# Fashion MNIST

**Group Member 1 Name:** Ian Driscoll **Group Member 1 SID:** 3031896752

The dataset contains $n = 18,000$ different $28 \times 28$ grayscale images of clothing, each with a label of either *shoes*, *shirt*, or *pants* (6000 of each). If we stack the features into a single vector, we can transform each of these observations into a single $28 * 28 = 784$ dimensional vector. The data can thus be stored in a $n \times p = 18000 \times 784$ data matrix $\mathbf{X}$, and the labels stored in a $n \times 1$ vector $\mathbf{y}$.

Once downloaded, the data can be read as follows.

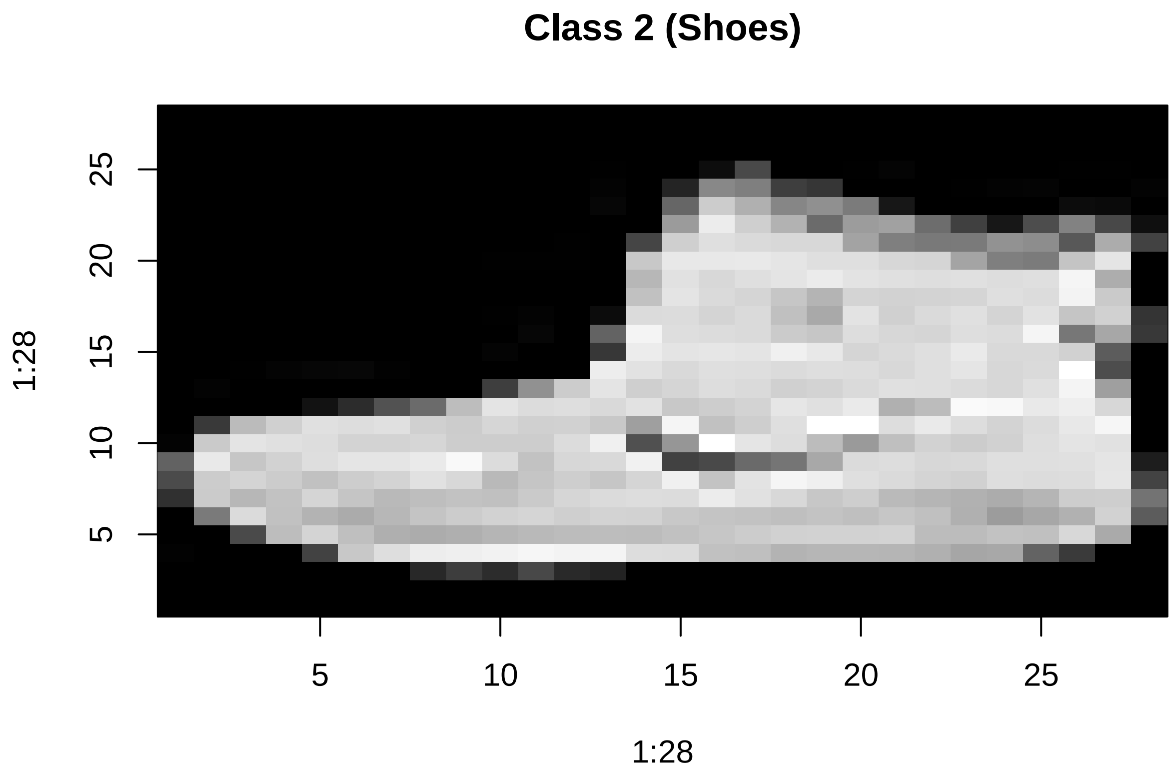
```r
library(readr)
FMNIST <- read_csv("FashionMNIST.csv")
```

```
## Parsed with column specification:
## cols(
##   .default = col_double()
## )

## See spec(...) for full column specifications.
```

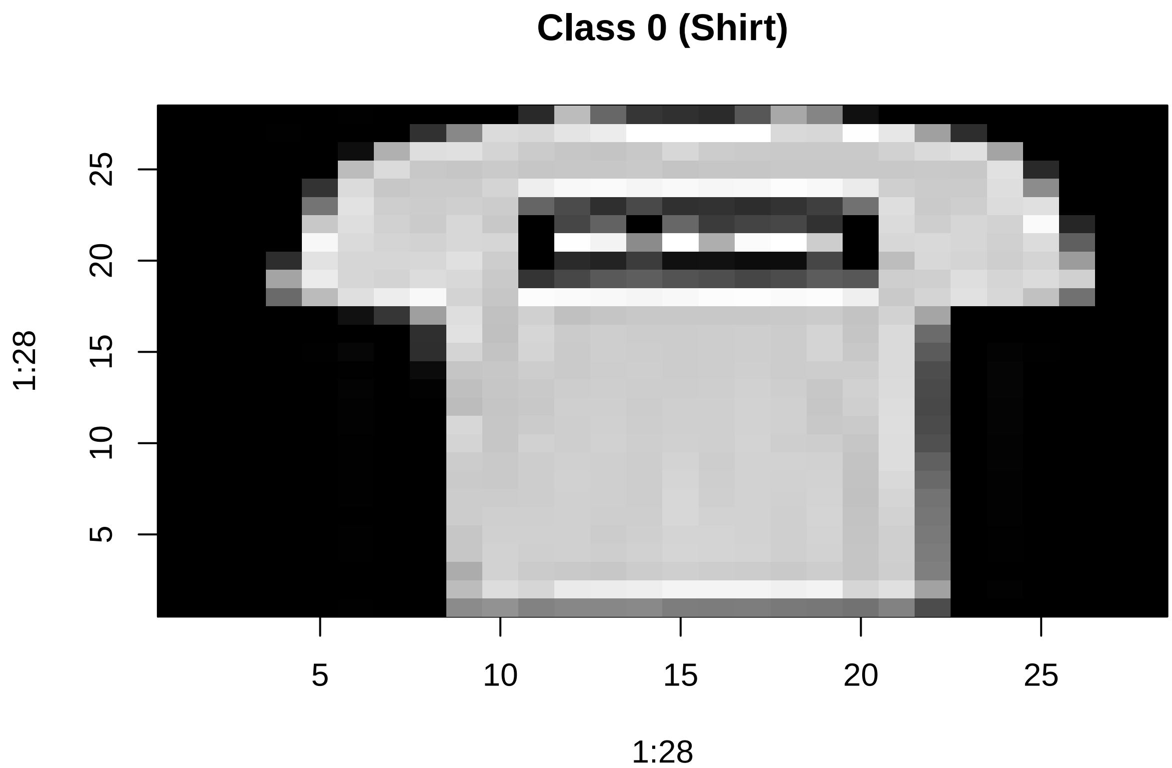
```r
y <- FMNIST$label
X <- subset(FMNIST, select = -c(label))
#rm('FMNIST') #remove from memory -- it's a relatively large file
print(dim(X))
```

We can look at a few of the images:

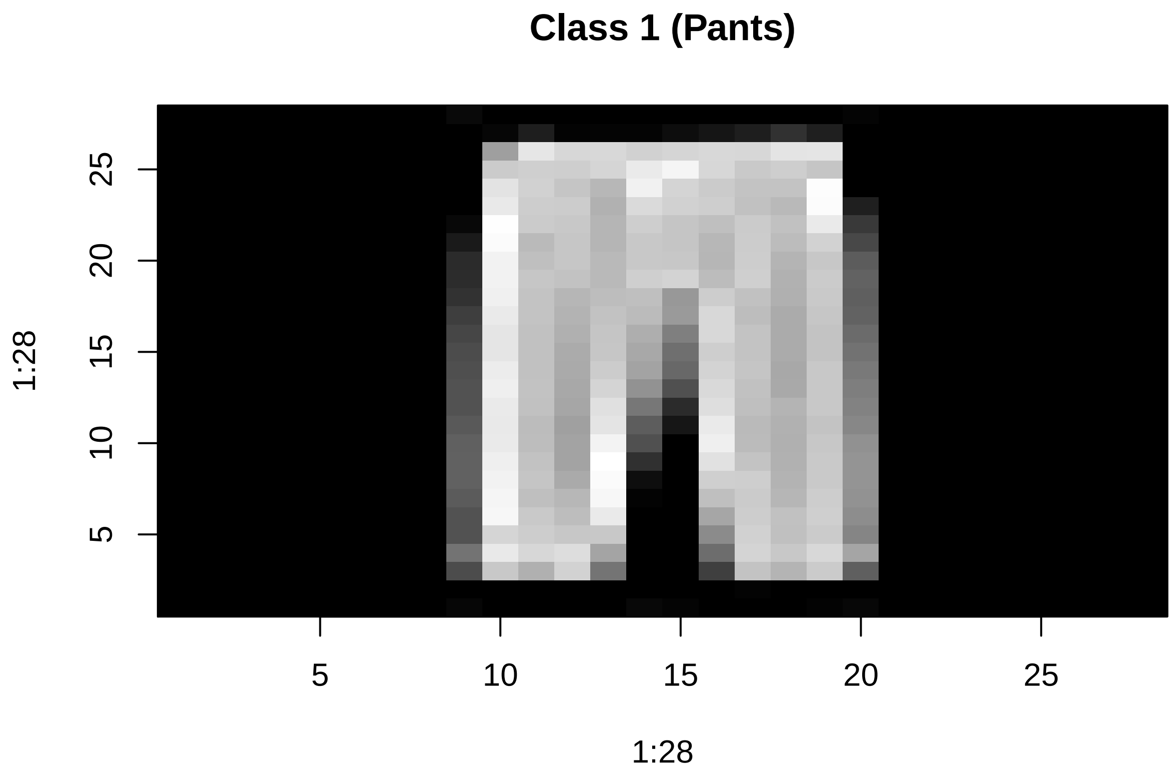
```r
X2 <- matrix(as.numeric(X[1,]), ncol=28, nrow=28, byrow = TRUE)
X2 <- apply(X2, 2, rev)
image(1:28, 1:28, t(X2), col=gray((0:255)/255), main='Class 2 (Shoes)')
```




```
X0 <- matrix(as.numeric(X[2,]), ncol=28, nrow=28, byrow = TRUE)
X0 <- apply(X0, 2, rev)
image(1:28, 1:28, t(X0), col=gray((0:255)/255), main='Class 0 (Shirt)')
```




```r
X1 <- matrix(as.numeric(X[10,]), ncol=28, nrow=28, byrow = TRUE)
X1 <- apply(X1, 2, rev)
image(1:28, 1:28, t(X1), col=gray((0:255)/255), main='Class 1 (Pants)')
```




## Data exploration and dimension reduction

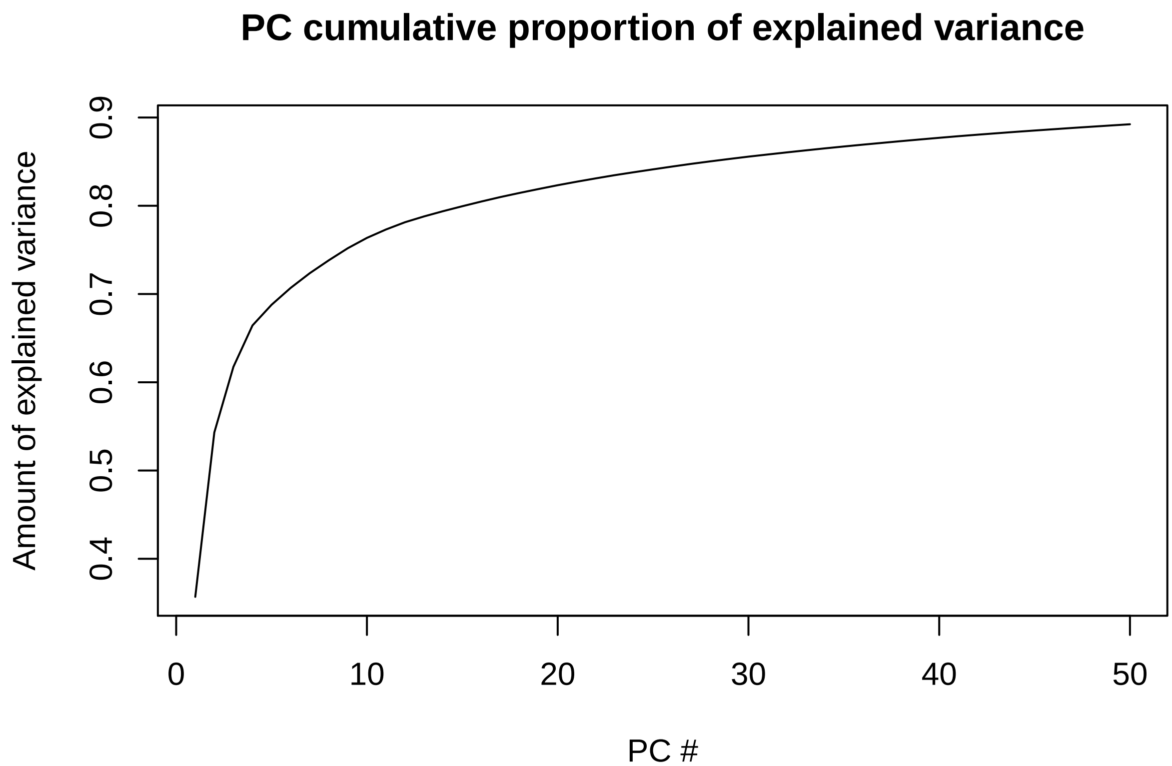
In this step, I will use PCA in order to reduces the dimensions of this dataset.

```
library(MASS)
pca = prcomp(X)
importance = summary(pca)
cumprop = cumsum(importance$sdev^2 / sum(importance$sdev^2))
plot(cumprop[0:50],
     xlab = "PC #",
     ylab = "Amount of explained variance",
     main = "PC cumulative proportion of explained variance",
     type="l")
cumprop[49]
```

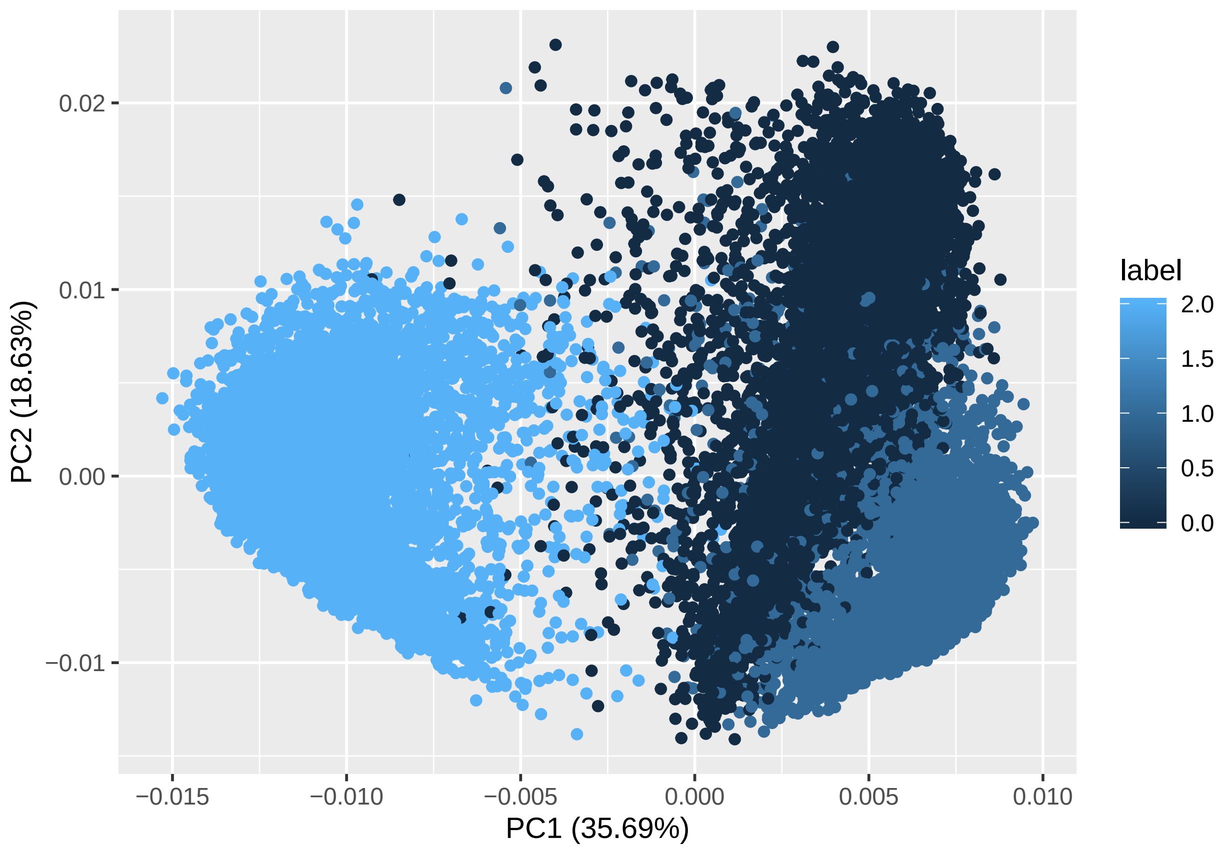
```
## [1] 0.8910128
```

```
reducedX = data.frame(pca$x[,1:49])
library(ggfortify)
```

```
## Loading required package: ggplot2
```




```r
data = data.frame(FMNIST)
autoplot(pca, data = data, colour = 'label',)
```




By applying PCA and inspecting the scree plot, we can see that about 90% of the variance is explained by the first 50 PCs. If we think about this logically, we can think of that as a 7 by 7 grid by including 49 PCs, which is reducing the dimensions overall by a factor of 16. Including 49 components accounts for approximately 89.1% of the variance. Additionally, in the plot above, we can see that the first two PCs alone display a bit of separation between the three classes.

# Classification task

## Binary classification

Our first step will be to split our dataset into a training set and test set. I will use the proportion of 75/25.

```
smp_size <- floor(0.75 * nrow(reducedX))

set.seed(123)
train_ind = sample(seq_len(nrow(reducedX)), size = smp_size)

trainX = reducedX[train_ind, ]
testX = reducedX[-train_ind, ]
trainY = y[train_ind]
testY = y[-train_ind]
```

We will first look at binary classifiers for shirt(class 0) vs. pants(class 1). I am using the reduced dataset of the first 49 components. Due to the nature of the reduced data, the two classifiers I will compare are K-Nearest Neighbors and Random Forest.

```r
trainX01 = trainX[trainY != 2,]
testX01 = testX[testY != 2,]
trainY01 = trainY[trainY != 2]
testY01 = testY[testY != 2]

# KNN
library(class)
library(caret)
```

```
## Loading required package: lattice
```

```r
train = data.frame(trainX01, "class"=trainY01)
test = data.frame(testX01, "class"=testY01)
output = knn(train, test, cl=trainY01)
a = confusionMatrix(as.factor(output), as.factor(testY01))
a
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1480   12
##          1   12 1493
##
##                Accuracy : 0.992
##                  95% CI : (0.9881, 0.9949)
##     No Information Rate : 0.5022
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.984
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9920
##             Specificity : 0.9920
##          Pos Pred Value : 0.9920
##          Neg Pred Value : 0.9920
##              Prevalence : 0.4978
##          Detection Rate : 0.4938
##    Detection Prevalence : 0.4978
##       Balanced Accuracy : 0.9920
##
##        'Positive' Class : 0
##
```

```r
library(randomForest)
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
m1 = randomForest(
  formula = class ~ .,
  x = train[,1:49],
  y = as.factor(train[,50]),
  data = train,
  xtest = testX01,
  ytest = as.factor(testY01)
)

m1
```

```
##
## Call:
##  randomForest(x = train[, 1:49], y = as.factor(train[, 50]), xtest = testX01,      ytest = as.factor
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 1.22%
## Confusion matrix:
##      0    1 class.error
## 0 4495   13 0.002883762
## 1   97 4398 0.021579533
##                 Test set error rate: 0.97%
## Confusion matrix:
##      0    1 class.error
## 0 1485    7 0.004691689
## 1   22 1483 0.014617940
```

We will next examine a binary classifier between shirt(0) and shoes(2).

```
trainX02 = trainX[trainY != 1,]
testX02 = testX[testY != 1,]
trainY02 = trainY[trainY != 1]
testY02 = testY[testY != 1]

# KNN
library(class)
library(caret)
train = data.frame(trainX02, "class"=trainY02)
test = data.frame(testX02, "class"=testY02)
output = knn(train, test, cl=trainY02)
confusionMatrix(as.factor(output), as.factor(testY02))
```

```
## Confusion Matrix and Statistics
```

```
##
##           Reference
## Prediction    0    2
##          0 1490    1
##          2    2 1502
##
##                Accuracy : 0.999
##                  95% CI : (0.9971, 0.9998)
##     No Information Rate : 0.5018
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.998
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9987
##             Specificity : 0.9993
##          Pos Pred Value : 0.9993
##          Neg Pred Value : 0.9987
##              Prevalence : 0.4982
##          Detection Rate : 0.4975
##    Detection Prevalence : 0.4978
##       Balanced Accuracy : 0.9990
##
##        'Positive' Class : 0
##
```

```r
library(randomForest)
m1 = randomForest(
  formula = class ~ .,
  x = train[,1:49],
  y = as.factor(train[,50]),
  data = train,
  xtest = testX02,
  ytest = as.factor(testY02)
)

m1
```

```
##
## Call:
##  randomForest(x = train[, 1:49], y = as.factor(train[, 50]), xtest = testX02,      ytest = as.factor
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 0.12%
## Confusion matrix:
##      0    2 class.error
## 0 4503    5 0.001109139
## 2    6 4491 0.001334223
##                 Test set error rate: 0.17%
## Confusion matrix:
##      0    2 class.error
```

```
## 0 1490    2 0.001340483
## 2    3 1500 0.001996008
```

Finally, we will look at pants(1) and shoes(2).

```
trainX12 = trainX[trainY != 0,]
testX12 = testX[testY != 0,]
trainY12 = trainY[trainY != 0]
testY12 = testY[testY != 0]

# KNN
library(class)
library(caret)
train = data.frame(trainX12, "class"=trainY12)
test = data.frame(testX12, "class"=testY12)
output = knn(train, test, cl=trainY12)
confusionMatrix(as.factor(output), as.factor(testY12))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    1    2
##          1 1505    0
##          2    0 1503
##
##                Accuracy : 1
##                  95% CI : (0.9988, 1)
##     No Information Rate : 0.5003
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0000
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 1.0000
##              Prevalence : 0.5003
##          Detection Rate : 0.5003
##    Detection Prevalence : 0.5003
##       Balanced Accuracy : 1.0000
##
##        'Positive' Class : 1
##
```

```
library(randomForest)
m1 = randomForest(
  formula = class ~ .,
  x = train[,1:49],
  y = as.factor(train[,50]),
  data = train,
  xtest = testX12,
```

```r
  ytest = as.factor(testY12)
)

m1
```

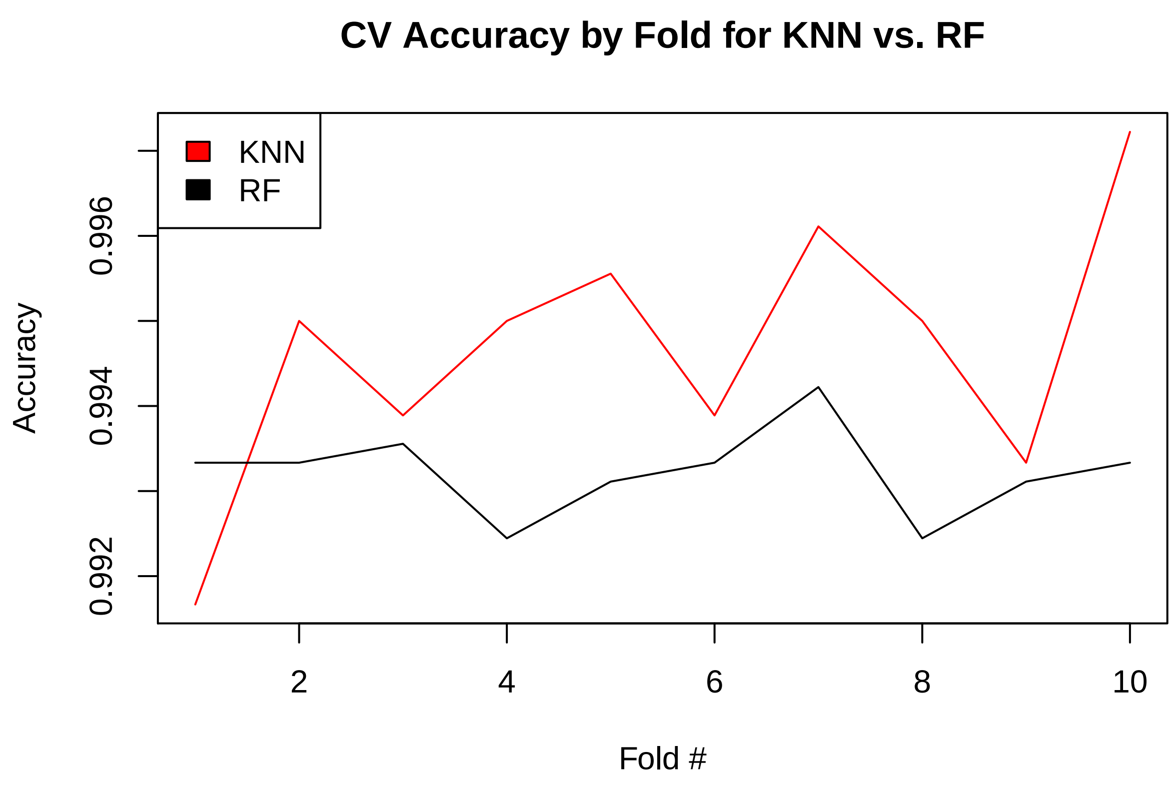
```
##
## Call:
##  randomForest(x = train[, 1:49], y = as.factor(train[, 50]), xtest = testX12,      ytest = as.factor
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 0.11%
## Confusion matrix:
##      1    2  class.error
## 1 4489    6 0.0013348165
## 2    4 4493 0.0008894819
##                 Test set error rate: 0.07%
## Confusion matrix:
##      1    2 class.error
## 1 1505    0 0.000000000
## 2    2 1501 0.001330672
```

## Multiclass classification

Examining the binary classification, we can see that both models perform very highly on our training data. We will use both KNN and Random Forest to apply a multiclass classifier to our entire dataset. In order to validate our performance, we will use 10-fold cross validation, splitting the data into 10 folds and using the Leave-One-Out method of training/testing.

```r
library(class)
library(caret)


train = data.frame(trainX, "class"=trainY)
test = data.frame(testX, "class"=testY)
data = rbind(train, test)
set.seed(123)
data = data[sample(nrow(data)),]

folds = cut(seq(1,nrow(data)),breaks=10,labels=FALSE)

accs_knn = c()
accs_rf = c()
for(i in 1:10) {
  testIndexes = which(folds==i,arr.ind=TRUE)
  testData = data[testIndexes, ]
  trainData = data[-testIndexes, ]
  output = knn(trainData, testData, cl=trainData$class)
  result = confusionMatrix(as.factor(output), as.factor(testData$class))
  accs_knn = c(accs_knn, result$overall["Accuracy"])
  library(randomForest)
```

```r
  m1 = randomForest(
    formula = class ~ .,
    x = train[,1:49],
    y = as.factor(train[,50]),
    data = train,
    xtest = testX,
    ytest = as.factor(testY)
  )

  min_index = which.min(m1$test$err.rate[,1])
  acc_rf = 1 - m1$test$err.rate[,1][min_index]
  accs_rf = c(accs_rf, acc_rf)
}

plot(accs_knn, type="l", col="red",
     main="CV Accuracy by Fold for KNN vs. RF",
     xlab="Fold #", ylab="Accuracy")
lines(accs_rf)
legend("topleft", c("KNN","RF"), fill=c("red", "black"))
```




```r
paste0("KNN CV 10 fold average accuracy: ", toString(mean(accs_knn)))
```

```
## [1] "KNN CV 10 fold average accuracy: 0.994666666666667"
```

```r
paste0("RF CV 10 fold average accuracy: ", toString(mean(accs_rf)))
```

```
## [1] "RF CV 10 fold average accuracy: 0.993222222222222"
```

We can see that averaged across all folds, the K-Nearest Neighbors classifier performs slightly higher than the Random Forest classifier. Therefore, our optimal model will be a K-Nearest Neighbors classifier built on the reduced dataset we obtained with the first 49 Principal Components.