final_exam

Stewart Renehan

6/7/2019

1

a)

The marginal distribution of x is a uniform distribution with probability 1/4 in the range {-4, -2}, and a uniform distribution with probability 1/4 in the range {2, 4}, with probability 0 everywhere else.

b)

The bayes rule is

$$f_b(x) = \{1\ if\ \ -4 < x < -2;\ 2\ if\ 2 < x < 4\}$$

The risk is 0 because there is 0 probability of y = 2 occuring when x is between -4 and -2, and 0 probability of y = 1 occuring if x is between 2 and 4, and 0 probability of observing x outside of these two ranges.

c)

If there is at least one observation in each bin, the risk is 0 because when the observed x is -2, the closest possible observation to the y=2 region, the furthest possible training data for the y=1 region is 2 points away, while the closest possible observation for the y = 2 region (which would result in an incorrect prediction) is 4 points away. So, as long as there is at least one trainig data point for y = 1 and at least one training data point for y =2, there is 0 probability of an incorrect prediction.

However, when there are zero training data points in one bin, the risk is nonzero. It is equivalent to the probability of obtaining zero training points in one bin = $(\frac{1}{2})^{n-1}$ times the probability of the test value being in the opposite bin = 1/2.

Thus, the total risk is $(\frac{1}{2})^n$.

d)

By the same argument in part (c), if there are 3 or more training observations in each bin, then the expected risk is zero. Again, but the same argument in part (c), if there are 0 observations in one bin, then the risk is $(\frac{1}{2})^{n+1}$. If there is only two observations in one of the bins, then the risk is zero because by majority vote, 2/3 of the closest observations will be of the correct class. However, if one of the bins has only one observation, then the observations in that bin will get the incorrect prediction because 2 of the 3 closest neighbors are in the other bin. So in this situation, the risk is the probability of one of the bins having exactly one observation. This is equal to 2, the number of bins, times the probability of one bin getting oen observation, which is $n(\frac{1}{2})^n$, where the n factor accounts for the fact that there are n possible timings of the draw for the bin with 1 observation. This is multiplied by the probability of the prediction being for the bin with only one observation which is 1/2. So, the risk for the case where the one of the bins only has one observation is $n(\frac{1}{2})^n$.

The total risk is the sum of the two cases, $(n+1)(\frac{1}{2})^n$

e)

The one nearest neighbor method has lower risk because misclassifications only happen when one bin has 0 training observations, while the 3 nearest neighbor has misclassifications when one bin has 0 or 1 training observations.

2

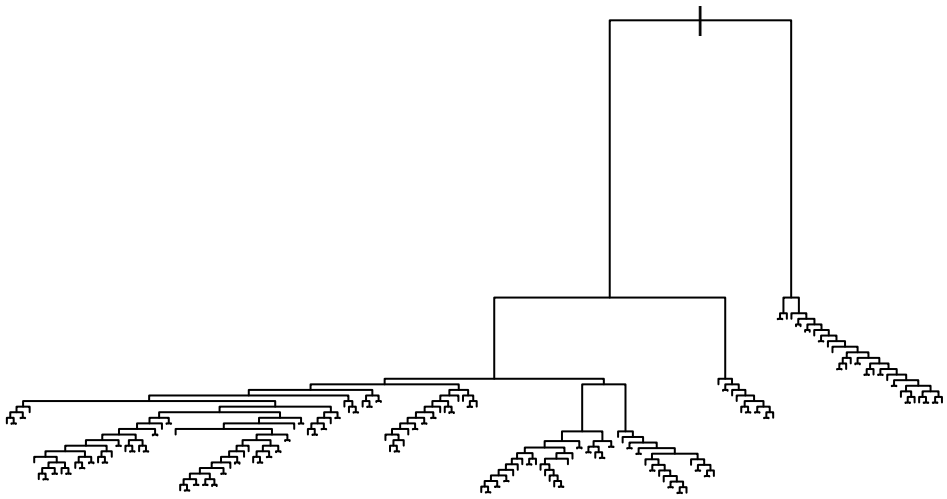
a)

```
spam.data = read.table('/Users/stewart/Downloads/spam.data')
traintest = read.table('/Users/stewart/Downloads/spam.traintest')

test = spam.data[traintest== 1,]
train = spam.data[traintest==0,]
# install.packages('rpart')
require(rpart)
```

```
## Loading required package: rpart
```

```
# help(rpart)
model = rpart(V58~., data = train, method="class", cp=0, minsplit=0)
plot(model)
```



```
print('Cant print the names of the splits, there are too many!')
```

```
## [1] "Cant print the names of the splits, there are too many!"
```

```
# text(model)

train.error = sum((predict(model, train)[,1] == 0) != train$V58) / nrow(train)
test.error = sum((predict(model, test)[,1] == 0) != test$V58) / nrow(test)
print(paste0('Train Error Rate: ', train.error));
```

```
## [1] "Train Error Rate: 0.000978792822185971"
```

```
print(paste0('Test Error Rate: ', test.error));
```

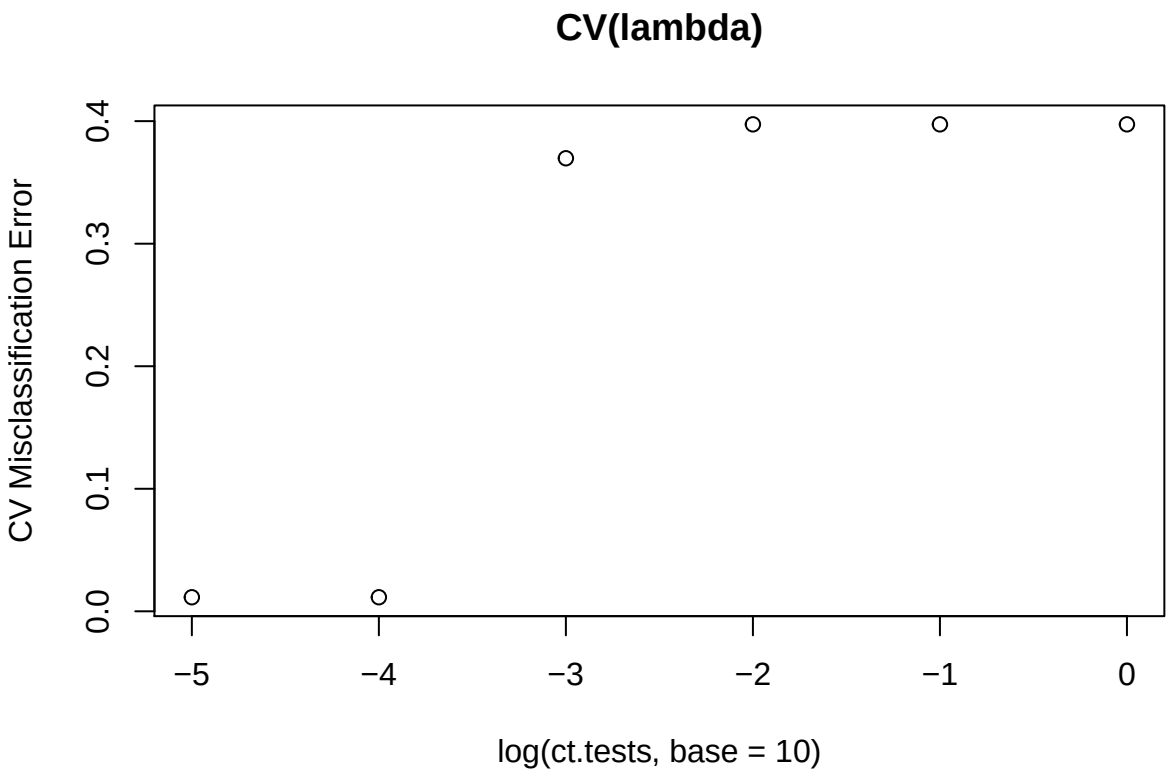
```
## [1] "Test Error Rate: 0.083984375"
```

b)

```
folds <- cut(sample(seq(1,nrow(train))),breaks=10,labels=FALSE)
ct.tests = c(0, 0.00001, 0.0001, 0.001, 0.01, 0.1, 1);
errors = c();
for (lambda in ct.tests) {
  val.errors = c();
  for (fold in 1:10) {
    model =  rpart(V58~., data = train[folds !=fold,], method='class', cp=lambda, minsplit=0);
    val.error = sum((predict(model, train[fold == fold,])[,1] == 0) != train[fold == fold,]$V58) / nrow
    val.errors = c(val.errors, val.error);
  }
  errors = c(errors, mean(val.errors))
  print(mean(val.errors))
}
```

```
## [1] 0.0114845
## [1] 0.0114845
## [1] 0.0114845
## [1] 0.3697227
## [1] 0.3973899
## [1] 0.3973899
## [1] 0.3973899
```

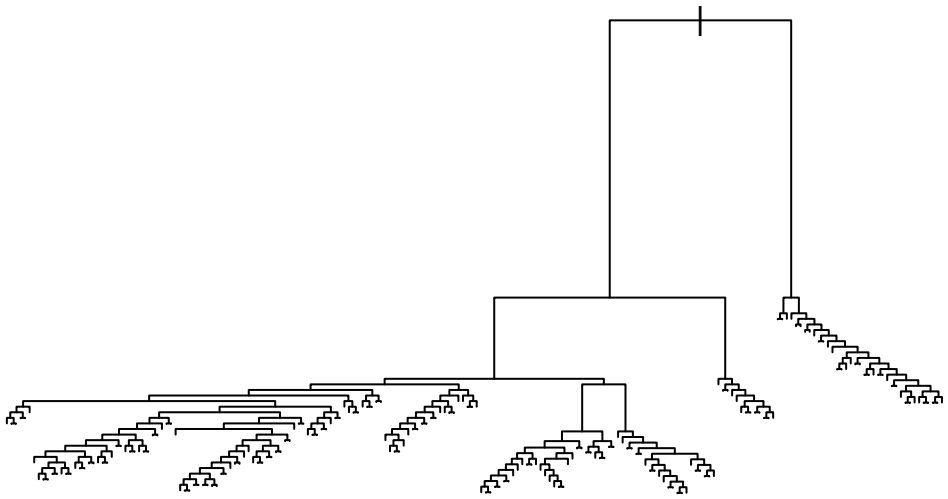
```
plot(log(ct.tests, base=10), errors, ylab='CV Misclassification Error', main='CV(lambda)');
```




```
print(paste0('Optimal lambda', 0.0001))
```

```
## [1] "Optimal lambda1e-04"
```

```
model.optimal = rpart(V58~., data = train, method='class', cp=0.0001, minsplit=0)
plot(model.optimal)
```



```
train.error = sum((predict(model.optimal, train)[,1] == 0) != train$V58) / nrow(train)
test.error = sum((predict(model.optimal, test)[,1] == 0) != test$V58) / nrow(test)
print(paste0('Train Error Rate: ', train.error));
```

```
## [1] "Train Error Rate: 0.000978792822185971"
```

```
print(paste0('Test Error Rate: ', test.error));
```

```
## [1] "Test Error Rate: 0.083984375"
```

3

```
require(glmnet)
```

```
## Loading required package: glmnet
```

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

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

```
## Loaded glmnet 2.0-12
```

```
model = cv.glmnet(as.matrix(train[,1:57]), train$V58, nfolds=10, alpha=0, lambda.min.ratio=0)
print(paste0('Optimal Lambda: ', model$lambda.1se))
```

```
## [1] "Optimal Lambda: 0.319934862675335"
```

```
preds = predict(model, as.matrix(train[,1:57]), s="lambda.1se")

best.c <- 0
best.error <- 'inf'
for (pred in preds) {
  err = (sum((preds > pred) != train$V58));
  if (pred < best.error) {
    best.error = err;
    best.c = pred;
  }
}
print(paste0('Optimal value for C: ', best.c))
```

```
## [1] "Optimal value for C: 0.307935997362605"
```

```
train.error = best.error / nrow(train);
test.error =sum((predict(model, newx=as.matrix(test[,1:57]), s="lambda.1se") > best.c) != test$V58) / n
print(paste0('Resubstitution Error on Training Data ', train.error))
```

```
## [1] "Resubstitution Error on Training Data 0.152691680261011"
```

```
print(paste0('Resubstitution Error on Testing Data ', test.error))
```

```
## [1] "Resubstitution Error on Testing Data 0.16796875"
```

```
ls.model = lm('V58~.', data=train)
preds = predict(ls.model, data=train)
best.c <- 0
best.error <- 'inf'
for (pred in preds) {
  err = (sum((preds > pred) != train$V58));
  if (pred < best.error) {
    best.error = err;
    best.c = pred;
  }
}
print(paste0('Optimal value for C for least squares model: ', best.c))
```

```
## [1] "Optimal value for C for least squares model: 0.257817571968291"
```

```
train.error = best.error / nrow(train);
test.error = sum((predict(ls.model, newdata=test) > best.c) != test$V58) / nrow(test)
print(paste0('Least Squares Resubstitution Error on Training Data ', train.error))
```

```
## [1] "Least Squares Resubstitution Error on Training Data 0.180750407830343"
```

```r
print(paste0('Least Squares Resubstitution Error on Testing Data ', test.error))
```

```
## [1] "Least Squares Resubstitution Error on Testing Data 0.200520833333333"
```

Ridge regression outperforms least squares on the testing data by about 2%.

4

a)

```r
require('ISLR')
```

```
## Loading required package: ISLR
```

```r
data('College')
sample <- sample.int(n = nrow(College), size = floor(.8*nrow(College)), replace = F)
train <- College[sample, ]
test  <- College[-sample, ]

require('leaps')
```

```
## Loading required package: leaps
```

```r
folds <- cut(sample(seq(1,nrow(train))),breaks=10,labels=FALSE)
best.n.features = 0;
best.error = 'inf'
possible_features = c();
for (name in names(train)) {
  if (name != 'Outstate') {
    possible_features = c(possible_features, name);
  }
}

for (n_features in 1:(ncol(train) - 1)) {
  errors = c();
  for (fold in 1:10) {
    optimal_features = summary(
      regsubsets(Outstate~., data=train[folds != fold,], method='forward', nvmax=n_features, intercept =
    )$which[n_features,];
    model = lm(as.formula(paste("Outstate~", paste(possible_features[optimal_features], collapse="+")))
    preds = predict(model, newdata=train[folds == fold,])
    mse = mean((preds - train[folds == fold, 'Outstate'])^2)
    errors= c(errors, mse);
  }
  if (mean(errors) < best.error) {
    best.error = mean(errors);
    best.n.features = n_features;
  }
}
print(paste0('Best n features using forward selection w/ 10 fold cv: ', best.n.features));
```

```
## [1] "Best n features using forward selection w/ 10 fold cv: 16"
```

```
optimal_features = summary(
  regsubsets(Outstate~., data=train, method='forward', nvmax=best.n.features, intercept = F)
)$which[best.n.features,];
print('Optimal features: ')
```

```
## [1] "Optimal features: "
```

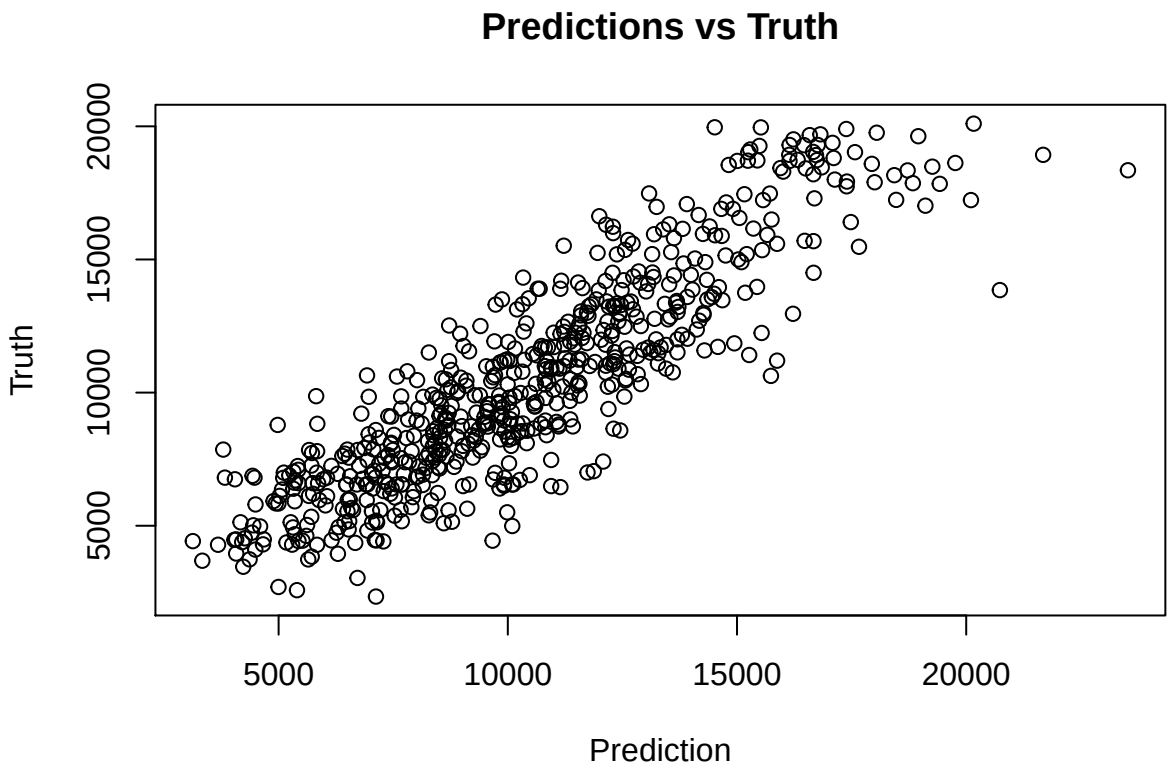
```
print(possible_features[optimal_features])
```

```
##  [1] "Private"     "Apps"        "Accept"      "Enroll"      "Top10perc"
##  [6] "Top25perc"   "F.Undergrad" "Room.Board"  "Books"       "Personal"
## [11] "PhD"         "Terminal"    "S.F.Ratio"   "perc.alumni" "Expend"
## [16] "Grad.Rate"
```

b)

```
model = lm(as.formula(paste("Outstate~", paste(possible_features[optimal_features], collapse="+"))), da

plot(predict(model, train), train$Outstate, xlab = 'Prediction', ylab='Truth', main='Predictions vs Tru
```



c)

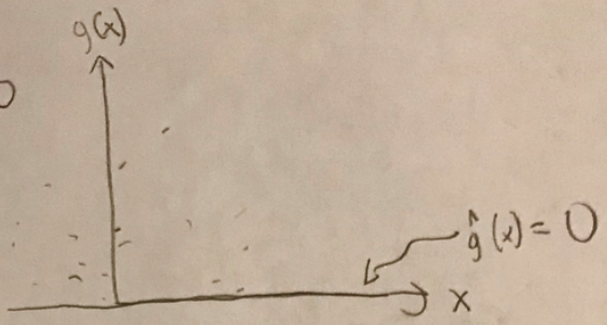
```
rmse = mean((predict(model, test) - test$Outstate)^2) ^.5
print(paste0('On the test data, the RMSE is ', rmse, ' so the predictions are usually within this far o
```

```
## [1] "On the test data, the RMSE is 2138.53802907092 so the predictions are usually within this far o
```

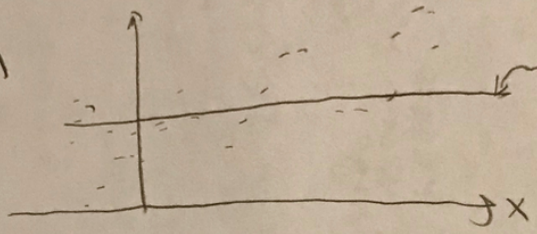
5

Problem 5

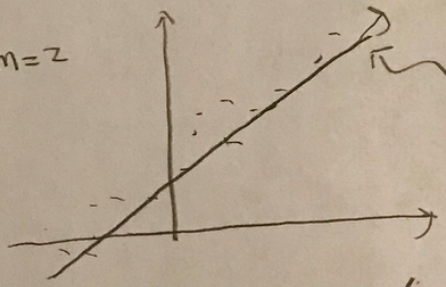
1. $\lambda = \infty, m = 0$



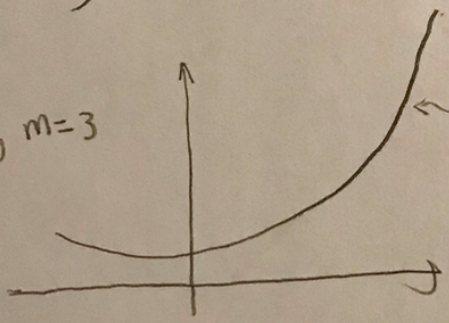
2. $\lambda = \infty, m = 1$



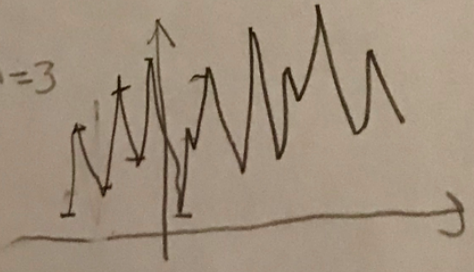
3. $\lambda = \infty, m = 2$



4. $\lambda = \infty, m = 3$



5. $\lambda = 0, m = 3$



$\hat{g}(x)$ will interpolate the training data.

6

a)

```
K = 10;
knn.classifier <- function(X.train, y.train, X.test, k.try = 1, pi = rep(1/K,K), CV = F){
  classes = sort(unique(y.train));
  X.train = as.matrix(X.train);
  X.test = as.matrix(X.test);
  n.is = c();
  for (class in classes) {
    n.is = c(n.is, sum(y.train == class));
  }
  output = matrix(nrow=nrow(X.test), ncol=length(k.try));
  for (row.test in 1:(nrow(X.test))) {
    # print(row.test);
    distances = rowSums(sweep(X.train, 2, as.vector(X.test[row.test,])) ^ 2)
    if (CV == T) {
      distances[row.test] = max(distances);
    }
    preds = y.train[sort(distances, index.return=TRUE)$ix];
    for (k.try.idx in 1:length(k.try)) {
      knns = preds[1:k.try[k.try.idx]];
      max.n.occurences = 0;
      best.prediction = -1;
      for (pred in unique(knns)) {
        pred.idx = match(pred, classes)
        n_occurences = sum(knns == pred) * pi[pred.idx] / n.is[pred.idx];
        if (n_occurences > max.n.occurences) {
          best.prediction = pred;
          max.n.occurences = n_occurences;
        }
      }
      output[row.test, k.try.idx] = best.prediction;
    }
  }
  return(output);
}
```

b)

```
library(datasets);
data(iris);
X.train = iris[,names(iris) != 'Species']
X.test = X.train
y.train = iris$Species
K = length(unique(y.train))
pi = c();
for (cls in sort(unique(y.train))) {
  pi = c(pi, sum(y.train == cls) / nrow(iris));
}
knn.nocv = knn.classifier(X.train, y.train, X.test, k.try = 5, pi = pi, CV = F);
knn.cv = knn.classifier(X.train, y.train, X.test, k.try = 1, pi = pi, CV = T);
```

```
print(paste0('Iris No CV misclassifications (out of 150 predictions): ', sum(knn.nocv != y.train)))
```

```
## [1] "Iris No CV misclassifications (out of 150 predictions): 5"
```

```
print(paste0('Iris CV misclassifications (out of 150 predictions): ', sum(knn.cv != y.train)))
```

```
## [1] "Iris CV misclassifications (out of 150 predictions): 6"
```

c)

For parts c and d, it took too long to evaluate the algorithm

```
k.try = c(1, 3, 7, 11, 15, 21, 27, 35, 43);
train = read.table('/Users/stewart/Downloads/zip-train.dat');
X.train = train[,2:257]
X.test = X.train
y.train = train[,1]
pi = c();
for (cls in sort(unique(y.train))) {
  pi = c(pi, sum(y.train == cls) / nrow(train));
}
knn.out = knn.classifier(X.train, y.train, X.test, k.try = k.try, pi = pi, CV = T);

best.k = 0;
best.error = 'inf';
for (k.idx in 1:length(k.try)) {
  preds = knn.out[,k.idx];
  n_errors = sum(preds != y.train);
  print(paste0('N errors for k = ', k.try[k.idx], ': ', n_errors));
  if (n_errors < best.error) {
    best.k = k.try[k.idx];
    best.error = n_errors;
  }
}
```

```
## [1] "N errors for k = 1: 70"
## [1] "N errors for k = 3: 83"
## [1] "N errors for k = 7: 101"
## [1] "N errors for k = 11: 121"
## [1] "N errors for k = 15: 138"
## [1] "N errors for k = 21: 157"
## [1] "N errors for k = 27: 186"
## [1] "N errors for k = 35: 214"
## [1] "N errors for k = 43: 232"
```

```
print(paste0('Optimal k: ', best.k));
```

```
## [1] "Optimal k: 1"
```

d)

```
test = read.table('/Users/stewart/Downloads/zip-test.dat')
test.preds = knn.classifier(X.train, y.train, test[,2:257], k.try=best.k, pi=pi, CV=F);

test.error = sum(test.preds != test[, 1]) / nrow(test);
print(paste0('Estimated Test Error Rate ', test.error));
```

```
## [1] "Estimated Test Error Rate 0.0792227204783259"
```