Fashion MNIST

Group Member 1 Name: Edward Zamora **Group Member 1 SID:** 3032137148

The dataset contains $n = 18,000$ different 28×28 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.

```
set.seed(1)
library(readr)
library(ISLR)
library(tree)
library(randomForest)
```

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

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

```
FMNIST <- read_csv("FashionMNIST.csv")
```

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

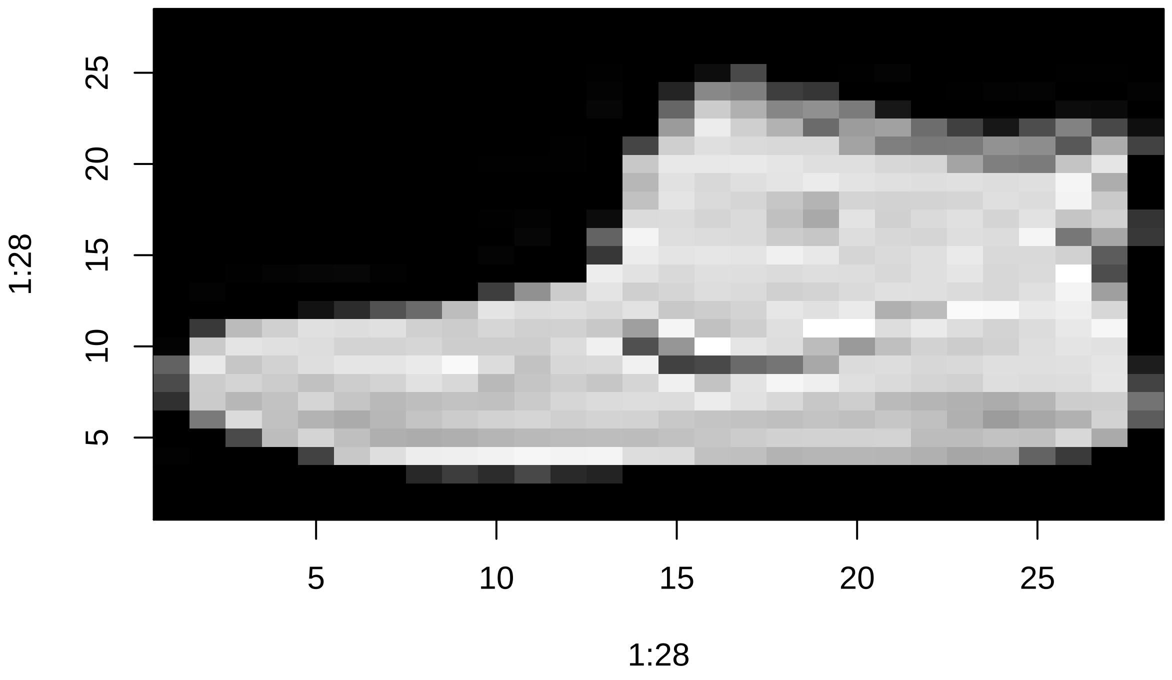
```
## See spec(...) for full column specifications.
```

```
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:

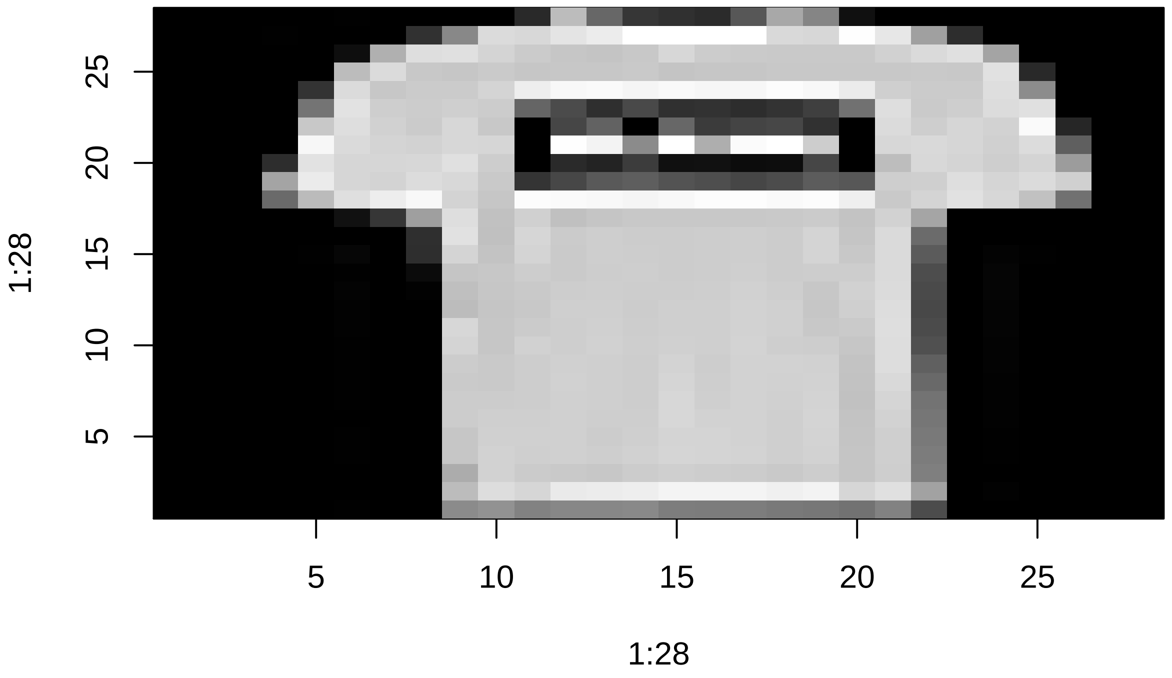
```
X2 <- matrix(as.numeric(X[1,]), ncol=28, nrow=28, byrow = TRUE)
X2 <- apply(X2, 2, rev)
image(1:28, 1:28, t(X2), col=gray((0:255)/255), main='Class 2 (Shoes)')
```

Class 2 (Shoes)



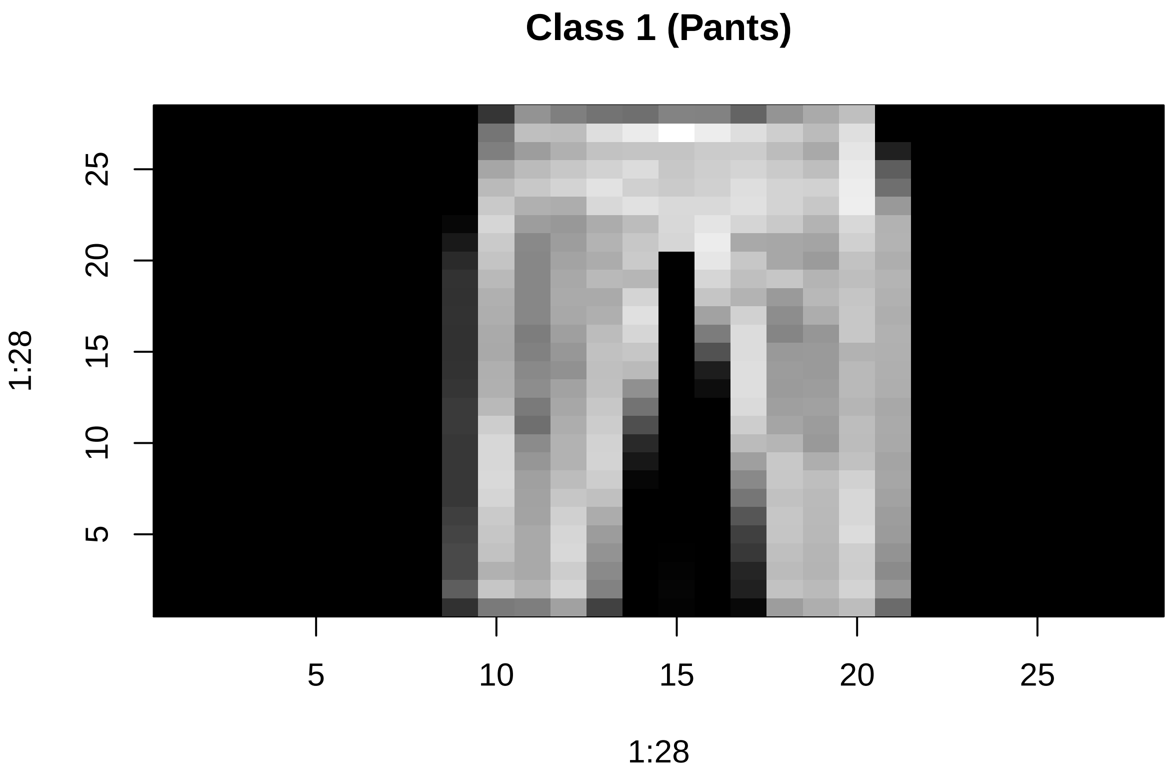
```r
X0 <- matrix(as.numeric(X[2,]), ncol=28, nrow=28, byrow = TRUE)
X0 <- apply(X0, 2, rev)
image(1:28, 1:28, t(X0), col=gray((0:255)/255), main='Class 0 (Shirt)')
```

Class 0 (Shirt)



```r
X1 <- matrix(as.numeric(X[8,]), ncol=28, nrow=28, byrow = TRUE)
X1 <- apply(X1, 2, rev)
```

```r
image(1:28, 1:28, t(X1), col=gray((0:255)/255), main='Class 1 (Pants)')
```



Data exploration and dimension reduction

In this section, you will experiment with representing the images in fewer dimensions than $28 * 28 = 784$. You can use any of the various dimension reduction techniques introduced in class. How can you visualize these lower dimensional representations as images? How small of dimensionality can you use and still visually distinguish images from different classes?

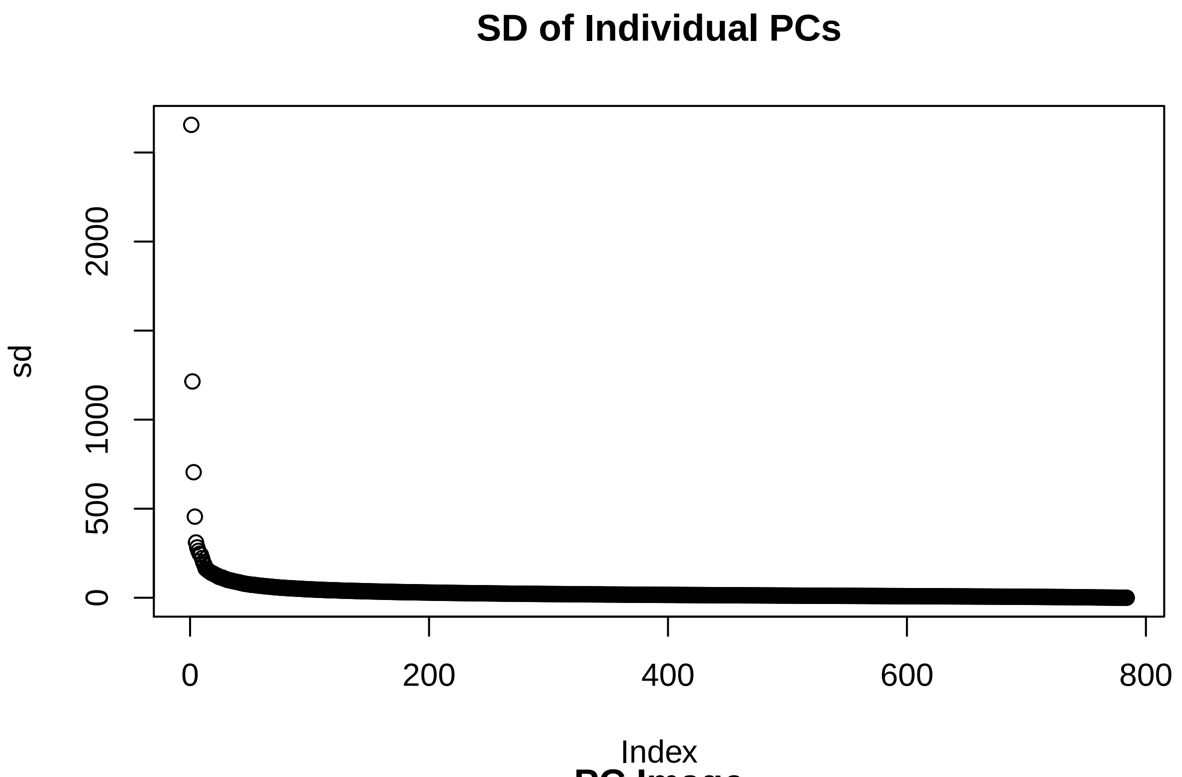
Holdout

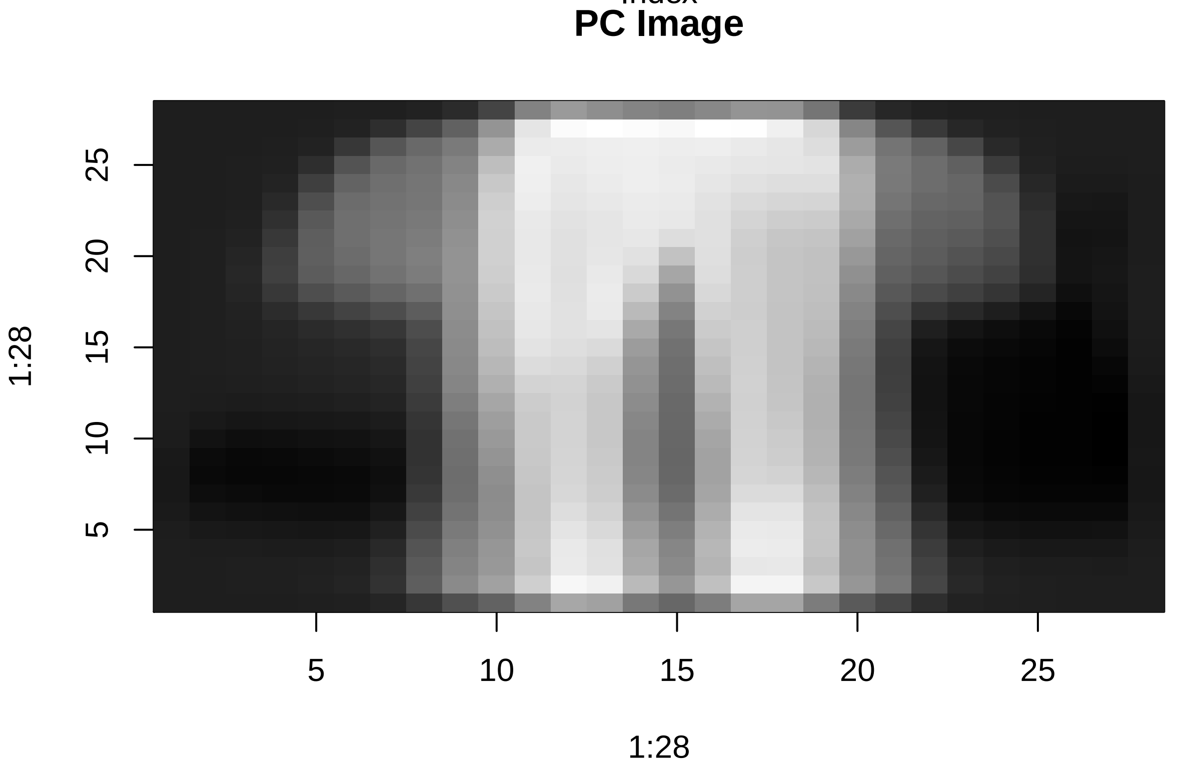
```r
train <- sample(18000,18000*.7)
test <- c(1:18000)[-train][sample(18000*.3,18000*.15)]
validation <- c(1:18000)[-c(train,test)]
```

The first step of my analysis consists of splitting the given data into three segments, a training set, a validation set, and a test set. The reason we use a 3 way split rather than a simple training and test set is that, aside from training the models, we must be able to compare the results returned by each evaluated on the validation set and, once the model has been chosen, we can determine its expected accuracy by evaluating on the test set.

PCA

```r
pcamod <- prcomp(X[train,],scale. = FALSE, center = FALSE)
```

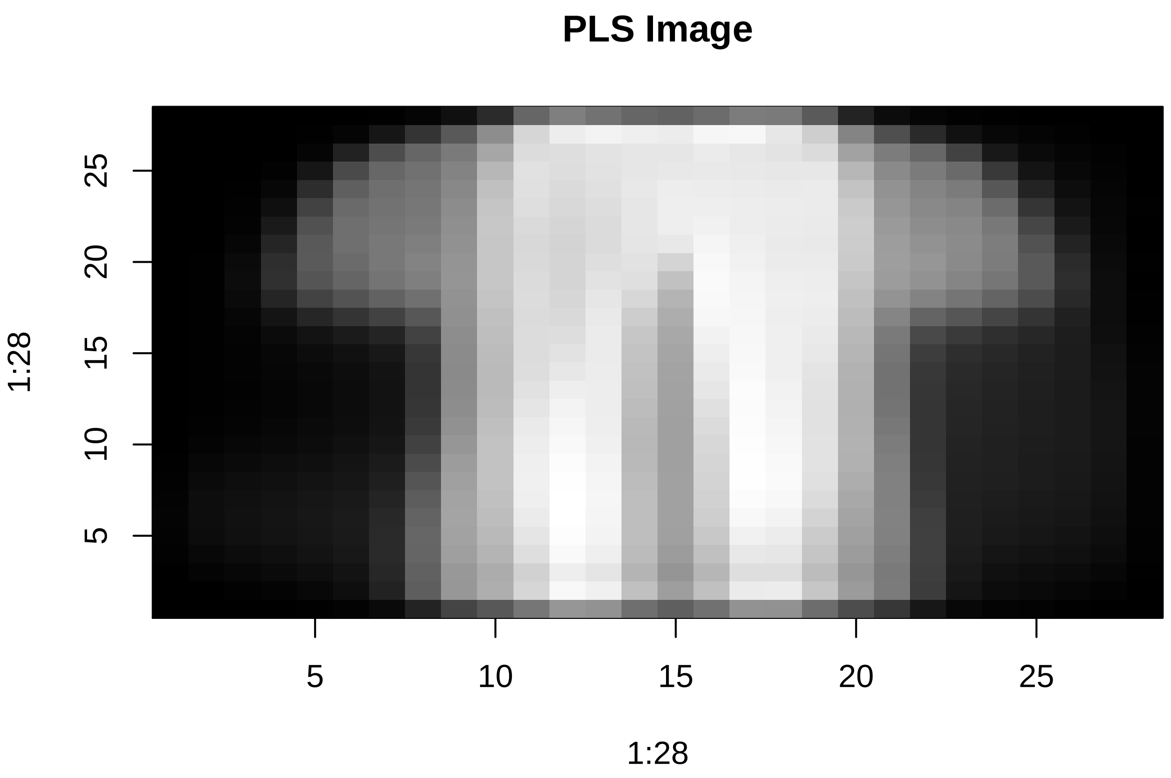




One of the main methods for dimension reduction discussed in the course was principle component analysis. I didn't find it necessary to scale or center the data in any way since all predictors considered are simply variations of the same pixel and centering would also have the adverse effect of creating negative values which are not considered within the grayscale range of each pixel. Of the principle components returned, only 2 dimensions were necessary in reconstructing a comprehensible representation of the original image, as depicted above. (Normally, we would choose a greater number of pc's according to some predetermined minimum value or to increase the total amount of variance captured but, for the purposes of the only making the image recognizable, 2 were needed.)

PLS

```
x <- as.matrix(X[train,])
Y <- as.matrix(y[train])
r <- 2
z <- matrix(nrow = nrow(x) ,ncol = r)
proj <- matrix(0,nrow = nrow(x),ncol = ncol(x))
for (h in 1:r){
  w <- (t(x)%*%Y)/(t(Y)%*%Y)[1]
  w <- w/sqrt(sum(w^2))
  z[,h] <- x%*%w
  p <- t(x)%*%z[,h]/(t(z[,h])%*%z[,h])[1]
  b <- t(Y)%*%z[,h]/(t(z[,h])%*%z[,h])[1]
  proj <- proj+z[,h]%*%t(p)
  x <- x-z[,h]%*%t(p)
  Y <- Y-t(b%*%z[,h])
}
```

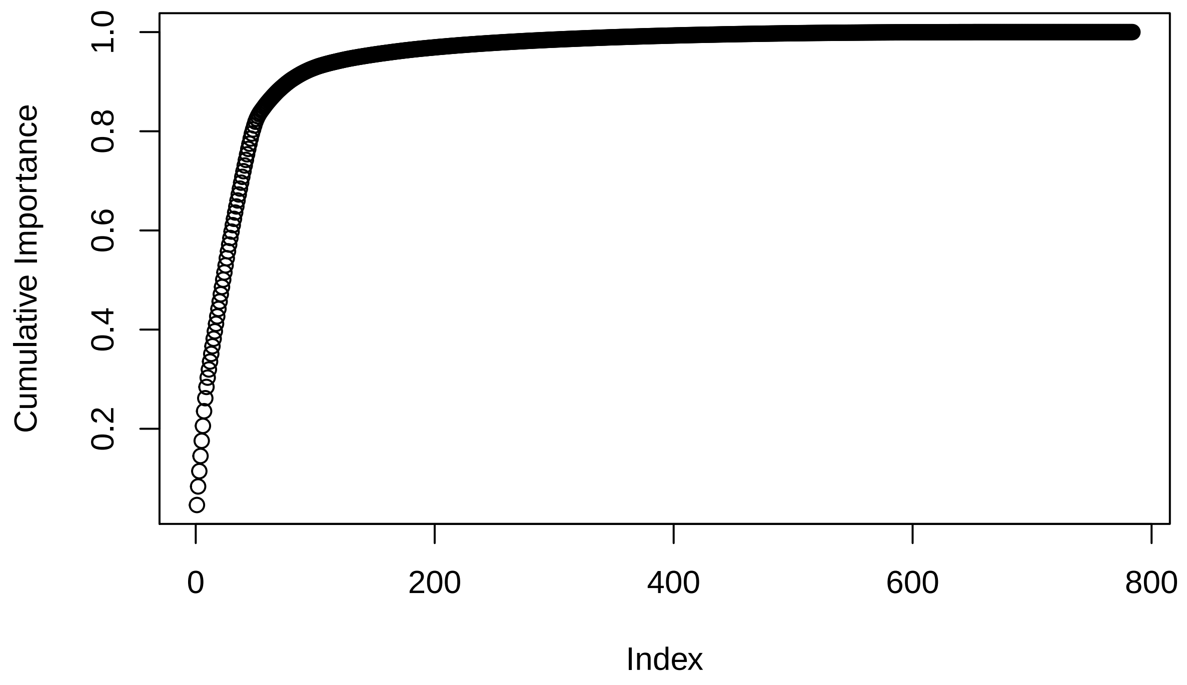


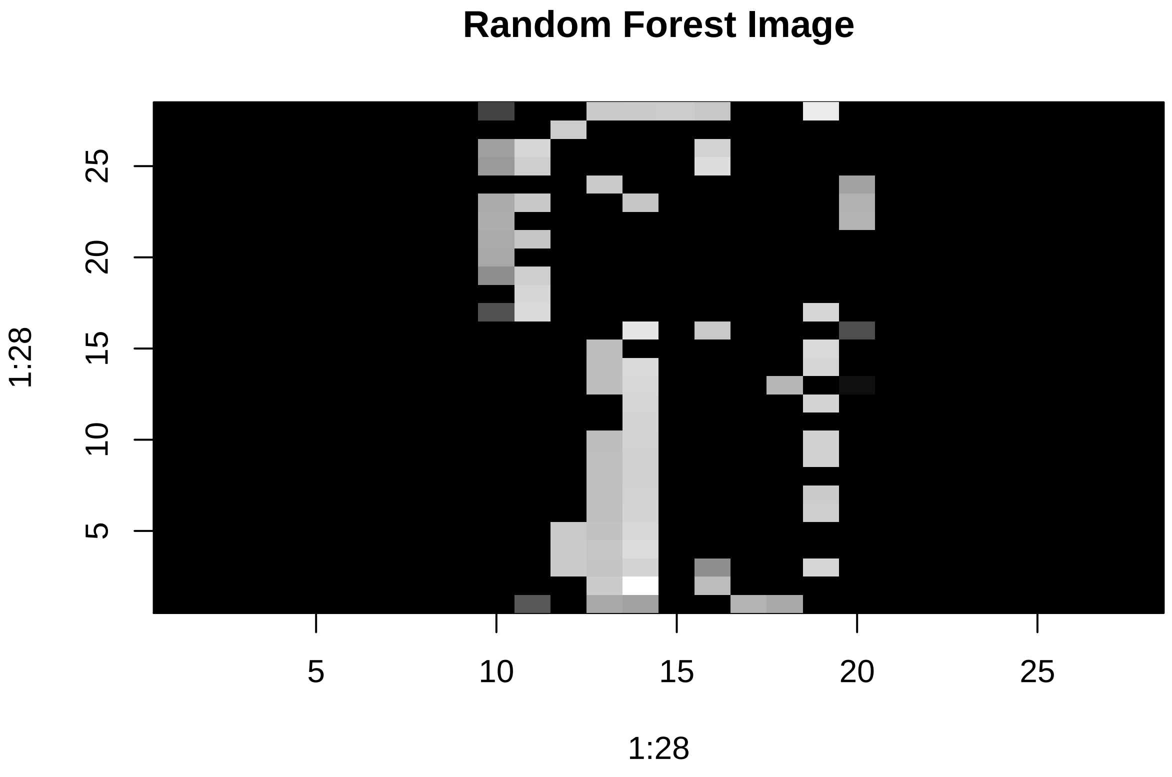
The second dimension reduction method attempted was partial least squares. Although usually used in regression settings, we are able to recreate a reduced version of the original image from the components returned very similarly to the PCA method. By projecting the original data onto these components, we can form a compressed version of the image. Since there was no straightforward way to determine the amount of variance explained by the components, determining the appropriate number was mainly by eye. From the given image, correct item of clothing can be clearly seen so only 2 dimensions are needed from pls. Interestingly, this image is very similar to the image given by PCA suggesting that 2 dimensions are the most efficient reduction to the data.

Random Forest

```
clothes <- ifelse(y==2,'shoe',ifelse(y==1,'pants','shirt'))
img <- data.frame(X,y=clothes)
forest.img =randomForest(y~.,img,subset = train,mtry=28,ntree=30)
```

Cumulative Importance of Random Forest





The last method I attempted for dimension reduction was choosing the most important predictors from a random tree model. Setting the number of random predictors sampled for each tree to the square root of the total number of predictors, 28, I found the minimum number of most important predictors that still allowed me to recognize the image by estimating the value based on cumulative imirtance, then systematically finding the minimum number which allowed for reasonable recognition. For the missing values, I choose a default value of 0 in the image's reconstruction. As expected, this method returned the least efficient means of dimension reductionas it relies on no transformations to the data, only the elimination of certain values.

Classification task

Binary classification

In this section, you should use the techniques learned in class to develop a model for binary classification of the images. More specifically, you should split up the data into different pairs of classes, and fit several binary classification models. For example, you should develop a model to predict shoes vs shirts, shoes vs pants, and pants vs shirts.

Remember that you should try several different methods, and use model selection methods to determine which model is best. You should also be sure to keep a held-out test set to evaluate the performance of your model.

Must use either logistic regression, random forrests, discriminant analysis

Segment Data

```
img_train_01 <-  data.frame(X[train,],label=y[train])[y[train]!=2,]
img_val_01 <- data.frame(X[validation,],label=y[validation])[y[validation]!=2,]
img_test_01 <-  data.frame(X[test,],label=y[test])[y[test]!=2,]

img_train_12 <-  data.frame(X[train,],label=y[train])[y[train]!=0,]
img_val_12 <- data.frame(X[validation,],label=y[validation])[y[validation]!=0,]
img_test_12 <-  data.frame(X[test,],label=y[test])[y[test]!=0,]

img_train_02 <-  data.frame(X[train,],label=y[train])[y[train]!=1,]
img_val_02 <- data.frame(X[validation,],label=y[validation])[y[validation]!=1,]
img_test_02 <-  data.frame(X[test,],label=y[test])[y[test]!=1,]
```

The first step in my binary classification was separating the data into appropriate subsets that only contained 1 of 2 classifications. Using the 3 way partition created in the previous part, i was able to segment these even further into training, validation, and test sets for each classification pair.

01 (Shirts and Pants):

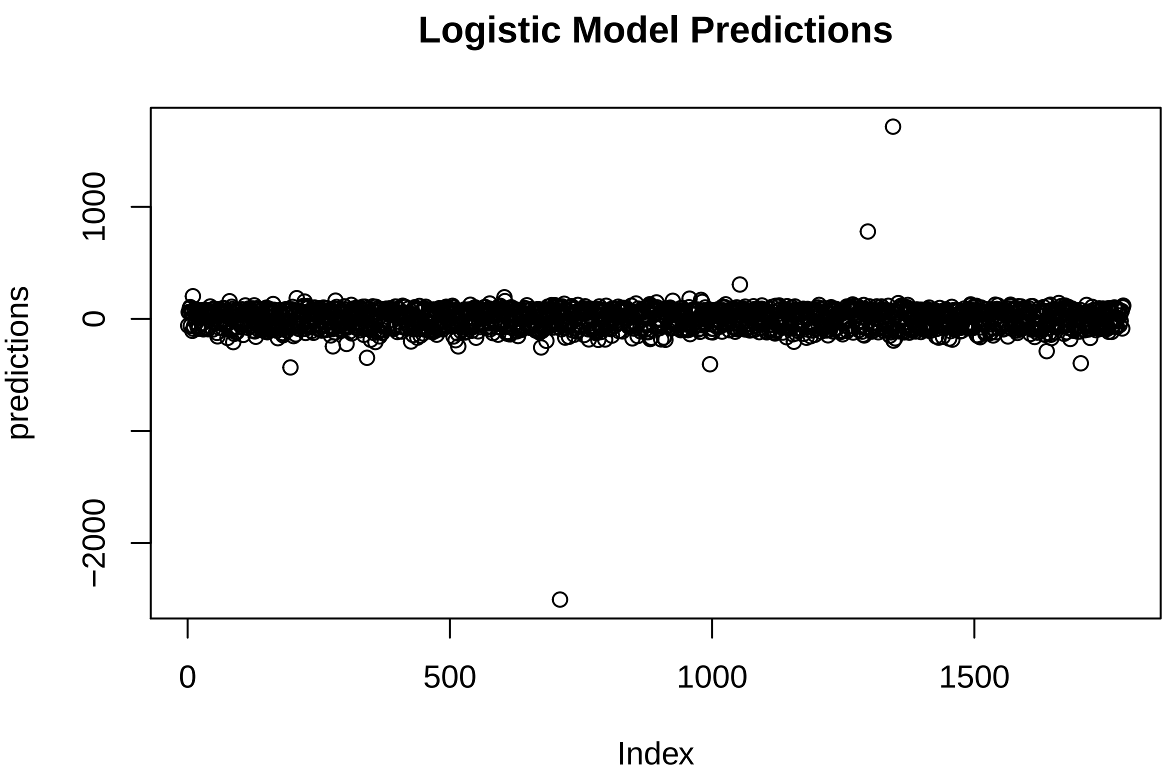
Logistice Regression

Create Complete Logistic Model (On All Data)

```
og_reg <-glm(label~.,family = binomial, data = img_train_01)
```

```
## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

Calculate Complete Logistic Model Accuracy



```
## [1] 0.9523543
```

The first predictive model for classification that I employed was a logistic regression on all values classified as either shirts or pants. Using the appropriate dataset, I predicted the classes of my validation set on the trained model. Once the predictions were plotted, there was a clear separation at the value 0 so any predictions above this threshhold were given a value of 1, and the rest 0. Once the classes were chosen, I assessed the predicted classes against the actual classes to obtain an accuracy rate. Though the model was extremely accurate, fitting was very slow making it somwhat undesirable to employ.

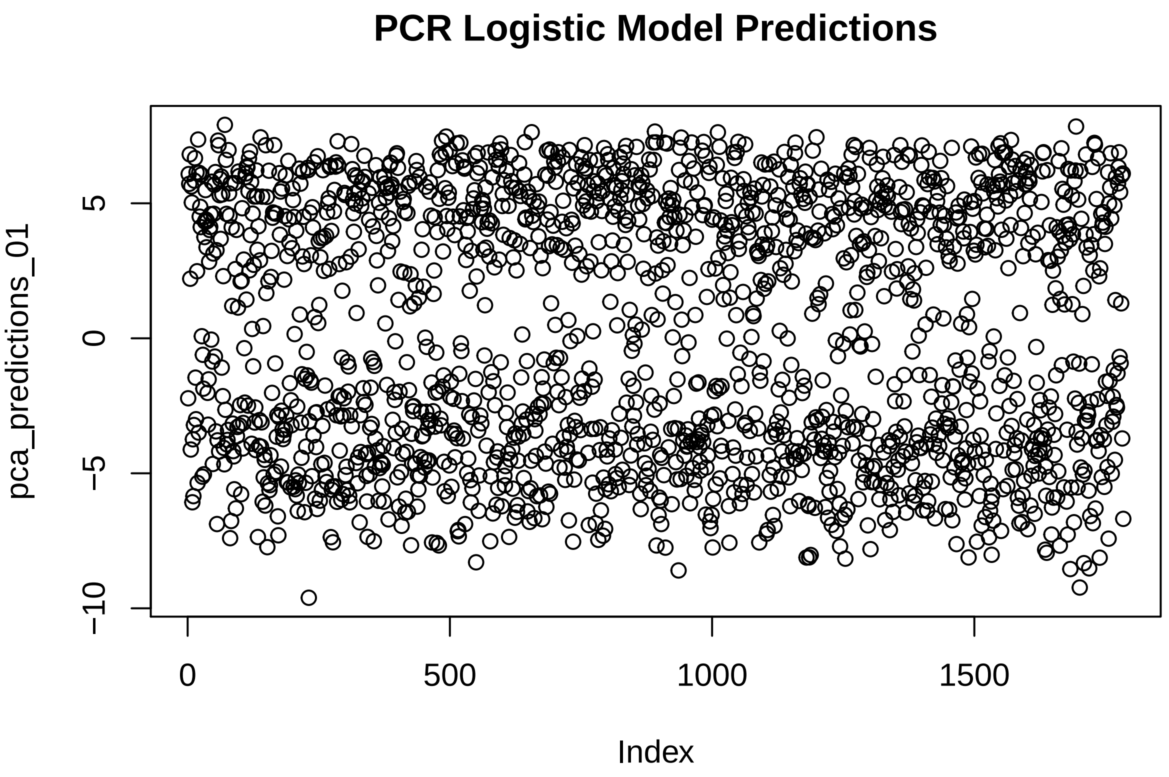
PCR

Create PCR Logistic Model

```
#Find PCs
pcamod01 <- prcomp(subset(img_train_01, select = -c(label)),scale. = FALSE, center = FALSE)

#Use first 2 to fit regression
pca_data01 <- as.data.frame(cbind(pcamod01$x[,1:2],y=img_train_01$label))
pca_og_reg01 <-glm(y~.,family = binomial, data = pca_data01)
```

The second model I used was another logistic regression, this time on the dimension reduced data from our PCA. I only used PCA from the dimension reducing techniques explored in the first part of the project since it offered the greatest reduction in dimensions while maintaing the integrity of the image though PLS would have worked as well. Once I calculated the appropriate PC's, only using the first 2 just as before, I ran a logistic regression on the reduced data. Noticeably, the logistic regression took much less time to fit on the reduced data, with the tradeoff of now having to compute the principle components.

Assess PCR Logistic Model Accuracy



```
## [1] 0.955157
```

After the model was fit, I used the validation set to gain a better understanding of the models accuracy. Using the first two eigenvectors from corresponding to the first two principle components, I projected the validation data onto the new subspace and used my model to predict. As in before, there was a clear boundary at the 0 value, albeit with a little bit more noise. Still, using this threshhold I again obtained class predictions and determined my accuracy rate. With the PCR model, there was only a slight decrease in accuracy but I think this was fairly supplemented by the slight increase in the model's efficiency.

CDA

Create and Assess CDA Model

```
X01 <- as.matrix(subset(img_val_01, select = -c(label)))
#centroids
g0 <- colMeans(subset(img_train_01, select = -c(label))[img_train_01$label==0,])
g1 <- colMeans(subset(img_train_01, select = -c(label))[img_train_01$label==1,])
#covariance matrix
W<- cov(subset(img_train_01, select = -c(label)))

cda_pred_01 <- (X01%*%solve(W)%*%(g0-g1))>((t(g0+g1)%*%solve(W)%*%(g0-g1)/2)[1])
cda_pred_01 <- as.numeric(cda_pred_01)

1-sum(cda_pred_01==img_val_01$label)/length(cda_pred_01)
```

```
## [1] 0.9809417
```

Now having used logistic regression for both original and reduced data, I wanted to compare how the models faired against dscriminant analysis. Since there were only two values and the previous logistic models were fairly accurate, I assumed a linear boundary would be sufficient for my model. I decided to use CDA rather than LDA since CDA is the less computationally intensive model and I could simply increase the complexity if the resulting CDA predictions were not accurate. Setting up the analysis was fairly simple as I only needed to calculate the two class centroids and the covariance matrix. After I classified each value in the validation set according to the inequality establisheed in lecture for 2 class classification, I was able to find the accuracy rate. I subtracted the initially calculated proportion from 1 since the initial value given was extremely low. Since there are only two possible values, only flipping the class values in my prediction was sufficient to give the true accuracy rate. The CDA model offered significant improvements upon the logistic regression in both the time it took to fit the model and in accuracy.

KNN

Create KNN (Non Parametric Approach) Model

```
euclidean_dist <- function(x,y){sqrt(sum((x-y)^2))}

my_kmeans<-function(X,K){
  n<-nrow(X)
  init<-sample(1:n,K)
  centroids<-X[init,]

  assignment<-apply(X,1,function(obs){
    distances<-apply(centroids,1,function(centroid){
      euclidean_dist(obs,centroid)})
    which.min(distances)
  })

  repeat{
    prev<-assignment
    mat_list<-split(X,assignment)
    centroids<-t(sapply(mat_list,colMeans))

     assignment<-apply(X,1,function(obs){
      distances<-apply(centroids,1,function(centroid){
        euclidean_dist(obs,centroid)
        })
      which.min(distances)
      })
    if (all(assignment==prev)){
      break
    }
  }

  return(list(cluster_sizes=as.vector(table(assignment)),cluster_means=centroids,clustering_vector=assi
  }
```

```
km <- my_kmeans(as.data.frame(pcamod01$x[,1:2]),2)
```

Assess KNN Model Accuracy

```
## [1] 0.8436099
```

The last method I attempted for binary classification was a nonparametric k nearest neighbors model in which I clustered the initial data, determined which cluster was closest to each point in the validation set, and classified accordingly. However, since the model is non parametric, it would take a significant amount of time to compute each point. Since the previous models were already fairly accurate and quick, I decided parametric would only be a viablealternative if it were at least as fast as the previous models. To circumvent the computation issue, I again applied the model to the principle components. Using the reduced images also have the advantage of requiring less data points since the ratio of predictors to data points is significantly higher. While the resulting model was comparable to logistic regression in terms of speed, it was significantly less accurate.

Comparing the results of the calculted accuracies based on the validation sets, it appears that CDA is the most efficient and accuarate model for binary classification. We can now evaluate our test set under the chosen CDA model to obtain an unbiased estimate of it accuracy.

Final Evaluation Using CDA

```
## [1] 0.9785915
```

To avoid repetition, the binary classifications for the other two class pairs were done in eactly the same manner as to this one (although the class values were adjusted to values of 0 or 1 so logistic regression would work).

02 (Shirts and Shoes)

Logistic Regression

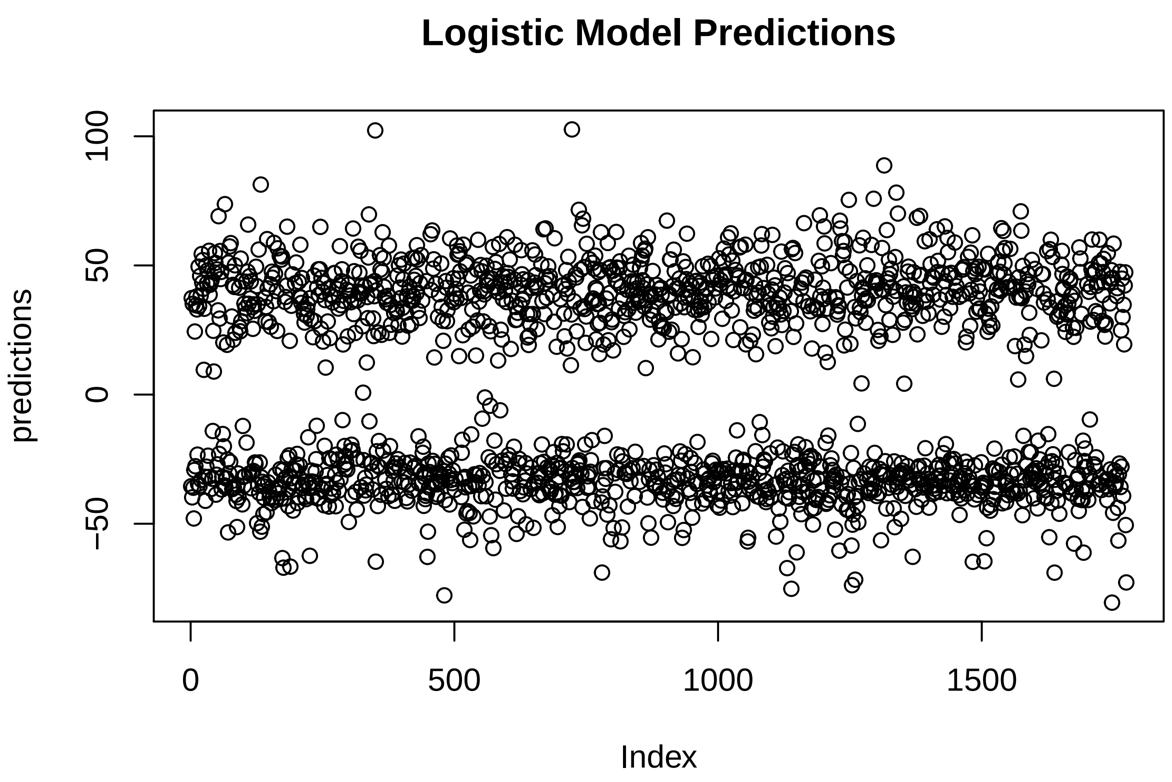
Create Complete Logistic Model (On All Data)

```
img_train_02$label[img_train_02$label==2] <- 1
img_val_02$label[img_val_02$label==2] <- 1


og_reg <-glm(label~.,family = binomial, data = img_train_02)
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

Calculate Complete Logistic Model Accuracy



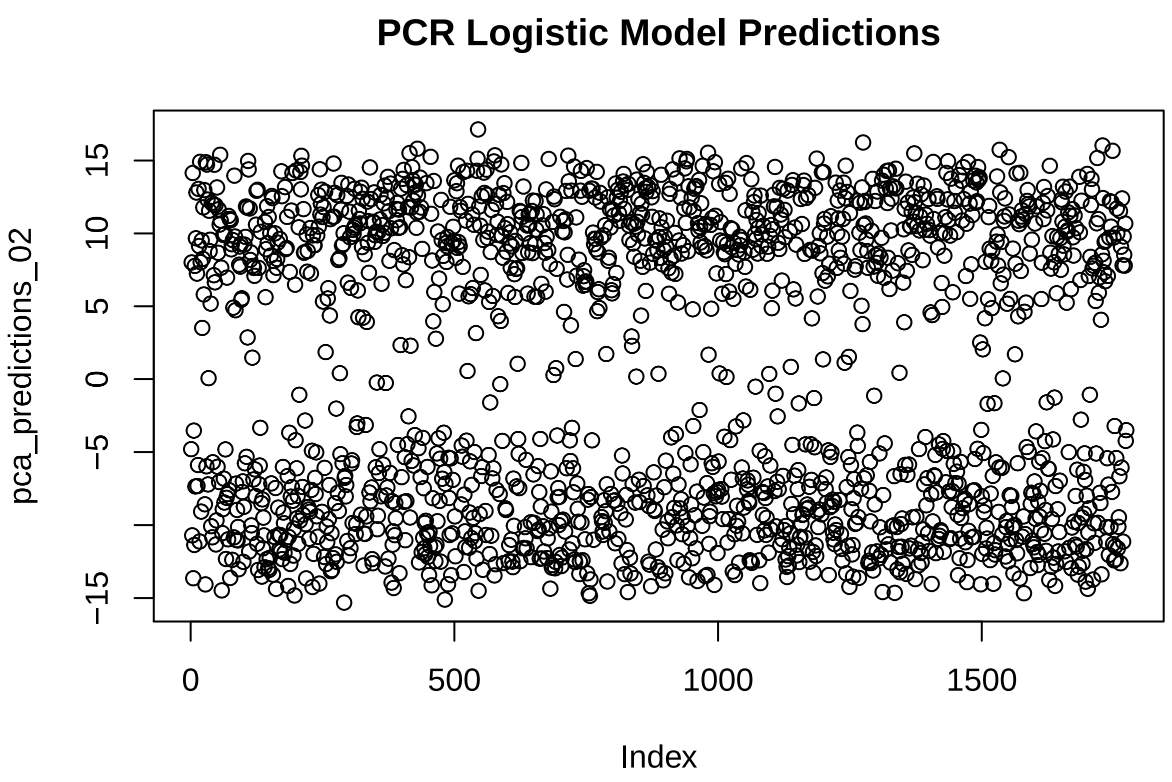
```
## [1] 0.9966178
```

PCR

Create PCR Logistic Model

```
pcamod02 <- prcomp(subset(img_train_02, select = -c(label)),scale. = FALSE, center = FALSE)

pca_data02 <- as.data.frame(cbind(pcamod02$x[,1:2],y=img_train_02$label))
pca_og_reg02 <-glm(y~.,family = binomial, data = pca_data02)
```

Assess PCR Logistic Model Accuracy



```
## [1] 0.9909808
```

CDA

Create and Assess CDA Model

```
X02 <- as.matrix(subset(img_val_02, select = -c(label)))
g0 <- colMeans(subset(img_train_02, select = -c(label))[img_train_02$label==0,])
g2 <- colMeans(subset(img_train_02, select = -c(label))[img_train_02$label==1,])
W<- cov(subset(img_train_02, select = -c(label)))

cda_pred_02 <- (X02%*%solve(W)%*%(g0-g2))>((t(g0+g2)%*%solve(W)%*%(g0-g2)/2)[1])
cda_pred_02 <- as.numeric(cda_pred_02)

1-sum(cda_pred_02==img_val_02$label)/length(cda_pred_02)
```

```
## [1] 0.9977452
```

KNN

Create KNN (Non Parametric Approach) Model

```
km <- my_kmeans(as.data.frame(pcamod02$x[,1:2]),2)
```

Assess KNN Model Accuracy

```
## [1] 0.9780158
```

Final Evaluation Using CDA

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

12 (Pants and Shoes)

Logitstic Regression

Create Complete Logistic Model (On All Data)

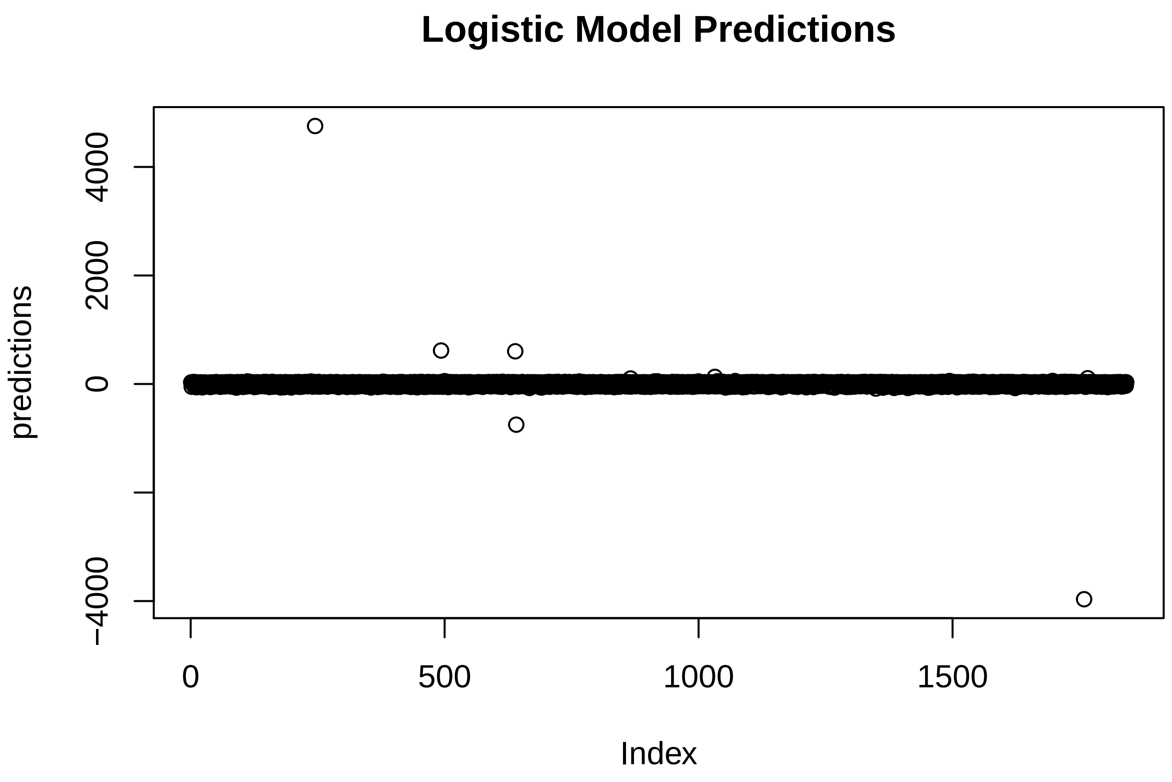
```
img_train_12$label[img_train_12$label==2] <- 0
img_val_12$label[img_val_12$label==2] <- 0


og_reg <-glm(label~.,family = binomial, data = img_train_12)
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

Calculate Complete Logistic Model Accuracy

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
## == : prediction from a rank-deficient fit may be misleading
```



```
## [1] 0.9940282
```

PCR

Create PCR Logistic Model

```r
pcamod12 <- prcomp(subset(img_train_12, select = -c(label)),scale. = FALSE, center = FALSE)

pca_data12 <- as.data.frame(cbind(pcamod12$x[,1:2],y=img_train_12$label))
pca_og_reg12 <-glm(y~.,family = binomial, data = pca_data12)
```

Assess PCR Logistic Model Accuracy

```
## [1] 0.9945711
```

CDA

Create and Assess CDA Model

```r
X12 <- as.matrix(subset(img_val_12, select = -c(label)))
g1 <- colMeans(subset(img_train_12, select = -c(label))[img_train_12$label==1,])
g2 <- colMeans(subset(img_train_12, select = -c(label))[img_train_12$label==0,])
W<- cov(subset(img_train_12, select = -c(label)))

s <- svd(W)
inv_s <- s$v[,1:735]%*%solve(diag(s$d[1:735]))%*%t(s$u[,1:735])

cda_pred_12 <- (X12%*%inv_s%*%(g1-g2))>((t(g1+g2)%*%inv_s%*%(g1-g2)/2)[1])
cda_pred_12 <- as.numeric(cda_pred_12)

sum(cda_pred_12==img_val_12$label)/length(cda_pred_12)
```

```
## [1] 0.9967427
```

Note: In this model, the inverse of W is actually unsolvable. Therefore, SVD was used to find an approximation of the inverse. However, the acucracy of the CDA model is still better than any of the other competitors.

KNN

Create KNN (Non Parametric Approach) Model

```r
km <- my_kmeans(as.data.frame(pcamod12$x[,1:2]),2)
```

Assess KNN Model Accuracy

```
## [1] 0.9945711
```

Final Evaluation Using CDA

```r
X12 <- as.matrix(subset(img_test_12, select = -c(label)))

cda_pred_12 <- (X12%*%inv_s%*%(g1-g2))>((t(g1+g2)%*%inv_s%*%(g1-g2)/2)[1])
cda_pred_12 <- as.numeric(cda_pred_12)

sum(cda_pred_12==img_test_12$label)/length(cda_pred_12)
```

```
## [1] 0.4942779
```

The discrepancy between the low test classification rate and higher validation rate suggest that the difference between classes 1 and 2 are are harder to detect.

Multiclass classification

Discriminant analysis, random forrests

In this section, you will develop a model to classify all three classes simultaneously. You should again try several different methods, and use model selection methods to determine which model is best. You should also be sure to keep a held-out test set to evaluate the performance of your model. (Side question: how could you use the binary models from the previous section to develop a multiclass classifier?)

Binary Classifiers

CDA

Create and Assess CDA Model

```
X_train <- as.matrix(X[train,])
g0 <- colMeans(X_train[y[train]==0,])
g1 <- colMeans(X_train[y[train]==1,])
g2 <- colMeans(X_train[y[train]==2,])
W<- cov(X_train)

f0 <-as.matrix(X[validation,])%*%solve(W)%*%g0-rep(g0%*%solve(W)%*%g0/2,length(validation))
f1 <-as.matrix(X[validation,])%*%solve(W)%*%g1-rep(g2%*%solve(W)%*%g1/2,length(validation))
f2 <-as.matrix(X[validation,])%*%solve(W)%*%g2-rep(g2%*%solve(W)%*%g2/2,length(validation))

fs <- data.frame(f0,f1,f2)
cd_pred <- apply(fs,1,order)[3,]-1

sum(cd_pred==y[validation])/length(validation)
```

```
## [1] 0.9662963
```

Unlike in the binary regression, multiclass regression is unsuited for logistic regression due to its increased complexity. Therefore, more comple models like CDA must be used. Once each matrix for CDA was calulated based on the training data, they were evaluated on the validation data. The class with the highest evaluated equation then became the predicted class.

LDA

Create and Assess LDA Model

```
f0 <- log((table(y[train])/length(train))[1])+f0
f1 <- log((table(y[train])/length(train))[2])+f1
f2 <- log((table(y[train])/length(train))[3])+f2

fslda <- data.frame(f0,f1,f2)
cd_pred <- apply(fslda,1,order)[3,]-1

sum(cd_pred==y[validation])/length(validation)
```

```
## [1] 0.9677778
```

To account for any increased complexity offered by a multiclass classifier, both LDA and QDA were both employed as well. LDA was a simple extension since it simply required the addition of the log of the class proportion to each class equation before prediction. This method did not appear to improve the model by any significant amount, implying that CDA was sufficient.

QDA

Create and Assess QDA Model

```
X_train <- as.matrix(X[train,])
g0 <- colMeans(X_train[y[train]==0,])
g1 <- colMeans(X_train[y[train]==1,])
g2 <- colMeans(X_train[y[train]==2,])

W0<- cov(X_train[y[train]==0,])
W1<- cov(X_train[y[train]==1,])
W2<- cov(X_train[y[train]==2,])

p0 <- log((table(y[train])/length(train))[1])
p1 <- log((table(y[train])/length(train))[2])
p2 <- log((table(y[train])/length(train))[3])

m0 <- matrix(rep(g0,2700),nrow = 2700, byrow = TRUE)
m1 <- matrix(rep(g1,2700),nrow = 2700, byrow = TRUE)
m2 <- matrix(rep(g2,2700),nrow = 2700, byrow = TRUE)

f0 <- p0-.5*diag(as.matrix(X[validation,]-m0)%*%solve(W0)%*%t(X[validation,]-m0))


s1 <- svd(W1)
inv_s1 <- s1$v[,1:550]%*%solve(diag(s1$d[1:550]))%*%t(s1$u[,1:550])
f1 <- p1-.5*diag(as.matrix(X[validation,]-m1)%*%inv_s1%*%t(X[validation,]-m1))


s2 <- svd(W2)
inv_s2 <- s2$v[,1:735]%*%solve(diag(s2$d[1:735]))%*%t(s2$u[,1:735])
f2 <- p2-.5*diag(as.matrix(X[validation,]-m2)%*%inv_s2%*%t(X[validation,]-m2))

fs <- data.frame(f0,f1,f2)
cd_pred <- apply(fs,1,order)[3,]-1

sum(cd_pred==y[validation])/length(validation)
```

```
## [1] 0.9603704
```

Though CDA was determined to be sufficient, I still used QDA considering that the decision boundary may not be linear. Once I fit the appropriate equations based on the training data, I predicted the validation data as before. Surprisingly, this model performed worse than the previous two, most likely because of a lack of sufficient data for a more complex model or overfitting on the training data.

Random Forest

Evaluate Random Forest

```r
sum(predict(forest.img,img[validation,])==img$y[validation])/length(validation)
```

```
## [1] 0.9940741
```

Another classification model discussed in class was the decision tree. Although useful for determining predictor importance, it wa noted that decision trees alone have poor predictive capabilities. Therefore, I utilized the random forest created in the exploratory part of my analysis for predicting my validation data. This method had the lowest prediction error by far.

KNN

Create KNN (Non Parametric Approach) Model

```r
pcamod <- prcomp(X[train,],scale. = FALSE, center = FALSE)

set.seed(2)
km <- my_kmeans(data.frame(pcamod$x[,1:2]),3)
```

Assess KNN Model Accuracy

```
## [1] 0.8111111
```

The last approach I used was a nonparametric approach to see if the decision boundary was not appropriately modeled. This model performed the worst in terms of prediction, most likely for a lack of data or overfitting.

Choosing the Random Forest as the most appropriate model for my prediction, I predicted the test data to achieve my final prediction rate estimate.

Final Evaluation by Random Forest

```r
sum(predict(forest.img,img[test,])==img$y[test])/length(validation)
```

```
## [1] 0.9944444
```