Fashion MNIST

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.

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

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

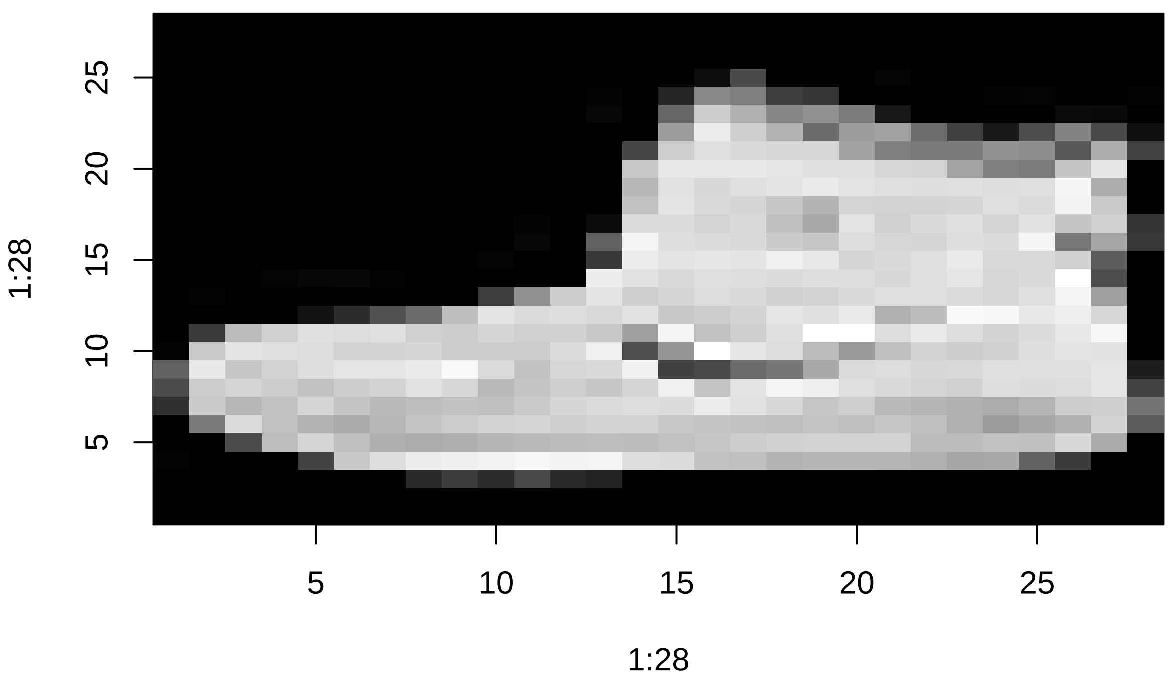
```
## See spec(...) for full column specifications.
```

```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:

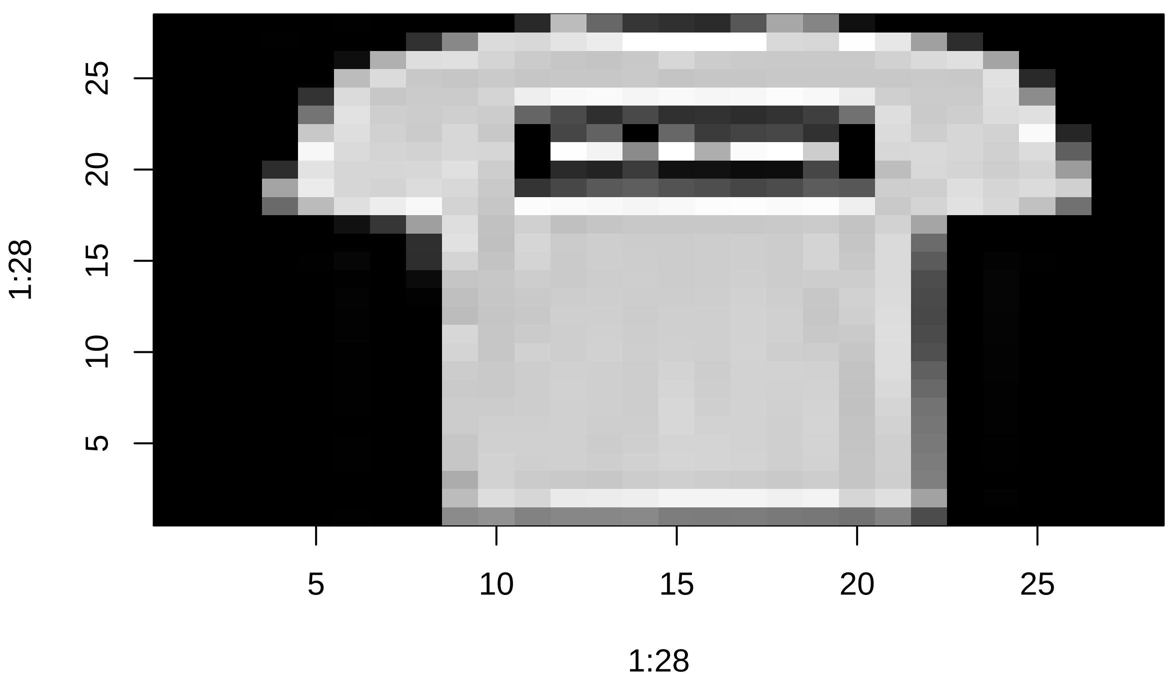
```r
X2 <- matrix(as.numeric(X[1,]), ncol=28, nrow=28, byrow = TRUE)
X2 <- apply(X2, 2, rev)
image(1:28, 1:28, t(X2), col=gray((0:255)/255), main='Class 2 (Shoes)')
```

Class 2 (Shoes)



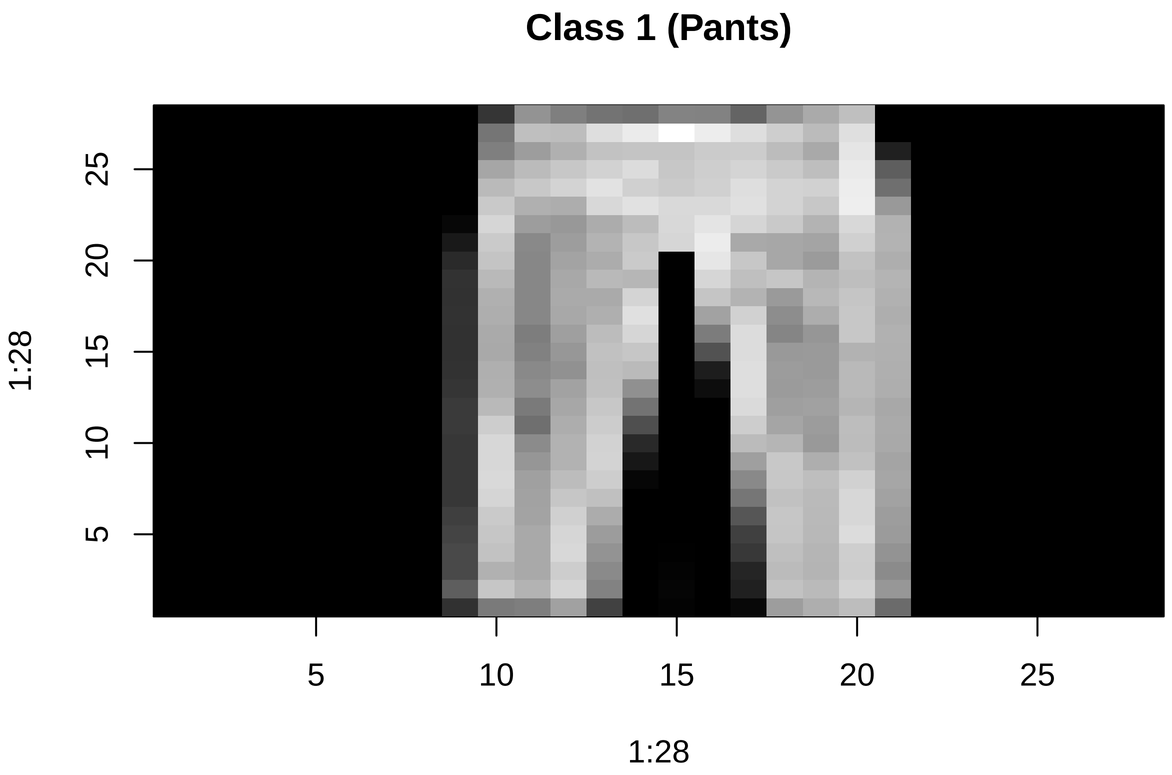
```r
X0 <- matrix(as.numeric(X[2,]), ncol=28, nrow=28, byrow = TRUE)
X0 <- apply(X0, 2, rev)
image(1:28, 1:28, t(X0), col=gray((0:255)/255), main='Class 0 (Shirt)')
```

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?

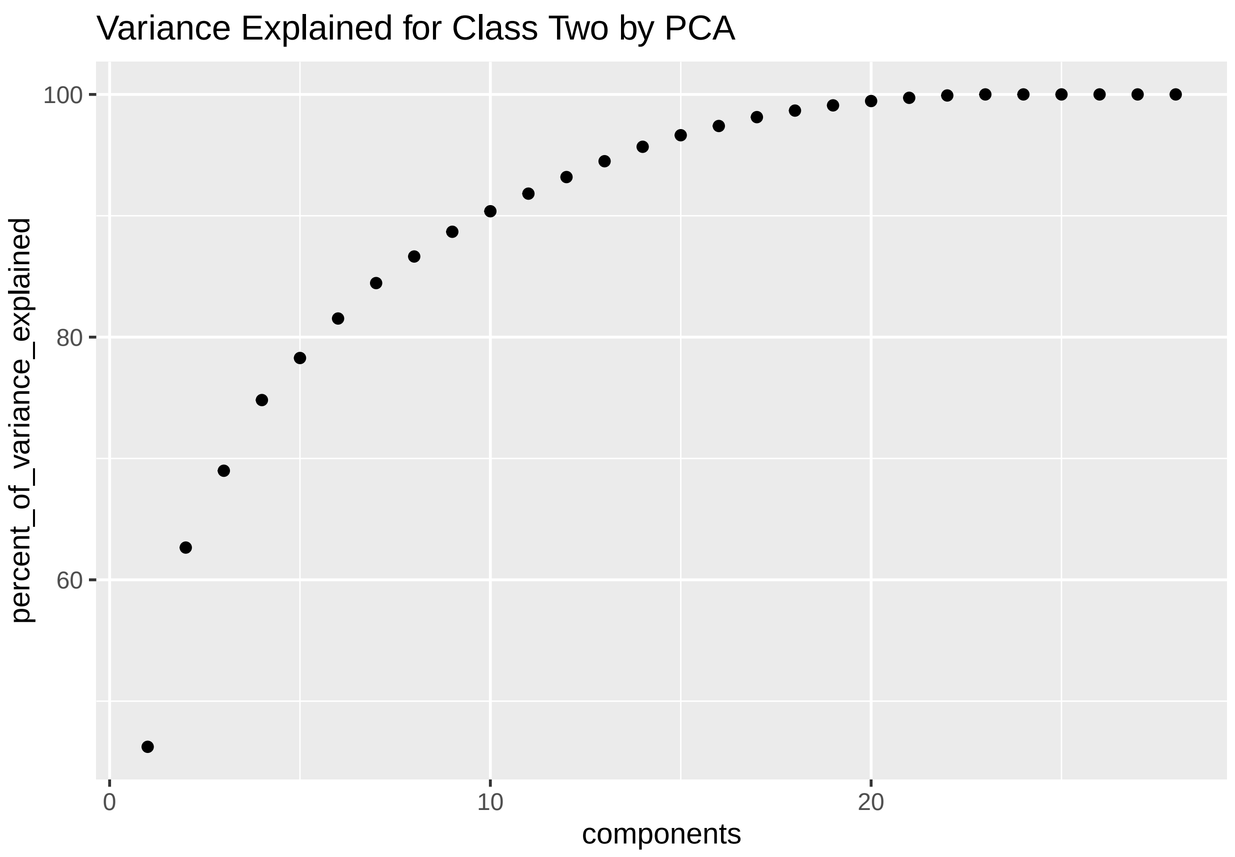
PCA for Dimension Reduction

Principal Component Analysis is a dimension reduction technique that we covered in class. PCA results in orthogonal vectors that maximize the variance given an amount of components. The principal components can then be used to visualize the data in a certain amount of dimensions. The original images that we visualize are 28×28, but using PCA we may reduce these images to only include the data of the first 8 principal components or possibly even lower. In order to visually distinguish images from different classes, I would say that being able to have about 80 percent of the variance explained in the number of components that we cover should be enough to be able to distinguish between the different classes. I'll visualize lower dimensional representations as images using just the number of components that allow for about 80 percent of the variation so if 10 components allow for 80 percent of the data to be accounted for the image would be plotted from those components only. When we perform PCA, the data is no longer in its 28×28 format and in order to be able to visualize it again we have to project the compressed data matrix using a certain number of components back into the original dimensions. For these matrices of data for each image I chose not to scale or center the data because the variation between columns and also across rows accounts for the difference in coloring, so we would want to preserve that.

```r
library(ggplot2)
```

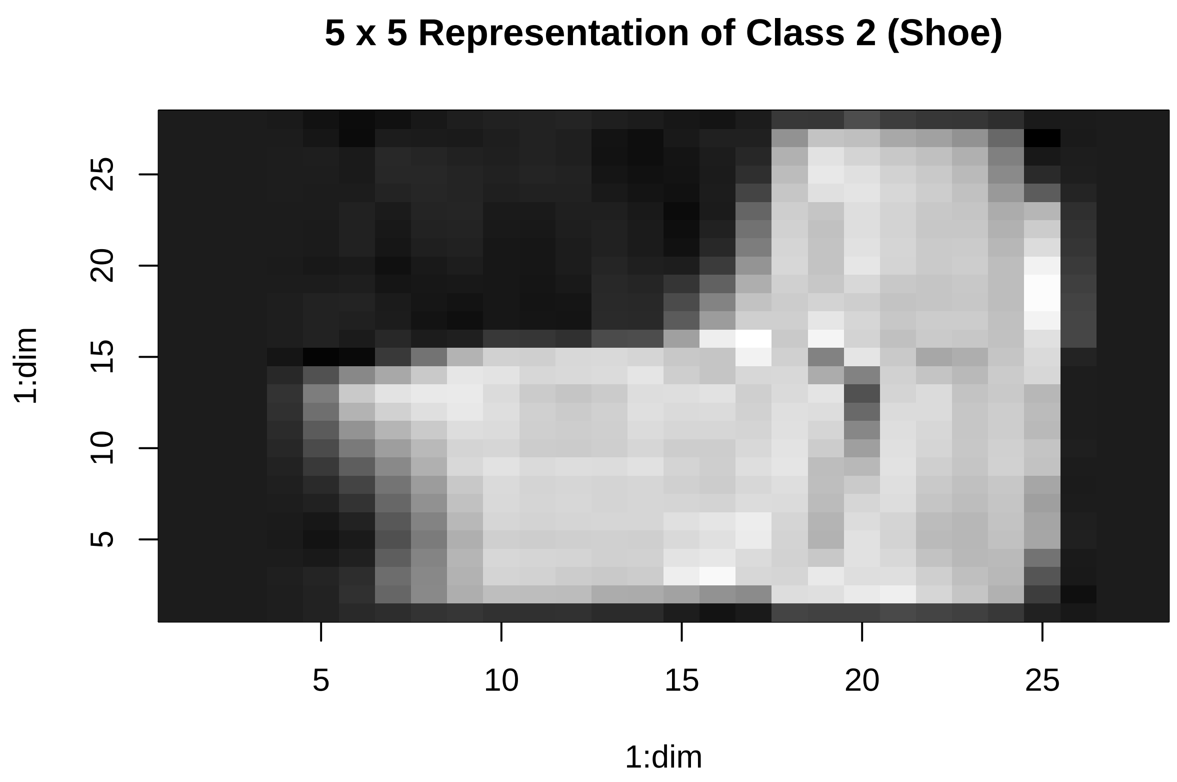
```
plot_image <- function(dvec, title) {
  tot_d <- length(dvec)
  dim <- sqrt(tot_d)
  mat <- matrix(as.numeric(dvec), ncol = dim, nrow = dim, byrow = TRUE)
  mat <- apply(mat, 2, rev)
  image(1:dim, 1:dim, t(mat), col = gray((0:255) / 255), main = title)
}
```

```
class_two <- matrix(as.numeric(X[1,]), ncol=28, nrow=28, byrow = TRUE)
class_two_pca <- prcomp(class_two, scale = FALSE, center = FALSE)
class_two_var_df <- data.frame(cbind(1:28, cumsum(class_two_pca$sdev) / sum(class_two_pca$sdev) * 100))
names(class_two_var_df) <- c('components', 'percent_of_variance_explained')
ggplot(class_two_var_df, aes(x = components, y = percent_of_variance_explained)) + geom_point() + ggtit
```

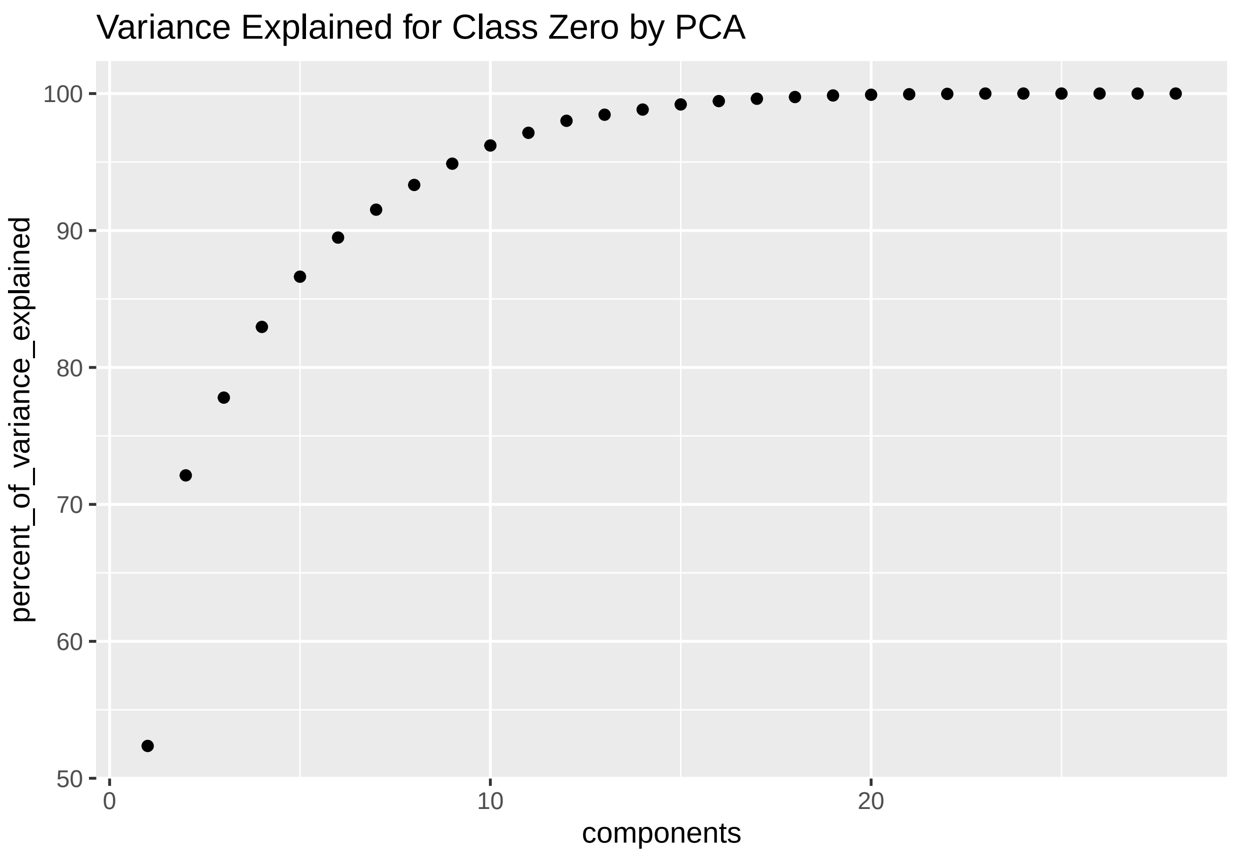


Using the plot above we can see that by using 5 principal components we are able to explain about 80 percent of the variance in the data. I used t

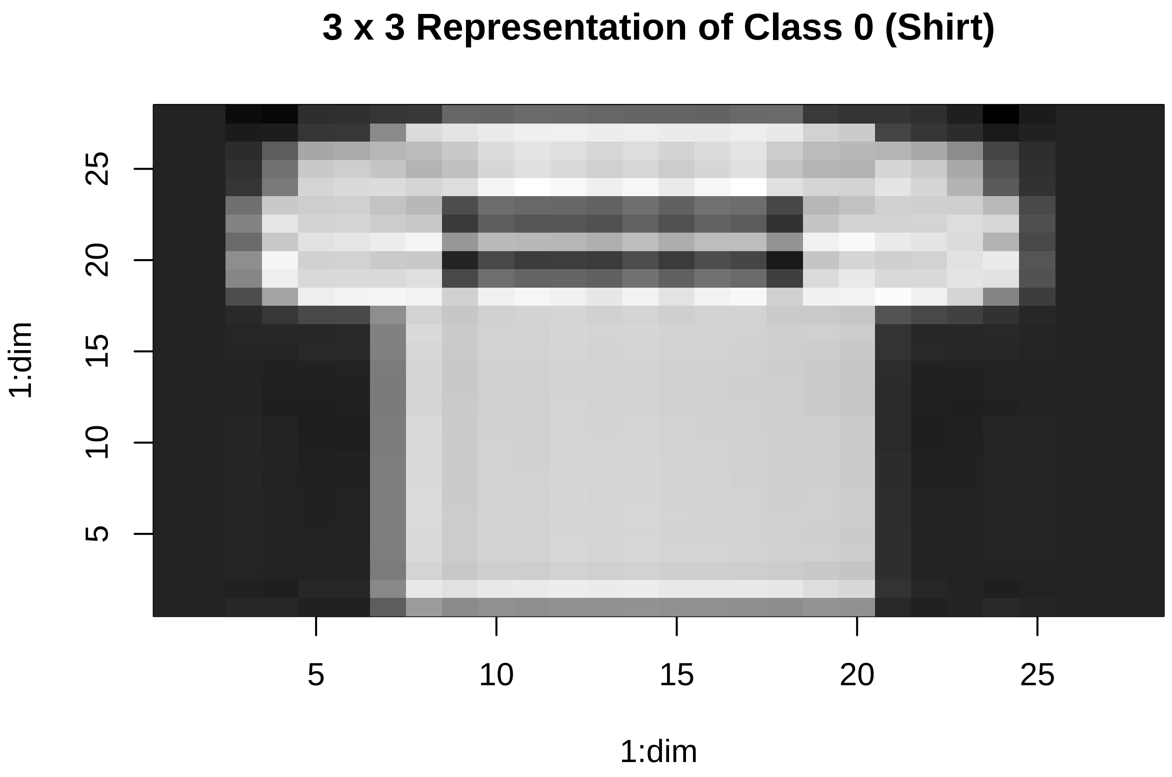
```
class_two_dr <- class_two_pca$x[, 1:5] %*% t(class_two_pca$rotation[, 1:5])
plot_image(class_two_dr, '5 x 5 Representation of Class 2 (Shoe)')
```



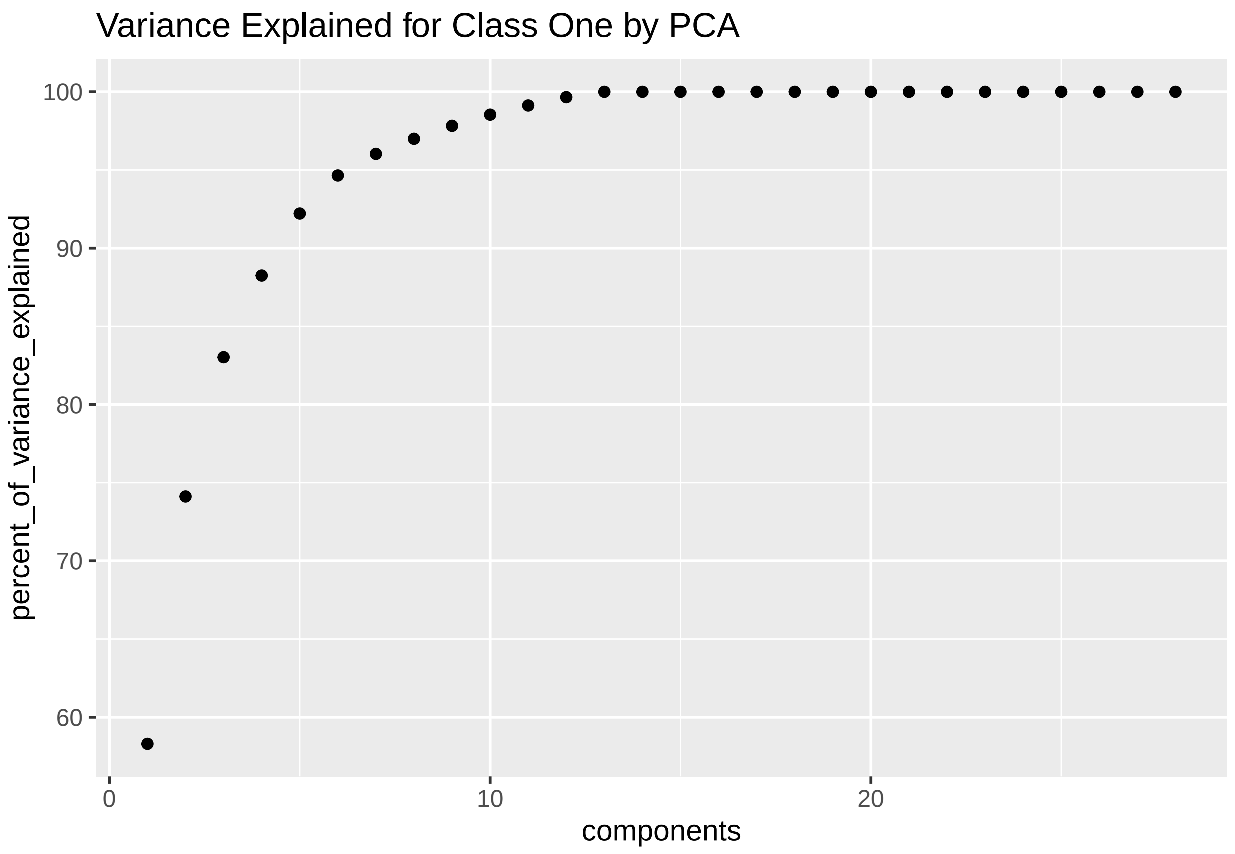
```
class_zero <- matrix(as.numeric(X[2,]), ncol=28, nrow=28, byrow = TRUE)
class_zero_pca <- prcomp(class_zero, scale = FALSE, center = FALSE)
class_zero_var_df <- data.frame(cbind(1:28, cumsum(class_zero_pca$sdev) / sum(class_zero_pca$sdev) * 10
names(class_zero_var_df) <- c('components', 'percent_of_variance_explained')
ggplot(class_zero_var_df, aes(x = components, y = percent_of_variance_explained)) + geom_point() + ggti
```



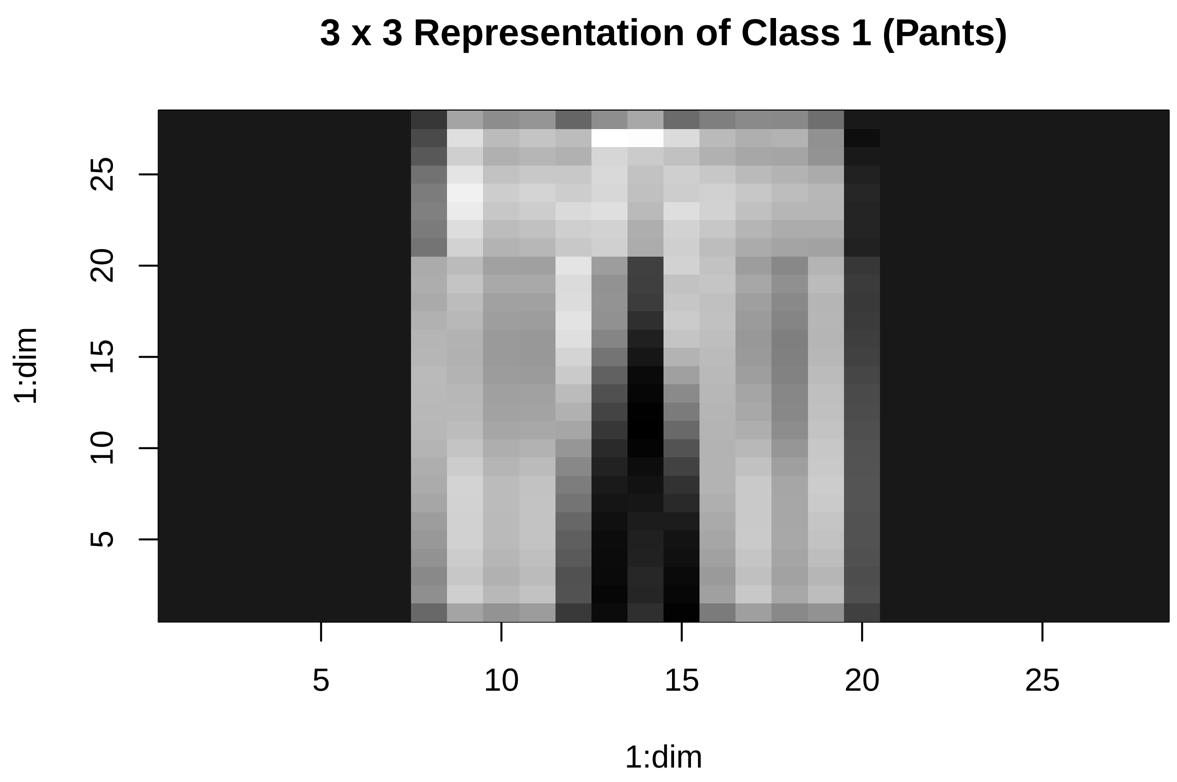
```r
class_zero_dr <- class_zero_pca$x[, 1:3] %*% t(class_zero_pca$rotation[, 1:3])
class_zero_dr <- apply(class_zero_dr, 1, rev)
plot_image(class_zero_dr, '3 x 3 Representation of Class 0 (Shirt)')
```



```r
class_one <- matrix(as.numeric(X[8,]), ncol=28, nrow=28, byrow = TRUE)
class_one_pca <- prcomp(class_one, scale = FALSE, center = FALSE)
class_one_var_df <- data.frame(cbind(1:28, cumsum(class_one_pca$sdev) / sum(class_one_pca$sdev) * 100))
names(class_one_var_df) <- c('components', 'percent_of_variance_explained')
ggplot(class_one_var_df, aes(x = components, y = percent_of_variance_explained)) + geom_point() + ggtit
```



```r
class_one_dr <- class_one_pca$x[, 1:3] %*% t(class_one_pca$rotation[, 1:3])
class_one_dr <- apply(class_one_dr, 1, rev)
plot_image(class_one_dr, '3 x 3 Representation of Class 1 (Pants)')
```



```r
convert_row_to_mat_28 <- function(row) {
  return(matrix(as.numeric(row), ncol = 28, nrow = 28, byrow = TRUE))
}
```

```r
convert_row_to_mat_5 <- function(row) {
  return(matrix(row, ncol = 5, byrow = FALSE))
}
```

```r
# keep only 5 principal components for each photo
X_5 <- matrix(nrow = nrow(X), ncol = 5 * 28)
for (i in 1:nrow(X)) {
  mat <- convert_row_to_mat_28(X[i, ])
  mat_pca <- prcomp(mat, scale = FALSE, center = FALSE)
  X_5[i, ] <- as.vector(mat_pca$x[, 1:5])
}
```

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.

Splitting the data into pairs of classes

```r
ss_ind <- (y == 2) | (y == 0)
sp_ind <- (y == 2) | (y == 1)
ps_ind <- (y == 1) | (y == 0)
ss_lab <- y[ss_ind]
# shirts 0, shoes 1
ss_lab <- ifelse(ss_lab == 0, 0, 1)
sp_lab <- y[sp_ind]
# shoes 0, pants 1
sp_lab <- ifelse(sp_lab == 2, 0, 1)
ps_lab <- y[ps_ind]
# pants 1 shirts 0
shoes_shirts <- X_5[ss_ind, ]
shoes_pants <- X_5[sp_ind, ]
pants_shirts <- X_5[ps_ind, ]
```

```r
nrow(shoes_shirts)
```

```
## [1] 12000
```

```r
nrow(shoes_pants)
```

```
## [1] 12000
```

```r
nrow(pants_shirts)
```

```
## [1] 12000
```

```r
training <- 1:9000
test <- 9001:12000
```

```r
shuffle1 <- sample(1:12000, size = 12000)
shuffle2 <- sample(1:12000, size = 12000)
shuffle3 <- sample(1:12000, size = 12000)
ss_train <- shoes_shirts[shuffle1[training], ]
ss_test <- shoes_shirts[shuffle1[test], ]
ss_train_lab <- ss_lab[shuffle1[training]]
ss_test_lab <- ss_lab[shuffle1[test]]
sp_train <- shoes_pants[shuffle2[training], ]
sp_test <- shoes_pants[shuffle2[test], ]
sp_train_lab <- sp_lab[shuffle2[training]]
sp_test_lab <- sp_lab[shuffle2[test]]
ps_train <- pants_shirts[shuffle3[training], ]
ps_test <- pants_shirts[shuffle3[test], ]
ps_train_lab <- ps_lab[shuffle3[training]]
ps_test_lab <- ps_lab[shuffle3[test]]
```

Logistic Regression

Logistic regression is one of the classic binary classification methods. It does not have many assumptions such as linearity or any other sort of relationship between the predictors and the target. It does assume independent observations though as well as very little multicolinearity in order to work well. We are basically predicting the conditional distribution of of the response given the values of x, so it outputs a probability vector for the observations and a threshold of say like 0.5 can be chosen in order to classify the observations into a class. I decided to run k-fold cross validation on the data using k = 6 so that each fold has 1500 observations in order to determine a good threshold of probability in order to classify the object into a given class.

```r
create_table <- function(matrix, labels) {
  df <- data.frame(cbind(matrix, labels))
  names(df) <- c(1:140, 'label')
  return(df)
}
```

```r
log_cv <- function(train, train_lab, percent) {
  per <- c()
  for (i in 0:5) {
    ind <- ((1500 * i) + 1):(1500 * (i + 1))
    tst_k <- train[ind, ]
    tst_k_lab <- train_lab[ind]
    tst_k_df <- create_table(tst_k, tst_k_lab)
    train_k <- train[-ind, ]
    train_k_lab <- train_lab[-ind]
    train_k_df <- create_table(train_k, train_k_lab)
    train_log <- glm(label ~ ., train_k_df, family = binomial())
    pred <- ifelse(predict(train_log, newdata = tst_k_df, type = 'response') >= percent, 1, 0)
    per <- c(per, sum(tst_k_lab == pred) / 1500)
  }
  return(mean(per))
}
```

```r
log_cv_per_ss <- c()
log_cv_per_sp <- c()
log_cv_per_ps <- c()
percents <- c(0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9)
for (p in percents) {
  log_cv_per_ss <- c(log_cv_per_ss, log_cv(ss_train, ss_train_lab, p))
  log_cv_per_sp <- c(log_cv_per_sp, log_cv(sp_train, sp_train_lab, p))
  log_cv_per_ps <- c(log_cv_per_ps, log_cv(ps_train, ps_train_lab, p))
}
```

```r
# shoes and shirts
log_cv_per_ss
```

```
## [1] 0.8707778 0.9035556 0.9155556 0.9200000 0.9203333 0.9155556 0.9058889
## [8] 0.8848889
```

```r
percents[log_cv_per_ss == max(log_cv_per_ss)]
```

```
## [1] 0.6
```

For a trained logistic model on just the shoes and shirts data it seems that 0.6 for the probability cutoff yielded the best results although the different thresholds that we experimented with the accuracy percent was relatively similar and actually also very low in this situation. Using principal components we are able to ensure that the components are orthogonal to each other and therefore uncorrelated, so I wouldn't expect the assumption of no collinearity between the predictive features to cause the poor performance, but in this case the elements of each component are multiplied by the same linear coefficients as linear combinations of the original predictors so in this case as the elements are see as their predictor many of them would be highly correlated causing a huge multicollinearity problem.

```r
# shoes and pants
log_cv_per_sp
```

```
## [1] 0.9326667 0.9375556 0.9396667 0.9393333 0.9353333 0.9290000 0.9197778
## [8] 0.8936667
```

```r
percents[log_cv_per_sp == max(log_cv_per_sp)]
```

```
## [1] 0.4
```

A trained model for classifying shoes and pants, the probability cutoff that yielded the best results was 0.9, but similarly to the classifier between shoes and shirts the accuracy across different percentage thresholds was very similar, but also very poor results.

```r
# pants and shirts
log_cv_per_ps
```

```
## [1] 0.7657778 0.8116667 0.8378889 0.8480000 0.8528889 0.8480000 0.8255556
## [8] 0.7530000
```

```r
percents[log_cv_per_ps == max(log_cv_per_ps)]
```

```
## [1] 0.6
```

The trained model for pants and shirts also yielded a model and similar to the two previous classifier the accuracy level did not change much but also produced rather poor results.

Linear Discriminant Analysis

Unlike logistic regression, linear discrimant analysis we use bayes theorem to estimate the conditional probablity of y given certain values of x by calculating first the distribution of x given a certain y. If the classes are well seperated logistic regression can be very unstable, so this is why LDA is more often preferred. Another reason that LDA is sometimes preferred because it is able to provide a more stable solution using a smaller number of observations especially if the distribution of predictors are roughly normal. LDA is also able to provide classification for multiple classes. Hopefully these results will be able to produce better results than our logistic regression models because there is no assumption of no correlation between predictors although there is the assumption that the covariance matrix between classes is the same.

```r
library(MASS)
```

```r
lda_cv <- function(train, train_lab, percent) {
  per <- c()
  for (i in 0:5) {
    ind <- ((1500 * i) + 1):(1500 * (i + 1))
    test_k <- train[ind, ]
    test_k_lab <- train_lab[ind]
    train_k <- train[-(ind), ]
    train_k_lab <- train_lab[-(ind)]
    train_lda <- lda(train_k, train_k_lab)
    pred <- ifelse(predict(train_lda, test_k)$posterior[, 1] >= percent, 0, 1)
    per <- c(per, sum(test_k_lab == pred) / 1500)
  }
  return(mean(per))
}
```

```r
lda_cv_per_ss <- c()
lda_cv_per_sp <- c()
lda_cv_per_ps <- c()
for (p in percents) {
  lda_cv_per_ss <- c(lda_cv_per_ss, lda_cv(ss_train, ss_train_lab, p))
  lda_cv_per_sp <- c(lda_cv_per_sp, lda_cv(sp_train, sp_train_lab, p))
  lda_cv_per_ps <- c(lda_cv_per_ps, lda_cv(ps_train, ps_train_lab, p))
}
```

```
lda_cv_per_ss
```

```
## [1] 0.9190000 0.9253333 0.9288889 0.9295556 0.9276667 0.9255556 0.9194444
## [8] 0.9017778
```

```
percents[lda_cv_per_ss == max(lda_cv_per_ss)]
```

```
## [1] 0.5
```

For shirts and shoes the LDA classifier performed best using a 0.4 threshold cutoff for probability of one class over the other. The average cross validated accuracy scores are relatively similar over the cutoffs chosen, but there is a slight decline as the values differed further from 0.5.

```
lda_cv_per_sp
```

```
## [1] 0.9443333 0.9457778 0.9454444 0.9455556 0.9445556 0.9447778 0.9448889
## [8] 0.9454444
```

```
percents[lda_cv_per_sp == max(lda_cv_per_sp)]
```

```
## [1] 0.3
```

The LDA classifier for shoes and pants did similarly well across the different thresholds chosen, but the cutoff of 0.6 was deemed the best.

```
lda_cv_per_ps
```

```
## [1] 0.8416667 0.8534444 0.8548889 0.8504444 0.8393333 0.8273333 0.7965556
## [8] 0.7268889
```

```
percents[lda_cv_per_ps == max(lda_cv_per_ps)]
```

```
## [1] 0.4
```

For the classifier between pants and shirts, the best probability threshold was 0.4. This classfier did seem to do quite a bit worse than the binary classifier between shirts and shoes as well as shoes and pants, but this could be considered since pants and shirts covered a more similar area in the images than the other pairs that we had.

Quadratic Discriminant Analysis

QDA similar to LDA assumes that observations are drawn a multivariate normal distribution, but there is no assumption that the covariance matrix for each of the classes is the same. In this way QDA is more flexible than LDA and could be prone to more overfitting. It might be more computationally expensive though since we are estimating a different covariance matrix for each class and estimating this takes up computing power that LDA nor logistic regression called for. The flexibility might be useful in this case as the images are quite different and less assumptions on the model might produce better results.

```
qda_cv <- function(train, train_lab, percent) {
  per <- c()
  for (i in 0:5) {
    ind <- ((1500 * i) + 1):(1500 * (i + 1))
    test_k <- train[ind, ]
    test_k_lab <- train_lab[ind]
    train_k <- train[-(ind), ]
    train_k_lab <- train_lab[-(ind)]
    train_qda <- qda(train_k, train_k_lab)
    pred <- ifelse(predict(train_qda, test_k)$posterior[, 1] >= percent, 0, 1)
    per <- c(per, sum(test_k_lab == pred) / 1500)
  }
  return(mean(per))
}
```

```
qda_cv_per_ss <- c()
qda_cv_per_sp <- c()
qda_cv_per_ps <- c()
for (p in percents) {
  qda_cv_per_ss <- c(qda_cv_per_ss, qda_cv(ss_train, ss_train_lab, p))
  qda_cv_per_sp <- c(qda_cv_per_sp, qda_cv(sp_train, sp_train_lab, p))
  qda_cv_per_ps <- c(qda_cv_per_ps, qda_cv(ps_train, ps_train_lab, p))
}
```

```
qda_cv_per_ss
```

```
## [1] 0.9817778 0.9817778 0.9817778 0.9817778 0.9816667 0.9816667 0.9818889
## [8] 0.9818889
```

```
percents[qda_cv_per_ss == max(qda_cv_per_ss)]
```

```
## [1] 0.8 0.9
```

The QDA classifier built to distinguish between shirts and shoes did fairly well across the different probability thresholds and in comparison to our previous classifiers. Using cross validation the best accuracy was using 0.9.

```
qda_cv_per_sp
```

```
## [1] 0.9736667 0.9736667 0.9737778 0.9737778 0.9736667 0.9735556 0.9736667
## [8] 0.9736667
```

```
percents[qda_cv_per_sp == max(qda_cv_per_sp)]
```

```
## [1] 0.4 0.5
```

For the shoes and pants QDA classifier did best using a probability threshold of 0.2, but similar to all of our previous classifiers there was not a lot of variation in the accuracy using different thresholds.

```
qda_cv_per_ps
```

```
## [1] 0.9351111 0.9351111 0.9353333 0.9351111 0.9351111 0.9351111 0.9350000
## [8] 0.9346667
```

```
percents[qda_cv_per_ps == max(qda_cv_per_ps)]
```

```
## [1] 0.4
```

The pants and shirts QDA classfier did better than the LDA classifier in this situation I think the flexibility in the different covariance matrices helped quite a bit on average we say about a 10 percent increase in accuracy. The probability threshold didn't seem to make a very large difference as there is a tie between the best probability threshold.

k-Nearest Neighbors

Using the k-Nearest Neighbors classifier, there are no assumption on the distribution of any of the features of the predictor variables and this instad using a measure of distance in order to classify the given observation. The training set makes up all of the possible neighbors that each of our "test" observations would be calculating distance from. The training data in this case as well as the number of neighbors chosen matters quite a bit in this case. The training data should not be incredibly imbalanced since if we have very little observations of one class it might be difficult for that class to ever be the majority and be the class we predict.

For example if we have just 3 observations of shoes in the training data, but we use 7 neighbors than we will never correctly predict the uncommon class (shoes) because the other 4 oobservations will always outvote teh 3 votes for shoes. Training set accuracy is also extremely biased if we choose to keep the original observation in because one of the nearest neighbors for our "test" observation in this case would be itself since it is in the training data, so there is already a correct vote for the class since that vote is itself.

```
library(class)
```

```
knn_cv <- function(train, train_lab, k) {
  per <- c()
  for (i in 0:5) {
    ind <- ((1500 * i) + 1):(1500 * (i + 1))
    test_k <- train[ind, ]
    test_k_lab <- train_lab[ind]
    train_k <- train[-(ind), ]
    train_k_lab <- train_lab[-(ind)]
    pred <- knn(train_k, test_k, train_k_lab, k)
    per <- c(per, sum(test_k_lab == pred) / 1500)
  }
  return(mean(per))
}
```

```
knn_cv_per_ss <- c()
knn_cv_per_sp <- c()
knn_cv_per_ps <- c()
ks <- c(5, 10, 15)
for (k in ks) {
  knn_cv_per_ss <- c(knn_cv_per_ss, knn_cv(ss_train, ss_train_lab, k))
  knn_cv_per_sp <- c(knn_cv_per_sp, knn_cv(sp_train, sp_train_lab, k))
  knn_cv_per_ps <- c(knn_cv_per_ps, knn_cv(ps_train, ps_train_lab, k))
}
```

```
knn_cv_per_ss
```

```
## [1] 0.9936667 0.9930000 0.9916667
```

```
ks[knn_cv_per_ss == max(knn_cv_per_ss)]
```

```
## [1] 5
```

```
knn_cv_per_sp
```

```
## [1] 0.9927778 0.9911111 0.9905556
```

```
ks[knn_cv_per_sp == max(knn_cv_per_sp)]
```

```
## [1] 5
```

```
knn_cv_per_ps
```

```
## [1] 0.9402222 0.9346667 0.9316667
```

```
ks[knn_cv_per_ps == max(knn_cv_per_ps)]
```

```
## [1] 5
```

After running some cross validation, the use of 5 neighbors produced the best average test accuracy across the 6 folds, but it is important to note that the use fo 10 neighbors and 15 neighbors did not seem to change the accuracy percentage much.

Using some cross validation to tune hyper parameters, using parametric models all three of the QDA models outperformed the logistic regression models as well as the LDA models when it came to model accuracy. The best hyper parameters for the paired binary QDA classifiers are 0.9 for shirts and shoes, 0.2 for shoes and pants, and 0.7, 0.8, and 0.9 tie for the shirts and pants. For nonparametric models in this case that had no assumptions for the distribution of the X features, I tried out the k-Nearest Neighbor classifier which produced the highest accuracy across all of the different classifiers using 5 neighbors. It was more computationally intensive as it took the longest to run, so if I were consider both computational time as well as accuracy the QDA classfier seems best, so it will be the one that I train our 3 pair training sets on and test the accuracy of the hold out test set that we saved.

train_qda <- qda(train_k, train_k_lab) pred <- ifelse(predict(train_qda, test_k)$posterior[, 1] >= percent, 0, 1) per <- c(per, sum(test_k_lab == pred) / 1500)

```r
ss_qda <- qda(ss_train, ss_train_lab)
sp_qda <- qda(sp_train, sp_train_lab)
ps_qda <- qda(ps_train, ps_train_lab)
```

```r
# hold out test accuracy for shoe shirt qda classfier
sum(ifelse(predict(ss_qda, ss_test)$posterior[, 1] >= 0.9, 0, 1) == ss_test_lab) / length(ss_test_lab)
```

```
## [1] 0.9753333
```

```r
# hold out test accuracy for shoe pants qda classfier
sum(ifelse(predict(sp_qda, sp_test)$posterior[, 1] >= 0.2, 0, 1) == sp_test_lab) / length(sp_test_lab)
```

```
## [1] 0.9773333
```

```r
# hold out test accuracy for pants shirt qda classfier
sum(ifelse(predict(ps_qda, ps_test)$posterior[, 1] >= 0.8, 0, 1) == ps_test_lab) / length(ps_test_lab)
```

```
## [1] 0.9373333
```

Multiclass classification

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?)

```r
shuffled <- sample(1:18000, size = 18000)
shuffled_data <- X_5[shuffled, ]
shuffled_labels <- y[shuffled]
data_train <- shuffled_data[1:13500, ]
data_train_labels <- shuffled_labels[1:13500]
data_test <- shuffled_data[13501:18000, ]
data_test_labels <- shuffled_labels[13501:18000]
```

LDA for Multiclass Classification

As what was summarized in the section above regarding LDA, LDA can also be used for multiclass classification.

```r
lda_cv_m <- function(train, train_lab) {
  per <- c()
  for (i in 0:5) {
    ind <- ((2250 * i) + 1):(2250 * (i + 1))
    test_k <- train[ind, ]
    test_k_lab <- train_lab[ind]
    train_k <- train[-(ind), ]
    train_k_lab <- train_lab[-(ind)]
    train_lda <- lda(train_k, train_k_lab)
    pred <- predict(train_lda, test_k)$class
    per <- c(per, sum(test_k_lab == pred) / 2250)
  }
  return(per)
}
```

```r
lda_cv_m(data_train, data_train_labels)
```

```
## [1] 0.8586667 0.8515556 0.8582222 0.8555556 0.8542222 0.8622222
```

QDA for Multiclass Classification

```r
qda_cv_m <- function(train, train_lab) {
  per <- c()
  for (i in 0:5) {
    ind <- ((2250 * i) + 1):(2250 * (i + 1))
    test_k <- train[ind, ]
    test_k_lab <- train_lab[ind]
    train_k <- train[-(ind), ]
    train_k_lab <- train_lab[-(ind)]
    train_qda <- qda(train_k, train_k_lab)
    pred <- predict(train_qda, test_k)$class
    per <- c(per, sum(test_k_lab == pred) / 2250)
  }
  return(per)
}
```

```r
qda_cv_m(data_train, data_train_labels)
```

```
## [1] 0.9484444 0.9462222 0.9453333 0.9453333 0.9382222 0.9351111
```

k-Nearest Neighbor Multiclass Classification

```r
knn_cv_m <- function(train, train_lab, k) {
  per <- c()
  for (i in 0:5) {
    ind <- ((2250 * i) + 1):(2250 * (i + 1))
    test_k <- train[ind, ]
    test_k_lab <- train_lab[ind]
    train_k <- train[-(ind), ]
    train_k_lab <- train_lab[-(ind)]
    pred <- knn(train_k, test_k, train_k_lab, k)
    per <- c(per, sum(test_k_lab == pred) / 2250)
  }
  return(mean(per))
}
```

```r
ks <- c(3, 4, 5, 6, 7, 8, 9, 10)
knn_cv_m_per <- c()
for (k in ks) {
  knn_cv_m_per <- c(knn_cv_m_per, knn_cv_m(data_train, data_train_labels, k))
}
knn_cv_m_per
```

```
## [1] 0.9594815 0.9560741 0.9556296 0.9539259 0.9536296 0.9517037 0.9504444
## [8] 0.9481481
```

```
ks[knn_cv_m_per == max(knn_cv_m_per)]
```

```
## [1] 3
```

The k-Nearest Neighbor classifier using k = 3 provided the best accuracy using cross validation on the training set.

```
# the hold out test accuracy on the best multiclass model which was the k-NN with k = 3
knn_multi_pred <- knn(data_train, data_test, data_train_labels, 3)
sum(knn_multi_pred == data_test_labels) / length(data_test_labels)
```

```
## [1] 0.9602222
```