

K-NEAREST NEIGHBOR REGRESSION AND CLASSIFICATION

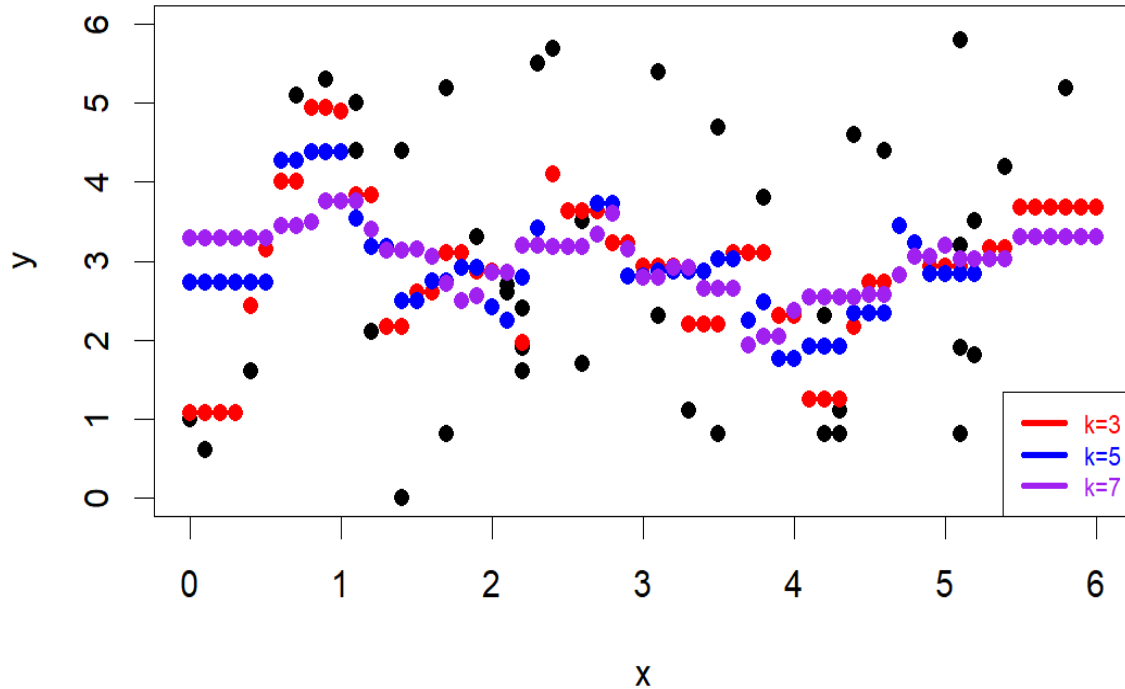
For regression, the **k Nearest-neighbor (kNN) algorithm** works as follows: for any point (on a grid) in the space of predictor variables, we find k nearest neighbors using the regular Euclidean distance. The **Euclidean distance** between two d -dimensional vectors $\mathbf{v} = (v_1, \dots, v_d)$ and $\mathbf{w} = (w_1, \dots, w_d)$ is defined as $distance(\mathbf{v}, \mathbf{w}) = \sqrt{(v_1 - w_1)^2 + (v_2 - w_2)^2 + \dots + (v_d - w_d)^2}$. The predicted value of y for this fixed point on a grid is determined as the average of the target values of the k nearest neighbors.

We illustrate how kNN regression works on a sample set of points with one predictor variable x and a target variable y . The black points in figure below represent the observed data on a scatterplot.

We consider each point on a grid between 0 and 6 with a step size of 0.1 and compute the mean values of y for $k = 3, 5$, and 7 nearest neighbors. The predicted values are shown as colored dots in the figure below.

Note that as k increases, the predicted values approach a horizontal line. Indeed if k is very large, all points are considered as nearest neighbors for every grid point, resulting in a single predicted value equal to the mean of all observed y -values.

Illustration of k-Nearest Neighbor Regression



Here is the R code that produced this figure.

```
sampldots.reg<- read.csv(file="./SamplePoints_KNN_reg.csv", sep=",", header=TRUE)

plot(sampldots.reg$x, sampldots.reg$y, xlim=c(0,6), ylim=c(0,6), xlab="x",
ylab="y", main="Illustration of k-Nearest Neighbor Regression", pch=16)

for (grid in seq(0, 6, 0.1)) {
  distance<- c()

  for (z in 1:length(sampldots.reg$x))
    distance[z]<- abs(grid - sampldots.reg$x[z])

  min.dist3<- sort(distance)[1:3] # 3 nearest neighbors
  y.pred3<- mean(sampldots.reg$y[which(distance %in% min.dist3)])
  points(grid,y.pred3,col="red", pch=16)

  min.dist5<- sort(distance)[1:5] # 5 nearest neighbors
  y.pred5<- mean(sampldots.reg$y[which(distance %in% min.dist5)])
  points(grid,y.pred5,col="blue", pch=16)

  min.dist7<- sort(distance)[1:7] # 7 nearest neighbors
  y.pred7<- mean(sampldots.reg$y[which(distance %in% min.dist7)])
  points(grid,y.pred7,col="purple", pch=16)

}
legend("bottomright",c("k=3","k=5","k=7"),lty=1,lwd=3,col=c("red","blue","purple"),
cex = 0.7, text.col=c("red","blue","purple"))
```

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;
```

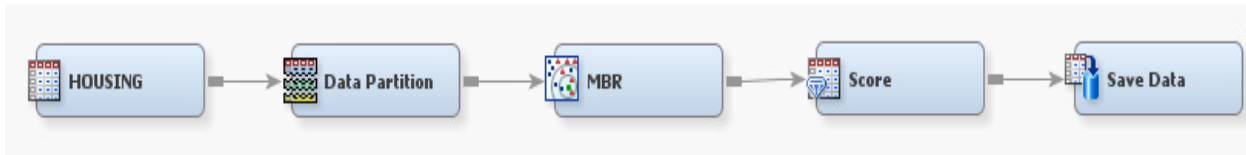
Then we use SAS Enterprise miner to fit a k nearest-neighbor regression (termed **Memory-Based Reasoning (MBR)**), using the path diagram:



For the "Data Partition" node, we specify to split the data into 80% training, and 20% testing sets. We run the paths and note the value for the Root MSE for the testing set (summarized in the table below).

Number of neighbors	Root MSE
7	77519.70
9	77038.58
10	76995.94

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



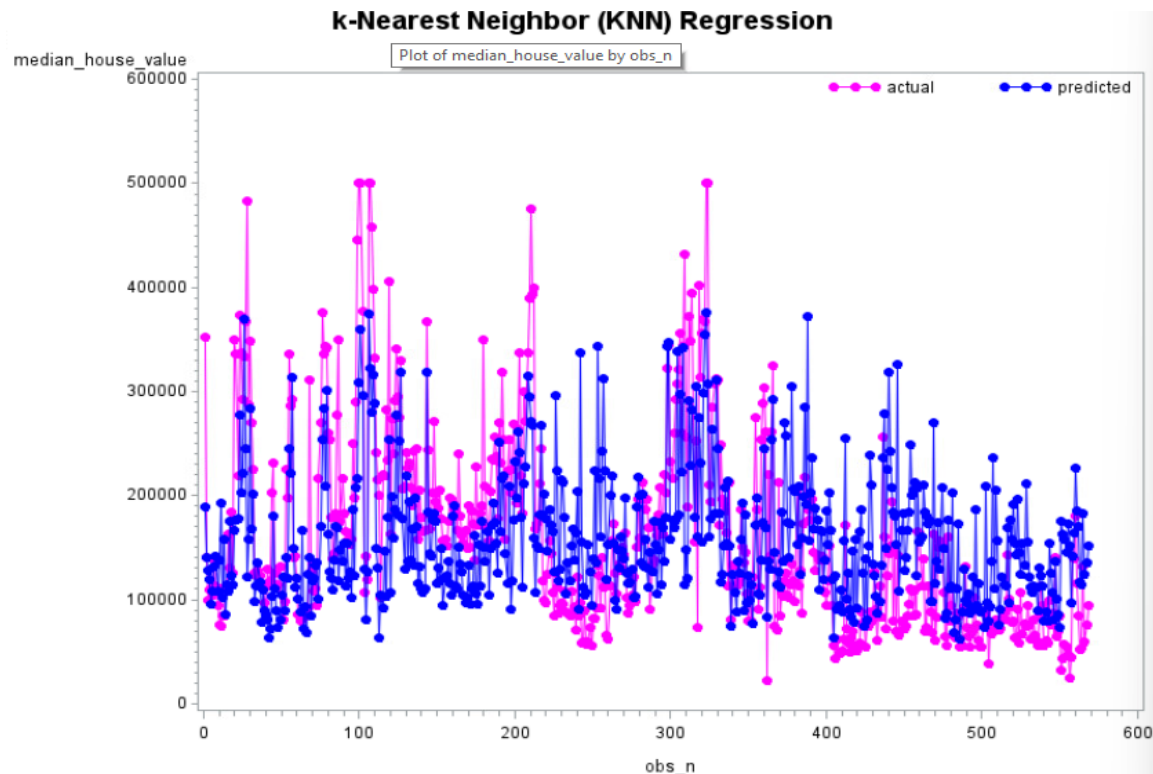
Next, we locate the file with predictions `"/Workspaces/EMWS1/EMSave/em_save_test.sas7bdat"`, open it in SAS (in the library "Tmp1") and run the code below to compute the proportion of predictions with 10%, 15%, and 20% of the actual values, and plot the actual and predicted values.

```
/*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
0.151408	0.214789	0.294014

```
/*PLOTING 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
```

```
library(caTools)
```

```
set.seed(753388)
```

```
sample<- sample.split(housing.data, SplitRatio=0.8)
```

```
train<- subset(housing.data, sample==TRUE)
```

```
test<- subset(housing.data, sample==FALSE)
```

```
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  82477.13
7  80267.18
9  79186.48
```

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))
```

```
0.1265823
```

```
#accuracy within 15%
```

```
accuracy15<- ifelse(abs(test.y-pred.y)<0.15*test.y,1,0)
```

```
print(mean(accuracy15))
```

```
0.2165963
```

```
#accuracy within 20%
```

```
accuracy20<- ifelse(abs(test.y-pred.y)<0.20*test.y,1,0)
```

```
print(mean(accuracy20))
```

```
0.2897328
```

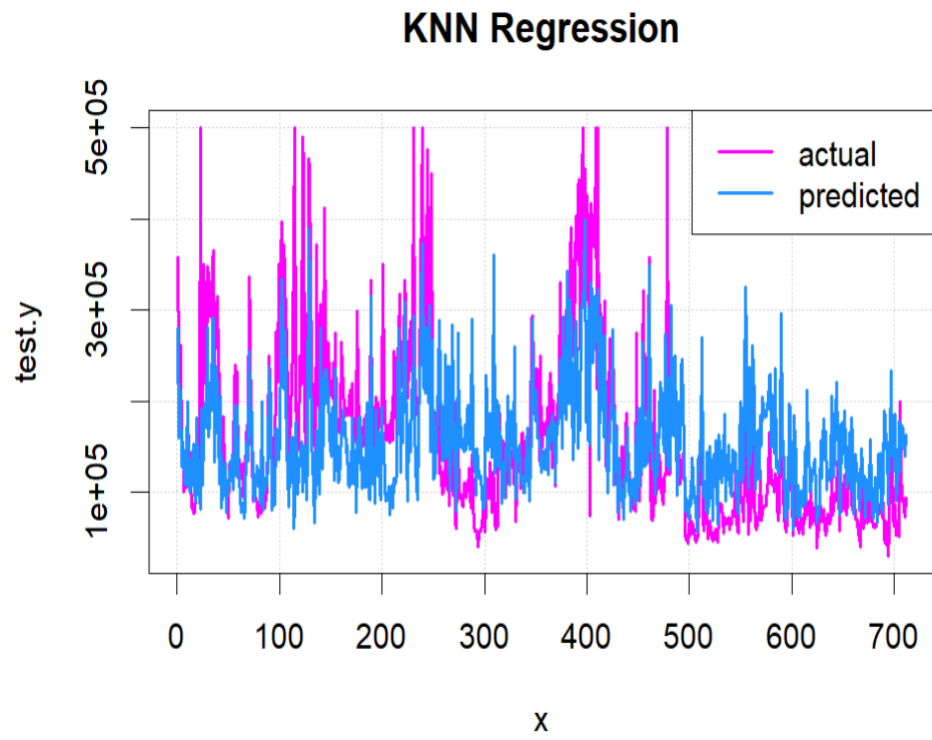
```
#PLOTING ACTUAL AND PREDICTED VALUES FOR TESTING DATA
```

```
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"))
```



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=9)
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))

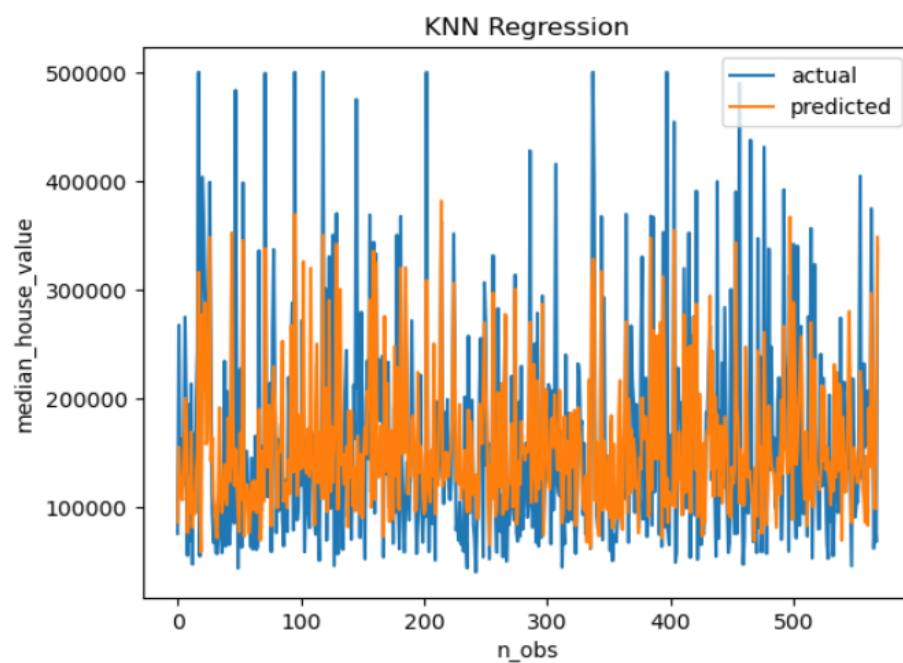
#accuracy within 20%
accuracy20=mean(ind20)
print('accuracy within 20% =', round(accuracy20,4))

```

accuracy within 10% = 0.1511
accuracy within 15% = 0.2443
accuracy within 20% = 0.297


```
#plotting actual and predicted obsevation 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()
```



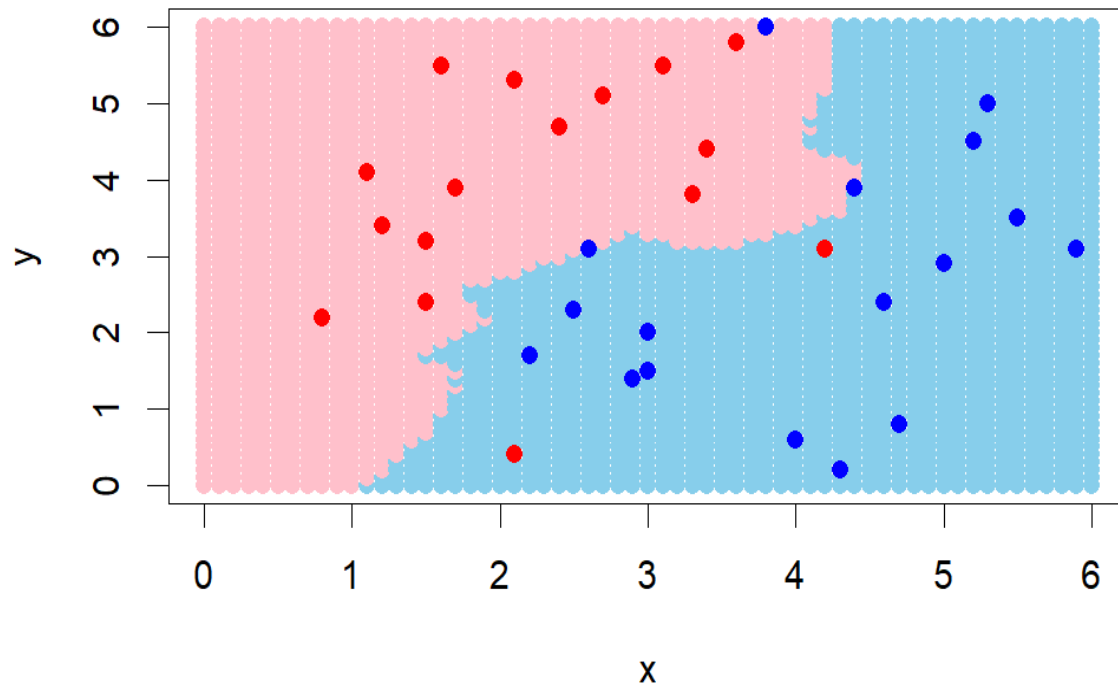
□

k-Nearest Neighbor Binary Classification

For binary classification, each point on a grid is assigned the class that is most frequent among its k nearest neighbors. Euclidean distance is used to measure the distances between grid points and data points.

We illustrate the algorithm using points of two colors: red and blue. The predictor variables are x and y . For each point on a two-dimensional grid with a step size of 0.1, the k nearest neighbors are identified, and the most frequent color among them is assigned to that grid point.

Illustration of k-Nearest Neighbor Binary Classification



Here is the R code that produced this illustration.

```
sampldots.bc<- read.csv(file="./SamplePoints_KNN_biclass.csv", sep="," , header=TRUE)

plot(sampldots.bc$x, sampldots.bc$y, xlim=c(0,6), ylim=c(0,6), xlab="x",
ylab="y", main="Illustration of k-Nearest Neighbor Binary Classification",
col=sampldots.bc$color, pch=16)

k<- 5 # five nearest neighbors
for (i in seq(0, 6, 0.1)) {
  for (j in seq(0, 6, 0.1)) {
    distance.sq<- c()
    for (z in 1:length(sampldots.bc$x))
```

```

distance.sq[z]<- (i-sampledots.bc$x[z])^2+(j-sampledots.bc$y[z])^2
min.values<- sort(distance.sq)[1:k]
min.colors<- sampledots.bc$color[which(distance.sq %in% min.values)]
pred.color<- names(sort(table(min.colors), decreasing=TRUE)[1])
if (pred.color=="red") col.dot="pink" else col.dot="skyblue"
points(i,j,col=col.dot, pch=16)
}
}
points(sampledots.bc$x, sampledots.bc$y, xlim=c(0,6), ylim=c(0,6),
col=sampledots.bc$color, pch=16)

```

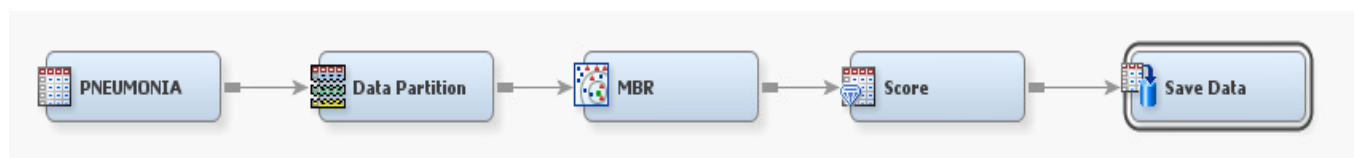
Example. For the data set "pneumonia_data.csv", we build the kNN binary classifier.

In SAS:

In SAS Enterprise Miner, we partition the data into 80% training, and 20% testing sets, and run the MBR node, varying the number of neighbors and record the misclassification rate for the testing set. The results are summarized here:

Number of neighbors	Misclassification Rate
5	0.312
7	0.303
9	0.269
11	0.286

We choose to utilize $k = 9$ neighbors 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
0.731214

Note that we already computed the misclassification rate for $k=9$ neighbors (0.269) for the testing set, so the above calculation of the accuracy rate is actually redundant.

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
```

```
library(caTools)
```

```
set.seed(704467)
```

```
sample<- sample.split(pneumonia.data, SplitRatio=0.8)
```

```
train<- subset(pneumonia.data, sample==TRUE)
```

```
test<- subset(pneumonia.data, sample==FALSE)
```

```
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  0.6775963

```

```
7  0.6817137
9  0.6809037
```

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was $k = 7$.

```
#FITTING OPTIMAL KNN BINARY CLASSIFIER (K=7)
knn.class<- knnreg(train.x, train.y, k=7)
```

```
#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= 0.7101"
```

```
#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= 0.7101"
```

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=7
biclass=KNeighborsClassifier(n_neighbors=7)
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))

```

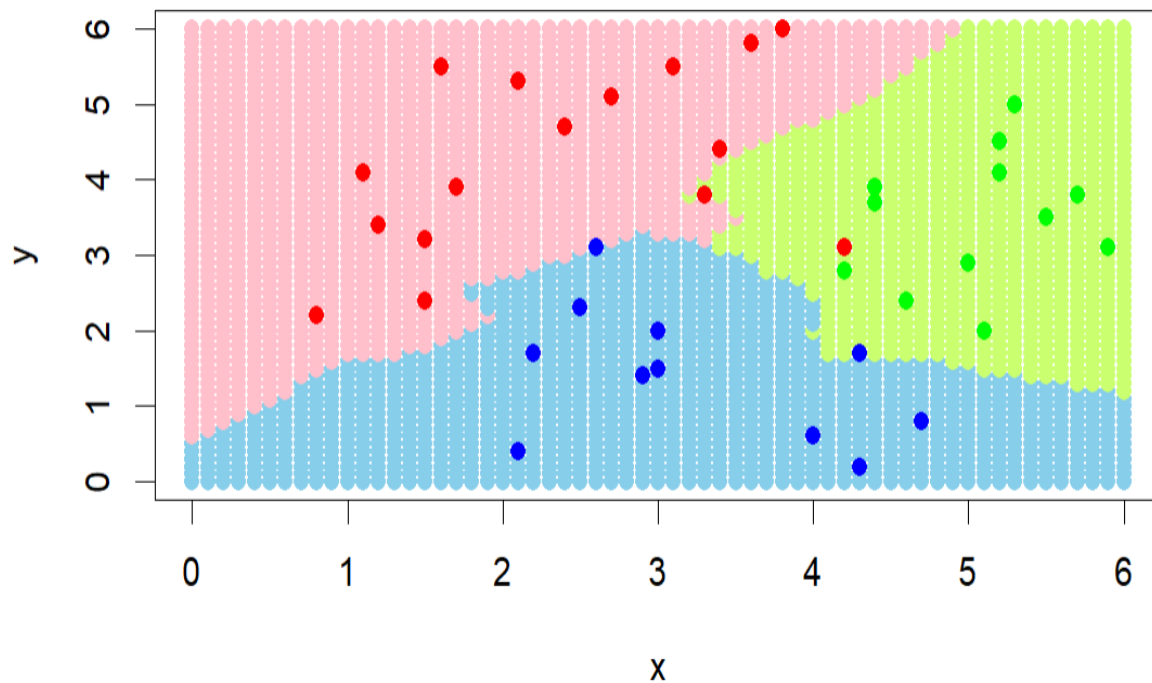
accuracy= 0.7369942196531792

□

k-Nearest Neighbor Multinomial Classification

The kNN algorithm works with multinomial classification in a manner similar to binary classification. The predicted class for a grid point is determined by a majority vote among the classes of its k nearest neighbors. We illustrate this below using three colors: red, blue, and green.

Illustration of k-Nearest Neighbor Multinomial Classification



Here is the R code that produced this illustration.

```
sampldots.bc<- read.csv(file="./SamplePoints_KNN_multiclass.csv", sep=",", header=TRUE)

plot(sampldots.bc$x, sampldots.bc$y, xlim=c(0,6), ylim=c(0,6), xlab="x",
ylab="y", main="Illustration of k-Nearest Neighbor Multinomial Classification",
col=sampldots.bc$color, pch=16)

k<- 5 # five nearest neighbors
for (i in seq(0, 6, 0.1)) {
  for (j in seq(0, 6, 0.1)) {

    distance.sq<- c()
    for (z in 1:length(sampldots.bc$x))
      distance.sq[z]<- (i-sampldots.bc$x[z])^2+(j-sampldots.bc$y[z])^2

    min.values<- sort(distance.sq)[1:k]
    min.colors<- sampldots.bc$color[which(distance.sq %in% min.values)]
```

```

pred.color<- names(sort(table(min.colors), decreasing=TRUE)[1])

if (pred.color=="red") col.dot="pink"
if (pred.color=="blue") col.dot="skyblue"
if (pred.color=="green") col.dot="darkolivegreen1"
points(i,j,col=col.dot, pch=16)
}
}

points(sampledots.bc$x, sampledots.bc$y, xlim=c(0,6), ylim=c(0,6),
col=sampledots.bc$color, pch=16)

```

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 $k = 7, 9$, and 11 , and compare the misclassification rates for the testing set.

Number of neighbors	Misclassification Rate
7	0.726
9	0.713
11	0.720

We use $k = 9$ as it gives the smallest misclassification rate and run the full path depicted in the figure below. Again, since we already know the misclassification rate for the testing set, running the lines below is not really necessary.



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
0.286624

In R:

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

```
#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
```

```
library(caTools)
```

```
set.seed(499251)
```

```
sample<- sample.split(movie.data, SplitRatio=0.8)
```

```
train<- subset(movie.data, sample==TRUE)
```

```
test<- subset(movie.data, sample==FALSE)
```

```
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=9 reasonably maximizes prediction accuracy for testing set
```

```
library(caret)
```

```
knn.mclass<- knnreg(train.x, train.y, k=9)
```

```
#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= 0.1974
```

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=799842)

#FITTING kNN MULTINOMIAL CLASSIFIER
multiclass=KNeighborsClassifier(n_neighbors=9)
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= 0.32894736842105265

□