```

```
0 0 0 1 1 1 2 2 2 3 3 3
```

```
print(test.y)
```

```
0 0 0 1 1 1 2 2 2 3 3 3
```

```
print(paste("accuracy=", round(1-mean(test.y!=pred.class),digits=4)))
```

```
"accuracy= 1"
```

□

Example. The built-in data set "MNIST" (short for "Modified National Institute of Standards and Technology") contains a collection of 70,000, 28×28 images of handwritten digits from 0 to 9.

These data are already split into a training set (60,000 images) and a testing set (10,000 images). The R and Python codes below train a CNN model and classify digits in the testing set. The prediction accuracy is computed.

In R:

```
library(keras)

mnist<- dataset_mnist()
train.x<- mnist$train$x #x=images
train.y<- mnist$train$y #y=labels
test.x<- mnist$test$x
test.y<- mnist$test$y

str(train.x)
```

```
int [1:60000, 1:28, 1:28] 0 0 0 0 0 0 0 0 0 0 ...
```

```
str(train.y)
```

```
int [1:60000(1d)] 5 0 4 1 9 2 1 3 1 4 ...
```

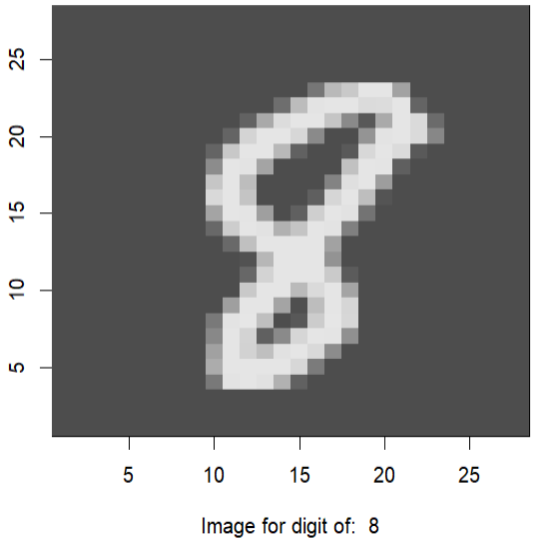
```
str(test.x)
```

```
int [1:10000, 1:28, 1:28] 0 0 0 0 0 0 0 0 0 0 ...
```

```
str(test.y)
```

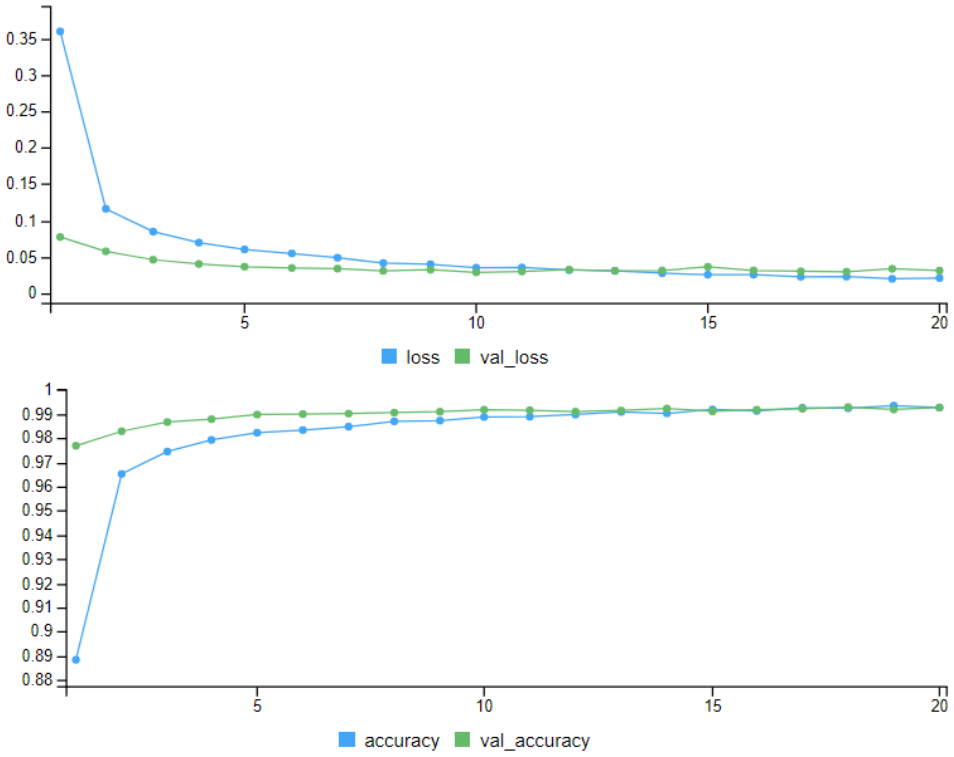
```
int [1:10000(1d)] 7 2 1 0 4 1 4 9 5 9 ...
```

```
index.image<- 2700 #picked randomly
input.matrix<- train.x[index.image,1:28,1:28]
output.matrix<- apply(input.matrix, 2, rev)
output.matrix<- t(output.matrix)
image(1:28, 1:28, output.matrix, col=gray.colors(256), xlab=paste('Image for digit of: ', train.y[index.image]),
ylab="")
```




```
#specifying parameters
batch.size<- 128
num.classes<- 10
epochs<- 20
img.rows<- 28
img.cols<- 28

train.x<- array_reshape(train.x, c(nrow(train.x), img.rows, img.cols, 1))
test.x<- array_reshape(test.x, c(nrow(test.x), img.rows, img.cols, 1))
input.shape<- c(img.rows, img.cols, 1)

#rescaling images
train.x<- train.x/255
test.x<- test.x/255

#converting class vectors to binary class matrices
train.y<- to_categorical(train.y, num.classes)
test.lab<- to_categorical(test.y, num.classes)

#defining model architecture
cnn_model<- keras_model_sequential() %>%
layer_conv_2d(filters=32, kernel_size=c(3,3), activation='relu', input_shape=input.shape) %>%
layer_max_pooling_2d(pool_size=c(2, 2)) %>%
layer_conv_2d(filters=64, kernel_size=c(3,3), activation='relu') %>%
layer_max_pooling_2d(pool_size=c(2, 2)) %>%
```

```
layer_dropout(rate=0.25) %>% layer_flatten() %>%
layer_dense(units=128, activation='relu') %>% layer_dropout(rate=0.5) %>%
layer_dense(units=num.classes, activation='softmax')

#compiling model
cnn_model %>% compile(loss=loss_categorical_crossentropy, optimizer=optimizer_adam(), metrics=c('accuracy'))

#training model
cnn_model %>% fit(train.x, train.y, batch_size=batch.size, epochs=epochs, validation_split=0.2)
```




```
#computing prediction accuracy
cnn_model %>% evaluate(test.x, test.lab)
pred.class<- as.array(cnn_model %>% predict(test.x) %>% k_argmax())
head(pred.class, n=50)
```

```
 [1] 7 2 1 0 4 1 4 9 5 9 0 6 9 0 1 5 9 7 3 4 9 6 6 5 4 0 7 4 0
[30] 1 3 1 3 4 7 2 7 1 2 1 1 7 4 2 3 5 1 2 4 4
```

```
head(test.y, n=50)
```

```
 [1] 7 2 1 0 4 1 4 9 5 9 0 6 9 0 1 5 9 7 3 4 9 6 6 5 4 0 7 4 0
[30] 1 3 1 3 4 7 2 7 1 2 1 1 7 4 2 3 5 1 2 4 4
```

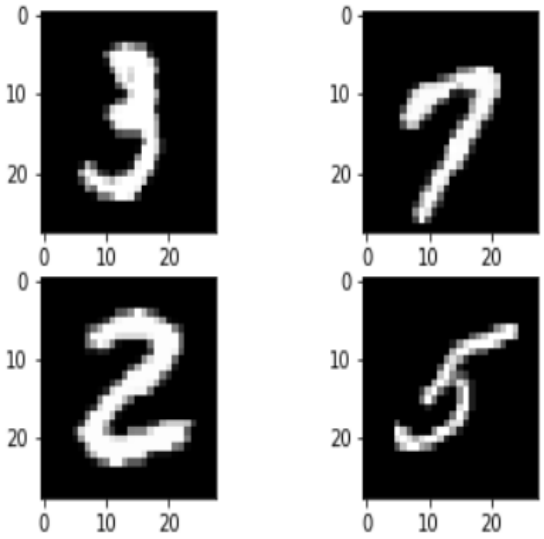
```
print(paste("accuracy=", round(1-mean(test.y!=pred.class), digits=4)))
```

```
"accuracy= 0.9924"
```

```
#displaying misclassified images
missed.image<- mnist$test$x[pred.class != mnist$test$y,,]
missed.digit<- mnist$test$y[pred.class != mnist$test$y]
missed.pred<- pred.class[pred.class != mnist$test$y]

index.image<- 270
input.matrix<- missed.image[index.image,1:28,1:28]
output.matrix<- apply(input.matrix, 2, rev)
output.matrix <- t(output.matrix)
image(1:28, 1:28, output.matrix, col=gray.colors(256), xlab=paste('Image for digit ', missed.digit[index.image],',
wrongly predicted as ', missed.pred[index.image]), ylab="")
```

In Python:

```
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt
#loading the MNIST dataset
(X_train, y_train), (X_test, y_test) = mnist.load_data()
#plotting some four images on gray scale
plt.subplot(221)
plt.imshow(X_train[10], cmap=plt.get_cmap('gray'))
plt.subplot(222)
plt.imshow(X_train[15], cmap=plt.get_cmap('gray'))
plt.subplot(223)
plt.imshow(X_train[25], cmap=plt.get_cmap('gray'))
plt.subplot(224)
plt.imshow(X_train[35], cmap=plt.get_cmap('gray'))
plt.show()
```



```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
from tensorflow.keras.utils import to_categorical
```

```
#loading training and testing sets (X=images, y=labels)
(train_X, train_y),(test_X, test_y)=mnist.load_data()
```

```
#flattening 28x28 images to a 784 vector for each image
num_pixels=train_X.shape[1]*train_X.shape[2]
train_X=train_X.reshape((train_X.shape[0],num_pixels)).astype('float32')
test_X=test_X.reshape((test_X.shape[0],num_pixels)).astype('float32')

#preparing labels
train_y=to_categorical(train_y)
test_y=to_categorical(test_y)
num_classes=test_y.shape[1]
```

```
#building CNN model
def baseline_model():
    model = Sequential()
    model.add(Dense(num_pixels, input_shape=(num_pixels,), kernel_initializer='normal', activation='relu'))
    model.add(Dense(num_classes, kernel_initializer='normal', activation='softmax'))
    model.compile(loss='categorical_crossentropy', metrics=['accuracy'])
    return model

model = baseline_model()
model.fit(train_X, train_y, validation_data=(test_X, test_y), epochs=10, batch_size=200)

#computing prediction accuracy
accuracy=model.evaluate(test_X,test_y)
print("Prediction Accuracy:", round(accuracy[1]*100,2),"%")
```

```
Prediction Accuracy: 97.25 %
```

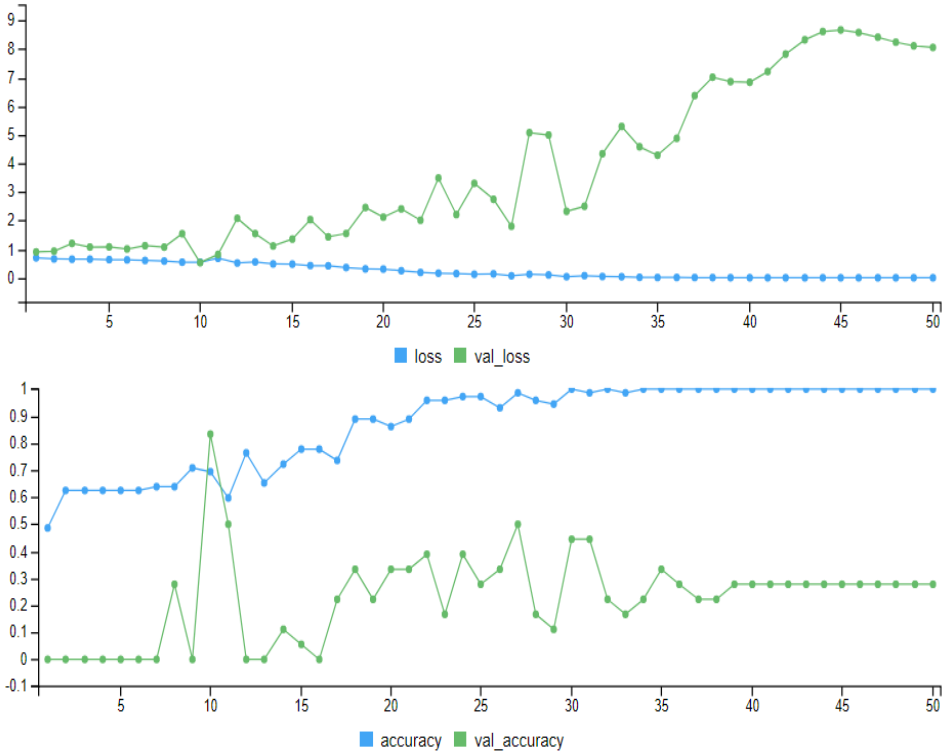
□

Example. The folder "PetsImages" contains the training set of 45 images of cats and 45 images of dogs, and a testing set of 34 images of our own pets (12 images of cats and 22 images of dogs). The R and Python codes below train a CNN model and classify the images in the testing set.

In R:

```
library(keras)
library(EBImage)

train.files<- Sys.glob(file.path("./PetsImages/train/*.jpg"))
train.labels<- substring(basename(train.files), 1,3) #extracting label: 'cat' or 'dog'
train.lab<- as.numeric(ifelse(train.labels=='cat',1,0))
```

```
setwd("./PetsImages/train")
img.pets<- sample(dir());
train.pets<- list(NULL);
for(i in 1:length(img.pets)) {
    train.pets[[i]]<- readImage(img.pets[i])
    train.pets[[i]]<- resize(train.pets[[i]], 100, 100)
}

train<- aperm(combine(train.pets), c(4,1,2,3)) #permuting dimensions

#building model
cnn.model<- keras_model_sequential() %>%
layer_conv_2d(filters=32, kernel_size=c(3, 3), activation="relu",
input_shape=c(100, 100, 3)) %>% layer_max_pooling_2d(pool_size=c(2, 2)) %>%
layer_conv_2d(filters=64, kernel_size=c(3, 3), activation="relu") %>%
layer_max_pooling_2d(pool_size=c(2, 2)) %>%
layer_conv_2d(filters=128, kernel_size=c(3, 3), activation="relu") %>%
layer_max_pooling_2d(pool_size=c(2, 2)) %>%
layer_conv_2d(filters=128, kernel_size=c(3, 3), activation="relu") %>%
layer_max_pooling_2d(pool_size=c(2, 2)) %>% layer_flatten() %>%
layer_dense(units=512, activation="relu") %>% layer_dense(units=1, activation="sigmoid")

cnn.model %>% compile(loss="binary_crossentropy", optimizer=optimizer_adam(), metrics="accuracy")

history<- cnn.model %>% fit(train, train.lab, epochs=50, batch_size=40, validation_split=0.2)
```




```
#computing prediction accuracy for testing set
setwd("./PetsImages/ourpets")
img.pets<- sample(dir());
test.pets<- list(NULL);
for(i in 1:length(img.pets)) {
    test.pets[[i]]<- readImage(img.pets[i])
    test.pets[[i]]<- resize(test.pets[[i]], 100, 100)
}

test<- aperm(combine(test.pets), c(4,1,2,3))
test.lab<- c(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1)

cnn.model %>% evaluate(test, test.lab)
pred.prob<- cnn.model %>% predict(test)

true.class<- ifelse(test.lab==1,'cat','dog')
pred.class<- c()
match<- c()
for (i in 1:length(pred.prob)) {
    pred.class[i]<- ifelse(pred.prob[i]>0.5,'cat','dog')
```

```
    match[i]<- ifelse(true.class[i]==pred.class[i],1,0)
}

print(true.class)
```

```
[1] "dog" "dog" "dog" "dog" "dog" "dog" "dog" "dog" "dog" "dog" "dog" "dog" "dog" "dog"
[15] "dog" "dog" "dog" "dog" "dog" "dog" "dog" "dog" "cat" "cat" "cat" "cat" "cat" "cat"
[29] "cat" "cat" "cat" "cat" "cat" "cat"
```

```
print(pred.class)
```

```
 [1] "dog" "cat" "cat" "cat" "dog" "cat" "cat" "dog" "dog" "dog" "cat" "dog" "dog" "cat"
[15] "cat" "dog" "cat" "cat" "dog" "dog" "cat" "cat" "dog" "dog" "dog" "dog" "cat" "cat"
[29] "cat" "dog" "cat" "cat" "dog" "dog"
```

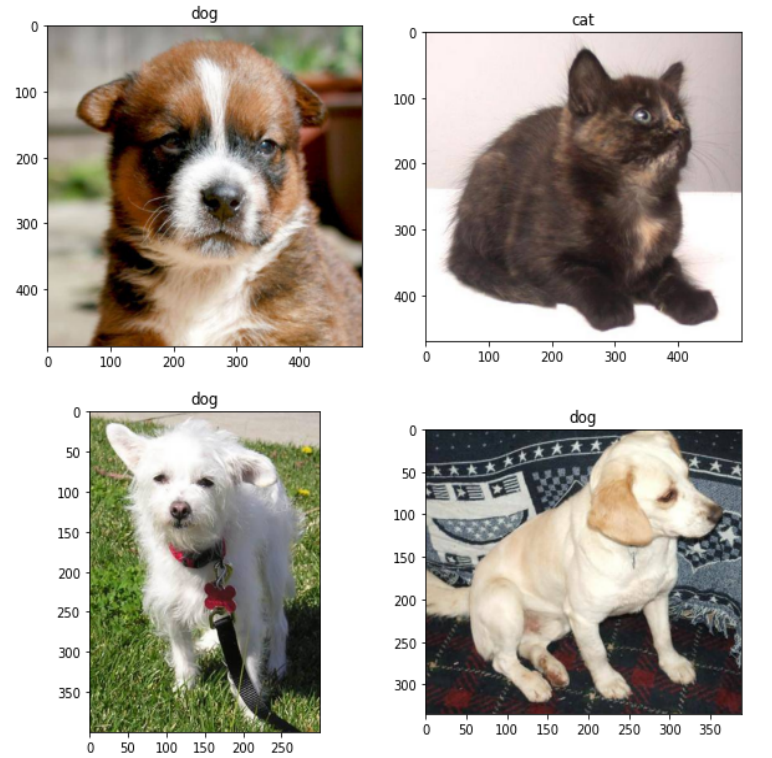
```
print(paste("accuracy=", round(mean(match), digits=4)))
```

```
"accuracy= 0.4412"
```

In Python:

```
import numpy
import pandas
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img
from tensorflow.keras.utils import plot_model
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import random
import os
from PIL import Image
import glob
import zipfile
```

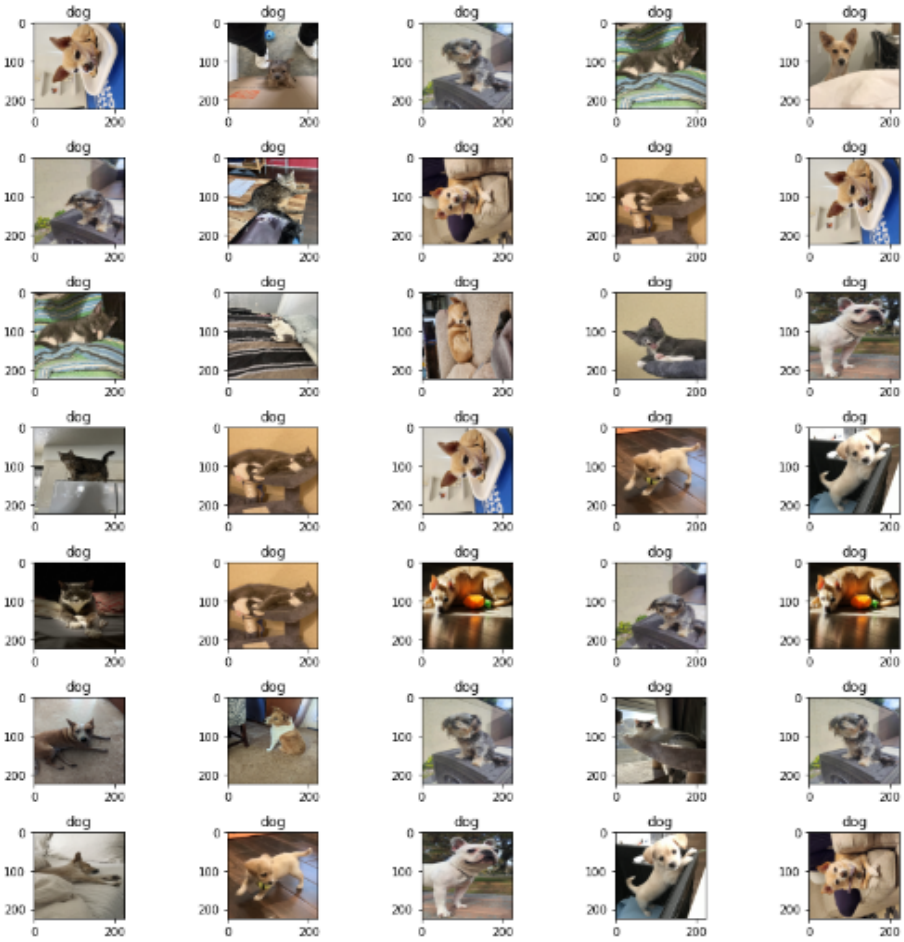
```
train_files=glob.glob('./PetsImages/train/*.jpg')
train_labels=[i.strip('./PetsImages/train/')[1:4] for i in train_files]
train_df=pandas.DataFrame({'filename': train_files, 'class': train_labels})
fig,axs=plt.subplots(2, 2, figsize=(10,10))
axs=axs.ravel()
for i in range(0,4):
    idx = random.choice(train_df.index)
    axs[i].imshow(Image.open(train_df['filename'][idx]))
    axs[i].set_title(train_df['class'][idx])
```




```
train_datagen=ImageDataGenerator(rotation_range=5,rescale=1./255, horizontal_flip=True, shear_range=0.2,
zoom_range=0.2, validation_split=0.2)

img_height,img_width=224, 224
batch_size=64

train_generator=train_datagen.flow_from_dataframe(train_df, target_size=(img_height, img_width),
batch_size=batch_size, class_mode='categorical', subset='training')

validation_generator=train_datagen.flow_from_dataframe(train_df, target_size=(img_height, img_width),
batch_size=batch_size, class_mode='categorical', subset='validation')
```

```
#training CNN model
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense, Activation, BatchNormalization

model=Sequential()

model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(img_width, img_height, 3)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2 ,2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(2, activation='softmax'))

model.compile(loss='binary_crossentropy', metrics=['accuracy'])

history=model.fit(train_generator, epochs=50, validation_data=validation_generator)
```

```
test_files=glob.glob('./PetsImages/ourpets/*.jpg')
test_df=pandas.DataFrame({'filename': test_files})
test_gen=ImageDataGenerator(rescale=1./255)
test_generator=test_gen.flow_from_dataframe(test_df, x_col='filename', y_col=None,
class_mode=None, target_size=(img_height,img_width), batch_size=batch_size)

def visualize_predictions(test_generator, model):
    plt.figure(figsize=(12,12))
    for i in range(0, 35):
        plt.subplot(7, 5, i+1)
        for X_batch in test_generator:
            prediction=model.predict(X_batch)[0]
            image=X_batch[0]
            plt.imshow(image)
            plt.title('cat' if numpy.argmax(prediction)==0 else 'dog')
            break
    plt.tight_layout()
    plt.show()

visualize_predictions(test_generator, model)
```



Note that every image is classified as a dog.
□

Further Reading

1. Generative Adversarial Networks (GANs)
   - https://machinelearningmastery.com/impressive-applications-of-generative-adversarial-networks/
   - https://realpython.com/generative-adversarial-networks/

2. Self Organizing Maps (SOMs)
   - https://davis.wpi.edu/∼matt/courses/soms/#Introduction

3. Restricted Boltzmann Machines (RBMs)
   - https://en.wikipedia.org/wiki/Restricted_Boltzmann_machine
   - https://wiki.pathmind.com/restricted-boltzmann-machine

4. Deep Belief Networks (DBNs)
   - https://en.wikipedia.org/wiki/Deep_belief_network
   - https://www.analyticsvidhya.com/blog/2022/03/an-overview-of-deep-belief-network-dbn-in-deep-learning/

5. AutoEncoders
   - https://towardsdatascience.com/introduction-to-autoencoders-7a47cf4ef14b

6. Learning Vector Quantization (LVQ)
   - https://machinelearningmastery.com/learning-vector-quantization-for-machine-learning/