Fashion MNIST

Group Member 1 Name: Khang V. Tran **Group Member 1 SID:** 25181590

Group Member 2 Name: Christian Philip Hoeck **Group Member 2 SID:** 3035385003

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("../data_files/FashionMNIST.csv")
```

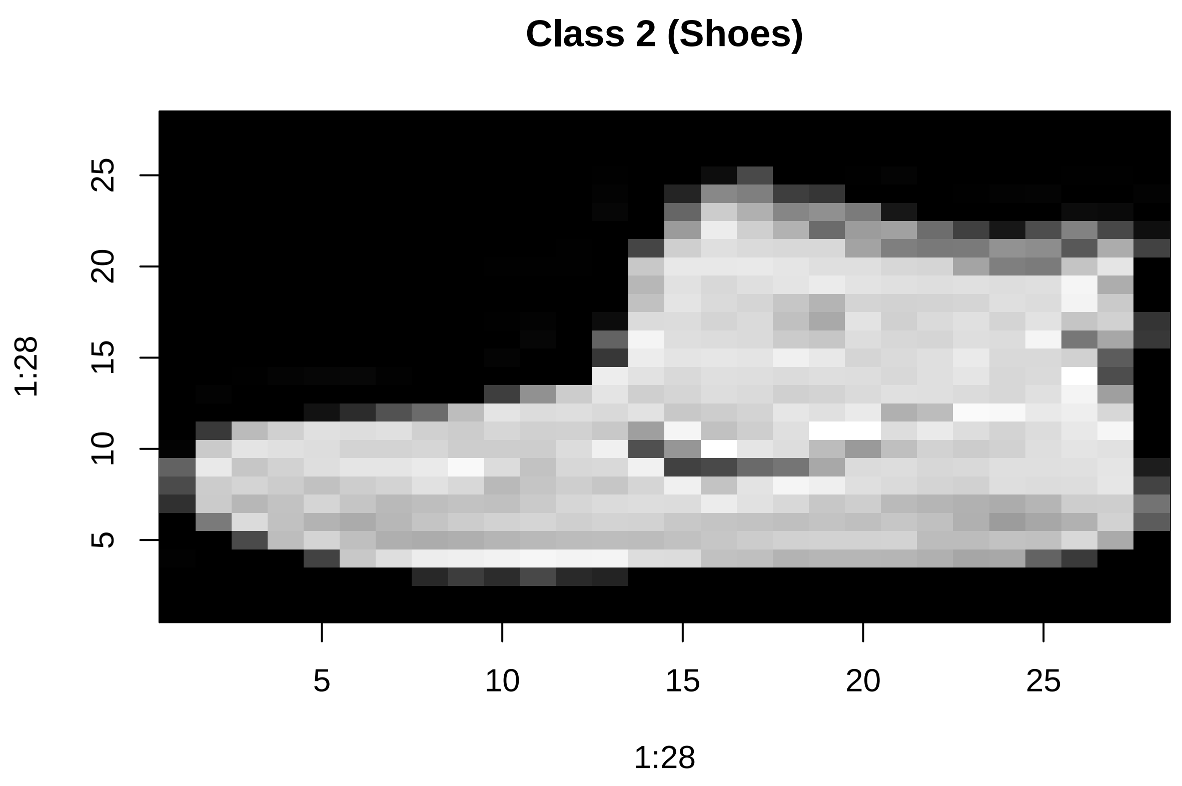
```
## Parsed with column specification:
## cols(
##   .default = col_double()
## )
```

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

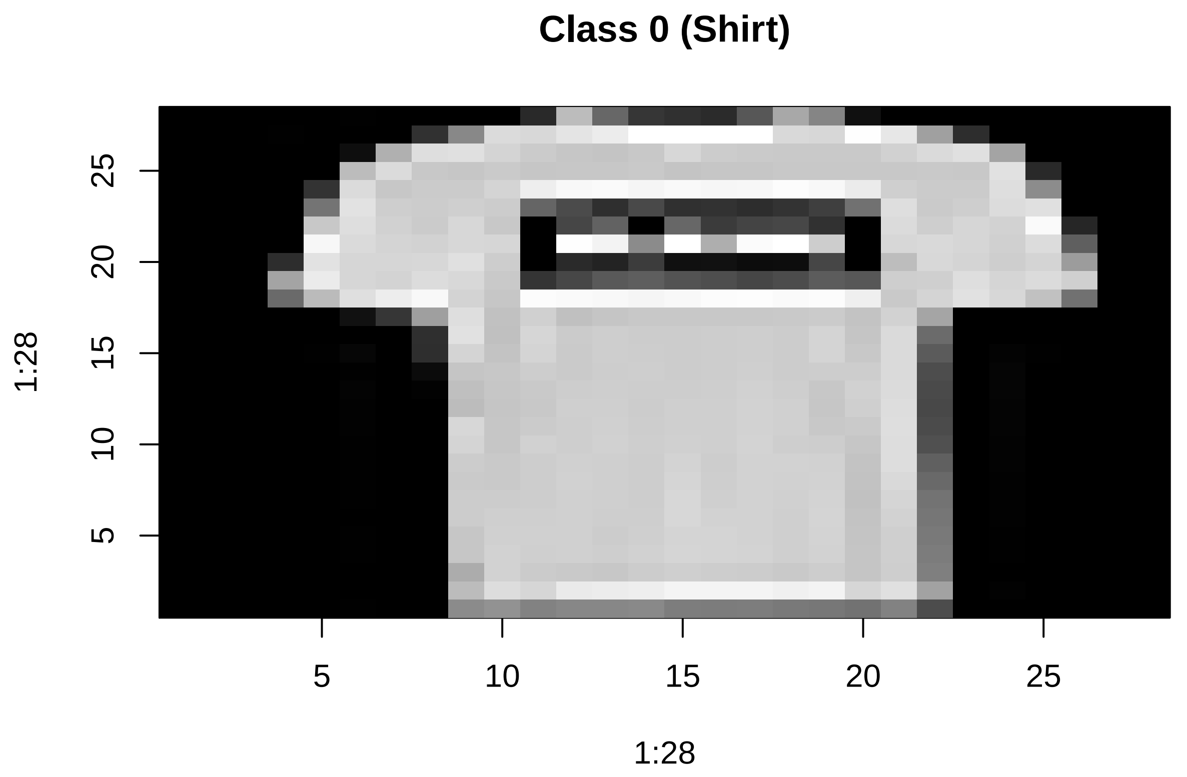
```r
y <- FMNIST$label
X <- subset(FMNIST, select = -c(label))
rm('FMNIST') #remove from memory -- it's a relatively large file
#print(dim(X))
```

We can look at a few of the images:

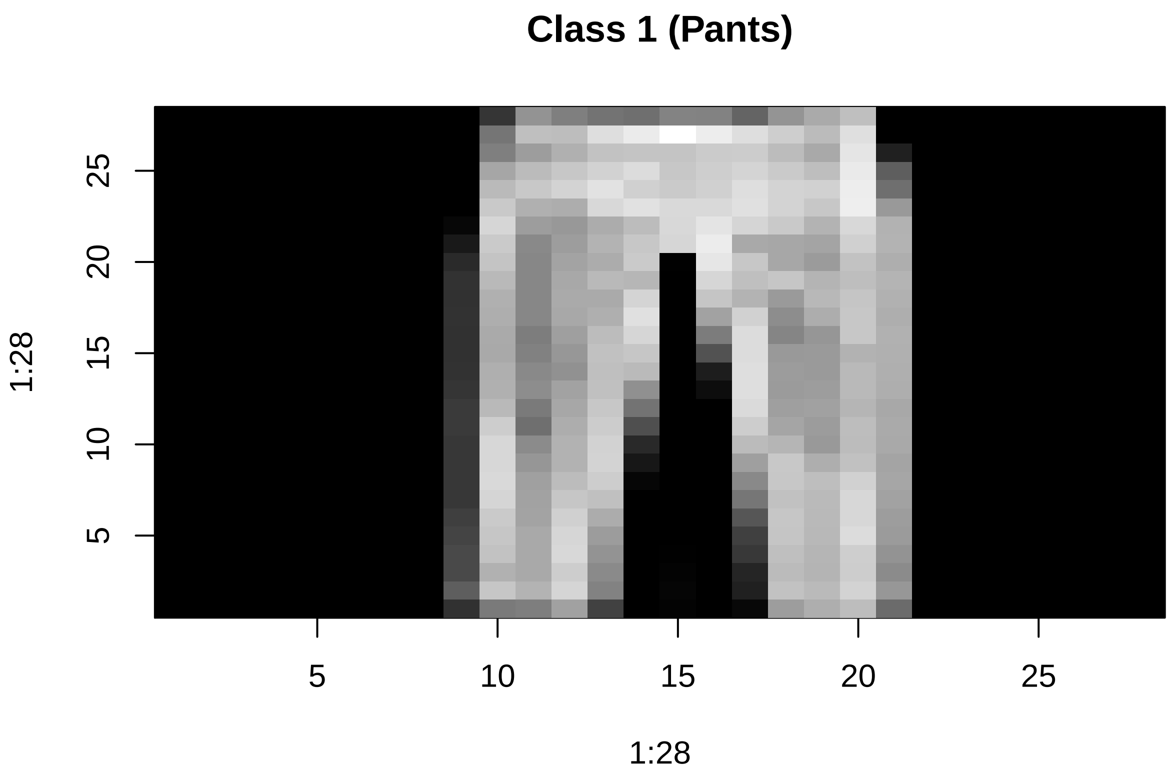
```r
X2 <- matrix(as.numeric(X[1,]), ncol=28, nrow=28, byrow = TRUE)
X2 <- apply(X2, 2, rev)
image(1:28, 1:28, t(X2), col=gray((0:255)/255), main='Class 2 (Shoes)')
```



```
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)')
```



```
X1 <- matrix(as.numeric(X[8,]), 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 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?

The Data and PCA

As mentioned in the describtion, the data consists of 18,000 grayscale 28x28 pixel images of clothing items with corresponding labels, i.e. shoe, shirt and pants. A grayscale 28x28 pixel image is essentially an obeservation with 784 features. Our data set has 6,000 observations for each catagory, so using all 784 feauteres will likely lead to severe overfitting, thereby increasing the variance. The assignment asks us to perform some form of dimension reduction in the exploritory part. The reasons for this are two fold: First it might simply gives us a better intuivtive understanding of which areas of the pictures vary the most, since each new dimension will through its weights corresond to different areas of the picture lighting up or not. Second, the new feautures can be used in the classification task later, as a way to, hopefully, trade of a small increase in bias, for a large reduction in variance.

The dimension reduction method we choose to use is Principal Component Analysis (PCA). Intuitively this method finds the dimension in the original 784-dimensional with the largest variance, and then projects each obeservation onto this dimension, to extract the first principal component. We can then extract the second principal commpoent by finding the next dimension along which most variation occurs, conditional on it being orthogonal to the first dimension.

Technaically this can be don by performing an single value decomposition of the mean-centered data $\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^\top$. The 748 collumn vectors of $\mathbf{V}$ will then correspond to the loading of the corresponding principal components, such that $\mathbf{Z} = \mathbf{X}\mathbf{V}$. By only using a few of the principal components, we wil still keep most of the variation in the data, but reduce the amounts of features used in the classification task.

In now continue to perform PCA. Often one standardizes the data before performing PCA, since it is not scale invariant. However, in this case all the feauteres are levels of white in the corresponding pixel, and they therefore have the same scale. I therefore simply mean-center the data.

```r
library(FactoMineR)

X.centered <- scale(X,center=TRUE, scale=FALSE)

pca <- PCA(X.centered ,scale.unit = FALSE, graph = FALSE,ncp = 200)
```

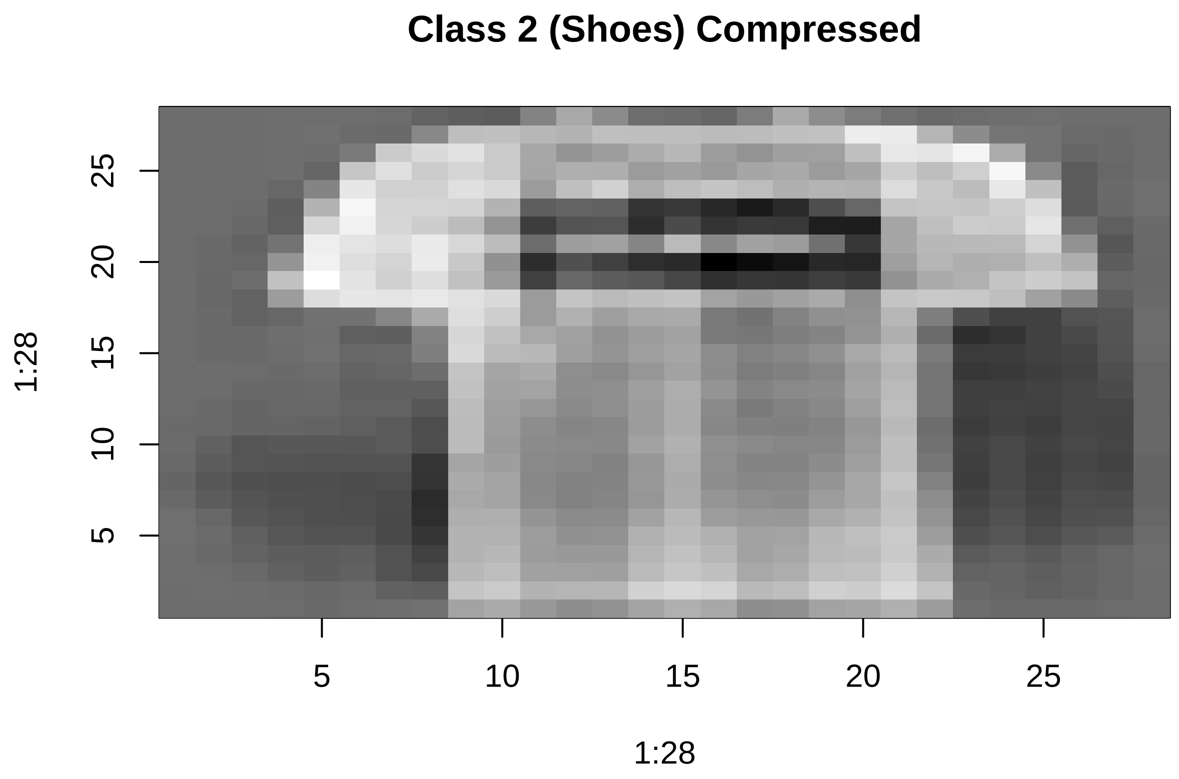
Using the the principal components and the loading vectors we can recover low-dimensional versions of the orginal pictures. This is done by using that $Z_k = XV_k \Rightarrow Z_k V_k^\top \approx X$

where k is the number of retained principal components.

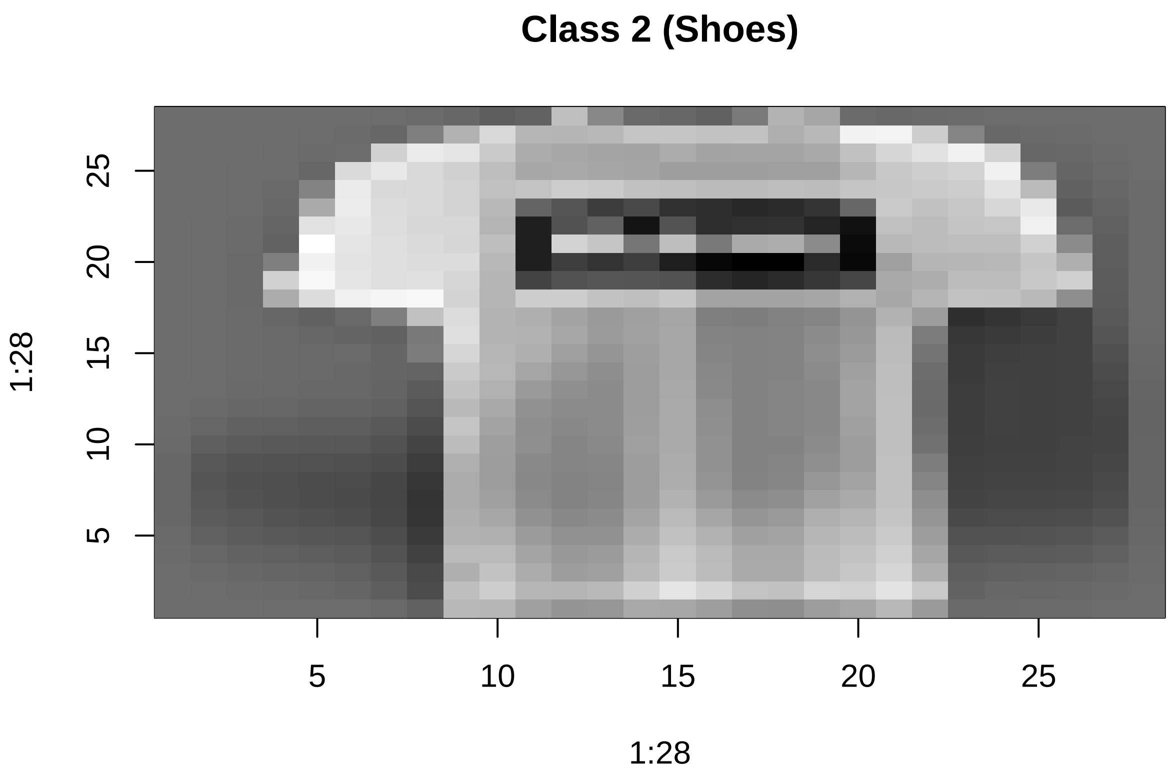
```r
V <- pca$svd$V
Z <- pca$ind[["coord"]]
Zdat <- data.frame(Z)
X.test <- data.frame(Z%*%t(V))
```

These low-dimensional versions can then be viewed in the same way as before. Since the PCA need mean-centered data, we also plot a mean-centered version of the high-dimensional picture.

```r
X2 <- matrix(as.numeric(X.test[2,]), 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) Compressed')
```



```
X2 <- matrix(as.numeric(X.centered[2,]), 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)')
```

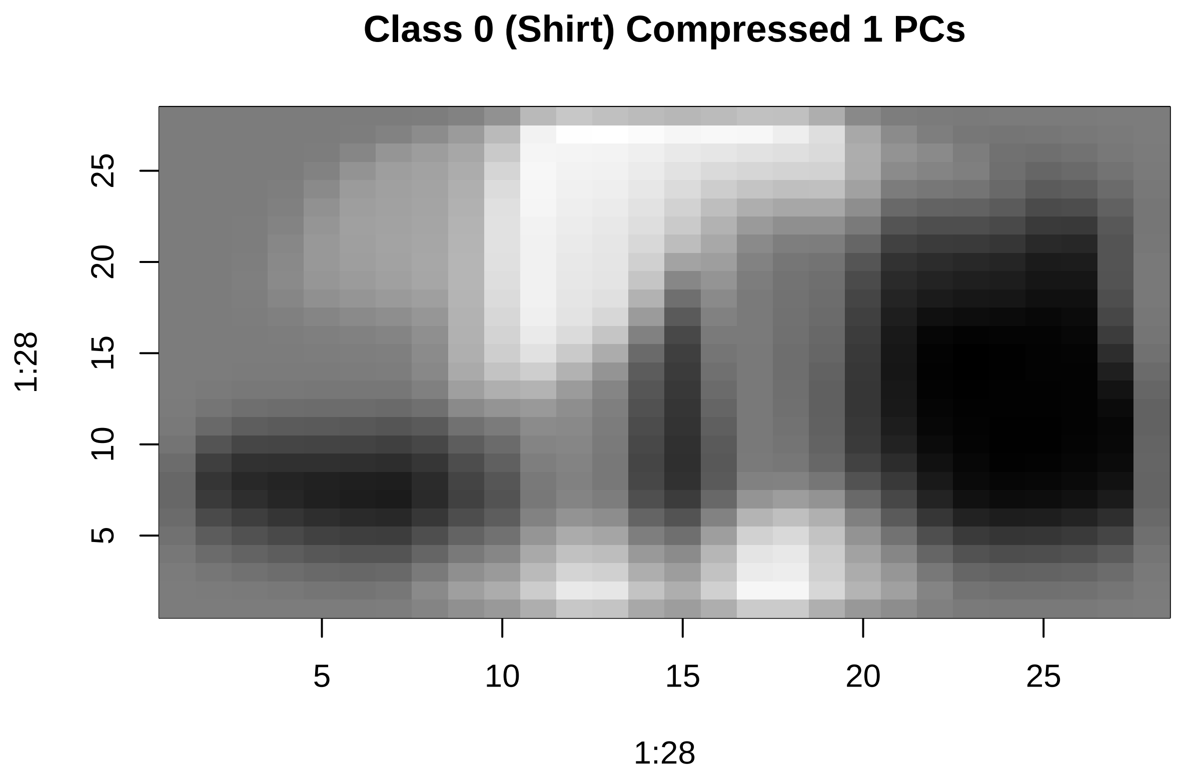


We then examine the number of components needed for classes to still be visually distinguishable. This is different from the way, we normally choose the number of components to keep, which is done using rules, like keep all components with variance over a certain threshold or add compoents until a certain percentage of total variation is explained.

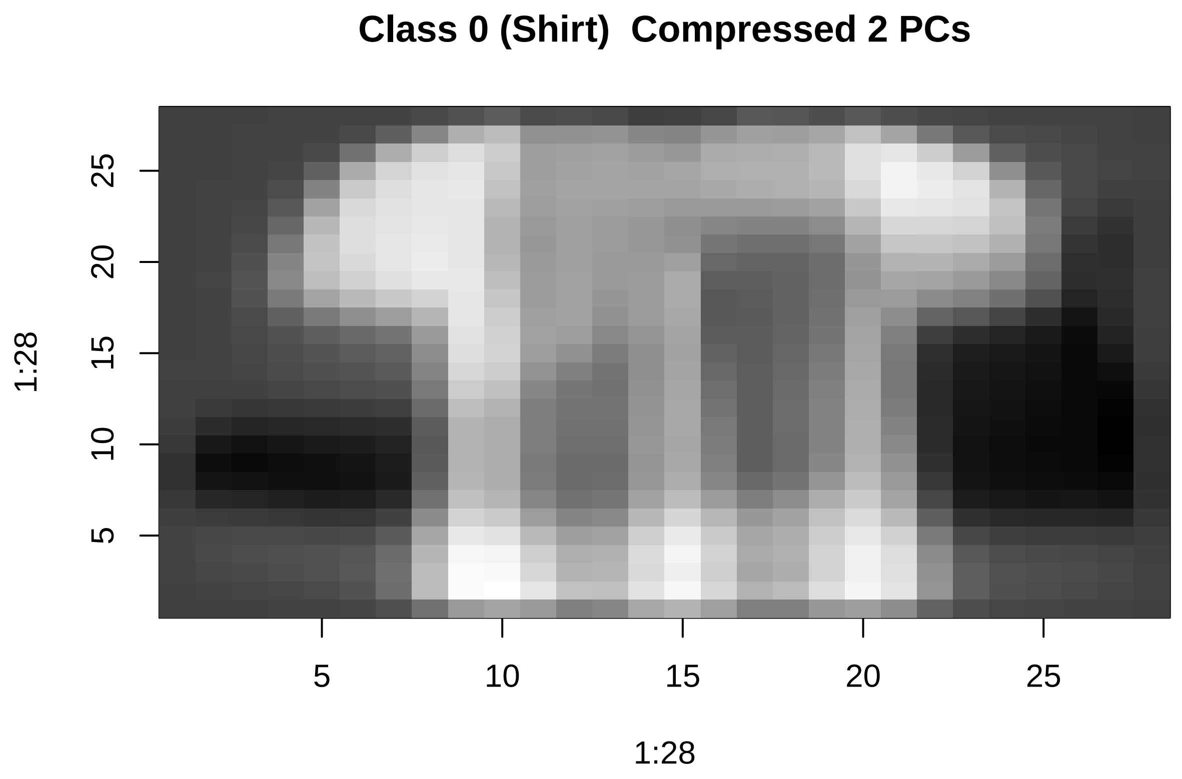
Here I show the the compressed images using 1, 2, and 100, pcs. We see that the using only 1 PC. It is hard to distinguish that it is a shirt. Actually it looks more like pants, but with a less intensive shirt. We also see that a shoe is blacked out. This indicates that the first PC is positively (Negatively) correlated with the pixels containing pants and shirts, and negatively (Positively) correlated with the pixels containing shoes with the opposite sign. When using 2 pcs the shirt is clearly distinguishable. The second pc is therefore positively (negatively) correlated with the pixels related to shirts and negatively (positively) correlated with the pixels related to pants. We also see that the pixels on related to shoes are not affected. In the image using 100 pcs we see that the logo of the shirt is recovered. However it is clear, that only 2 pcs are needed to distinguish the classes from each other. We can therefore reduce a 784-dimesional space to a 2-dimensional space, and still keep the relevant variation.

```r
X.1 <- data.frame(Z[,1]%*%t(V[,1]))
X.2 <- data.frame(Z[,1:2]%*%t(V[,1:2]))
X.3 <- data.frame(Z[,1:3]%*%t(V[,1:3]))
X.100 <- data.frame(Z[,1:100]%*%t(V[,1:100]))
```

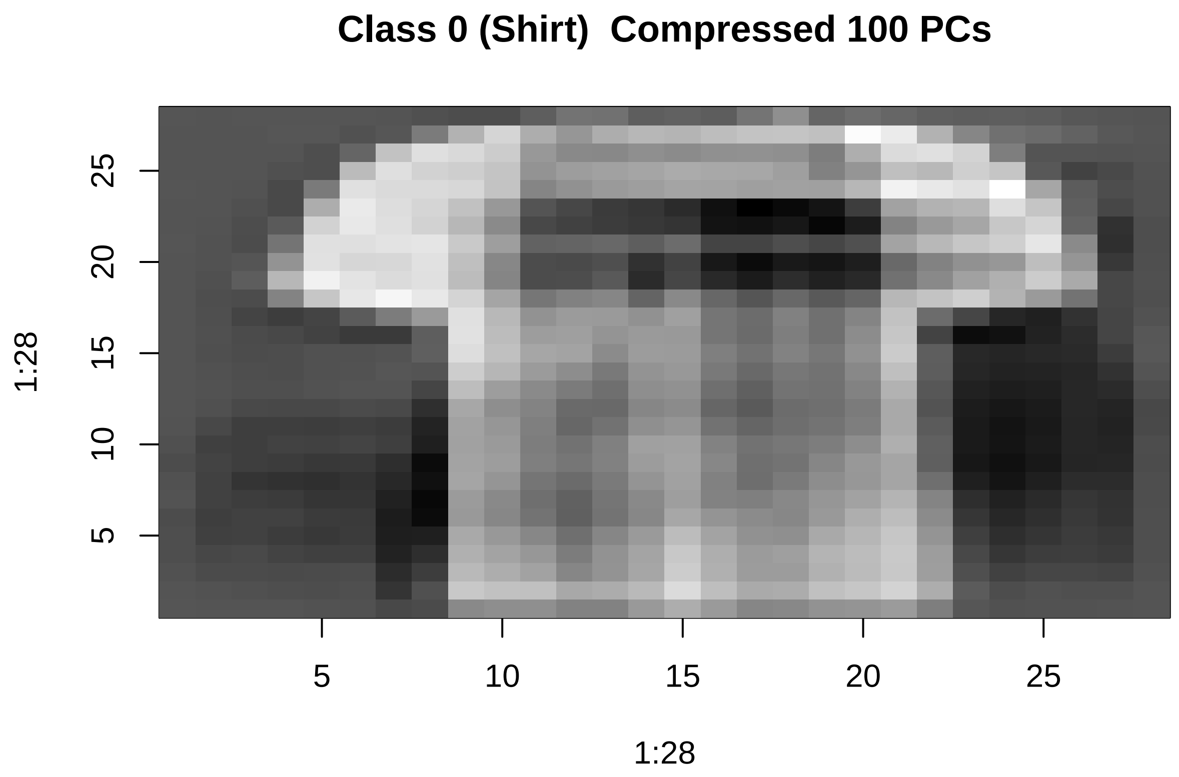
```r
X.image <- matrix(as.numeric(X.1[2,]), ncol=28, nrow=28, byrow = TRUE)
X.image <- apply(X.image, 2, rev)
image(1:28, 1:28, t(X.image), col=gray((0:255)/255), main='Class 0 (Shirt) Compressed 1 PCs')
```



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



```
X.image <- matrix(as.numeric(X.100[2,]), ncol=28, nrow=28, byrow = TRUE)
X.image <- apply(X.image, 2, rev)
image(1:28, 1:28, t(X.image), col=gray((0:255)/255), main='Class 0 (Shirt)  Compressed 100 PCs')
```



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.

Our code

We start by dividing the dataset into three parts: The Training set (60 pct.), Validation set (20 pct.) and Test set (20 pct.). The training set is used to fit the models to the data, and perform tuning of hyperparamets with cv when necessary. The validation set is then used to determine which model has the best performance. If we did not have a seperate validation set, our estimate for the error rate when evalutating which models is best would be downwards biased. This is especially true for very flexible methods which tend to overfit noise in the data. After selecting our prefered model, the test set is used to evaluate its performance. If we just used the error rate used for model selection, the estimate would be downwards biased as this is the estimate condtional on the model winning.

```r
set.seed(1)
train_idx <- sample(nrow(X), nrow(X)*0.6)
test_idx <- sample(seq(1,nrow(X),1)[-train_idx], nrow(X)*0.2)
val_idx <- seq(1,nrow(X),1)[-c(train_idx,test_idx)]
```

We will use four different methods for predicting the binary responses, Lineary Discriminant Analysis, K-Nearest Neigbor, Logistic Regression and Random Forrest. These method have different strengths and weakness, with LDA and Logistic Regressionhaving relatively strong assumptions and therefore high bias, but low variance. KNN and Random forest are on the other hand very flexible, and tend to have low bias. The variance of Random forest is however decreased by it inheirent bagging.

We start by preparing the data, and splitting it into the three sets, i.e. shirt vs. pants (Case 1), shoes vs. shirt (Case 2) and pants vs. shoes (Case 3).

```r
# Bind response and covariates in one dataframe
Xy <- cbind(X,"y" = as.factor(y))
names(Xy) <- make.names(names(Xy))

# Bind response and PCs in one dataframe
Zy <- cbind(Zdat,"y" = as.factor(y))
names(Zy) <- make.names(names(Zy))


# Create three datasets for each binary classification task.
Xy.split <- split(Xy, y)
Xy.1 <- rbind(Xy.split$`0`,Xy.split$`1`)
Xy.2 <- rbind(Xy.split$`0`,Xy.split$`2`)
Xy.3 <- rbind(Xy.split$`1`,Xy.split$`2`)

# Need to refactorize so only to levels exist
Xy.1$y <- factor(Xy.1$y)
Xy.2$y <- factor(Xy.2$y)
Xy.3$y <- factor(Xy.3$y)


# Create three  PC datasets for each binary classification task.
Zy.split <- split(Zy, y)
Zy.1 <- rbind(Zy.split$`0`,Zy.split$`1`)
Zy.2 <- rbind(Zy.split$`0`,Zy.split$`2`)
Zy.3 <- rbind(Zy.split$`1`,Zy.split$`2`)

# Need to refactorize so only to levels exist
Zy.1$y <- factor(Zy.1$y)
Zy.2$y <- factor(Zy.2$y)
Zy.3$y <- factor(Zy.3$y)

# Create relevant subsets of existing training, validation and test subsets
train_idx.1 <- row.names(Xy.1) %in% subset(train_idx, train_idx %in% row.names(Xy.1))
train_idx.2 <- row.names(Xy.2) %in% subset(train_idx, train_idx %in% row.names(Xy.2))
train_idx.3 <- row.names(Xy.3) %in% subset(train_idx, train_idx %in% row.names(Xy.3))

test_idx.1 <- row.names(Xy.1) %in% subset(test_idx, test_idx %in% row.names(Xy.1))
test_idx.2 <- row.names(Xy.2) %in% subset(test_idx, test_idx %in% row.names(Xy.2))
test_idx.3 <- row.names(Xy.3) %in% subset(test_idx, test_idx %in% row.names(Xy.3))
```

```r
val_idx.1 <- row.names(Xy.1) %in% subset(val_idx, val_idx %in% row.names(Xy.1))
val_idx.2 <- row.names(Xy.2) %in% subset(val_idx, val_idx %in% row.names(Xy.2))
val_idx.3 <- row.names(Xy.3) %in% subset(val_idx, val_idx %in% row.names(Xy.3))
```

LDA

We start with Linear Discriminant (LDA) analysis. Simply put LDA works by assuming that all the covariates are drawn from different gaussian distribution depending on their class. Using the gaussian assumption we can estimate the parameters of the distribution, and thereby recover an estimated density function

$$f_k(x)$$

. Using the estimated density and Bayes theorem one can then calculate the posterior density, i.e. the probability that an observation belongs to a specific class conditional on the control variables,

$$Pr(y = k, X = x) = \frac{\pi_k f_k(x)}{\sum_{j=1}^{K} \pi_l f_l(x)}$$

, where

$$pi_k$$

is the prior density, which is simply estimated as the proportion of each class in the sample. From the Bayes Classifier, we know that the class with the highest posterior probability is the best prediction, and that is how the actual class prediction is done. The difference between LDA and QDA, is that for LDA you assume a common covariance structure across classes and only different means, while QDA allows for different covariance structures for each class. LDA is therefore less flexible, which of course results in higher bias but lower variance. Which one is prefered depends on the given setting. For LDA we use the `lda()` function which is part of the MASS library.

Since some of the pixels are constant for all pictures in Case 3, I temporarily remove them. This changes nothing as the dont contain information relevant for Case 3, and does not put lda at a disadvatange.

```r
library(MASS)

# Case 1: 0 vs. 1
# Estimating LDA model and performing prediction
lda.1 <- lda(y~.,Xy.1,subset = train_idx.1)

# Performing prediction and calculating Confusion matrix and error rate for validation set
yhat.lda.1 <- predict(lda.1,newdata = Xy.1[val_idx.1,])
confusion.lda.1 <- table(obs = Xy.1[val_idx.1,"y"], pred = yhat.lda.1$class)
error_rate.lda.1 <- 1 - (sum(diag(confusion.lda.1)) / sum(val_idx.1))



# Case2: 0 vs. 2
# Estimating LDA model and performing prediction
lda.2 <- lda(y~.,Xy.2,subset = train_idx.2)

# Performing prediction and calculating Confusion matrix and error rate for validation set
yhat.lda.2 <- predict(lda.2,newdata = Xy.2[val_idx.2,])
confusion.lda.2 <- table(obs = Xy.2[val_idx.2,"y"], pred = yhat.lda.2$class)
error_rate.lda.2 <- 1 - (sum(diag(confusion.lda.2)) / sum(val_idx.2))

# Case 3: 1 vs. 2
# Estimating LDA model and performing prediction
# Temporarily removing constant features
tmp.Xy.3 <- Xy.3[,apply(Xy.3[train_idx.3,], 2, var, na.rm=TRUE) != 0]
lda.3 <- lda(y~.,tmp.Xy.3,subset = train_idx.3)

# Performing prediction and calculating Confusion matrix and error rate for validation set
yhat.lda.3 <- predict(lda.3,newdata = Xy.3[val_idx.3,])
confusion.lda.3 <- table(obs = Xy.3[val_idx.3,"y"], pred = yhat.lda.3$class)
error_rate.lda.3 <- 1 - (sum(diag(confusion.lda.3)) / sum(val_idx.3))
```

KNN

K-Nearest Neighbor (KNN), is a very simple method, which can be seen as an extremely non-parametric estimation of the posterior probabilities of each class. Given a test observation it simply find the K training observations that are nearest in the covariate space given a specified metric, and calculates the posterior probablity as the proportions of each class. Like Bayes' classifier it then predicts the class with the highest probability. KNN depends on two decision made by the researcher: The metric used and the number of neighborghs included. Most of the time euclidian distance is used, but depending one the data other norms might be more relevant. The most important parameter is the number of neighbor, K, as it decides how flexible or rigid the method is. It can range from

$$k = 1$$

, which is so flexible that it perfectly fits the training sample, to

$$k = n$$

which simply amount to predicting the class with the highest frequency in the sample. The choice of k, is clearly a classic bias-variance tradeoff, and we choose to find the optimal k, using 5-fold cross-validation. Since KNN generally has poor performance in high-dimensional problems, we will use the PCs recovered in the first part, as predictors. The amount of PCs to include will be decided based on 5-fold cross-validation. The range used for k is 1-10, and the range for PC is 2-20 with step size 2. This results in 10x10 grid of parameter settings among which to decide which is best.

The function used is `knn` from the `class` package.

```r
knitr::opts_chunk$set(cache = TRUE)
#### WARNING: This is computationally heavy and will take along time
library(class)

# We first find the optimal numbers of neighbors using 5-fold crossvailidation.
K <- 5
set.seed(1)
# Generate indices of holdout observations
holdout.1 <- split(sample(1:length(train_idx.1)),1:K)
holdout.2 <- split(sample(1:length(train_idx.2)),1:K)
holdout.3 <- split(sample(1:length(train_idx.3)),1:K)

# Initialize CV-error vectors
CV_err.1 <- matrix(rep(NaN,100),10)
CV_err.2 <- matrix(rep(NaN,100),10)
CV_err.3 <- matrix(rep(NaN,100),10)

CV_err_temp.1 <-rep(NaN,K)
CV_err_temp.2 <-rep(NaN,K)
CV_err_temp.3 <-rep(NaN,K)


for(j in 1:10){
for(i in 1:10){

for(k in 1:5){
# Create current training index
cv_idx.1 <- train_idx.1[-holdout.1[[k]]]
cv_idx.2 <- train_idx.2[-holdout.2[[k]]]
cv_idx.3 <- train_idx.3[-holdout.3[[k]]]

# Perfom prediction

pred.knn.1<-knn(Zy.1[cv_idx.1,1:(j*2)],Zy.1[holdout.1[[k]],1:(j*2)],Zy.1[cv_idx.1,"y"],k=i)
pred.knn.2<-knn(Zy.2[cv_idx.2,1:(j*2)],Zy.2[holdout.2[[k]],1:(j*2)],Zy.2[cv_idx.2,"y"],k=i)
pred.knn.3<-knn(Zy.3[cv_idx.3,1:(j*2)],Zy.3[holdout.3[[k]],1:(j*2)],Zy.3[cv_idx.3,"y"],k=i)


# Calculate Confusion matrices and error rates
confusion.knn.1 <- table(obs = Zy.1[holdout.1[[k]],"y"], pred = pred.knn.1)
confusion.knn.2 <- table(obs = Zy.2[holdout.2[[k]],"y"], pred = pred.knn.2)
confusion.knn.3 <- table(obs = Zy.3[holdout.3[[k]],"y"], pred = pred.knn.3)

CV_err_temp.1[k] <- 1 - (sum(diag(confusion.knn.1)) / length(holdout.1[[k]]))
CV_err_temp.2[k] <- 1 - (sum(diag(confusion.knn.2)) / length(holdout.2[[k]]))
CV_err_temp.3[k] <- 1 - (sum(diag(confusion.knn.3)) / length(holdout.3[[k]]))




}

# Calculate mean cv-error rate
CV_err.1[i,j] <- mean(CV_err_temp.1)
CV_err.2[i,j] <- mean(CV_err_temp.2)
CV_err.3[i,j] <- mean(CV_err_temp.3)
}
}

par.opt.1 <- which(CV_err.1 == min(CV_err.1), arr.ind = TRUE)
par.opt.2 <- which(CV_err.2 == min(CV_err.2), arr.ind = TRUE)
par.opt.3 <- which(CV_err.3 == min(CV_err.3), arr.ind = TRUE)

message("Optimal Parameters Case 1")
```

```
## Optimal Parameters Case 1
```

```r
par.opt.1
```

```
##      row col
## [1,]   1  10
```

```
message("Optimal Parameters Case 2")
```

```
## Optimal Parameters Case 2
```

```
par.opt.2
```

```
##      row col
## [1,]   1   6
## [2,]   2   6
```

```
message("Optimal Parameters Case 3")
```

```
## Optimal Parameters Case 3
```

```
par.opt.3
```

```
##       row col
##  [1,]   1   3
##  [2,]   1   4
##  [3,]   1   5
##  [4,]   5   5
##  [5,]   6   5
##  [6,]   1   6
##  [7,]   5   6
##  [8,]   1   7
##  [9,]   5   7
## [10,]   1   8
## [11,]   1   9
## [12,]   1  10
## [13,]   2  10
```

We see that case 2 and 3 has multiple minima. This probably happens because so few errors are made, so most the remaining error are irreducible noise. We simply choose the first minima for each case, which corresponds to 1 neigbor and 10x2=20 PCs for Case 1, 1 neighbor and 6x2=12 PC for Case 2 and 1 neigbor and 3x2 = 6 PCs for Case 3. We now proced to estimate validation error, using our optimal parameters and the validation set.

```
# Perform prediction with optimal parameters
pred.knn.1 <- knn(Zy.1[train_idx.1,1:(par.opt.1[1,2]*2)],Zy.1[val_idx.1,1:(par.opt.1[1,2]*2)],Zy.1[trai
pred.knn.2 <- knn(Zy.2[train_idx.2,1:(par.opt.2[1,2]*2)],Zy.2[val_idx.2,1:(par.opt.2[1,2]*2)],Zy.2[trai
pred.knn.3 <- knn(Zy.3[train_idx.3,1:(par.opt.3[1,2]*2)],Zy.3[val_idx.3,1:(par.opt.3[1,2]*2)],Zy.3[trai


# Calculate the confusion matrix and error rate for the validation set.
confusion.knn.1 <- table(obs = Zy.1[val_idx.1,"y"], pred = pred.knn.1)
confusion.knn.2 <- table(obs = Zy.2[val_idx.2,"y"], pred = pred.knn.2)
confusion.knn.3 <- table(obs = Zy.3[val_idx.3,"y"], pred = pred.knn.3)

error_rate.knn.1 <- 1 - (sum(diag(confusion.knn.1)) / sum(val_idx.1))
error_rate.knn.2 <- 1 - (sum(diag(confusion.knn.2)) / sum(val_idx.2))
error_rate.knn.3 <- 1 - (sum(diag(confusion.knn.3)) / sum(val_idx.3))
```

Logistic Regression

We then turn to logistic regression. It models the probability of being class

$$y$$

conditional on

$$X$$

, as a logitstic function of a linear combination of the features,

$$p(y = 1|X = x) = \phi(\beta^\top x_i)$$

. These probabilities are used to create a likelyhood function

$$\prod_{i=1}^{n} \phi(\beta^\top x_i)^{y_i}(1 - \phi(\beta^\top x_i))^{1-y_i}$$

. One can then use gradient decent to maximize the log-likelihood function to find the optimal weights. Often LDA and logistic regression performs similar as they both create linear decision boundaries. However, if the gaussian asumption made when using LDA does not hold, logistic regression will tend to perfom better (ISL p. 151). The opposite is true if the assumption approximately holds. To perfom logistic regression we use the `glm` function.

```r
knitr::opts_chunk$set(cache = TRUE)
# Case 1: 0 vs. 1
# Estimating logistic model
log.1 <- glm(y~.,data=Xy.1,subset = train_idx.1,family = binomial, maxit = 100)

# Performing prediction and calculating Confusion matrix and error rate for validation set
yhat.log.1 <- round(predict(log.1,newdata = Xy.1[val_idx.1,],type="response"))
confusion.log.1 <- table(obs = Xy.1[val_idx.1,"y"], pred = yhat.log.1)
error_rate.log.1 <- 1 - (sum(diag(confusion.log.1)) / sum(val_idx.1))

# Case 2: 0 vs. 2
# Estimating logistic model
log.2 <- glm(y~.,data=Xy.2,subset = train_idx.2,family = binomial, maxit = 100)

# Performing prediction and calculating Confusion matrix and error rate for validation set
yhat.log.2 <- round(predict(log.2,newdata = Xy.2[val_idx.2,],type="response"))
confusion.log.2 <- table(obs = Xy.2[val_idx.2,"y"], pred = yhat.log.2)
error_rate.log.2 <- 1 - (sum(diag(confusion.log.2)) / sum(val_idx.2))

# Case 3: 1 vs. 2
# Estimating logistic model
log.3 <- glm(y~.,data=Xy.3,subset = train_idx.3,family = binomial, maxit = 100)

# Performing prediction and calculating Confusion matrix and error rate for validation set
yhat.log.3 <- round(predict(log.3,newdata = Xy.3[val_idx.3,],type="response"))
confusion.log.3 <- table(obs = Xy.3[val_idx.3,"y"], pred = yhat.log.3)
error_rate.log.3 <- 1 - (sum(diag(confusion.log.3)) / sum(val_idx.3))
```

Random Forest

When using Random Forest one creates B new samples of data using bootstrapping of the original data and then fits a decision tree to each one. In addition to the randomness induced by the different bootstrap samples, one tries to uncorrelate the trees even more, by restricting the number of features considered at each node-split to a small subsample of the features, usually $\sqrt{p}$ features. By making the trees less correlated, Random Forest becomes a method that has the same low bias of the decision trees, but has a much lower variance, since the average of 50 or 500 trees varies less from the mean hypothesis than a single tree does. Another benifit of this method, and bagging in general, is that one can use the so called "Out-of-bag" error as an estimate for the Test MSE. This means that one can use the entire sample for training. However, for this exercise we will use the same training sample for all the methods and and estimate the Test and Validation MSE using the hold-out method, in addition to reporting th out-of-bag error.

For Random Forest and use the `randomForest` package. The number of features considered at each split is set at the default

$$\sqrt{p}$$

, which has been shown to be the theoretical optimal. Each random forest consists of 500 trees.

```r
knitr::opts_chunk$set(cache = TRUE)
library(randomForest)
```

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

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

```r
set.seed(1)

# Case 1: 0 vs. 1
# Estimating Random forest model
rf.1 <- randomForest(y~.,Xy.1,subset = train_idx.1)

# Performing prediction and calculating Confusion matrix and error rate for validation set
yhat.rf.1 <- predict(rf.1,newdata = Xy.1[val_idx.1,])
confusion.rf.1 <- table(obs = Xy.1[val_idx.1,"y"], pred = yhat.rf.1)
error_rate.rf.1 <- 1 - (sum(diag(confusion.rf.1)) / sum(val_idx.1))


# Case 2: 0 vs. 2
# Estimating Random forest model
rf.2 <- randomForest(y~.,Xy.2,subset = train_idx.2)

# Performing prediction and calculating Confusion matrix and error rate for validation set
yhat.rf.2 <- predict(rf.2,newdata = Xy.2[val_idx.2,])
confusion.rf.2 <- table(obs = Xy.2[val_idx.2,"y"], pred = yhat.rf.2)
error_rate.rf.2 <- 1 - (sum(diag(confusion.rf.2)) / sum(val_idx.2))


# Case 3: 1 vs. 2
# Estimating Random forest model
rf.3 <- randomForest(y~.,Xy.1,subset = train_idx.3 )

# Performing prediction and calculating Confusion matrix and error rate for validation set
yhat.rf.3 <- predict(rf.3,newdata = Xy.1[val_idx.3,])
confusion.rf.3 <- table(obs = Xy.1[val_idx.3,"y"], pred = yhat.rf.3)
error_rate.rf.3 <- 1 - (sum(diag(confusion.rf.3)) / sum(val_idx.3))
```

As mentioned the test error rate can be estimated as the Out-of-Bag error rate for all bagging methods. The OOB error rates for the three cases are

```r
mean(rf.1$err.rate[,"OOB"])
```

```
## [1] 0.01038749
```

```r
mean(rf.2$err.rate[,"OOB"])
```

```
## [1] 0.001109157
```

```r
mean(rf.3$err.rate[,"OOB"])
```

```
## [1] 0.00852798
```

We see that the random forests generally perfom very well. A deeper analysis of the errors and confusion matrices is done in the model selection section.

Method selection

The validation the confusion matrix and validation error for each method and each case now presented, and the best one is chosen.

Case 1: 0 vs. 1

Below are all the confusion matrices and a table of the validation errors. We see that all the models generally perfom well with a error rate of around 2 pct. for LDA and 5 pct. for Logistic Regression. KNN has an even small error rate of 1 pct. Random forest, however, has the lowest with an error rate of 0.7 pct. From the confusion matrix we see that LDA, KNN and Random Forest methods generally mistake pants for shirts more than shirts for pants, but logistic regression mistakes more shirts for pants. Our prefered model in this case is the Random forest.

```r
error.1 <- data.frame("LDA" = error_rate.lda.1,"KNN" = error_rate.knn.1 ,"Logistic Regression" = error_r
error.1
```

```
##          LDA        KNN Logistic.Regression Random.Forest
## 1 0.02293968 0.01062022          0.05055225    0.00722175
```

```r
message("LDA Confusion matrix")
```

```
## LDA Confusion matrix
```

```r
confusion.lda.1
```

```
##    pred
## obs    0    1
##   0 1151   21
##   1   33 1149
```

```
message("KNN Confusion matrix")
```

```
## KNN Confusion matrix
```

```
confusion.knn.1
```

```
##    pred
## obs    0    1
##   0 1162   10
##   1   15 1167
```

```
message("Logistic Regression Confusion matrix")
```

```
## Logistic Regression Confusion matrix
```

```
confusion.log.1
```

```
##    pred
## obs    0    1
##   0 1077   95
##   1   24 1158
```

```
message("Random Forest Confusion matrix")
```

```
## Random Forest Confusion matrix
```

```
confusion.rf.1
```

```
##    pred
## obs    0    1
##   0 1171    1
##   1   16 1166
```

Case 2: 0 vs. 2

I Case 2 performance is extremely good for all method. Random forest still has a very low error rate of 0.04 pct. LDA has an error rate of 0.1 pct., Logistic Regression has an error rate of 0.3 pct. KNN is winner with an error rate of 0. From the confusion matrix we see that allmost no errors are made, with random forrest only having one error. Our prefered model in this case is the KNN.

```
error.2 <- data.frame("LDA" = error_rate.lda.2,"KNN" = error_rate.knn.2, "Logistic Regression" = error_r
error.2
```

```
##           LDA KNN Logistic.Regression Random.Forest
## 1 0.001240695   0         0.003722084  0.0004135649
```

```
message("LDA Confusion matrix")
```

```
## LDA Confusion matrix
```

```
confusion.lda.2
```

```
##    pred
## obs    0    2
##   0 1170    2
##   2    1 1245
```

```
message("KNN Confusion matrix")
```

```
## KNN Confusion matrix
```

```
confusion.knn.2
```

```
##    pred
## obs    0    2
##   0 1172    0
##   2    0 1246
```

```
message("Logistic RegressionConfusion matrix")
```

```
## Logistic RegressionConfusion matrix
```

```
confusion.log.2
```

```
##    pred
## obs    0    1
##   0 1165    7
##   2    2 1244
```

```
message("Random Forest Confusion matrix")
```

```
## Random Forest Confusion matrix
```

```
confusion.rf.2
```

```
##    pred
## obs    0    2
##   0 1172    0
##   2    1 1245
```

Case 3: 1 vs. 2

In this case performance is also really good. KNN also has the best performance again, with an error rate of 0.04 pct. LDA has an error rate of 0.1 pct., Logistic Regression has an error rate of 0.3 pct and Random Forest has an error rate of 0.9 pct. KNN is therefore our prefered method for Case 3.

```
error.3 <- data.frame("LDA" = error_rate.lda.3,"KNN" = error_rate.knn.3,"Logistic Regression" = error_ra
error.3
```

```
##           LDA          KNN Logistic.Regression Random.Forest
## 1 0.003706755 0.0004118616         0.007413509   0.009472817
```

```r
message("LDA Confusion matrix")
```

```
## LDA Confusion matrix
```

```r
confusion.lda.3
```

```
##    pred
## obs    1    2
##   1 1176    6
##   2    3 1243
```

```r
message("KNN Confusion matrix")
```

```
## KNN Confusion matrix
```

```r
confusion.knn.3
```

```
##    pred
## obs    1    2
##   1 1181    1
##   2    0 1246
```

```r
message("Logistic RegressionConfusion matrix")
```

```
## Logistic RegressionConfusion matrix
```

```r
confusion.log.3
```

```
##    pred
## obs    0    1
##   1 1169   13
##   2    5 1241
```

```r
message("Random Forest Confusion matrix")
```

```
## Random Forest Confusion matrix
```

```r
confusion.rf.3
```

```
##    pred
## obs    0    1
##   0 1178    4
##   1   19 1227
```

Estimating test error

In Case 1 Random forest was the best binary classifier and in Case 2 and 3 KNN wast the best. We will therefore estimate the test error, using these models trained on the combined training and validation test, and use the test set wich has not been touched yet, to estiamte the out-of-sample error rate. For KNN we have to perform cross-validation again to determine the optimal parameters for the new set.

We start by combining the training and validation set.

```
# Combining training and validation sets
train_val_idx.1 <- train_idx.1|val_idx.1
train_val_idx.2 <- train_idx.2|val_idx.2
train_val_idx.3 <- train_idx.3|val_idx.3
```

We then fit a random forest model for Case 1 and estimate the out-of-sample error rate using the test set.

```
knitr::opts_chunk$set(cache = TRUE)
#Case 1:

# Estimating Random forest model
rf.1.test <- randomForest(y~.,Xy.1,subset = train_val_idx.1)

# Performing prediction and calculating Confusion matrix and error rate for test set
yhat.test.1 <- predict(rf.1.test,newdata = Xy.1[test_idx.1,])
confusion.test.1 <- table(obs = Xy.1[test_idx.1,"y"], pred = yhat.test.1)
error_rate.test.1 <- 1 - (sum(diag(confusion.test.1)) / sum(test_idx.1))
```

We then use cross validation to find the new optimal parameters for the KNN models for Case 2 and 3:

```
knitr::opts_chunk$set(cache = TRUE)
# Cross validation for Case 2 and 3:

# We first find the optimal numbers of neighbors using 5-fold crossvailidation.
K <- 5
set.seed(1)
# Generate indices of holdout observations
holdout.2 <- split(sample(1:length(train_val_idx.2)),1:K)
holdout.3 <- split(sample(1:length(train_val_idx.3)),1:K)

# Initialize CV-error vectors
CV_err.2 <- matrix(rep(NaN,100),10)
CV_err.3 <- matrix(rep(NaN,100),10)

CV_err_temp.2 <-rep(NaN,K)
CV_err_temp.3 <-rep(NaN,K)


for(j in 1:10){
for(i in 1:10){

for(k in 1:5){
# Create current training index
cv_idx.2 <- train_idx.2[-holdout.2[[k]]]
cv_idx.3 <- train_idx.3[-holdout.3[[k]]]
# Perfom prediction
pred.knn.2<-knn(Zy.2[cv_idx.2,1:(j*2)],Zy.2[holdout.2[[k]],1:(j*2)],Zy.2[cv_idx.2,"y"],k=i)
pred.knn.3<-knn(Zy.3[cv_idx.3,1:(j*2)],Zy.3[holdout.3[[k]],1:(j*2)],Zy.3[cv_idx.3,"y"],k=i)


# Calculate Confusion matrices and error rates
confusion.knn.2 <- table(obs = Zy.2[holdout.2[[k]],"y"], pred = pred.knn.2)
confusion.knn.3 <- table(obs = Zy.3[holdout.3[[k]],"y"], pred = pred.knn.3)


CV_err_temp.2[k] <- 1 - (sum(diag(confusion.knn.2)) / length(holdout.2[[k]]))
CV_err_temp.3[k] <- 1 - (sum(diag(confusion.knn.3)) / length(holdout.3[[k]]))

}

# Calculate mean cv-error rate

CV_err.2[i,j] <- mean(CV_err_temp.2)
CV_err.3[i,j] <- mean(CV_err_temp.3)
}
}


par.opt.2 <- which(CV_err.2 == min(CV_err.2), arr.ind = TRUE)
par.opt.3 <- which(CV_err.3 == min(CV_err.3), arr.ind = TRUE)

message("Optimal Parameters Case 2")
```

```
## Optimal Parameters Case 2
```

```
par.opt.2
```

```
##      row col
## [1,]   1   6
## [2,]   1   7
## [3,]   1   8
## [4,]   1  10
```

```
message("Optimal Parameters Case 3")
```

```
## Optimal Parameters Case 3
```

```
par.opt.3
```

```
##      row col
## [1,]   1   4
## [2,]   2   6
## [3,]   2   9
## [4,]   3   9
```

Again we have multipli minima, and we simply choose the first minima.

After determining the new optimal parameters, we then estimate the out-of-sample error-rate for Case 2 and 3 using the test sets.

```
#Case 2:
# Performing prediction and calculating Confusion matrix and error rate for test set


pred.knn.test.2 <- knn(Zy.2[train_val_idx.2,1:(par.opt.2[1,2]*2)],Zy.2[test_idx.2,1:(par.opt.2[1,2]*2)]

confusion.test.2 <- table(obs = Zy.2[test_idx.2,"y"], pred = pred.knn.test.2)
error_rate.test.2 <- 1 - (sum(diag(confusion.test.2)) / sum(test_idx.2))


#Case 3:
# Performing prediction and calculating Confusion matrix and error rate for test set
pred.knn.test.3 <- knn(Zy.3[train_val_idx.3,1:(par.opt.3[1,2]*2)],Zy.3[test_idx.3,1:(par.opt.3[1,2]*2)]

confusion.test.3 <- table(obs = Zy.3[test_idx.3,"y"], pred = pred.knn.test.3)
error_rate.test.3<- 1 - (sum(diag(confusion.test.3)) / sum(test_idx.3))
```

```
Model <- c("Random Forest","KNN","KNN")
error.test <- data.frame("Case 1" = error_rate.test.1,"Case 2" = error_rate.test.2, "Case 3" = error_ra
error.test <- rbind("Model" = Model, "Error Rate" = error.test)
error.test
```

```
##                          Case.1               Case.2               Case.3
## Model             Random Forest                  KNN                  KNN
## Error Rate 0.00584307178631049 0.000419991600168035 0.000412711514651276
```

```
message("Case 1 Confusion matrix")
```

```
## Case 1 Confusion matrix
```

```
confusion.test.1
```

```
##    pred
## obs    0    1
##   0 1172    5
##   1    9 1210
```

```
message("Case 2 Confusion matrix")
```

```
## Case 2 Confusion matrix
```

```
confusion.test.2
```

```
##    pred
## obs    0    2
##   0 1176    1
##   2    0 1204
```

```
message("Case 3 Confusion matrix")
```

```
## Case 3 Confusion matrix
```

```
confusion.test.3
```

```
##    pred
## obs    1    2
##   1 1219    0
##   2    1 1203
```

The Estimated out-of-sample error for Case 1 using Random Forest is 0.6 pct. For Case 2 and 3 the estimated error rate for the KNN models using the test data is 0.04 pct. In the confusion matrices we see that both the KNN models only has one error. Overall the final test performance for the prefered methods is really good, but the error rate in Case 1 is more than ten times higher than the other two cases.

Deployment

Final step would be to fit the three prefered models to 100 pct. of their corresponding subsets (Training, validation and Test), ie. the entire dataset. It would then be ready for deployment.

This concludes the binary classification tasks.

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?)

Our Code

For multiclass classification we will use most of the methods from the binary task as they easily allow for multible classes. However, we will not use Logistic Regression even though it performs multiclass problems, as this has not been covered in the course. All the methods have already been described when performing binary classification.

LDA

We start with Linear Discriminant (LDA) analysis.

```
library(MASS)
# Estimating LDA model and performing prediction
lda.mul <- lda(y~.,Xy,subset = train_idx)
yhat.lda.mul <- predict(lda.mul,newdata = Xy[test_idx,])

# Calculating Confusion matrix
confusion.lda.mul <- table(obs = y[test_idx], pred = yhat.lda.mul$class)
error_rate.lda.mul <- 1 - (sum(diag(confusion.lda.mul)) / length(test_idx))
```

KNN

We then estimate the KNN model. Again we use PCs as predictors and perfrom 5-fold cross-validation to determine the optimal number of neigbors and PCs used.

```r
knitr::opts_chunk$set(cache = TRUE)
library(class)

# We first find the optimal numbers of neighbors using 5-fold crossvailidation.
K <- 5
set.seed(1)
# Generate indices of holdout observations
holdout <- split(sample(1:length(train_idx)),1:K)

# Initialize CV-error vectors
CV_err <- matrix(rep(NaN,100),10)
CV_err_temp <-rep(NaN,K)
for(j in seq(1,10,1)){
for(i in 1:10){

for(k in 1:5){
# Create current training index
cv_idx <- train_idx[-holdout[[k]]]

# Perform prediction
pred.knn <- knn(Zy[cv_idx,1:(j*2)],Zy[holdout[[k]],1:(j*2)],Zy[cv_idx,"y"],k=i)

# Calculate Confusion matrices and error rates
confusion.knn.mul <- table(obs = y[holdout[[k]]], pred = pred.knn)
CV_err_temp[k] <- 1 - (sum(diag(confusion.knn.mul)) / length(y[holdout[[k]]]))

}

# Calculate mean error.
CV_err[i,j] <- mean(CV_err_temp)
}
}

par.opt <- which(CV_err == min(CV_err), arr.ind = TRUE)
par.opt
```

```
##      row col
## [1,]   1   9
```

We find that the optimal number of neigbors is 1, and the optimal number of PCs is 9*2 = 18. We then perform the prediction with theese parameters, and calculate the confusion matrix and error rate for the validation set.

```r
# Perform prediction with optimal parameters
pred.knn <- knn(Zy[train_idx,1:(par.opt[1,2]*2)],Zy[val_idx,1:(par.opt[1,2]*2)],Zy[train_idx,"y"],k=par

# Calculate the confusion matrix and error rate for the validation set.
confusion.knn.mul <- table(obs = y[val_idx], pred = pred.knn)
error_rate.knn.mul <- 1 - (sum(diag(confusion.knn.mul)) / length(y[val_idx]))
```

Random Forest

We then estimate a Random Forest model, using the same setting as for the binary classification task:

$$\sqrt{p}$$

feauteres considered each split, and 500 trees. We then calculate the confusion matrix and error rate for the validation set.

```r
knitr::opts_chunk$set(cache = TRUE)
library(randomForest)
#### WARNING: This is computationally heavy and will take along time
set.seed(1)
# Estimate random forest model
rf.mul <- randomForest(y~.,Xy,subset = train_idx)
yhat.rf.mul <- predict(rf.mul,newdata = Xy[val_idx,])

# Calculate the confusion matrix and error rate for the validation set.
confusion.rf.mul <- table(obs = y[val_idx], pred = yhat.rf.mul)
error_rate.rf.mul <- 1 - (sum(diag(confusion.rf.mul)) / length(test_idx))
```

Method selection

Below are all the confusion matrices and a table of the validation errors. We see that all the models generally perfom very well. LDA has the highest error rate of 2 pct. KNN and Random forest are very close. However, Random forest has the lowest with an error rate of 0.5 pct. while KNN has an error rate of 0.6 pct. Generally, this indicates that increased flexiblilty in KNN and Random forest is more important for this particular data than the reduction in variance when using more rigid methods like LDA.

From the confusion matrix we see that all methods generally mistake pants for shirts more than any other mistake. This also explains why Random Forest is the best for multiclass classification even though, KNN was better in 2 out of 3 of the binary classification task. Random forest was the best in Case 1, which was shirts vs. pants, and since this is task every method is worst at, the higher performance here matters more than performance when discriminating shoes from shirts and pants.

To summarize: Random Forest has the lowest validation error rate, and is therefore our prefered.

```r
error.mul <- data.frame("LDA" = error_rate.lda.mul,"KNN" =error_rate.knn.mul ,"Random Forest" = error_ra
error.mul
```

```
##      LDA         KNN Random.Forest
## 1 0.0125 0.008055556         0.005
```

```r
message("LDA Confusion matrix")
```

```
## LDA Confusion matrix
```

```r
confusion.lda.mul
```

```
##    pred
## obs    0    1    2
##   0 1168    7    2
##   1   29 1190    0
##   2    6    1 1197
```

```r
message("KNN Confusion matrix")
```

```
## KNN Confusion matrix
```

```r
confusion.knn.mul
```

```
##    pred
## obs    0    1    2
##   0 1160   12    0
##   1   16 1165    1
##   2    0    0 1246
```

```r
message("Random Forest Confusion matrix")
```

```
## Random Forest Confusion matrix
```

```r
confusion.rf.mul
```

```
##    pred
## obs    0    1    2
##   0 1171    1    0
##   1   16 1166    0
##   2    1    0 1245
```

Estimating test error

Random forest also turned out to be the best multiclass classifier. We train a random forest model on the combined training and validation set, and estiamte the out-of-sample error rate using the test set. We see that the test error for the the models is 0.4 pct. As mentioned before, it is clear from the confusion matrix, that the hardest task is discriminating shirts and pants. This was the one binary task random forest was best at, which is the reason why it is the best multiclass classifier for this dataset.

```r
#### WARNING: This is computationally heavy and will take along time
#  Combine training and validation set
train_val.idx <- rbind(train_idx,val_idx)

# Performing prediction and calculating Confusion matrix and error rate for test set
rf.mul.test <- randomForest(y~.,Xy,subset = train_val.idx)
yhat.test.mul <- predict(rf.mul.test,newdata = Xy[test_idx,])
confusion.test.mul <- table(obs = Xy[test_idx,"y"], pred = yhat.test.mul)
error_rate.test.mul <- 1 - (sum(diag(confusion.test.mul)) / length(test_idx))

error_rate.test.mul
```

```
## [1] 0.005
```

```r
confusion.test.mul
```

```
##    pred
## obs    0    1    2
##   0 1170    6    1
##   1    9 1210    0
##   2    1    1 1202
```

Deployment

The final step is to fit the Random Forest model to the entire dataset. It would then be ready for deployment.

```r
knitr::opts_chunk$set(cache = TRUE)
#### WARNING: This is computationally heavy and will take along time

# Fitting final model
rf.mul.final <- randomForest(y~.,Xy)
```

Side Question:

If we wanted to do multiclass classification using the binary classifcation models estimated in the second section, we would simply have to use each of the three binary models on the test data or new data that needs to be predicted, i.e. shoes vs. shirt, shirt vs. pants and pants vs. shoes. This would result in three predictions for each obeservation. Each prediction of a class is then counted as a vote on that class, and the class with the most vote becomes the final prediction. As the methods all produce posterior probabilities one could also average the probabilities from the three models and then pick the class with the highest probabilities as the prediction.