Fashion MNIST

Group Member 1 Name: _Miao Qi_ Group Member 1 SID: ___3035423408___

Group Member 2 Name: _*Kailin Wang* Group Member 2 SID: ___3035423603___

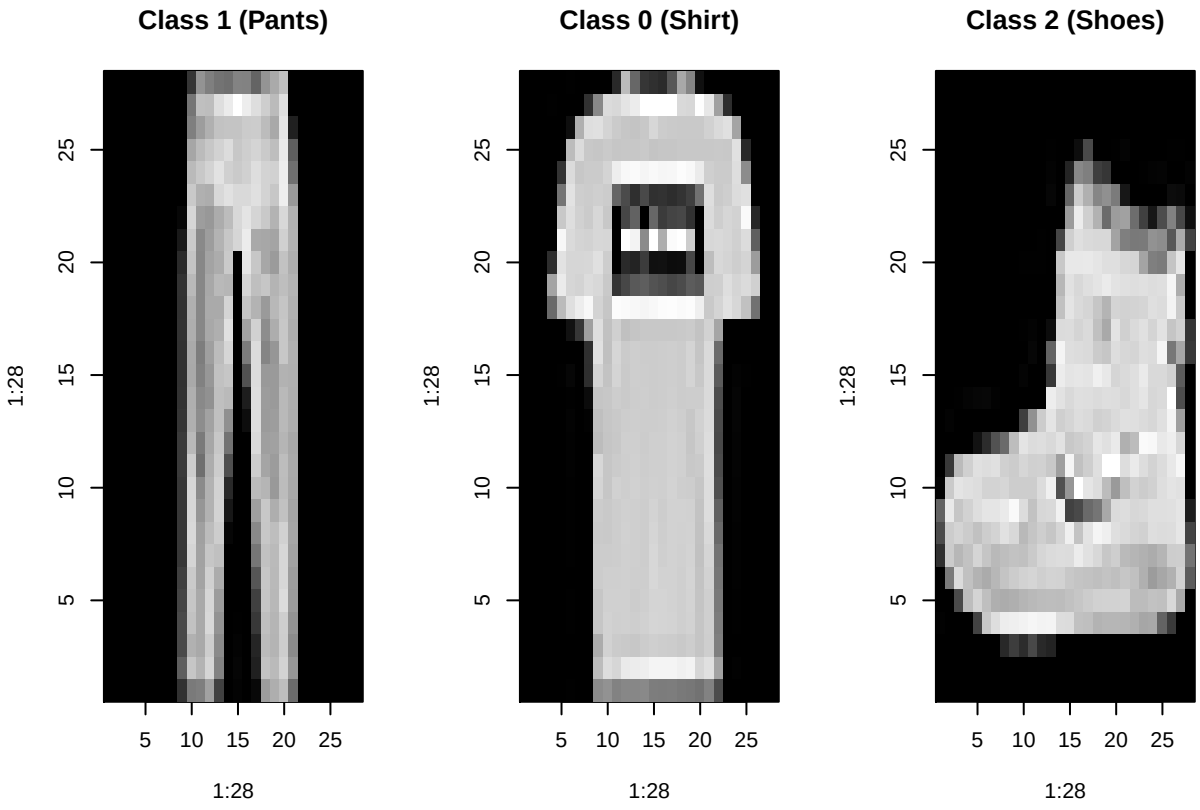
The dataset contains $n = 18,000$ different 28×28 grayscale images of clothing, each with a label of either *shoes*, *shirt*, or *pants* (6000 of each). If we stack the features into a single vector, we can transform each of these observations into a single $28 * 28 = 784$ dimensional vector. The data 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.

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

To begin with,let's take a look at a few of the images:

```
X2 <- matrix(as.numeric(X[1,]), ncol=28, nrow=28, byrow = TRUE)
X2 <- apply(X2, 2, rev)


X0 <- matrix(as.numeric(X[2,]), ncol=28, nrow=28, byrow = TRUE)
X0 <- apply(X0, 2, rev)


X1 <- matrix(as.numeric(X[8,]), ncol=28, nrow=28, byrow = TRUE)
X1 <- apply(X1, 2, rev)
par(mfrow = c(1,3))
image(1:28, 1:28, t(X1), col=gray((0:255)/255), main='Class 1 (Pants)')
image(1:28, 1:28, t(X0), col=gray((0:255)/255), main='Class 0 (Shirt)')
image(1:28, 1:28, t(X2), col=gray((0:255)/255), main='Class 2 (Shoes)')
```


Class 1 (Pants)
Class 0 (Shirt)
Class 2 (Shoes)
1:28
5
10
15
20
25

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?

Libraries

```
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loading required package: foreach
```

```
## Loaded glmnet 2.0-18
```

```
library(tree)
library(ISLR)
library(randomForest)
```

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

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

```
library(class)
library(MASS)
```

Data split

Firstly, we use stratified sampling to the number of each class is the same among the training set and the test set, which will not change the distribution of the original data.

```
dataset <- cbind(y, X)
dataset <- dataset[order(dataset$y), ]
set.seed(110)
index <- sample(1:6000, 5400)
indexs <- c(index, 6000+index, 12000+index)
test <- dataset[-indexs, ]

train <- dataset[indexs, ]
train <- as.data.frame(train)
```

PCA

We use principle component analysis to reduce the dimension of data by using some linear combination of all the features. Notice that when we perform the `prcomp` function, we only center the data because the color parameters are all on the same scale. And when we use the PC's to recover the picture, we only need to add the mean of the feature to the matrix. The deduction is below:

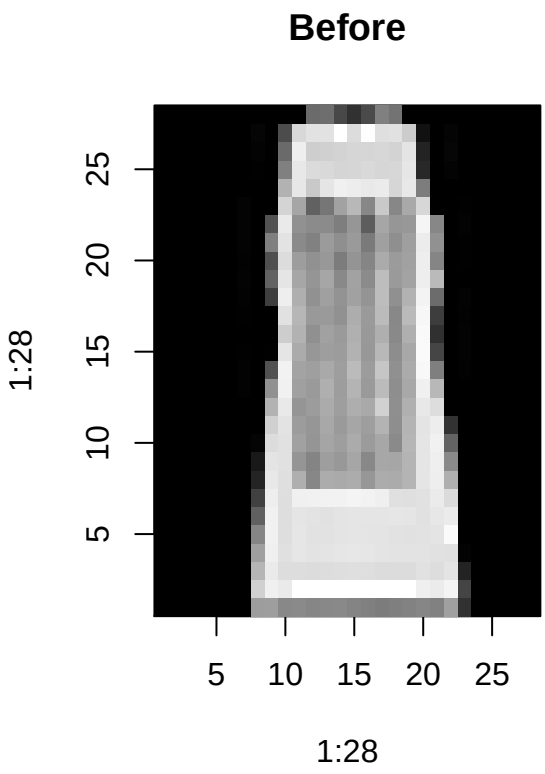
$$Z = (X - \bar{X})VVV^T = IX = ZV^T + \bar{X}$$

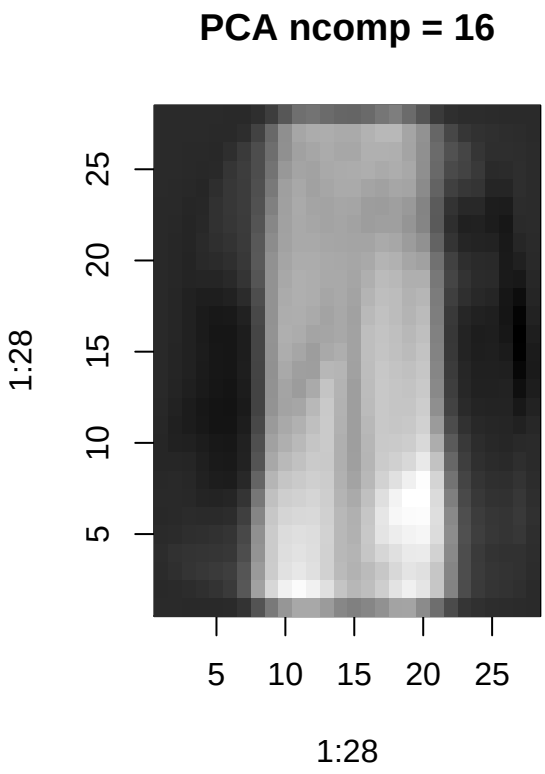
We choose the first 16 PC's whose variance amounts up to 80 percent of the variance of the original data.

```
#center
Xmean <- apply(train[,-1], 2, mean)
pca.dr <- prcomp(train[,-1], center = T, scale = F, retx = T)

#variance selection: 80%
VarExplain <- (pca.dr$sdev)^2 / sum((pca.dr$sdev)^2)
VarAccu <- 0
for(i in 1:length(VarExplain))
{
   VarAccu <- VarAccu + VarExplain[i]
   if(VarAccu > 0.8){
     NVar <- i
     break;
   }
}
NVar
```

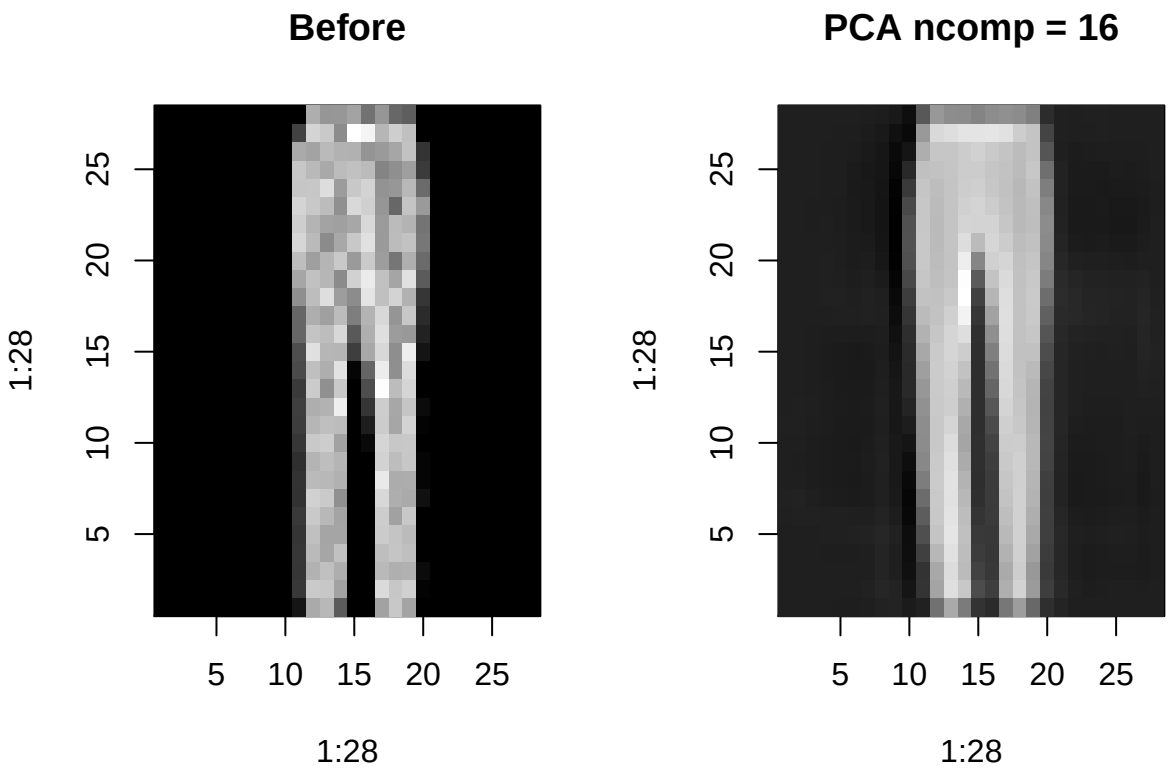
```
## [1] 16
```

```
#visualization
Image <- function(ncomp, index)
{
  loading <- pca.dr$rotation[,1:ncomp]
  score <- pca.dr$x[,1:ncomp]
  X1 <- score %*% t(loading)
  Xori <- t(apply(X1, 1, function(x) x+Xmean))

  par(mfrow = c(1,2))
  X1 <- matrix(as.numeric(train[index, -1]), ncol=28, nrow=28, byrow = TRUE)
  X1 <- apply(X1, 2, rev)
  image(1:28, 1:28, t(X1), col=gray((0:255)/255), main="Before")

  X1 <- matrix(as.numeric(Xori[index,]), ncol=28, nrow=28, byrow = TRUE)
  X1 <- apply(X1, 2, rev)
  image(1:28, 1:28, t(X1), col=gray((0:255)/255),
        main=paste0("PCA ncomp = ",ncomp))
}

#Test the visualization when ncomp = NVar (80% variance)
par(mfrow = c(1,3))
Image(NVar, 1) #shirt
```

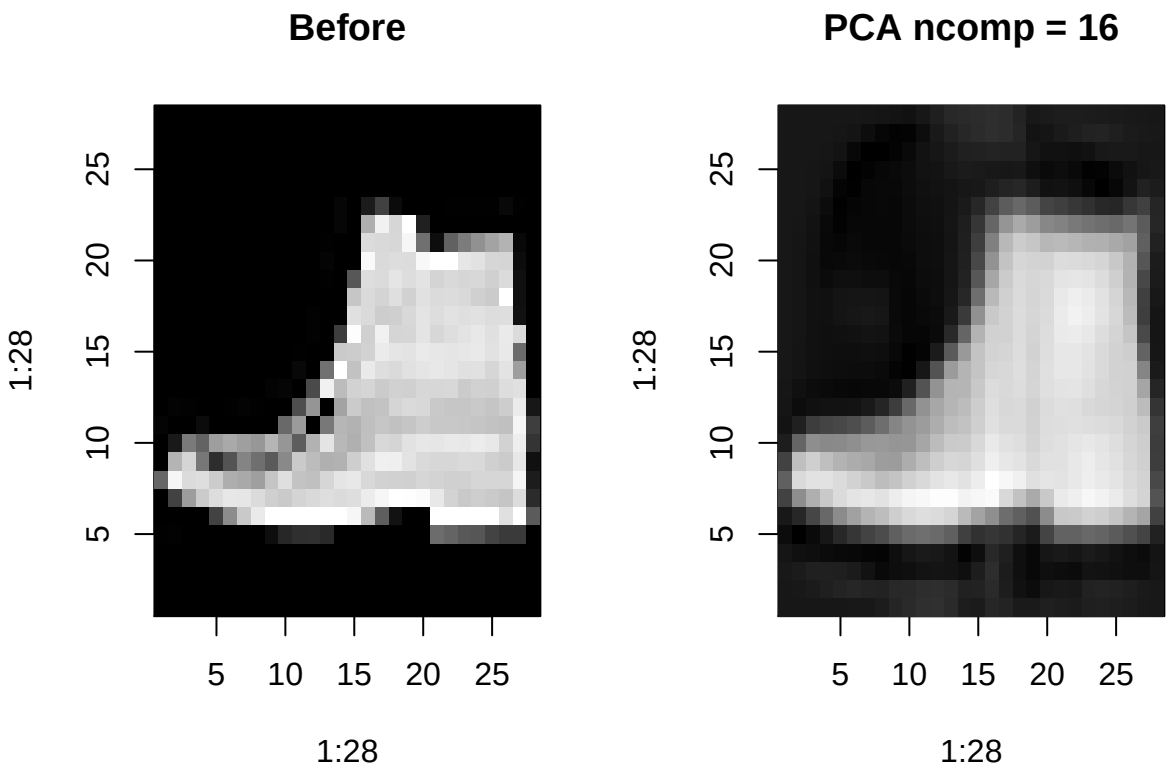





```
Image(NVar, 8000) #pants
```




```
Image(NVar, 13003) #shoes
```



From the images above, dimension reduction with 80% variance of the original can still preserve a lot of features of the images before.
We store the 16 PC's into `Z` together with the class they represent. The next several steps will be performed based on `Z`.

```
#the Dataset for the next parts
Z <- as.data.frame(pca.dr$x[,1:NVar])
Z$y <- train$y
```

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.

Criterion

In this dataset, our task is to perform classification on 3 kinds of pictures, among which each kind has 6000 samples. The sample is balanced. To identify the staff in the picture is not a complex problem, because we can simply focus on the accuracy of the classifier without giving the errors some "weights". So in this project, our criterion to measure the performance of our classifier will be the *misclassification rate*. Furthermore, in the conclusion part we will also discuss the performance of the chosen classifier from perspectives of its *time complexity* and its *interpretability*.

Logistic

- Introduction: The logistic model regress the logit $log(\frac{p(X)}{1-p(X)})$ on the predictors. It is a generalized linear model and its coefficients are estimated with Maximum Likelihood method. The intuition to use this method is that it is suitable for binary classification.

- Validation: We use cross validation to find the unbiased estimation of E_{out}.

- Detail: When fitting model `logit12` and `logit02`, R gives warning:

    1. The function did not converge. We raise the maximum of iteration to 100 and the problem is solved.

    2. The result is only numerically 0 and 1, which shows signs of overfitting. *Our speculation is that the decision boundary of those pairs are very clear, and is linear.* So the misclassification rate for `logit12` and `logit02` is very small, especially `logit12`.

```
cv.logistic <- function(nfold, X, y)
{
  #check y
  y <- factor(y, labels = c(0,1))

  #datasplit
  n <- nrow(X)
  x <- n %% nfold
  k <- (n - x) / nfold #size per fold
  id <- c(rep(1:nfold, k), rep(nfold, x))

  set.seed(10)
  id <- sample(id, n)
```

```
  Misrate <- array()
  for(i in 1:nfold)
  {
    tempdat <- as.data.frame(X[id != i,])
    tempdat$y <- y[id != i]
    log.fit <- glm(y ~ ., data = tempdat,
                   family = "binomial",
                   control=list(maxit=100))
    log.pred <- predict(log.fit, X[id == i, ], type = "response")
    class <- ifelse(log.pred > 0.5, 1, 0)

    Misrate[i] <- mean(class != y[id == i])
  }

  mean(Misrate)
}

logit01 <- cv.logistic(10,
                   as.data.frame(Z[train$y == 0 | train$y == 1, -(NVar+1)]),
                   train$y[which(train$y == 0 | train$y == 1)])
logit12 <- cv.logistic(10,
                   as.data.frame(Z[train$y == 1 | train$y == 2, -(NVar+1)]),
                   train$y[which(train$y == 1 | train$y == 2)])
logit02 <- cv.logistic(10,
                   as.data.frame(Z[train$y == 0 | train$y == 2, -(NVar+1)]),
                   train$y[which(train$y == 0 | train$y == 2)])

logit <- c(logit01, logit12, logit02)
names(logit) <- c("0 vs 1", "1 vs 2", "0 vs 2")
logit
```

```
##       0 vs 1       1 vs 2       0 vs 2
## 0.0195370370 0.0003703704 0.0010185185
```

Linear Discriminant Analysis

- Introduction: Linear discriminant analysis is a less direct method based on Bayes theorem that model the distribution of each of the predictors. From the result above, we can find that some classes are just well separated, and in this case, the coeffeicients in the logistic function may be unstable. We plot the distribution of all the PC's, and find that most of the PCs' distibutions are symmetry. In LDA, the discriminant function is a linear function with regard to x which assumes the mean of each class differs while their covariate matrix are the same.

- Validation: We use cross validation to find the estimation of E_{out}.

```
cv.lda <- function(nfold, X, y) {
    # datasplit
    n <- nrow(X)
    x <- n%%nfold
    k <- (n - x)/nfold  #size per fold
    id <- c(rep(1:nfold, k), rep(nfold, x))

    set.seed(10)
    id <- sample(id, n)

    Rrate <- array()
    Misrate <- array()
    for (i in 1:nfold) {
        fit <- lda(y[id != i] ~ ., data = X[id != i, ])
        pred <- predict(fit, X[id == i, ])$class

        Rrate[i] <- mean(pred == y[id == i])
        Misrate[i] = 1 - Rrate[i]
    }

    mean(Misrate)
}

lda01 <- cv.lda(10, as.data.frame(Z[train$y == 0 | train$y == 1, ]), train$y[which(train$y ==
    0 | train$y == 1)])
lda12 <- cv.lda(10, as.data.frame(Z[train$y == 1 | train$y == 2, ]), train$y[which(train$y ==
    1 | train$y == 2)])
lda02 <- cv.lda(10, as.data.frame(Z[train$y == 0 | train$y == 2, ]), train$y[which(train$y ==
    0 | train$y == 2)])
lda <- c(lda01, lda12, lda02)
names(lda) <- c("0 vs 1", "1 vs 2", "0 vs 2")
lda
```

```
      0 vs 1       1 vs 2       0 vs 2
0.0261111111 0.0005555556 0.0020370370
```

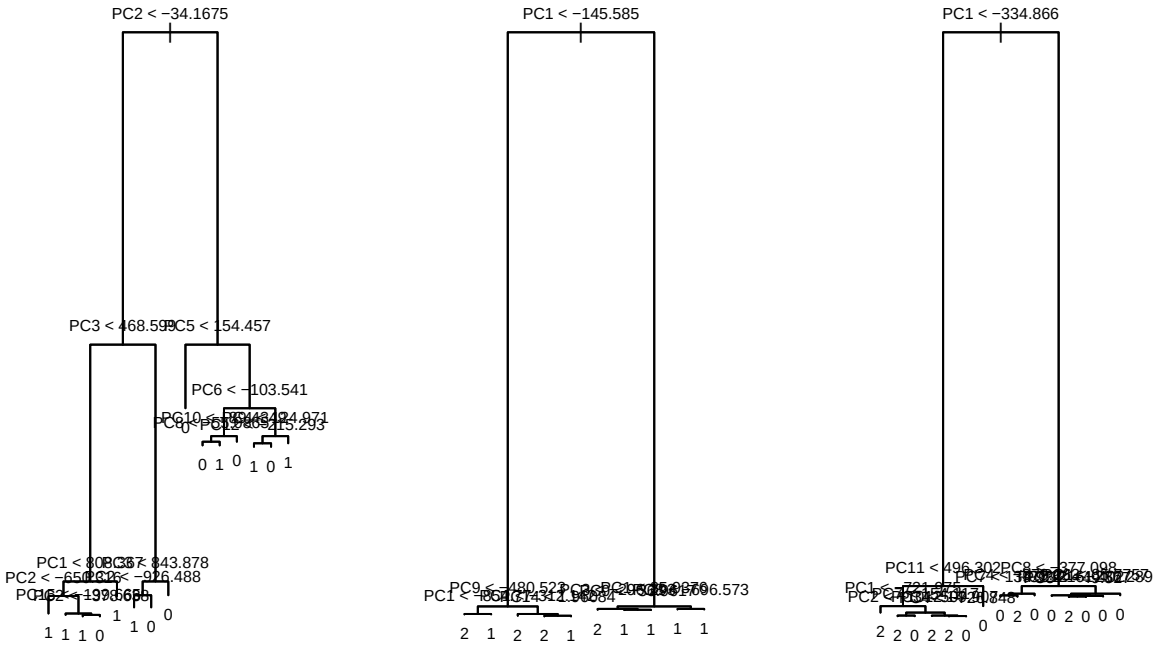
Classification Tree

From the logistic model, the *importance of variable* can be evaluated through the coefficients. And we find that some coefficients of the PC's are so small, however logistic model does not conduct variable selection.

- Introduction: Classification Tree is one kind of tree-based models. It is easy to be interpreted and visualized. Here we use it not only to find the accuracy of the method, but also to see the importance of the variables/select the variables through the process of pruning.
- Validation: 10-fold cross validation during pruning.
- Detail: In order to comparing those models on the same level, in fitting the tree, we set the parameter `mindev` to be 0.0001. This parameter means that the within-node deviance should be at this magnitude. The motivation to do that is because the default `tree()` will only use some of the PC's to give the result, however, it may seem untenable to compare a tree with only some PC's with other models. And to prevent overfitting, we use 10-fold cross validation to prune the tree. Finally, the number PC's used in each tree may still smaller than 16, but the pruning process is reasonable to avoid overfitting. Typically, we restrict the size(i.e. number of terminal nodes) of the tree up to 15.

```
tree01 <- tree(factor(y) ~ ., as.data.frame(Z[train$y == 0 | train$y == 1, ]),
    split = "deviance", control = tree.control(nrow(Z[train$y == 0 | train$y ==
        1, ]), mindev = 1e-04))
tree12 <- tree(factor(y) ~ ., as.data.frame(Z[train$y == 1 | train$y == 2, ]),
    control = tree.control(nrow(Z[train$y == 1 | train$y == 2, ]), mindev = 1e-04))
tree02 <- tree(factor(y) ~ ., as.data.frame(Z[train$y == 0 | train$y == 2, ]),
    control = tree.control(nrow(Z[train$y == 0 | train$y == 2, ]), mindev = 1e-04))
# prune and plot the tree
CVtree <- function(tree) {
    tree.prune <- cv.tree(tree, FUN = prune.misclass, K = 10)
    size <- tree.prune$size
    ind <- which(size <= 15)
    size <- size[ind]
    dev <- tree.prune$dev[ind]
    BEST <- size[which.min(dev)]
    tree.newfit <- prune.misclass(tree, best = BEST)
    summary1 <- summary(tree.newfit)
    {
        plot(tree.newfit)
        text(tree.newfit, pretty = 0, cex = 0.7)
        print("PC used")
        print(as.character(summary1$used))
    }
    return(summary1$misclass[1]/summary1$misclass[2])
}

# mis
par(mfrow = c(1, 3))
classtree <- c(CVtree(tree01), CVtree(tree12), CVtree(tree02))
```

```
[1] "PC used"
 [1] "PC2"  "PC3"  "PC1"  "PC15" "PC5"  "PC6"  "PC10" "PC8"  "PC4"  "PC12"

[1] "PC used"
[1] "PC1"  "PC9"  "PC14" "PC3"  "PC8"
```




```
[1] "PC used"
[1] "PC1"  "PC11" "PC7"  "PC2"  "PC3"  "PC4"  "PC8"  "PC12" "PC5"
```

```
names(classtree) <- c("0 vs 1", "1 vs 2", "0 vs 2")
classtree
```

```
      0 vs 1       1 vs 2       0 vs 2
0.0252777778 0.0005555556 0.0009259259
```

- Evaluation: After cross validation, the misclassification rate between class 0(shirt) and class 1(pants) is still large compared with another 2 trees. And this tree's size is our upper limit, which means that decision tree here may perform not well.

Model Selection

```
Binary <- as.data.frame(rbind(logit, lda, classtree))
row.names(Binary) <- c("Logistics", "LDA", "ClassificationTree")
Binary$mean <- apply(Binary, 1, mean)
Binary
```

```
                      0 vs 1       1 vs 2       0 vs 2        mean
Logistics          0.01953704 0.0003703704 0.0010185185 0.006975309
LDA                0.02611111 0.0005555556 0.0020370370 0.009567901
ClassificationTree 0.02527778 0.0005555556 0.0009259259 0.008919753
```

The result show that these models have a similar performance when classifying between class 1(pants) and class 2(shoes) with an extremely low misclassification rate. And logistics outperforms the other 2 between class 0(shirt) and class 1(pants). Classification tree has the lowest misclassification rate when discriminating between class 0(shirt) and class 2(shoes). In all, the mean misclassification rate of logistics is lower. So in this part, we will choose logistic regression as the classification model. In addition, the result tells that the Gaussian assumption does not hold true.

Model Evaluation

Based on the test set, model evaluation on the final selected model is conducted in this part. First, use the same loading vector in the training set to perform dimension reduction.

```
Ztest <- as.data.frame(scale(test[, -1], center = T, scale = F) %*% pca.dr$rotation[,
    1:NVar])
Ztest$y <- test$y
```

```
PredLogistics <- function(testx, testy, train) {
    testy <- factor(testy, labels = c(0, 1))
    train$y <- factor(train$y, labels = c(0, 1))

    fit <- glm(y ~ ., train, family = "binomial", control = list(maxit = 100))
    ypred <- predict(fit, testx, type = "response")
    ypred <- ifelse(ypred > 0.5, 1, 0)
    Eout <- mean(ypred != testy)

    return(list(model = fit, Eout = Eout))
}

Eout01 <- PredLogistics(testx = Ztest[test$y == 0 | test$y == 1, 1:16], testy = Ztest$y[test$y ==
    0 | test$y == 1], train = Z[which(train$y == 0 | train$y == 1), ])
Eout12 <- PredLogistics(testx = Ztest[test$y == 1 | test$y == 2, 1:16], testy = Ztest$y[test$y ==
    1 | test$y == 2], train = Z[which(train$y == 1 | train$y == 2), ])
```

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

```
Eout02 <- PredLogistics(testx = Ztest[test$y == 0 | test$y == 2, 1:16], testy = Ztest$y[test$y ==
    0 | test$y == 2], train = Z[which(train$y == 0 | train$y == 2), ])
```

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

```
MISS <- c(Eout01$Eout, Eout12$Eout, Eout02$Eout)
MISS <- c(MISS, mean(MISS))
names(MISS) <- c("0 vs 1", "1 vs 2", "0 vs 2", "Mean")
MISS
```

```
      0 vs 1       1 vs 2       0 vs 2         Mean
0.0191666667 0.0008333333 0.0008333333 0.0069444444
```

- Evaluation: The misclassification rate of our final model on test set is 0.0069 on average. Moreover, logistics model is quite excellent in the discrimination between class 2(shoes) and the others.

```
summary(Eout01$model)
```

```
##
## Call:
## glm(formula = y ~ ., family = "binomial", data = train, control = list(maxit = 100))
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.2039  -0.0171   0.0000   0.0164   3.8243
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.0448596  0.4401354  -2.374  0.01760 *
## PC1          0.0022228  0.0005196   4.278 1.88e-05 ***
## PC2         -0.0065907  0.0002985 -22.077  < 2e-16 ***
## PC3         -0.0077138  0.0004934 -15.633  < 2e-16 ***
## PC4          0.0007409  0.0006841   1.083  0.27879
## PC5          0.0006487  0.0004298   1.509  0.13124
## PC6          0.0056824  0.0004536  12.526  < 2e-16 ***
## PC7          0.0050431  0.0008802   5.729 1.01e-08 ***
## PC8          0.0006448  0.0006567   0.982  0.32610
## PC9         -0.0056896  0.0005046 -11.275  < 2e-16 ***
## PC10        -0.0036600  0.0006000  -6.099 1.06e-09 ***
## PC11         0.0013997  0.0006274   2.231  0.02568 *
## PC12        -0.0015958  0.0006316  -2.526  0.01152 *
## PC13        -0.0040008  0.0006321  -6.330 2.46e-10 ***
## PC14        -0.0020010  0.0008715  -2.296  0.02168 *
## PC15        -0.0044836  0.0007394  -6.064 1.33e-09 ***
## PC16        -0.0015374  0.0005095  -3.018  0.00255 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 14972.0  on 10799  degrees of freedom
## Residual deviance:  1118.7  on 10783  degrees of freedom
## AIC: 1152.7
##
## Number of Fisher Scoring iterations: 9
```

- Interpretation: Take model logit01. From the coefficients, we can find the importance of the PC's. PC1, 2, 3, 6, 7, 10-16 are significant. The negative coefficient means that when the value of those PC's increase, the probability of the class being "1" will decrease. The larger the absolute of the coefficient, the more important the PC is.

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?)

For multi-classification problems, it is not sure whether the decision boundary of the data is linear or non-linear. So we use both parametric methods and non-parametric methods to fit the model.

Discriminant Analaysis: LDA and QDA

- Introduction: The difference between QDA and LDA is that QDA assumes that the mean and the covariance structure in each class are both different, thus it produced a quadratic decision boundary. LDA can suffer from high bias, and from the model in binary cases we find that it may be poor in multiclass problem in this dataset. To fit a QDA model is necessary. Since the number of the predictors is 16 and the number of classes is 3, to estimate the covariance matrix for each class and each predictor is not that trouble. QDA is more flexible than LDA and it is quite useful in large datasets.

- Validation: We still use 10-fold cross validation here.

```
cv.da <- function(nfold, X, y, FUN) {
    # datasplit
    n <- nrow(X)
    x <- n%%nfold
    k <- (n - x)/nfold  #size per fold
    id <- c(rep(1:nfold, k), rep(nfold, x))

    set.seed(10)
    id <- sample(id, n)

    Misrate <- array()
    for (i in 1:nfold) {
        if (FUN == "lda")
            fit <- lda(factor(y[id != i]) ~ ., data = X[id != i, ]) else fit <- qda(factor(y[id != i]) 
        pred <- predict(fit, X[id == i, ])$class

        Misrate[i] = mean(pred != y[id == i])
    }

    mean(Misrate)
}

lda.multi <- cv.da(10, Z[, -1], Z$y, "lda")

qda.multi <- cv.da(10, Z[, -1], Z$y, "qda")

da.mis <- c(lda.multi, qda.multi)
names(da.mis) <- c("lda", "qda")
da.mis
```

```
       lda        qda
0.10444444 0.01592593
```

The missclassification rate shows that qda fits better.

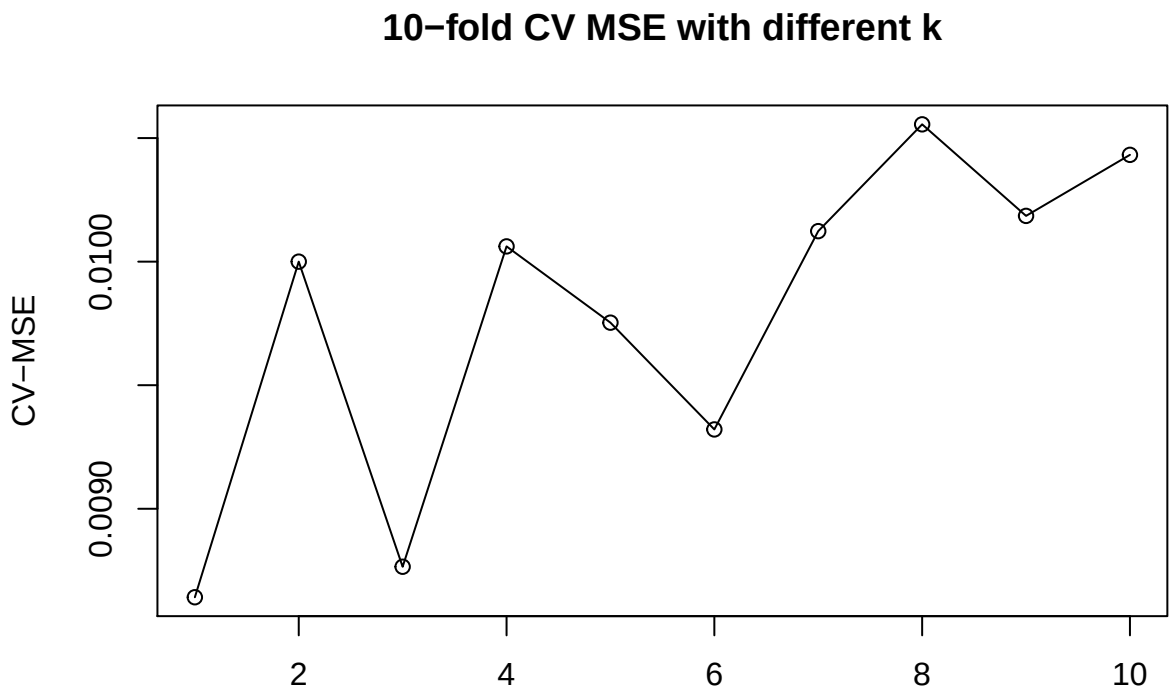
KNN

- Introduction: KNN is short for k-nearest neighbor, which is a non-parametric method and will perform better when the decision boundary of the classes are non-linear. And when dataset is much bigger, the method could become rather slow. Also, because of its non-liearity, KNN is more flexible than logistics, LDA, and QDA. However, KNN does not give us the coefficients, and we cannot tell which PC is more important. The basic problems with KNN is the chosen of k and the distance measure. Here we will use Euclidean distance.
- Validation: And we will use cross validation to choose k. The plot shows that when $k = 1$, the misclassification rate is the lowest.

```
cv.knn <- function(nfold, X, y, K) {
    # check y
    y <- factor(y)
    # datasplit
    n <- nrow(X)
    x <- n%%nfold
    k <- (n - x)/nfold  #size per fold
    id <- c(rep(1:nfold, k), rep(nfold, x))

    set.seed(10)
    id <- sample(id, n)

    Misrate <- array()
    temp <- array()
    for (k in 1:K) {
        for (i in 1:nfold) {
            fit <- knn(train = X[id != i, ], test = X[id == i, ], y[id != i],
                k)
            temp[i] <- mean(fit != y[id == i])
        }
        Misrate[k] <- mean(temp)
    }

    names(Misrate) <- paste("k =", 1:K)
    Misrate
}

knn.mis <- cv.knn(10, Z[, -1], Z$y, 10)
knn.mis
```

```
      k = 1       k = 2       k = 3       k = 4       k = 5       k = 6
0.008641975 0.010000000 0.008765432 0.010061728 0.009753086 0.009320988
      k = 7       k = 8       k = 9      k = 10
0.010123457 0.010555556 0.010185185 0.010432099
```

```
# plot knn
plot(knn.mis, main = "10-fold CV MSE with different k", xlab = "", ylab = "CV-MSE")
lines(knn.mis)
```




```
knn1.mis <- knn.mis[1]
```

Decision Tree

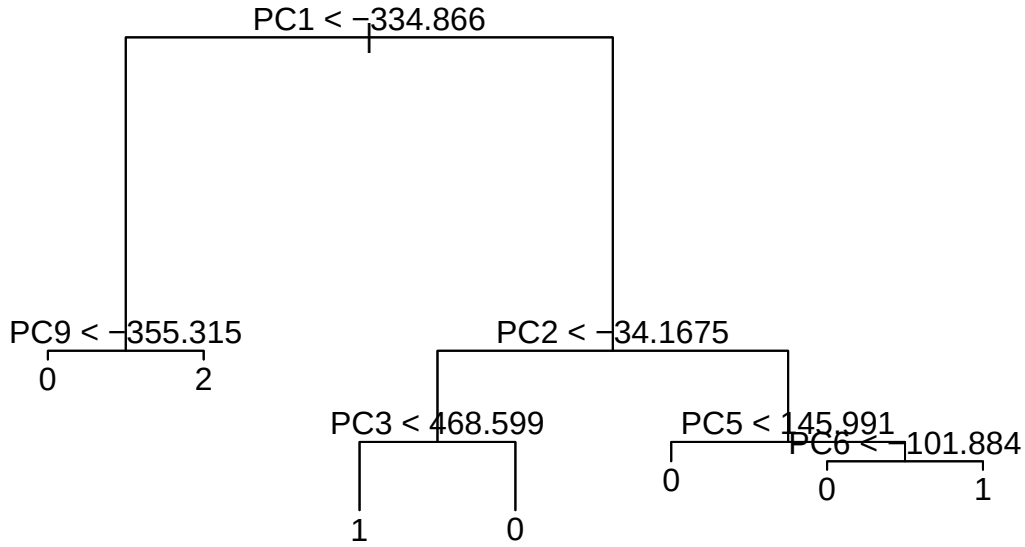
- Introduction: Classification tree can also be used in the multiclassification problem. One of its most obvious advantage is that tree-based methods are very easy to be interpreted. To prevent overfitting, we also prune it using the misclassification rate. The control parameter is the size of the tree (i.e the number of terminal nodes of the tree), and it equals 7, which means that no nodes/branches was pruned.

- Validation: 10-fold cross validation is used during the pruning process.

```
tree.fit <- tree(factor(y) ~ ., Z)
summary(tree.fit)
```

```
Classification tree:
tree(formula = factor(y) ~ ., data = Z)
Variables actually used in tree construction:
[1] "PC1" "PC9" "PC2" "PC3" "PC5" "PC6"
Number of terminal nodes:  7
Residual mean deviance:  0.2551 = 4132 / 16190
Misclassification error rate: 0.03531 = 572 / 16200
```

```
{
    plot(tree.fit)
    text(tree.fit, pretty = 0)
}
```



```r
# prune the tree
tree.prune <- cv.tree(tree.fit, FUN = prune.misclass, K = 10)
BEST <- tree.prune$size[which.min(tree.prune$dev)]
tree.newfit <- prune.misclass(tree.fit, best = BEST)
BEST
```

```
[1] 7
```

```r
# mis
dectree.mis <- tree.prune$dev[tree.prune$size == BEST]/nrow(Z)
names(dectree.mis) <- "ClassTree"
dectree.mis
```

```
 ClassTree
0.03777778
```

Random Forest

- Introduction: Random forest will boost the accuracy of the classification tree through combining a lot of trees together at the cost of some interpretability. This method introduces more randomness in 2 aspects:

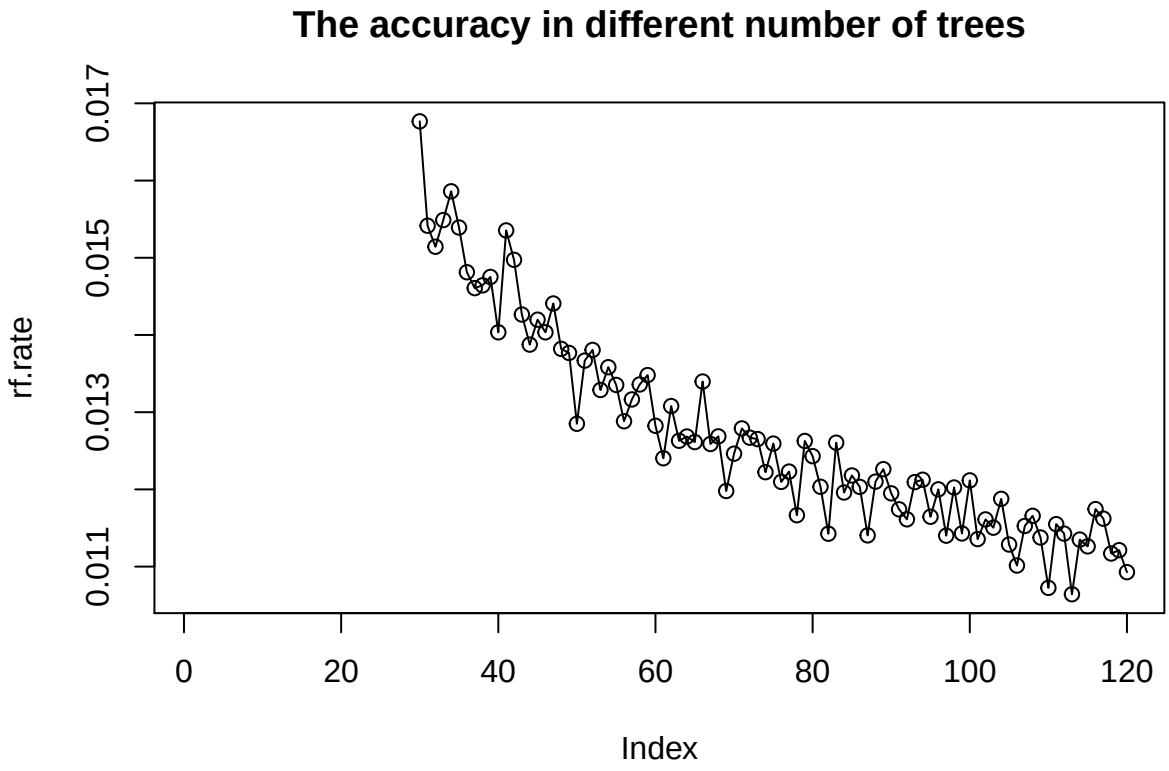
1. the randomness in sampling: bootstrap
2. the randomness in feature selection. In our classification tree, the number of feature chosen each time is $\sqrt{p}$, and p is the number of all the features. In our model, $p = 16$ (number of the components) so in each tree, 4 features will be used.

- Validation: Random forest uses bootstrap sampling and the expectation of out-of-bag error as the unbiased estimation of E_{out}.

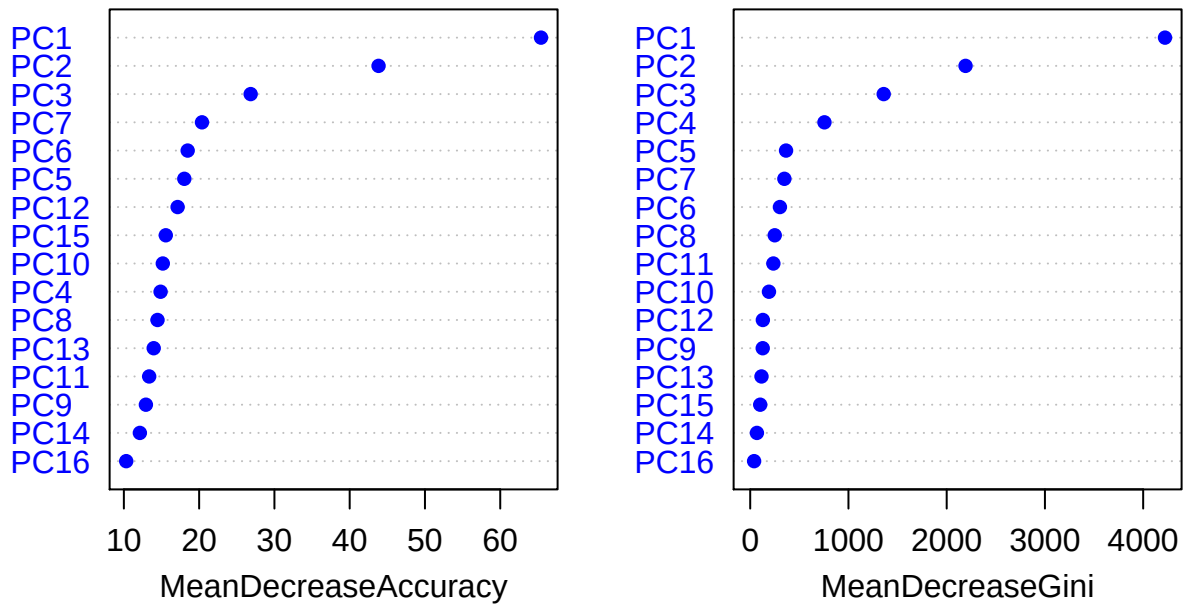
- Detail: In order to minimize the MSE, we adjust the model by making the number of tree equal 100, when the misclassification rate becomes stable at its lowest level. According to literature, the out-of-bag error and the test error will change similarly when the number of trees changed. However, the selection costs much in terms of run time. From the plot we find that the accuracy of the model increased when the number of tree is less than 100. When it becomes larger, the accuracy becomes stable. So 100 is enough to produce the best result.

literature: LIU Min, LANG Rongling, CAO Yongbin. Number of trees in random forest. Computer Engineering and Applications, 2015, 51(5): 126-131

```
set.seed(1216)
rf.rate <- array()
for (i in 30:120) {
    rf.fit <- randomForest(factor(y) ~ ., data = Z, importance = T, ntree = i)
    rf.rate[i] <- apply(rf.fit$err.rate, 2, mean)[1]
}
{
    plot(rf.rate, main = "The accuracy in different number of trees")
    lines(rf.rate)
}
```




```r
rf.fit <- randomForest(factor(y) ~ ., data = Z, importance = T, ntree = 100)

# result importance of PC's
varImpPlot(rf.fit, main = "Importance of PC's", col = "blue", pch = 16)
```




```
# misclassification rate
rf.mis <- apply(rf.fit$err.rate, 2, mean)[1]
names(rf.mis) <- "RandomForest"
rf.mis
```

```
RandomForest
  0.01177244
```

Model Selection

```
MIS <- c(da.mis, knn1.mis, dectree.mis, rf.mis)
MIS
```

```
        lda          qda        k = 1    ClassTree RandomForest
0.104444444  0.015925926  0.008641975  0.037777778  0.011772437
```

Model Evaluation

Our final model is KNN. KNN assumes that points that are closer to each other in the eigenspace is more similar, and every dimension in the space is of the same importance. Here we use Euclidean distance to measure the similarity of the data. According to the cross validation, the k we choose is 1, bringing small bias however very likely to cause overfitting. Luckily our data has little noise, so the result in this model is not so far away from the estimated E_{out}.

Accuracy

```
knn.pred <- knn(train = Z[, 1:NVar], test = Ztest[, 1:NVar], Z$y, 1)
Eout.knn <- mean(Ztest$y != knn.pred)
Eout.knn
```

```
[1] 0.01111111
```

The misclassification rate of KNN is 0.011. Recall that in our training data, it was 0.0087. It behaves well in this case.

Time complexity

Using *K-D tree*, the time complexity of KNN is $O(p\ log(N))$. The features in this dataset after dimension reduction is small (up to 16), so using KNN is simple and efficient here.

Interpretability

KNN is an instance-based learning method, which is very easy to interpret. It uses the vote of the class of his k nearest neighbor–in this case 1–to predict the class of the test data. And our data is the value of a image in its pixel grids, so it will be OK to skip the *scaling*.