Question 2.1

I, like many other people living in the car centric focused United States, have been facing the daunting task of purchasing a new vehicle. Although I consider myself to be rich in spirit, my financial situation does not follow suit. As a result, I have had to weigh many predictors to make the right choice for both my finances and my personal needs. Standing at a towering 5'7" 120lbs, I require a vehicle that matches my monster-like build. As such, I decided that an SUV or something similar would be best, however something including the following predictors have helped me limit my decisions:

1. Price (<$35,000)
2. Fuel Efficiency
3. Make and Model
4. Current Vehicle Mileage
5. Number of Accidents on Vehicle

Question 2.2

1. Below you will find the two snippets of code used to produce the results for support vector machine function kvsm. Figure 1 denotes the initial parameters used to calculate the accuracy, taken from the code given in HW1 Header. In Figure 2, in order to find the C value corresponding with the greatest accuracy, I utilized a for loop to test multiple extremes ranging from (1e-6 to 1e5). This method granted insight towards which order of magnitude the highest accuracy would fall. My simulations returned that 0.01, 1, and 100 had the greatest accuracy compared to the higher and lower extremes, by a factor of 0.0015314 and 0.3154671, respectively. Overall there were no surprises in the data or accuracies outputted, as extreme changes to the constant of the regularization term (i.e C-value) would result in decreased accuracies across the board.

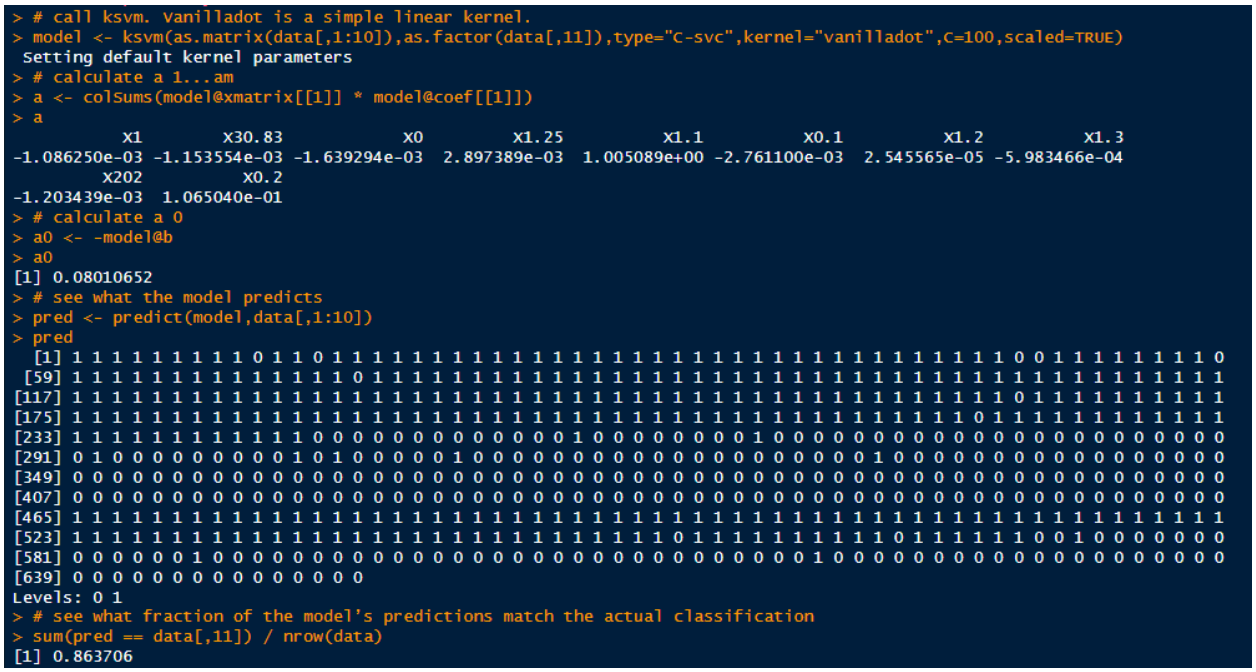

Figure 1. Initial Testing of kvsm Model.

```
> for (i in c(0.000001,0.0001,0.01,1,1000,100000))
+     {
+         # call ksvm. Vanilladot is a simple linear kernel.
+         model <- ksvm(as.matrix(data[,1:10]),as.factor(data[,11]),type="C-svc",kernel="vanilladot",C=i,scaled=TRUE)
+         # calculate a 1...am
+         a <- colSums(model@xmatrix[[1]] * model@coef[[1]])
+         # calculate a 0
+         a0 <- -model@b
+         # see what the model predicts
+         pred <- predict(model,data[,1:10])
+         pred
+         # see what fraction of the model's predictions match the actual classification
+         print(sum(pred == data[,11]) / nrow(data))
+         print(i)
+     }
 Setting default kernel parameters
[1] 0.5482389
[1] 1e-06
 Setting default kernel parameters
[1] 0.5482389
[1] 1e-04
 Setting default kernel parameters
[1] 0.863706
[1] 0.01
 Setting default kernel parameters
[1] 0.863706
[1] 1
 Setting default kernel parameters
[1] 0.8621746
[1] 1000
 Setting default kernel parameters
[1] 0.8621746
[1] 1e+05
```

Figure 2. Testing of Multiple C Values.

2. Despite not being required to be done, I chose to go above and beyond and try multiple different nonlinear kernels to determine the effect it would have on the accuracy of the kvsm model used on the given dataset. The tanh kernel sets the scale equal to 1 and the offset equal to 1. What this does is supply a convenient way of normalizing patterns without the need to modify the data itself. However, the greatest accuracy obtained from this method was 0.8621746, which is slightly less (0.0015314) than the highest accuracy obtained with the vanilla kernel (Figures 1 & 3). Next, the ANOVA kernel was used in the same fashion. Speaking in terms of the attributes of ANOVA, it sets the sigma (inverse kernel width) and the degree (of ANOVA function) equal to 1, and utilizes a Gaussian Radial Basis Function kernel to compute similarity between two points. Figure 4 displays the accuracies with the same range denoted in the previous trials. Out of all trials including the vanilla and tanh kernels, the ANOVA kernel showed the highest accuracy of them all with a value of 0.906585, at a C-value of 1000. In terms of why this result may have occurred, it can be seen in the way ANOVA interprets the model compared to the other kernels. For example, ANOVA allows for better control of the interactions in the model by determining the similarity between two data points, and supplying a more accurate classifier.

```
> for (i in c(0.000001,0.0001,0.01,1,1000,100000))
+     {
+         # call ksvm. vanilladot is a simple linear kernel.
+         model <- ksvm(as.matrix(data[,1:10]),as.factor(data[,11]),type="C-svc",kernel="tanhdot",C=i,scaled=TRUE)
+         # calculate a 1...am
+         a <- colSums(model@xmatrix[[1]] * model@coef[[1]])
+         # calculate a 0
+         a0 <- -model@b
+         # see what the model predicts
+         pred <- predict(model,data[,1:10])
+         pred
+         # see what fraction of the model's predictions match the actual classification
+         print(sum(pred == data[,11]) / nrow(data))
+         print(i)
+     }
 Setting default kernel parameters
[1] 0.5482389
[1] 1e-06
 Setting default kernel parameters
[1] 0.5482389
[1] 1e-04
 Setting default kernel parameters
[1] 0.8621746
[1] 0.01
 Setting default kernel parameters
[1] 0.7212864
[1] 1
 Setting default kernel parameters
[1] 0.7212864
[1] 1000
 Setting default kernel parameters
[1] 0.7212864
[1] 1e+05
```

Figure 3. Testing of tanh as a Classifier.

```
> for (i in c(0.000001,0.0001,0.01,1,1000,100000))
+     {
+         # call ksvm. Vanilladot is a simple linear kernel.
+         model <- ksvm(as.matrix(data[,1:10]),as.factor(data[,11]),type="C-svc",kernel="anovadot",C=i,scaled=TRUE)
+         # calculate a 1...am
+         a <- colSums(model@xmatrix[[1]] * model@coef[[1]])
+         # calculate a 0
+         a0 <- -model@b
+         # see what the model predicts
+         pred <- predict(model,data[,1:10])
+         pred
+         # see what fraction of the model's predictions match the actual classification
+         print(sum(pred == data[,11]) / nrow(data))
+         print(i)
+     }
 Setting default kernel parameters
[1] 0.5482389
[1] 1e-06
 Setting default kernel parameters
[1] 0.5482389
[1] 1e-04
 Setting default kernel parameters
[1] 0.8621746
[1] 0.01
 Setting default kernel parameters
[1] 0.863706
[1] 1
 Setting default kernel parameters
[1] 0.906585
[1] 1000
 Setting default kernel parameters
[1] 0.8667688
[1] 1e+05
```

Figure 4. Testing of ANOVA as a Classifier.

3. For the final portion of the homework, the results generated by the KNN are depicted in Figures 5&6. Figure 5 denotes (from left to right) each accuracy corresponding with a k-value following the pattern 1,2,3,..25. As depicted by the figure, the lower the k-value, the lower the accuracy up until k is greater than or equal to 5, and a decreasing accuracy as k becomes much larger (k>18). To take this even further, I decided to test the next accuracies for odd numbers only, as it would result in an automatic tie-break for each data point. Using this rationale, I expected to see greater results in the odd numbers, however they seemed in line with the even results generated in Figure 5. Unsurprisingly, Figure 6 did corroborate the notion that as the k-value reached magnitudes greater than 19, it resulted in a drop in accuracy compared to k-values between 5 and 18.

```
> acc <- rep(0,25) #initialize vector for printing accuracies
> for (X in 1:25){
+     acc[X] = check_accuracy(X) # test knn with X neighbors
+ }
>
> #print accuracies to console
>
> print(acc)
 [1] 0.8149847 0.8149847 0.8149847 0.8149847 0.8516820 0.8455657
 [7] 0.8470948 0.8486239 0.8470948 0.8501529 0.8516820 0.8532110
[13] 0.8516820 0.8516820 0.8532110 0.8516820 0.8516820 0.8516820
[19] 0.8501529 0.8501529 0.8486239 0.8470948 0.8440367 0.8455657
[25] 0.8455657
```

Figure 5. Printing of k-values from 1-25 with Associated Accuracies.

```
> xList <- c(seq(from=1, to=50, by=2)) #generate odd numbers
> acc <- rep(0,1) #initialize vector for printing accuracies
> for (X in xList){
+     acc[X] = check_accuracy(X) # test knn with X neighbors
+ }
>
> #print accuracies to console
>
> print(acc)
 [1] 0.8149847        NA 0.8149847        NA 0.8516820        NA
 [7] 0.8470948        NA 0.8470948        NA 0.8516820        NA
[13] 0.8516820        NA 0.8532110        NA 0.8516820        NA
[19] 0.8501529        NA 0.8486239        NA 0.8440367        NA
[25] 0.8455657        NA 0.8409786        NA 0.8394495        NA
[31] 0.8379205        NA 0.8348624        NA 0.8318043        NA
[37] 0.8318043        NA 0.8318043        NA 0.8318043        NA
[43] 0.8348624        NA 0.8394495        NA 0.8379205        NA
[49] 0.8394495
```

Figure 6. Printing of Odd Numbered (1-49) k-values with Associated Accuracies.

```
model=kknn(V11~V1+V2+V3+V4+V5+V6+V7+V8+V9+V10,data[-i,],data[
i,],k=X, scale = TRUE)
```

Figure 7. Code Utilized to Generate KNN Model.