STATS 315A HW#2

Adhitya Venkatesh

February 17, 2018

4. a.) When matrix X has its columns centered and linear regression is run on it, the resulting coefficients for all predictor variables will be equivalent to the corresponding coefficient values resulting from running OLS on the original matrix X. However, the intercept after column-centering will be the sample mean of the y vector whereas in the original case the intercept will be the sample mean of y minus the sum of products of coefficient estimates of each predictor variable multiplied by the mean value of that predictor variable in the given matrix X.
   b.) This is accomplished by computing the new RSS value from adding each of the p-q variables and choosing the one that causes the most significant reduction from the original RSS value. Each corresponding column of Z can be added via orthogonalization and run without affecting the parameter estimates of existing "q" predictors already present.

c.) In order to perform forward-stepwise regression as outlined in the slides, we follow the algorithm 3.1 as detailed in ESL. Basically, we calculate loop through each predictor currently in the model and regressing on residual vectors and then finally using the output to calculate the residual for the additional variable after adjusting with the residual vectors previously calculated. The added predictor yielding the smallest RSS is added to the subset and the process continues till termination.

7. a.)

```
#We load and subset the training and test data

ref_1 = "/Users/Adi/Documents/zip.train"
ref_2 = "/Users/Adi/Documents/zip.test"

train.3 = as.matrix(read.csv(paste(ref_1, "train.3", sep = ""), header = F))
train.5 = as.matrix(read.csv(paste(ref_1, "train.5", sep = ""), header = F))
train.8 = as.matrix(read.csv(paste(ref_1, "train.8", sep = ""), header = F))
test_data = as.matrix(read.table(file = ref_2))

x_train = rbind(train.3, train.5, train.8)
y_train = rep(c(3,5,8), c(nrow(train.3), nrow(train.5), nrow(train.8)))

y_test = test_data[,1]
x_test = test_data[y_test == 3 |y_test == 5 |y_test == 8]
y_test = ytest[y_test == 3 |y_test == 5 |y_test == 8]


#We fit the data to an LDA model
mod_1 = lda(y_train~x_train)
y_pred = predict(x_test, mod_1, type = "class")
classification_table = table(y_test, y_pred)
accuracy = sum(diag(classification_table))/sum(classification_table)
```

When run on all the raw components, the model yields a 1.59% error on the training data and 8.74% error on the test data.

b.)

```
#Calculating SVD of training data and transforming in terms of principal components
data_svd = svd(x_train)
```

```
v_leading = (data_svd$v)[:,30]
x_train_adj = x_train*v_leading
x_test_adj = x_test*v_leading


mod_2 = lda(y_train~x_train_adj)
y_train_pred = predict(x_train_adj, mod_2, type = "class")
classification_table_1 = table(y_train, y_train_pred)
train_accuracy = sum(diag(classification_table_1))/sum(classification_table_1)

y_test_pred = predict(x_test_adj, mod_2, type = "class")
classification_table_2 = table(y_test, y_test_pred)
test_accuracy = sum(diag(classification_table_2))/sum(classification_table_2)
```

When run on the leading 30 principal components, the model yields a 4.84% error on the training data and 8.51% error on the test data.

c.)

```
v_leading = (data_svd$v)[:,10]
x_train_adj = x_train*v_leading
x_test_adj = x_test*v_leading


mod_3 = lda(y_train~x_train_adj)
y_train_pred = predict(x_train_adj, mod_3, type = "class")
classification_table_1 = table(y_train, y_train_pred)
train_accuracy = sum(diag(classification_table_1))/sum(classification_table_1)

y_test_pred = predict(x_test_adj, mod_3, type = "class")
classification_table_2 = table(y_test, y_test_pred)
test_accuracy = sum(diag(classification_table_2))/sum(classification_table_2)
```

When run on the leading 10 principal components, the model yields a 5.24% error on the training data and 8.12% error on the test data.

d.)

```
#Pooling non-overlapping 4x4 blocks via average
pool_cells = function(M){
 M = matrix(M, 16, 16)
 r_0 = 4*(1:4)
 r_1 = 4*(0:3) + 1
 r_2 = 4*(0:3) + 2
 r_3 = 4*(0:3) + 3

 M = M[r_0,] + M[r_1,] + M[r_2,] + M[r_3,]
 M = M[,r_0] + M[,r_1] + M[,r_2] + M[,r_3]

 as.vector(M)/16
}
#fitting to model after filtering
x_train_filtered = t(apply(x_train, 1, pool_cells))
x_test_filtered = = t(apply(x_test, 1, pool_cells))
mod_4 = lda(x_train_filtered, y_train)$class
train_accuracy = sum(predict(mod_4) == y_train)/length(y_train)
```

```
test_accuracy = sum(predict(mod_4, x_test_filtered) == y_test)/length(y_test)
```

After filtering the data and running LDA on the training data and comparing the predicted and actual outputs, we obtain a training error of 4.12% and a test error of 7.34%.

e.)

```
library(glmnet)
y_glmnet = y_train

for i = 1:length(y_train){
  if y_train[i] == 3{
    y_glmnet = 1
  }
  else if y_train[i] == 5{
    y_glmnet = 2
  }
  else{
    y_glmnet = 3
  }

mod_5 = glmnet(x_train, y_train)
y_train_pred = predict.glmnet(x_train, mod_5)$class
train_accuracy = sum(y_train_pred == y_train)/length(y_train)

y_test_pred = predict.glmnet(x_test, mod_5)$class
test_accuracy = sum(y_test_pred == y_test)/length(y_test)
```

When run on all the raw components after modifying output in suitable form for glmnet, the model yields a 0.29% error on the training data and 9.15% error on the test data.