HW 6 Hints

Load Data

Load the data from the Kaggle digits data set. There is a train and test data set. For this example, I only use the train data set but run crossvalidation. Also note: this data set is large and so I reduce it by "percent" – see below.

```
#First load the training data in csv format, and then convert "label" to nominal variable.
filename <-"digit_train.csv"
DigitTotalDF <- read.csv(filename, header = TRUE, stringsAsFactors = TRUE)
DigitTotalDF$label<-as.factor(DigitTotalDF$label)
dim(DigitTotalDF)
```

```
## [1] 42000   785
```

```
#head(DigitTotalDF)
#Create a random sample of n% of train data set
```

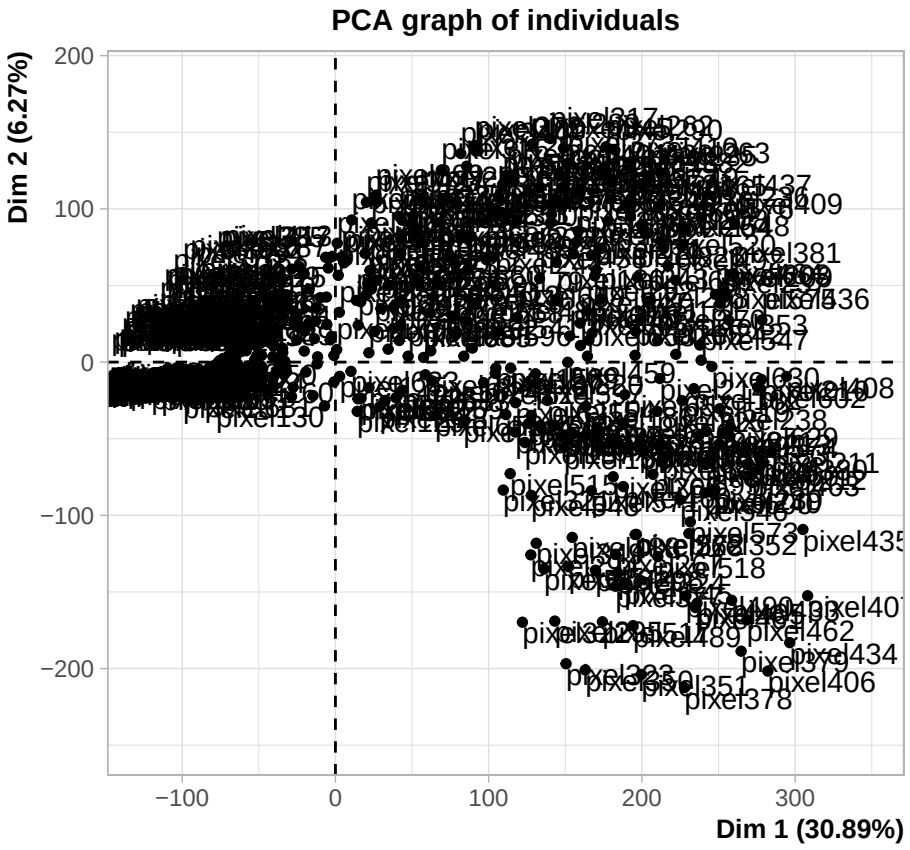
Dimensionality reduction and feature selection are important concepts when dealing with high dimensional data. The goal is to avoid the curse of high dimensionality. Principal Components Analysis is one way to select features based on variance. Here is an example! Lets reduce the dimensionality from 784 to 5!! Interestingly – results improve!!

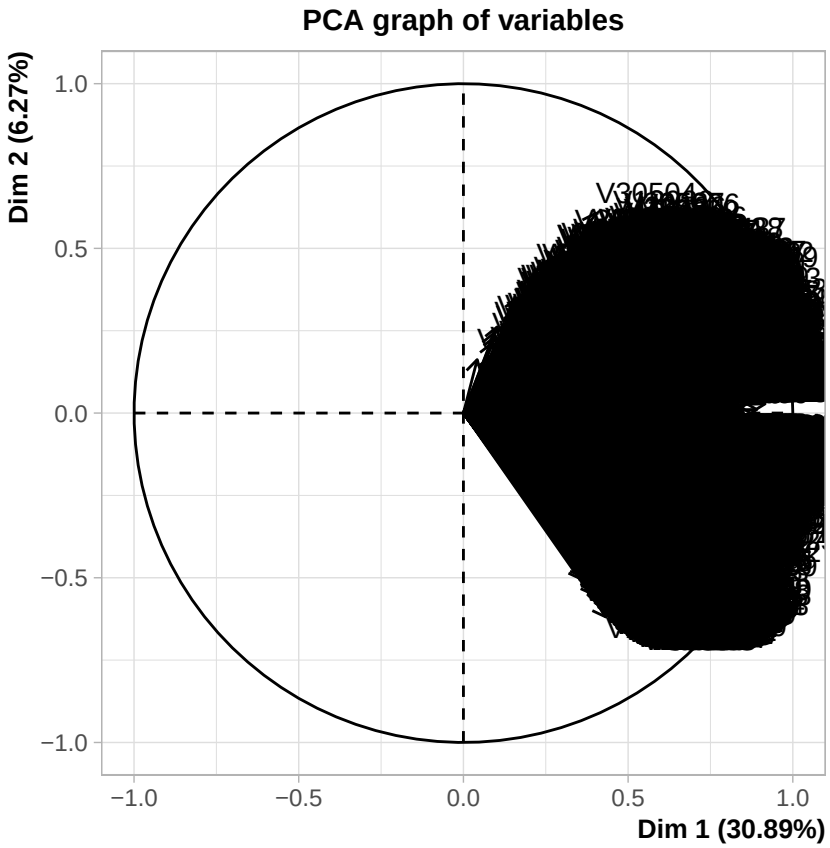
Read about the PCA parameters (from FactoMineR), and try to improve these results by running different variations of PCA. For example, try keeping 10 dimensions, try 15, ...

```
library(FactoMineR)
```

```
## Warning: package 'FactoMineR' was built under R version 3.5.3
```

```
pca_digits = PCA(t(select(DigitTotalDF,-label)))
```


PCA graph of individuals
Dim 2 (6.27%)
Dim 1 (30.89%)
200
100
0
−100
−200
−100
0
100
200
300
pixel130
pixel347
pixel381
pixel409
pixel436
pixel462
pixel434
pixel406
pixel378




```
DigitTotalDF =   data.frame(DigitTotalDF$label,pca_digits$var$coord)
```

Lets also reduce the total number of data samples used .... 10000 is quite a bit!!

```
percent <- .25
set.seed(275)
DigitSplit <- sample(nrow(DigitTotalDF),nrow(DigitTotalDF)*percent)
DigitDF <- DigitTotalDF[DigitSplit,]
dim(DigitDF)
```

```
## [1] 10500     6
```

```
#(head(DigitDF))
#(str(DigitDF))
(nrow(DigitDF))
```

```
## [1] 10500
```

Don't use the test data set in this example, but I encourage you to try and see what happens. Instead I simply run kfold crossvalidation using the data from the "train" csv file ...

```
filename <-"digit_test.csv"
TestTotalDF <- read.csv(filename, header = TRUE, stringsAsFactors = TRUE)
```

```
#TestTotalDF$label<-as.factor(TestTotalDF$label)
# Wont use test data, instead crossvalidation on train.
dim(TestTotalDF)
```

```
## [1] 28000   784
```

```
#(head(TestTotalDF))
```

EDA

Investigate the data here . . . what do you find???

Experimental Design

Try running k-fold crossvalidation. See code example below which uses split to determine the fold indexes.

```
# Feature Selection / Dim Reduction

# Feature Selection
```

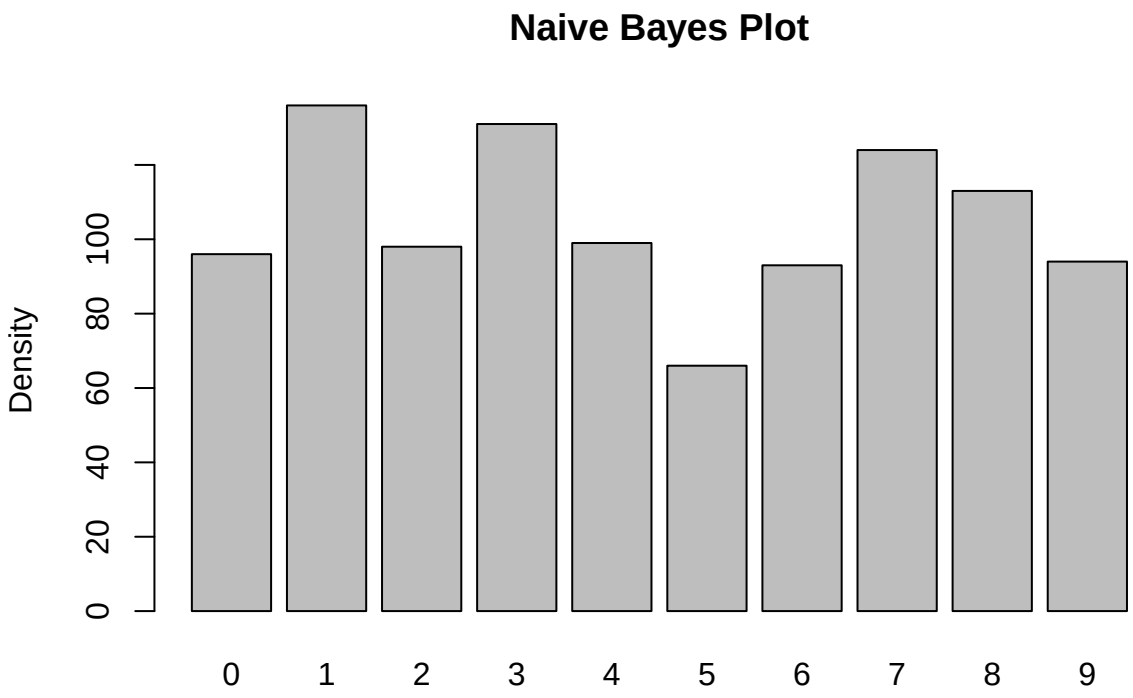
```
# Create k-folds for k-fold cross validation
## Number of observations
N <- nrow(DigitDF)
## Number of desired splits
kfolds <- 10
## Generate indices of holdout observations
## Note if N is not a multiple of folds you will get a warning, but is OK.
holdout <- split(sample(1:N), 1:kfolds)
#head(holdout)
```

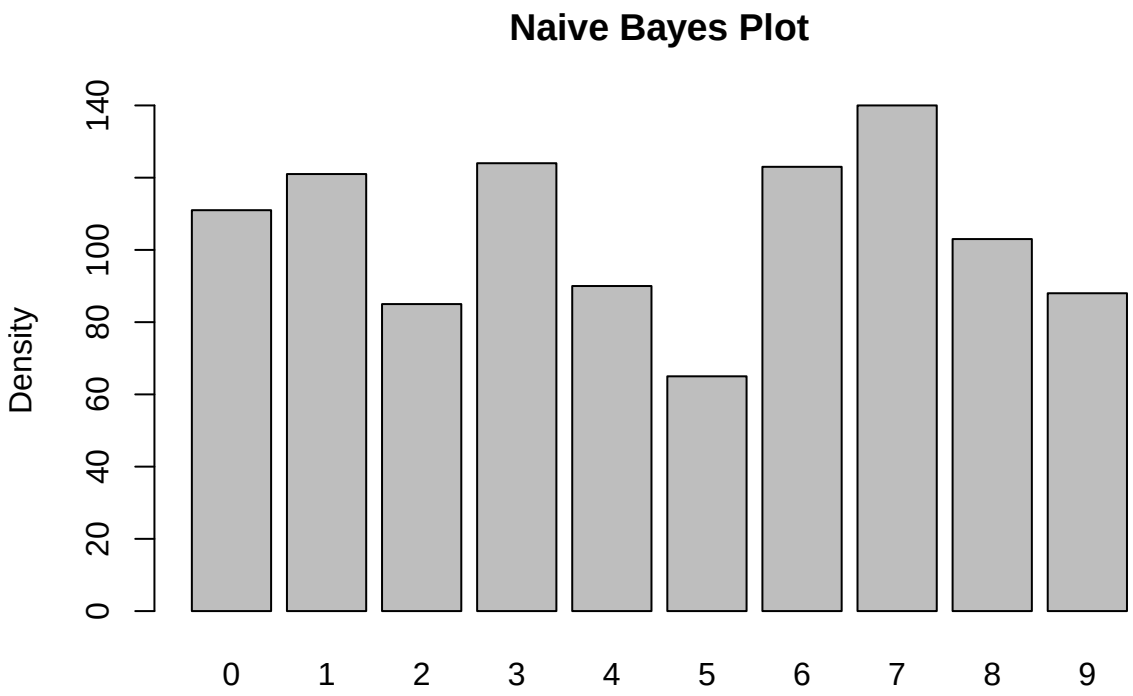
Crossvalidation results

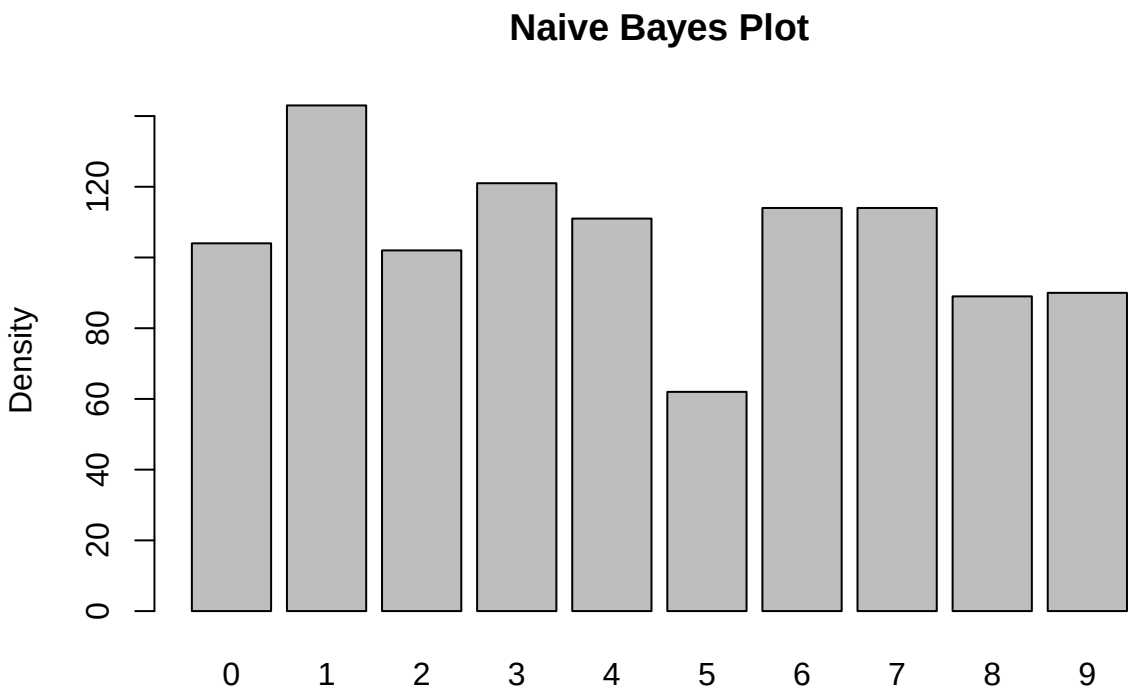
Build the train and test data sets for each fold. Next, train Naive Bayes Classifier on Train Set and test on Test Set. Lastly, present classification results using table and/or confusion matrix!!

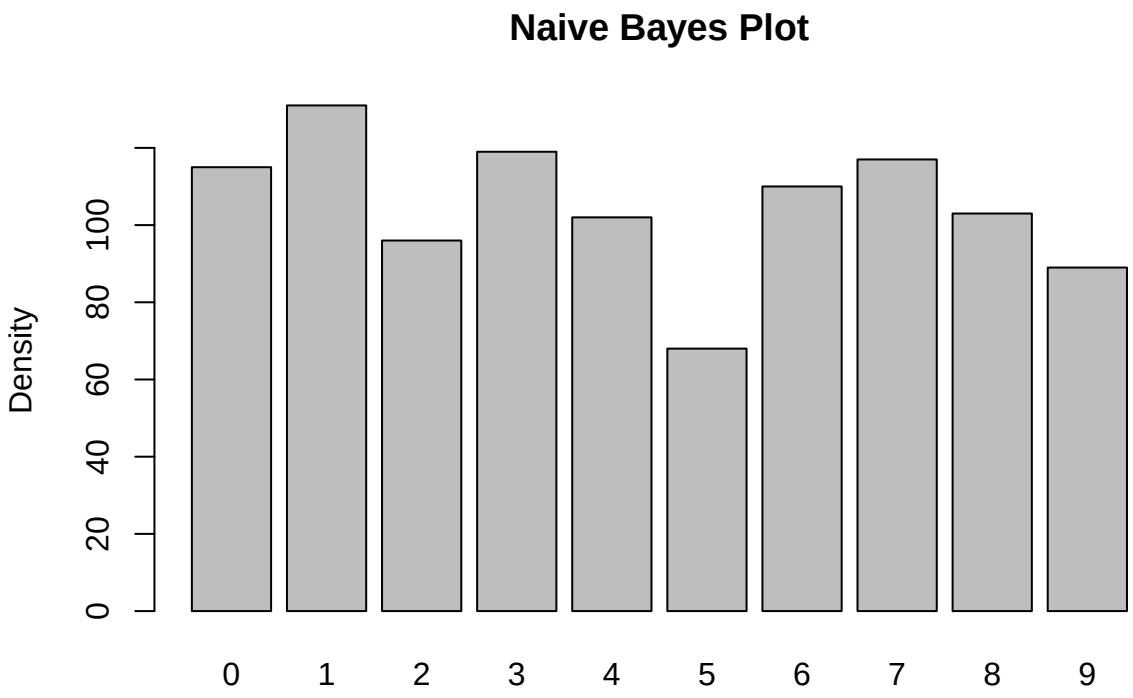
```
#Run training and Testing for each of the k-folds
AllResults<-list()
AllLabels<-list()
for (k in 1:kfolds){

DigitDF_Test <- DigitDF[holdout[[k]], ]
DigitDF_Train=DigitDF[-holdout[[k]], ]
## View the created Test and Train sets
#(head(DigitDF_Train))
#(table(DigitDF_Test$DigitTotalDF.label))
## Make sure you take the labels out of the testing data
#
DigitDF_Test_noLabel<-DigitDF_Test[-c(1)]
```

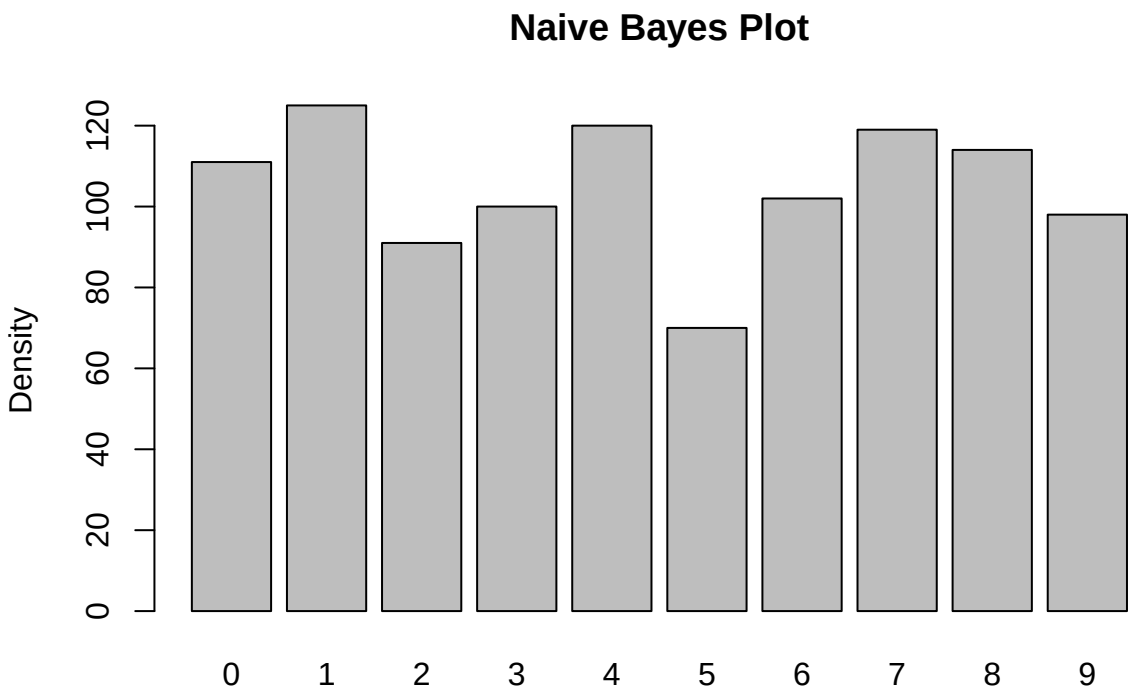
```
DigitDF_Test_justLabel<-DigitDF_Test$DigitTotalDF.label
#(head(DigitDF_Test_noLabel))

#### Naive Bayes prediction ussing e1071 package
#Naive Bayes Train model
train_naibayes<-naiveBayes(DigitTotalDF.label~., data=DigitDF_Train, na.action = na.pass)
#train_naibayes
#summary(train_naibayes)

#Naive Bayes model Prediction
nb_Pred <- predict(train_naibayes, DigitDF_Test_noLabel)
#nb_Pred


#Testing accurancy of naive bayes model with Kaggle train data sub set
(confusionMatrix(nb_Pred, DigitDF_Test$DigitTotalDF.label))

# Accumulate results from each fold, if you like
AllResults<- c(AllResults,nb_Pred)
AllLabels<- c(AllLabels, DigitDF_Test_justLabel)


##Visualize
plot(nb_Pred, ylab = "Density", main = "Naive Bayes Plot")

}
```

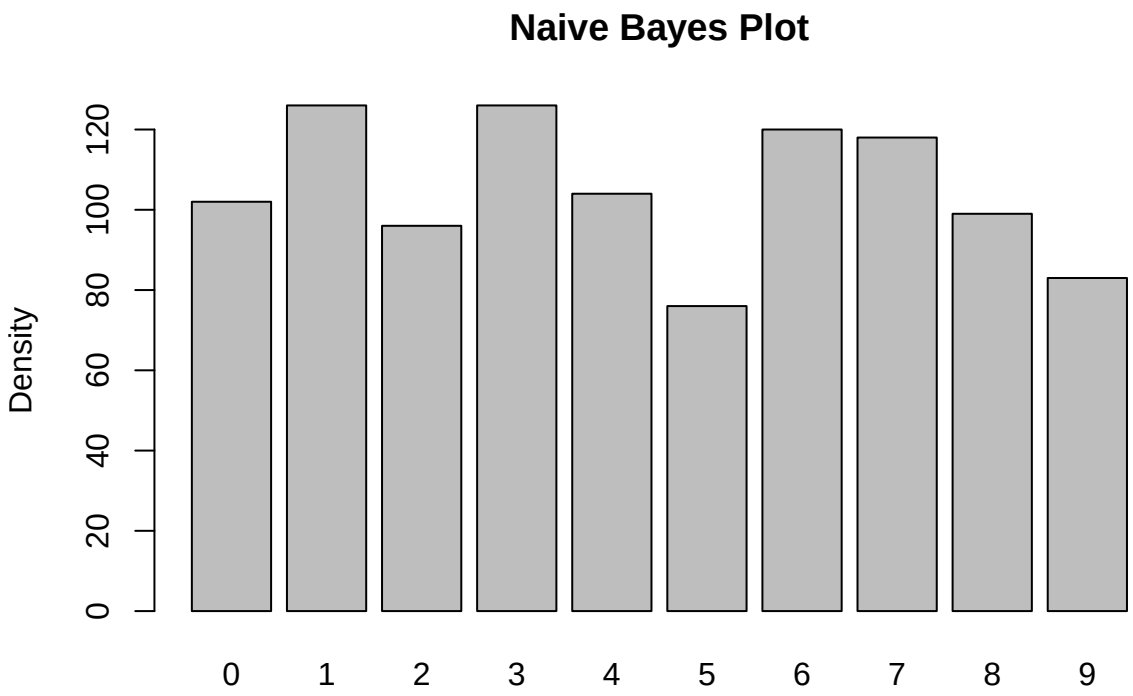

Naive Bayes Plot
Density
0
20
40
60
80
100
0
1
2
3
4
5
6
7
8
9

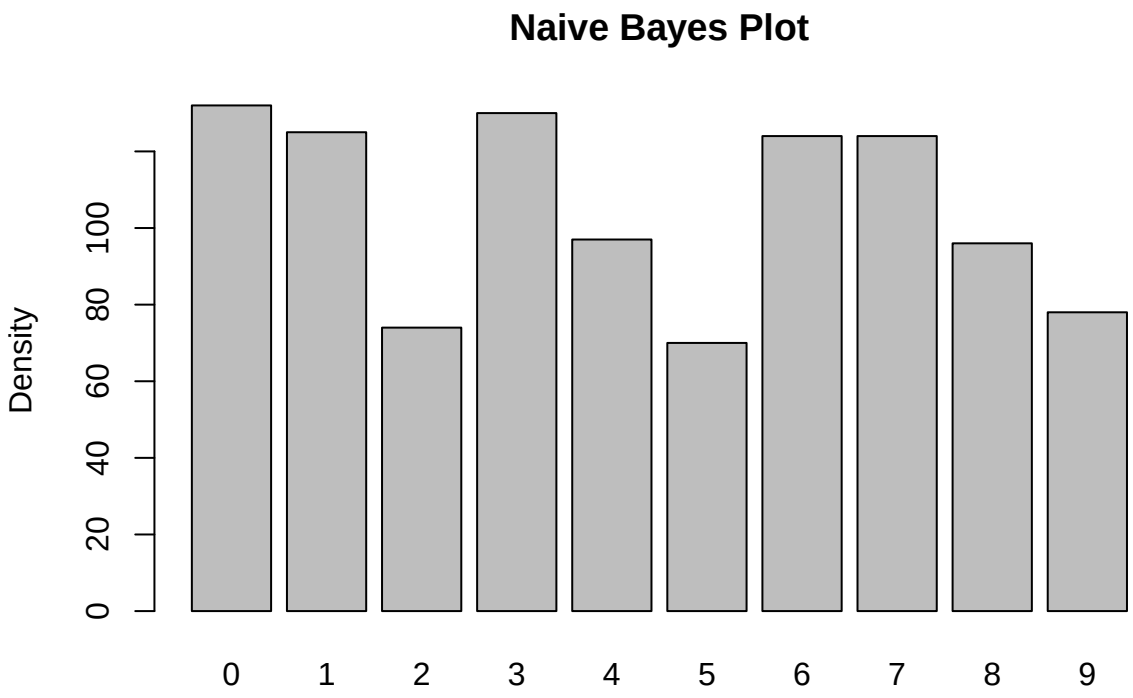

Naive Bayes Plot
Density
0
20
40
60
80
100
140
0
1
2
3
4
5
6
7
8
9

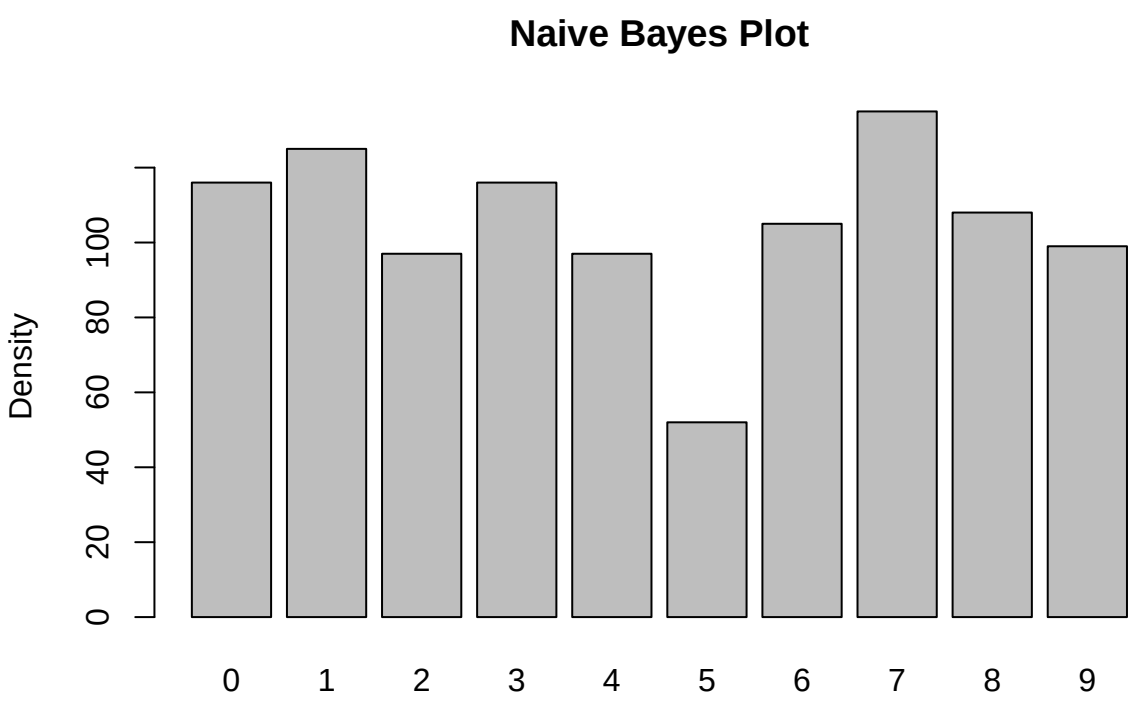

Naive Bayes Plot
Density
0
20
40
60
80
120
0
1
2
3
4
5
6
7
8
9

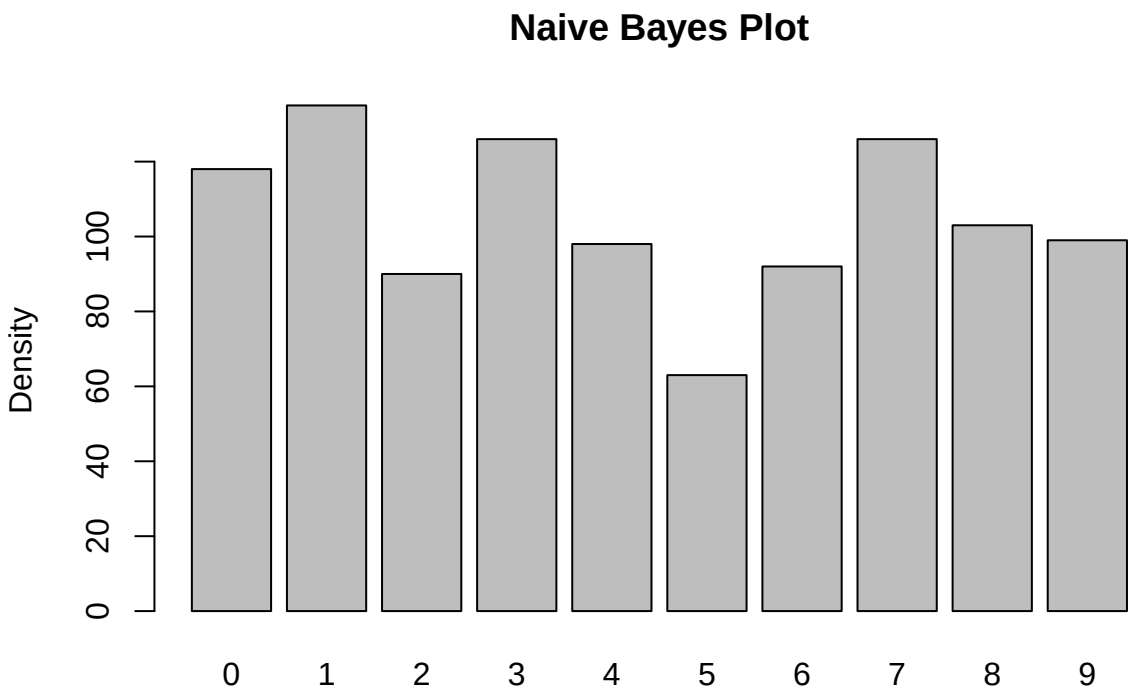

Naive Bayes Plot
Density
0
20
40
60
80
100
0
1
2
3
4
5
6
7
8
9

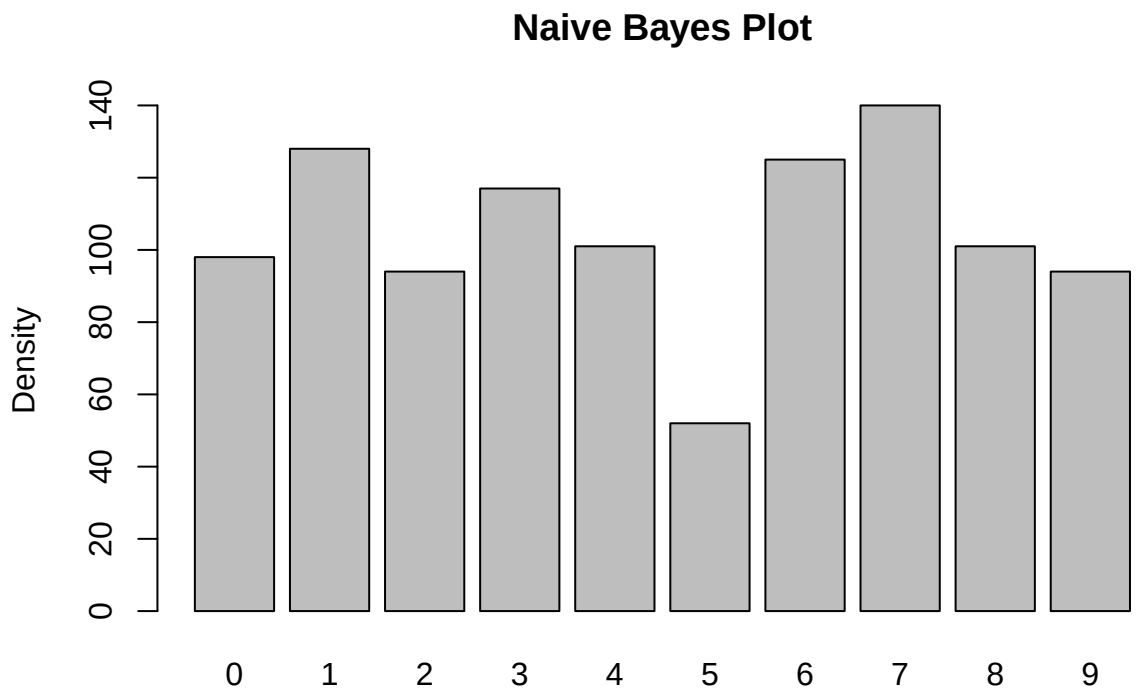

Naive Bayes Plot
Density
0
20
40
60
80
100
120
0
1
2
3
4
5
6
7
8
9


Naive Bayes Plot
Density
0
20
40
60
80
100
120
0
1
2
3
4
5
6
7
8
9


Naive Bayes Plot
Density
0
20
40
60
80
100
0
1
2
3
4
5
6
7
8
9


Naive Bayes Plot
Density
0
20
40
60
80
100
0
1
2
3
4
5
6
7
8
9


Naive Bayes Plot
Density
0
20
40
60
80
100
0
1
2
3
4
5
6
7
8
9



Confusion Matrix (across all folds)

```
### end crossvalidation -- present results for all folds
(table(unlist(AllResults),unlist(AllLabels)))
```

```
##
##        1    2    3    4    5    6    7    8    9   10
##   1  898    0   49    8    5   77   37   13   13    3
##   2    0 1075   20   22    8   33   25   33   57   22
##   3    5   61  533   30   18   40  155    7   72    2
##   4    3   35   43  773    1  183   18    1  138   15
##   5    8    1   41   10  536   27   50   92   19  235
##   6   56    2   22   81    1  407   12   20   34    9
##   7   14    0  293    8   69   19  649    5   29   22
##   8    0   15    3    3   91   71    0  790   10  274
##   9   27   27   59  152   19   53   99   18  556   19
##   10   1    1    2   22  228   14    1  133   46  464
```

```
#########Testing with Kaggle sample#########
#TestTotalDF_noLabel<-TestTotalDF[-c(1)]
#(head(TestTotalDF_noLabel))

#nb_Pred <- predict(train_naibayes, TestTotalDF_noLabel)
```

```
#nb_Pred

### Export naive Bayesbest result and run through Kaggle

#nbTestPred <- predict(train_naibayes,TestTotalDF, type = 'class')
#nbTestPred <- data.frame(nbTestPred)
#colnames(nbTestPred)[1] <- 'Label'
#nbTestPred$ImageId <- 1:nrow(nbTestPred)
#nbTestPred <- nbTestPred %>% select(ImageId, Label)

#write.csv(nbTestPred, 'Naive Bayes Classifier.csv', row.names = FALSE)
```

Next: Try running a similar cross validation experiment using Decision Trees!!! Compare the results ... which classifier performed best????