# Final Problem 3

*Michael Frasco*

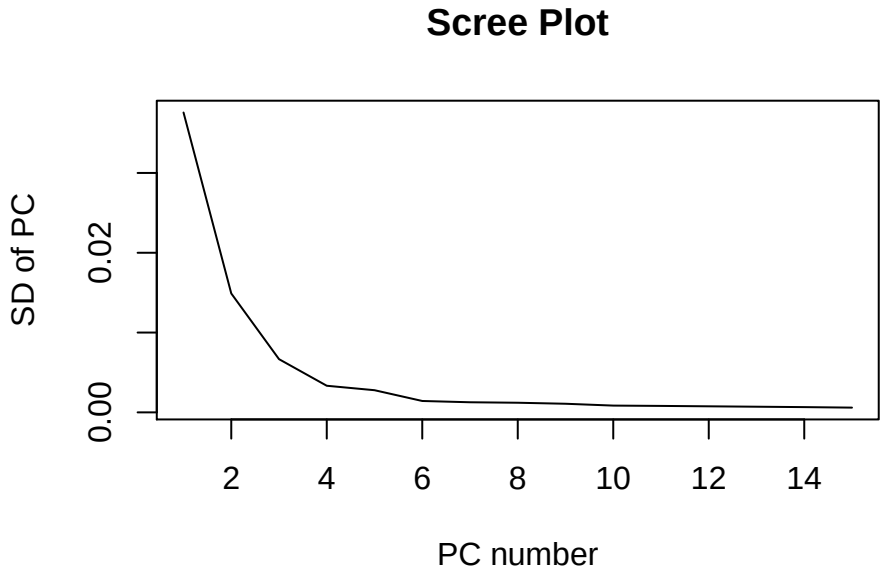
*December 4, 2014*

**Which of the variable selection or shrinking procedures seem suitable for this situation of n < p?** Stepwise procedures do not seem suitable for this situation. Since we have so many predictor variables, the total number of possible subsets for our predictors is too large to handle in a stepwise fashion. It is very likely that a greedy algorithm like stepwise gression would choose a subset that was much worse than the globally optimal subset. Principal component analysis, partial least squares, and lasso will all be able to handle the case where n is less than p. There are no restrictions on these shrinking methods in terms of how many observations must be included.

```
set.seed(1968)
randomRows = sample(166)
nirPermute = nir[randomRows, ]
trainData = nirPermute[1:126, ]
testData = nirPermute[127:166, ]
```

**Fit a principal component regression model, using both a scree plot and k-fold cross validation to select the number of components**

First, I create a scree plot resulting from principal compenents analysis. This plot is remarkable. With only 8 components, I can almost account for the entire variance across all 236 variables in the data set.



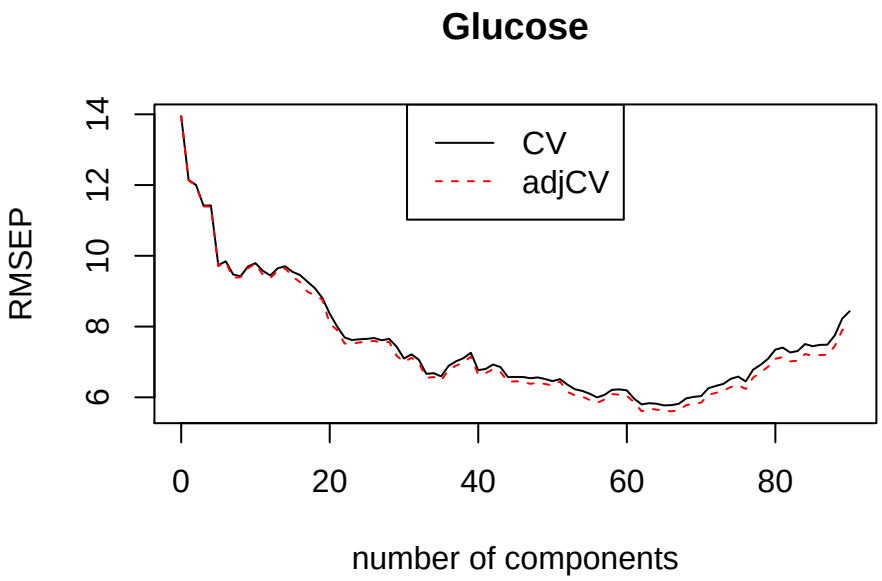

SUMMARY OF THE PLS PACKAGE AND CROSSVALIDATION. I created a function that allows me to specify the type of regression (pca or pls) and cross validation (K-fold or LOO). I have provided the code below. This function uses a built in R package which can perform cross validation on either pcr or pls. A very useful feature of the function is that it builds a model using every possible number of components from 1 to a max value I can set. By comparing the cross validation results of a model with only the first ten principal components to a model with the first 80 principal components, I can choose the best model. The cross validation works by first dividing the data into "k" partitions, or folds, (in the case of leave-one-out, the number of folds equals the number of observations). For each fold, k minus one folds are used to train a model, and then predictions are made on the fold left out. The R package I used calculates the root mean square error of prediction to evaluate the cross validated predictions. The function that I created returns

three things. The first is a graph of the cross-validation and adjusted cross-validation estimate of the RMSEP for the model with each number of principal components. Adjusted cross-validation corrects for the bias that might occur when a small number of folds are used. The second is the mean square error of the predictions on the training data. The third is the mean square error of the predictions on the forty observations in the testing data. This will allow me to assess bias and variance.

```
PLS_PCR_regression <- function(type, cv, k, numComp, graph = TRUE) {
      if (type == "pcr") {
            if (cv == "CV") {
                  model = pcr(Glucose ~ ., data = trainData,
                        validation = cv, ncomp = numComp, segments = k)
            } else {
                  model = pcr(Glucose ~ ., data = trainData,
                        validation = cv, ncomp = numComp)
            }
      } else {
            if (cv == "CV") {
                  model = plsr(Glucose ~ ., data = trainData,
                        validation = cv, ncomp = numComp, segments = k)
            } else {
                  model = plsr(Glucose ~ ., data = trainData,
                        validation = cv, ncomp = numComp)
            }
      }
      if (graph == TRUE) {
            validationplot(model, val.type = c("RMSEP"), legend= "top")
      }
      model_adjCV = RMSEP(model, "adjCV")$val
      best_num_comp = which.min(model_adjCV)
      train_RMSE = rmse(predict(model, ncomp = best_num_comp), trainData$Glucose)
      test_RMSE = rmse(predict(model, ncomp = best_num_comp, testData[-1]), testData$Glucose)
      return(c(best_num_comp, train_RMSE, test_RMSE))
}
```

Now, I call my function to perform pcr with 10-fold cross validation on models with one predictor component to 90 predictor components. I chose to perform ten fold cross validation over k = 5 and k = 15 because I ran multiple regressions with each value of k, and using k = 10 gave me consistent results with the lowest average testing errors.

```
set.seed(200)
PLS_PCR_regression(type = "pcr", cv = "CV", k = 10, numComp = 90)
```



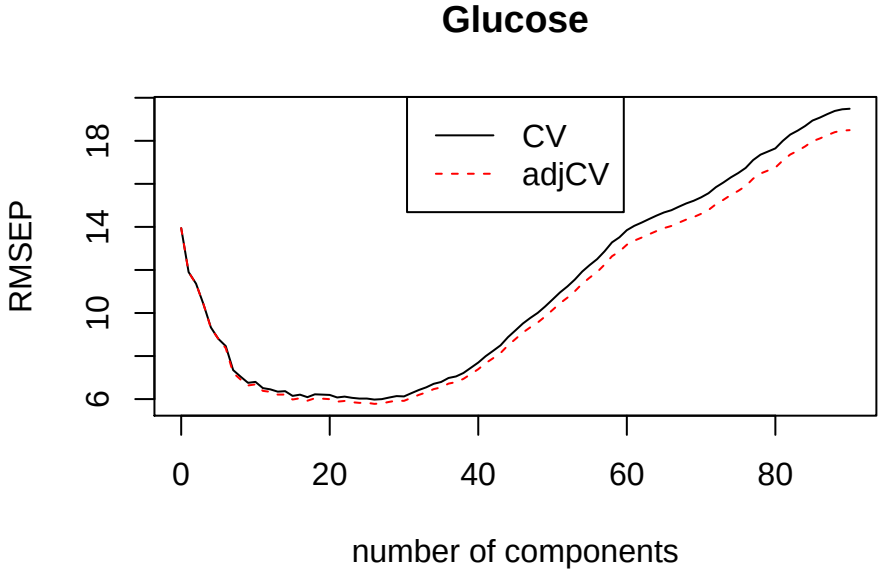

```
## [1] 67.000  2.601  5.537
```

The best model according to 10-fold cross validation has 67 parameters. The mean square error of the training data is 2.601 and 5.537 for the testing data. Since we have already seen a scree plot, it is surprising that using 67 parameters is the model that minimizes the cross validation error. However, the error that comes from using less than 10 componenets is much higher than the model with 67 components.

**Fit a partial least squares model, with a number of components chosen by a K-fold cross-validation approach, with the same number of K**

Now I can use my same function to quickly fit a partial least squares model. I'll choose the number of components to use based on 10-fold cross validation.

```
PLS_PCR_regression(type = "pls", cv = "CV", k = 10, numComp = 90)
```



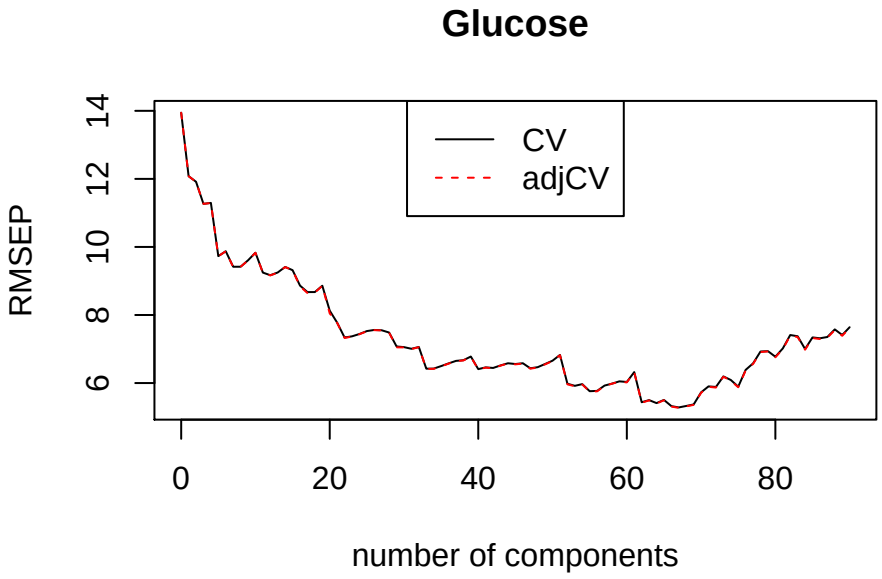

```
## [1] 27.000  2.433  5.422
```

The number of components used here is much less than the number in pcr. Our best model has 27 componenets, mean square error for the training set of 2.433, and a testing error of 5.422.

**For PCR and PLS redo the calculations using now a leave-one-out crossvalidation approach. Are the estimates of the error better or worse than the K-fold CV? What would you have expected**

I expect that the estimates of the training error will be better for leave-one-out cross validation, but the testing error will be worse. Put another way, leave-one-out crossvalidation will result in lower bias and higher variance. The reason it will have lower bias than k-fold cross validation is that more of the sample is being used to train the model each time. This allows the model to better resemble the true model. However, leave-one-out is also prone to overfitting, so we expect it to choose a model that has a greater testing error. Notice that in the validation plots for leave-one-out cross validation, the adjCV line is indentical to the CV line, since there is very little bias to adjust for.

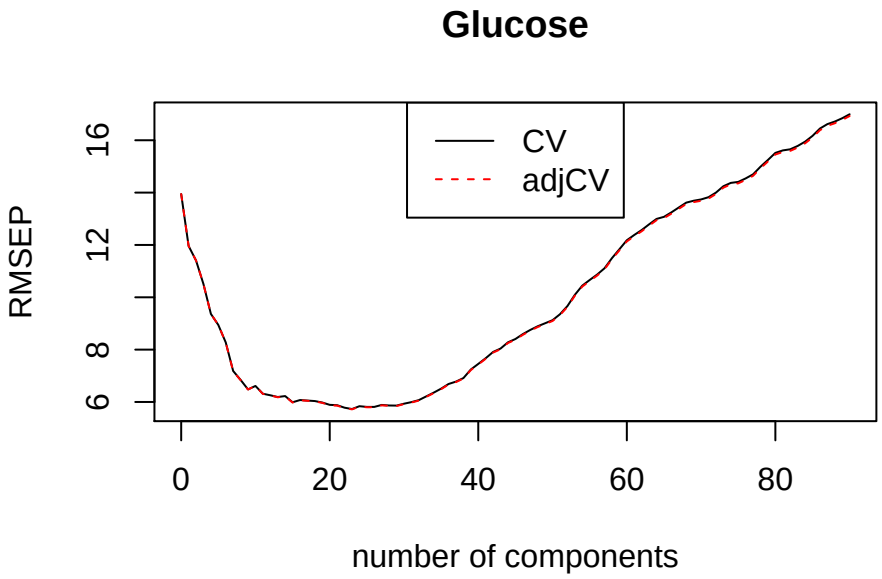
```
PLS_PCR_regression(type = "pcr", cv = "LOO", numComp = 90)
```




```
## [1] 68.000  2.582  5.643
```

As a result of leave-one-out cross validation, a model with 68 components was chosen. The model had a training set mean square error of 2.58, which is lower than the model selected by ten fold cross validation, and a testing error of 5.64, which is greater than the model selected by ten fold cross validation. This is exactly what we would have expected. Although, the results are very close.

```
PLS_PCR_regression(type = "pls", cv = "LOO", numComp = 90)
```



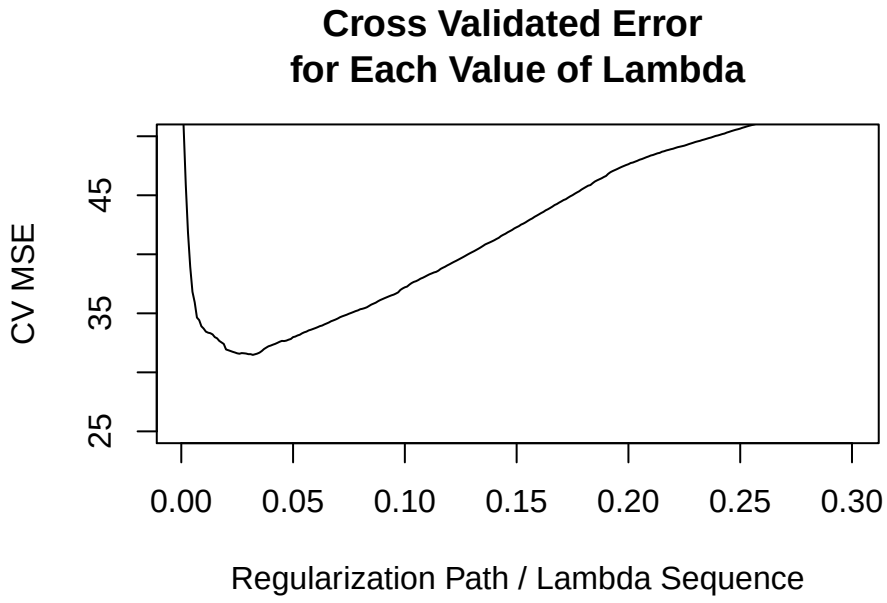

```
## [1] 24.000  2.593  5.264
```

When we perform leave one out cross validation for partial least squares, our best model has 24 components, a training error of 2.539, which is greater than that of k-fold cross validation, and a testing error of 5.264, which is less than that of k-fold. This is the opposite of what we would have expected. I repeated this regression multiple times, and the pattern is consistent.

**Fit a lasso model with the complexity parameter tuned by cross validation.**

SUMMARY OF LASSO FUNCTION: I chose to use the cv.glmnet function in the glmnet package to fit my lasso model with cross validation. This function takes a sequence of values for lambda (the regularization / complexity parameter) and fits a model at each value of lambda along the regularization path. It also allows us to perform k-fold cross validation based on the mean cross validated error for the model corresponding to each value of lambda. Similar to how we selected the complexity parameter in our tree model, we choose the model with the largest value of lambda whose cross validated error is with one standard deviation of the minimum cross validated error.

```
set.seed(666)
cvLasso <- cv.glmnet(x = as.matrix(trainData[,-1]), y = trainData[,1],
                     lambda = c(seq(from=8, to=1, by=-1), seq(from=0.3, to=0.001, by=-0.001)),
                     nfolds = 10, type.measure = "deviance")
```




```
##        num_coef.s211 complexity_parameter           train_rmse
##               34.000                 0.097                4.286
##            test_rmse
##                5.658
```

As a result of ten-fold cross validation, we get a lasso model with 34 non-zero coefficients, a complexity parameter of 0.097, a mean square error for the training data of 4.28, and a testing error of 5.65.

**Which of the three shrinking models do you prefer?**

I prefer the partial least squares model better than the principal components regression model and the lasso model. The primary reason that I prefer pls is that it results in the lowest training and testing errors, as seen in the summaries above. This makes sense seeing as how the algorithm of pls seeks to find the linear combination of the predictors that best explains the response. Even though pcr has training and testing errors that are similar to that of pls, the number of components used in the pls model (24) are much fewer than the number used in pcr (68) and lasso (34). Furthermore, pls and lasso have the advantage over pcr in the sense that they maintain the meaning of the predictors instead of transforming them.

**Theoretical benefits of K-fold vs leave one out**

In terms of computation time, leave one out is going to take longer than k-fold, since you have to do more cross validation regressions.

In terms of accuracy, the leave-one-out approach is going to have less bias but more variance than the k-fold approach. It has less bias because the subset used to train the data (i.e. all of the points except for the one being held out) better represents the entire training data than the subset being used to train the data in k-fold CV. We saw evidence of the bias-variance trade off in the validation plots that we created for each regression. The dotted line was mean square error adjusted for the bias. In the plots where we used leave-one-out cross-validation the adjustment is almost zero due to the fact that these models have very low bias. On the other hand, the adjustment in the k-fold plots is larger. As we increase K (approaching leave one out), the bias decreases.

However, the training error of leave-one-out will have higher variance that that of k-fold. This is because there is only one observation in the hold-out set, so the error on that fold could fluctuate more across all of the hold out sets. Leave-one-out cross validation also tends to overfit the training data, resulting in higher testing errors than k-fold cross validation, which we saw above. This is because the models fit by leave-one-out see

more data than those fit by k-fold, so they do a better job fitting the data they see, instead of the general trend that the training data represent.

Lastly, in terms of determinism, leave-one-out is completely deterministic since there is no random partitioning of the training data. You know that every point is going to be tested on a model built using every other point. On the other hand, as the value of K decreases, the cross validation becomes less deterministic and more random. The choice of which data points will be partitioned into the same fold is less clear. Thus, leave-one-out cross validation loses an aspect of randomness that is very important in statistics.