Lab 10 Classification: K-NN

Lusine Zilfimian

April 22 (Wednesday), 2020

Contents

Needed packages

```r
library(caret) # for createDataPartition() and trainControl(), train()
library(class) # for knn()
library(dplyr) # for data manipulation
library(MASS) # for Boston Dataset
library(FNN) # for knn.reg()
library(ggvoronoi) # for stat_voronoi()
library(ggplot2) # ok, you know it
```

Data Preparation / Undestanding the data

- Read the data Diabetes.csv

```r
diab <- read.csv("Diabetes.csv")
str(diab)
```

```
## 'data.frame':    768 obs. of  9 variables:
##  $ NTS  : int  6 1 8 1 0 5 3 10 2 8 ...
##  $ PGC  : int  148 85 183 89 137 116 78 115 197 125 ...
##  $ DBP  : int  72 66 64 66 40 74 50 0 70 96 ...
##  $ TSFT : int  35 29 0 23 35 0 32 0 45 0 ...
##  $ INS  : int  0 0 0 94 168 0 88 0 543 0 ...
##  $ BMI  : num  33.6 26.6 23.3 28.1 43.1 25.6 31 35.3 30.5 0 ...
##  $ DPF  : num  0.627 0.351 0.672 0.167 2.288 ...
##  $ Age  : int  50 31 32 21 33 30 26 29 53 54 ...
##  $ Class: Factor w/ 2 levels "Negative","Positive": 2 1 2 1 2 1 2 1 2 2
```

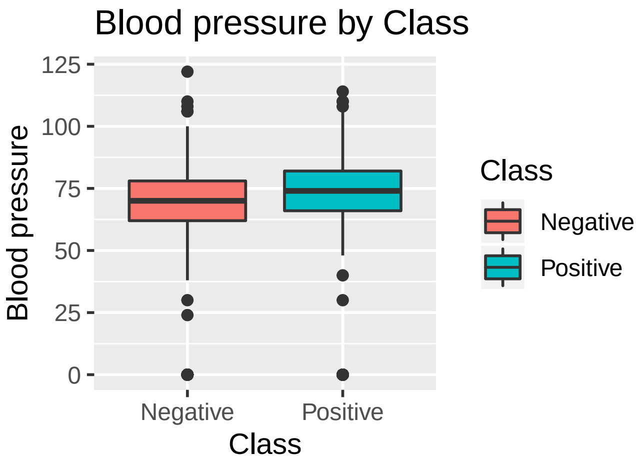
Undestanding the data

Source: https://www.kaggle.com/uciml/pima-indians-diabetes-database

- NTS - Number of times pregnant
- PGC - Plasma glucose concentration a 2 hours in an oral glucose tolerance test
- DBP - Diastolic blood pressure (mm Hg)
- TSFT - Triceps skin fold thickness (mm)
- INS - Hour serum insulin (mu U/ml)
- BMI - Body mass index (weight in kg/(height in m)^2)
- DPF - Diabetes pedigree function
- Age (years)
- Class variable

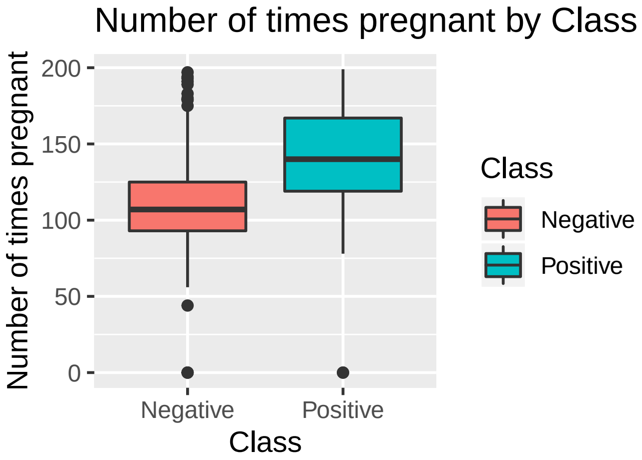
Undestanding the data

```
ggplot(data = diab, aes(x = Class, y = DBP, fill = Class))+
  geom_boxplot() + labs(x = "Class", y = "Blood pressure",
    title = "Blood pressure by Class")
```



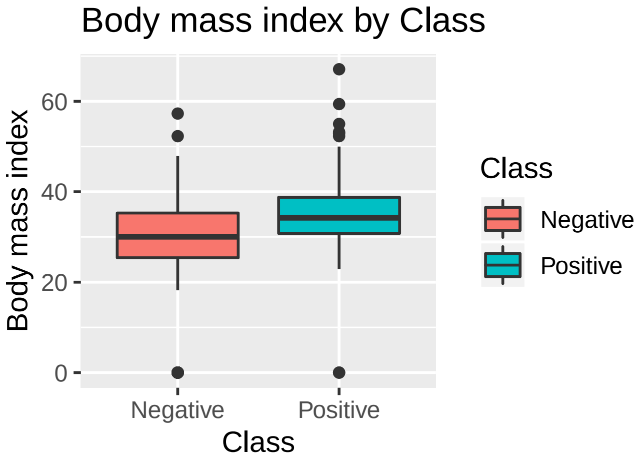
Undestanding the data

```
ggplot(data = diab, aes(x = Class, y = PGC, fill = Class))+
  geom_boxplot() + labs(x = "Class", y = "Number of times pregnant",
    title = "Number of times pregnant by Class")
```



Undestanding the data

```
ggplot(data = diab, aes(x = Class, y = BMI, fill = Class))+
  geom_boxplot() + labs(x = "Class", y = "Body mass index",
    title = "Body mass index by Class")
```



Data Preparation

- Divide the dataset into training and testing sets:

```r
set.seed(2708)
split <- diab$Class %>% createDataPartition(p = 0.8, list = FALSE)

train.data  <- diab[split, ]
test.data <- diab[-split, ]
```

k-NN Classification (Sample Validation)

- Make prediction on the testing set (single case):
- The function arguments:
  1. train: The training set without actual label
  2. test: the testing set without actual label
  3. cl: actual class labels from Train set
  4. k: the number of neighbors to consider

```
# Make prediction on the testing set (one observation)
(knn0 <- knn(train = train.data[,-9], test = test.data[1,-9],
  cl = train.data$Class, k = 5))
```

```
## [1] Negative
## attr(,"nn.index")
##      [,1] [,2] [,3] [,4] [,5]
## [1,]  604  355  382  126  606
## attr(,"nn.dist")
##          [,1]     [,2]     [,3]     [,4]     [,5]
## [1,] 10.55812 19.19861 22.04193 23.60764 24.60926
## Levels: Negative
```

k-NN Classification (Sample Validation)

- Make prediction for the whole Test dataset.
- The first column of each dataframe is the actual class, so we take it out

```
knn1 <- knn(train = train.data[,-9], test = test.data[,-9],
  cl = train.data$Class, k=5)
length(knn1) == dim(test.data)[2]
```

```
## [1] FALSE
```

```
head(knn1,10)
```

```
##  [1] Negative Negative Positive Negative Positive Negative Negative
##  [8] Negative Negative Negative
## Levels: Negative Positive
```

k-NN Classification (Sample Validation)

- Lets look at the accuracy

```
table(knn1, test.data$Class)
```

```
##
## knn1       Negative Positive
##   Negative       83       29
##   Positive       17       24
```

```
# Accuracy
(83 + 24)/153
```

```
## [1] 0.6993464
```

```
mean(knn1 == test.data$Class)
```

```
## [1] 0.6993464
```

```
table(test.data$Class)/153
```

```
##
##  Negative  Positive
## 0.6535948 0.3464052
```

k-NN Classification (Sample Validation)

- You can use knn function to predict probabilities of belonging to a class as well.
- Use argument prob = T in the function
- Access the probabilities using function attr, inside giving the object and attribute name “prob”
- Case 1: there were 5 neighbors, 4 of them were Negative cases so we predict the case as being Negative with the probability of 0.8 (4/5)

```r
knn2 <- knn(train = train.data[,-9], test = test.data[,-9],
  cl = train.data$Class, k = 6, prob = T)
df <- data.frame(class = knn1, probs=attr(knn2, "prob"))
head(df)
```

```
##      class     probs
## 1 Negative 0.6666667
## 2 Negative 0.8333333
## 3 Positive 0.6666667
## 4 Negative 0.6666667
## 5 Positive 0.6666667
## 6 Negative 0.8333333
```

k-NN Classification (Cross Validation)

- `trainControl` creates object that will be used in modeling. In this case we specify that we want to do cross-validation (method="cv") with 10 folds (number=10).

```r
set.seed(2708)
ctrl <- trainControl(method="cv", number = 10)
```

Function `train` from caret

- give the formula
- give the data
- set the method for training (method="knn)
- trControl specifies the training control object
- prePreprocess: calculate the Z-scores
- tuneLength: for how many parameters of k do you want to build model

k-NN Classification (Cross Validation)

The tuneLength parameter tells the algorithm to try different **default** values for the main parameter

```
set.seed(2708)
knn_c <- train( Class~.,  data = diab, method = "knn",
  trControl = ctrl, preProcess = c("center","scale"), tuneLength = 10)
```

k-NN Classification (Cross Validation)

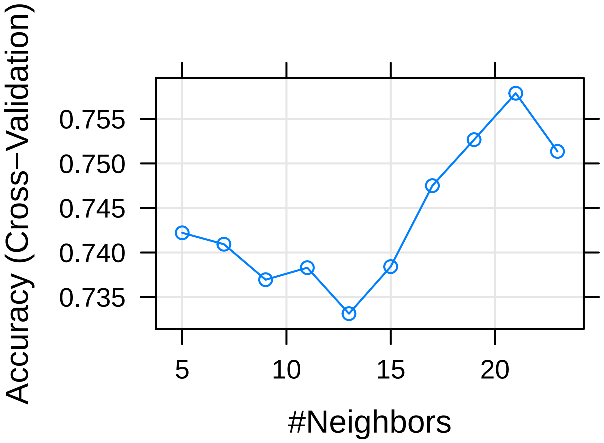
- The choice of parameters is always the same. The column k shows the number of neighbours that are used during the training
- Column Accuracy shows the average accuracy on the testing sets (total 10 testing sets)
- Overall 10 different values for k are used for modeling

```
knn_c$results[,1:2]
```

```
##     k  Accuracy
## 1   5 0.7422078
## 2   7 0.7409262
## 3   9 0.7369446
## 4  11 0.7382946
## 5  13 0.7331340
## 6  15 0.7384142
## 7  17 0.7475051
## 8  19 0.7526658
## 9  21 0.7578776
## 10 23 0.7513500
```

k-NN Classification (Cross Validation)

```
plot(knn_c)
```

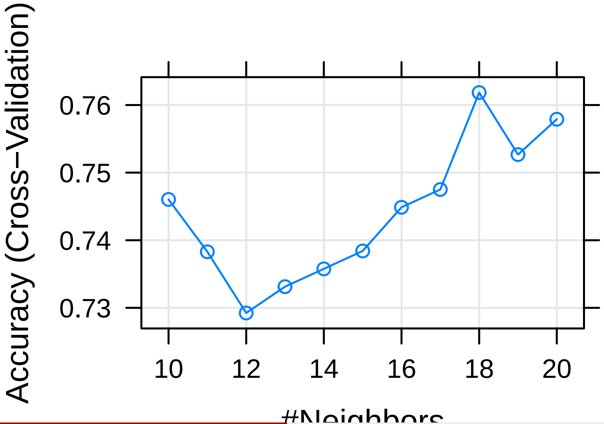


```
knn_c$bestTune
```

k-NN Classification (Cross Validation)

- To specify vectors of k's for the model, use `tuneGrid` (not `tuneLenght`)

```
grid <- expand.grid(k = 10:20); set.seed(2708)
knn_c1<-train(Class~., data = diab, method="knn",  trControl = ctrl,
  preProcess = c("center","scale"), tuneGrid = grid)
plot(knn_c1)
```



KNN Regression (Sample Validation)

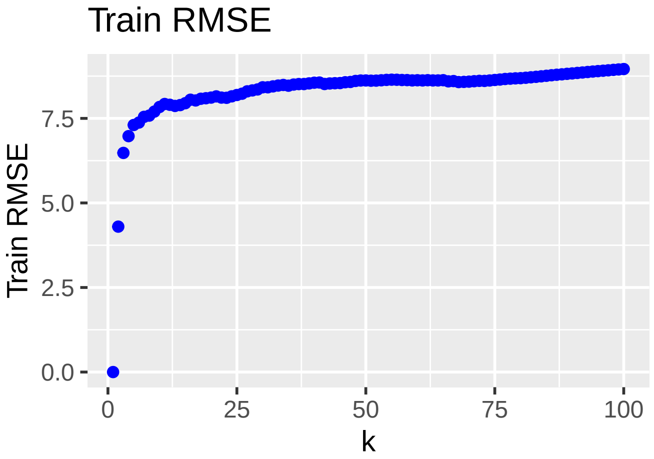
This dataset contains information collected by the U.S Census Service concerning housing in the area of Boston Mass.

```r
set.seed(42)
idx <- sample(1:nrow(Boston), size = 250)
trn_boston <- Boston[idx, ]
tst_boston <- Boston[-idx, ]
```

```r
RMSEtrain <- NULL
RMSEtest <- NULL
for (i in 1:100){
  trainknn <- knn.reg(train = trn_boston[, -1],
                      test = trn_boston[,-1], y = trn_boston$crim, k = i)
  testknn <- knn.reg(train = trn_boston[, -1],  test = tst_boston[,-1],
                     y = trn_boston$crim, k = i)
  RMSEtrain[i] <- sqrt(mean((trn_boston$crim - trainknn$pred)^2))
  RMSEtest[i] <- sqrt(mean((tst_boston$crim - testknn$pred)^2))
}
```

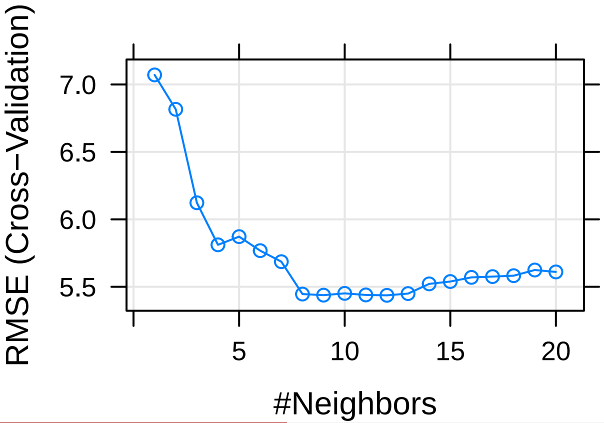
KNN Regression (Sample Validation)

```
df <- data.frame(x = 1:100, train = RMSEtrain, test = RMSEtest)
ggplot(data = df, aes(x = x, y = train)) +
  geom_point(color = "blue")+
  labs(y = "Train RMSE", x = "k", title="Train RMSE")
```



KNN Regression (Cross Validation)

```r
grid <- expand.grid(k = 1:20); set.seed(1)
model <- train( crim~., data = Boston, method = "knn",
  trControl = trainControl("cv", number = 10),
  preProcess = c("center","scale"), tuneGrid = grid)
plot(model)
```



KNN Regression (Cross Validation)

```r
head(model$results[,c(1,2)], n=10)
```

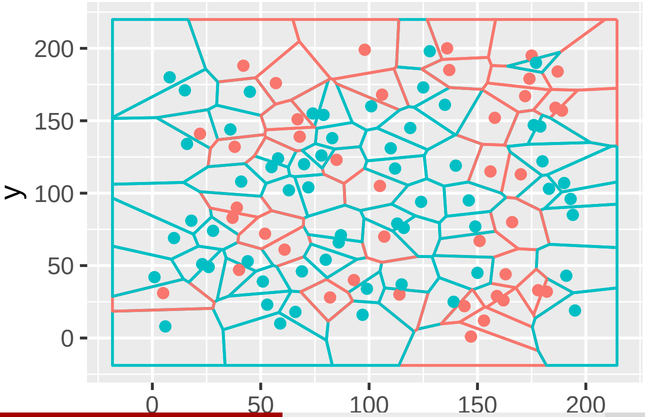
```
##     k     RMSE
## 1   1 7.070863
## 2   2 6.815805
## 3   3 6.122864
## 4   4 5.811194
## 5   5 5.872013
## 6   6 5.767924
## 7   7 5.686235
## 8   8 5.446108
## 9   9 5.438080
## 10 10 5.451263
```

```r
model$results[which.min(model$results$RMSE),]
```

```
##     k     RMSE  Rsquared      MAE   RMSESD RsquaredSD     MAESD
## 12 12 5.436749 0.6297794 1.804765 3.145291  0.2207016 0.6775957
```

Voronoi Tessellation

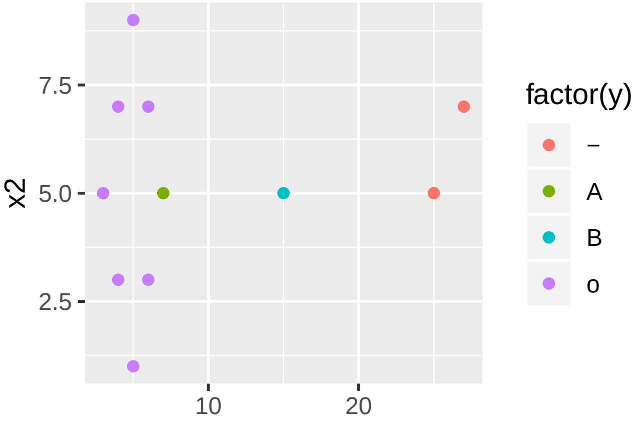
```r
set.seed(2708)
x <- sample(1:200,100); y <- sample(1:200,100)
z <- rbinom(n = 100, size = 1, prob = 0.6); df <- data.frame(x,y,z)
ggplot(df, aes(x,y, col=factor(z))) + stat_voronoi(geom="path")  +
  geom_point() + theme(legend.position = "none")
```



Question

1. If for B the nearest is A, does it mean that for A the nearest is B

```
x1 <- c(3,4,4,5,5,6,6,7,15,25,27); x2 <- c(5, 3, 7, 1,9,3,7,5, 5, 5,7 )
y <-c(rep("o", 7),"A", "B", rep("-", 2)); df <- data.frame(x1, x2, y)
ggplot(df, aes ( x = x1, y = x2, col = factor(y))) + geom_point()
```



Question

2. Problem: What to do?

```
x<-c(4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4)
y<-c(5, 3, 7, 1, 2, 9, 8, 6, 4, 0.5, -5)
df <- data.frame(x,y);
ggplot(df, aes ( x= x, y = y)) + geom_point()
```