HW 1 - Kelly "Scott" Sims

Question 2.1

Describe a situation or problem from your job, everyday life, current events, etc., for which a classification model would be appropriate. List some (up to 5) predictors that you might use.

> I have experimented with image detecting machine learning algorithms (CNN - Convolutional Neural Networks). There is a yearly ImageNet object detection competition that supplies terrabytes of labeled images for you to train a machine leanring algorithm on. I created a TensorFlow API algorithm that reads in an image and predicts what the image is (bird, dog, stove or car in this particular project). So the classification metric is a multiclass one in which the classifier predicts either bird, dog, stove, or car. The predictors for this particular classifier are all the same fortunately, and that is pixel intensities. An image is just a matrix of pixel values ranging from 0 to 255. Closer to 0, the darker a pixel is and closer to 255, the brighter a pixel is. Based on the intensities of neighboring pixels, the classifier finds edges. That is all a convolutional neural network is an edge detector. Depending on where these edges are and how bright they are (pixel value), it makes a prediction of one of the 4 classes. The overall image size determines how many predictors there are (pixels). If an image is 300x300, then there are 90,000 predictors that are fed into the model for a grayscale image. If the image is colored, then there are 300x300x3 (3 for RGB, red green and blue channels) or 270,000,000 predictors. My project can be seen on Github Here:

(https://github.com/Mooseburger1/Springboard-Data-Science-Immersive/tree/master/Capstone%202%20Project)

and

(https://github.com/Mooseburger1/Springboard-Data-Science-Immersive/blob/master/Capstone%202%20Project/Report/Exploring%20Computational%20Efficiency%20in%20Object%20Detection%20with%20Convolutional%20Neural%20Networks.pdf)

Question 2.2

The files credit_card_data.txt (without headers) and credit_card_data-headers.txt (with headers) contain a dataset with 654 data points, 6 continuous and 4 binary predictor variables. It has anonymized credit card applications with a binary response variable (last column) indicating if the application was positive or negative. The dataset is the "Credit Approval Data Set" from the UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/Credit+Approval) without the categorical variables and without data points that have missing values.

1. Using the support vector machine function *ksvm* contained in the R package *kernlab*, find a good classifier for this data. Show the equation of your classifier, and how well it classifies the data points in the full data set.

```
require("kernlab")
```

```
## Loading required package: kernlab
```

```
require("caret")
```

```
## Loading required package: caret
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
##
## Attaching package: 'ggplot2'
```

```
## The following object is masked from 'package:kernlab':
##
##     alpha
```

```
data <- read.csv("./credit_card_data-headers.csv", header = TRUE)
X <- as.matrix(data[,1:10])
y <- as.matrix(data[,11])

C = c(10, 50, 100, 500)

for(i in C){
cat("For C=", i, "\n")
model <- ksvm(x = X, y = y, type = "C-svc", kernel = "vanilladot",
              C = i, scaled = TRUE)

pred <- predict(model, X)


percentage <- sum(pred == y) / nrow(X)



a <- colSums(model@xmatrix[[1]] * model@coef[[1]])

a0 <- -model@b

cat("\nEquation\n")
cat("-----------------------------------------------------------------------\n")
cat(a0,"+",a[1],"+",a[2],"+",a[3],"+",a[4],"+\n",a[5],"+",a[6],"+",a[7],"+",a[8],"+",a[9],"+",a[10])

cat("\n\n")
print(confusionMatrix(table(pred, y), positive = "1"))
cat("\n\n")
}
```

```
## For C= 10
##  Setting default kernel parameters
##
## Equation
## -----------------------------------------------------------------
## 0.08157559 + -0.0009033671 + -0.0007891036 + -0.001697213 + 0.002611363 +
##  1.005022 + -0.002836302 + -0.0001569285 + -0.0003925964 + -0.001278444 + 0.1064387
##
## Confusion Matrix and Statistics
##
##     y
## pred   0   1
##    0 286  17
##    1  72 279
##
##                Accuracy : 0.8639
##                  95% CI : (0.8352, 0.8893)
```

```
##     No Information Rate : 0.5474
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7297
##  Mcnemar's Test P-Value : 1.041e-08
##
##             Sensitivity : 0.9426
##             Specificity : 0.7989
##          Pos Pred Value : 0.7949
##          Neg Pred Value : 0.9439
##              Prevalence : 0.4526
##          Detection Rate : 0.4266
##    Detection Prevalence : 0.5367
##       Balanced Accuracy : 0.8707
##
##        'Positive' Class : 1
##
##
##
## For C= 50
##  Setting default kernel parameters
##
## Equation
## ---------------------------------------------------------------------
## 0.08147145 + -0.001052363 + -0.001202513 + -0.001538266 + 0.0028762 +
##  1.005276 + -0.002495809 + 0.0001810245 + -0.0006514829 + -0.001375714 + 0.1064003
##
## Confusion Matrix and Statistics
##
##     y
## pred   0   1
##    0 286  17
##    1  72 279
##
##                Accuracy : 0.8639
##                  95% CI : (0.8352, 0.8893)
##     No Information Rate : 0.5474
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7297
##  Mcnemar's Test P-Value : 1.041e-08
##
##             Sensitivity : 0.9426
##             Specificity : 0.7989
##          Pos Pred Value : 0.7949
##          Neg Pred Value : 0.9439
##              Prevalence : 0.4526
##          Detection Rate : 0.4266
##    Detection Prevalence : 0.5367
##       Balanced Accuracy : 0.8707
##
##        'Positive' Class : 1
##
##
```

```
##
## For C= 100
##  Setting default kernel parameters
##
## Equation
## ---------------------------------------------------------------------
## 0.08158492 + -0.001006535 + -0.001172905 + -0.001626197 + 0.00300642 +
##  1.004941 + -0.002825943 + 0.0002600295 + -0.0005349551 + -0.001228376 + 0.1063634
##
## Confusion Matrix and Statistics
##
##     y
## pred   0   1
##    0 286  17
##    1  72 279
##
##                Accuracy : 0.8639
##                  95% CI : (0.8352, 0.8893)
##     No Information Rate : 0.5474
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7297
##  Mcnemar's Test P-Value : 1.041e-08
##
##             Sensitivity : 0.9426
##             Specificity : 0.7989
##          Pos Pred Value : 0.7949
##          Neg Pred Value : 0.9439
##              Prevalence : 0.4526
##          Detection Rate : 0.4266
##    Detection Prevalence : 0.5367
##       Balanced Accuracy : 0.8707
##
##        'Positive' Class : 1
##
##
##
## For C= 500
##  Setting default kernel parameters
##
## Equation
## ---------------------------------------------------------------------
## 0.08534036 + -0.0006306278 + -0.0001994861 + -0.0003750699 + 0.001615496 +
##  1.003398 + -0.0003814784 + 6.761309e-05 + -3.621798e-05 + -0.0001272743 + 0.1057258
##
## Confusion Matrix and Statistics
##
##     y
## pred   0   1
##    0 286  17
##    1  72 279
##
##                Accuracy : 0.8639
##                  95% CI : (0.8352, 0.8893)
```

```
##     No Information Rate : 0.5474
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7297
##  Mcnemar's Test P-Value : 1.041e-08
##
##             Sensitivity : 0.9426
##             Specificity : 0.7989
##          Pos Pred Value : 0.7949
##          Neg Pred Value : 0.9439
##              Prevalence : 0.4526
##          Detection Rate : 0.4266
##    Detection Prevalence : 0.5367
##       Balanced Accuracy : 0.8707
##
##        'Positive' Class : 1
##
```

*__Conclusion__ With the vanilladot kernel SVM classifier, regardless of the value C, it appears that the data is linearly separable at most 86%. Experimenting with values of C ranging from 10 to 500, the accuracy was always about 86%. The y- intercept from all equations was always about 0.08. From all the confusion matrices from each model, we can see that when the True value was 0, the model predicted zero 286 times out of 358. This means it was incorrect 72 times. And when the True value was 1, the model predicted one 279 times out of 296. This means it was incorrect 17 times. A non linear kernel SVM might perform better. Let's do that next.

2. You are welcome, but not required, to try other (nonlinear) kernels as well; we're not covering them in this course, but they can sometimes be useful and might provide better predictions than vanilladot.

```
require("kernlab")
require("caret")
data <- read.csv("./credit_card_data-headers.csv", header = TRUE)
X <- as.matrix(data[,1:10])
y <- as.matrix(data[,11])

C = c(10, 50, 100, 500)

for(i in C){
cat("For C=", i, "\n")
model <- ksvm(x = X, y = y, type = "C-svc", kernel = "rbfdot",
              C = i, scaled = TRUE)

pred <- predict(model, X)


percentage <- sum(pred == y) / nrow(X)



a <- colSums(model@xmatrix[[1]] * model@coef[[1]])

a0 <- -model@b

cat("\nEquation\n")
cat("-----------------------------------------------------------------\n")
```

```
cat(a0,"+",a[1],"+",a[2],"+",a[3],"+",a[4],"+\n",a[5],"+",a[6],"+",a[7],"+",a[8],"+",a[9],"+",a[10])

cat("\n\n")
print(confusionMatrix(table(pred, y), positive = "1"))
cat("\n\n")
}
```

```
## For C= 10
##
## Equation
## ----------------------------------------------------------------
## 0.4396519 + -3.028303 + -17.79666 + 3.65582 + 26.80679 +
##  32.04049 + -12.32479 + 16.22005 + -9.51336 + -32.94186 + 37.22976
##
## Confusion Matrix and Statistics
##
##     y
## pred   0   1
##    0 330  27
##    1  28 269
##
##                Accuracy : 0.9159
##                  95% CI : (0.8919, 0.936)
##     No Information Rate : 0.5474
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8303
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9088
##             Specificity : 0.9218
##          Pos Pred Value : 0.9057
##          Neg Pred Value : 0.9244
##              Prevalence : 0.4526
##          Detection Rate : 0.4113
##    Detection Prevalence : 0.4541
##       Balanced Accuracy : 0.9153
##
##        'Positive' Class : 1
##
##
##
## For C= 50
##
## Equation
## ----------------------------------------------------------------
## 0.7262287 + -8.786796 + -33.47718 + -4.669235 + 48.39777 +
##  38.06392 + -15.16716 + 11.57216 + -17.50918 + -49.12375 + 43.81893
##
## Confusion Matrix and Statistics
##
##     y
## pred   0   1
##    0 341  20
```

```
##    1  17 276
##
##                Accuracy : 0.9434
##                  95% CI : (0.9229, 0.9599)
##     No Information Rate : 0.5474
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8857
##  Mcnemar's Test P-Value : 0.7423
##
##             Sensitivity : 0.9324
##             Specificity : 0.9525
##          Pos Pred Value : 0.9420
##          Neg Pred Value : 0.9446
##              Prevalence : 0.4526
##          Detection Rate : 0.4220
##    Detection Prevalence : 0.4480
##       Balanced Accuracy : 0.9425
##
##        'Positive' Class : 1
##
##
##
## For C= 100
##
## Equation
## ---------------------------------------------------------------
## 0.6964003 + -18.57436 + -34.48544 + -7.652224 + 55.63805 +
##  51.52331 + -27.00694 + 21.77367 + -24.69934 + -56.56499 + 54.85537
##
## Confusion Matrix and Statistics
##
##     y
## pred   0   1
##    0 348  18
##    1  10 278
##
##                Accuracy : 0.9572
##                  95% CI : (0.9387, 0.9714)
##     No Information Rate : 0.5474
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9134
##  Mcnemar's Test P-Value : 0.1859
##
##             Sensitivity : 0.9392
##             Specificity : 0.9721
##          Pos Pred Value : 0.9653
##          Neg Pred Value : 0.9508
##              Prevalence : 0.4526
##          Detection Rate : 0.4251
##    Detection Prevalence : 0.4404
##       Balanced Accuracy : 0.9556
##
```

```
##        'Positive' Class : 1
##
##
##
## For C= 500
##
## Equation
## ---------------------------------------------------------------
## 0.7496402 + -40.56077 + -14.08409 + -26.0971 + 120.9104 +
##  71.48936 + -69.37463 + 92.05872 + -54.46091 + -71.15109 + 84.76695
##
## Confusion Matrix and Statistics
##
##     y
## pred   0   1
##    0 353  13
##    1   5 283
##
##                Accuracy : 0.9725
##                  95% CI : (0.9568, 0.9836)
##     No Information Rate : 0.5474
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.9443
##  Mcnemar's Test P-Value : 0.09896
##
##             Sensitivity : 0.9561
##             Specificity : 0.9860
##          Pos Pred Value : 0.9826
##          Neg Pred Value : 0.9645
##              Prevalence : 0.4526
##          Detection Rate : 0.4327
##    Detection Prevalence : 0.4404
##       Balanced Accuracy : 0.9711
##
##        'Positive' Class : 1
##
```

*__Conclusion__ Applying the rbf kernel or Radial Basis, the non linear classifier was able to achieve an accuracy of almost **97%** with a C-value of 500. Again looking at the confusion matrix, it misclassified values of 0 only 7 out of 358 observations. Conversely it misclassified values of 1 only 13 out of 296 times. This is a major improvement over the linear SVM

3. Using the k-nearest-neighbors classification function kknn contained in the R kknn package, suggest a good value of k, and show how well it classifies that data points in the full data set. Don't forget to scale the data (scale=TRUE in kknn).

```
require("kknn")
```

```
## Loading required package: kknn
```

```
##
## Attaching package: 'kknn'
```

```
## The following object is masked from 'package:caret':
##
##     contr.dummy
```

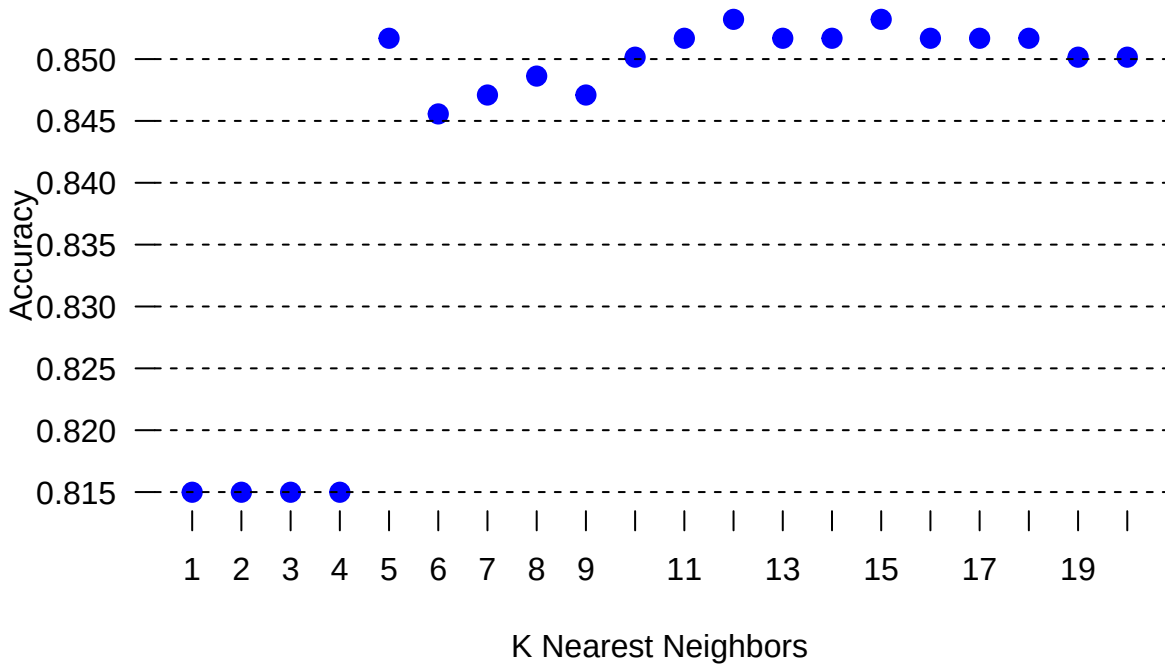
```
data = read.csv("./credit_card_data-headers.csv", header = TRUE)

accuracy_percentage <- rep(0,20)
for(j in 1:20){
      pred = rep(0, nrow(data))
      for(i in 1:nrow(data)){

            model <- kknn(R1~., data[-i,], data[i,], k = j, scale = TRUE)
            pred[i] <- round(fitted(model))

      }

      acc <- sum(pred == as.vector(data[,11])) / nrow(data)
      cat("Finished model with", j, "nearest neightbors.....................",(acc*100),"% Accuracy\n")
      accuracy_percentage[j] = acc

}
```

```
## Finished model with 1 nearest neightbors..... 81.49847 % Accuracy
## Finished model with 2 nearest neightbors..... 81.49847 % Accuracy
## Finished model with 3 nearest neightbors..... 81.49847 % Accuracy
## Finished model with 4 nearest neightbors..... 81.49847 % Accuracy
## Finished model with 5 nearest neightbors..... 85.1682 % Accuracy
## Finished model with 6 nearest neightbors..... 84.55657 % Accuracy
## Finished model with 7 nearest neightbors..... 84.70948 % Accuracy
## Finished model with 8 nearest neightbors..... 84.86239 % Accuracy
## Finished model with 9 nearest neightbors..... 84.70948 % Accuracy
## Finished model with 10 nearest neightbors..... 85.01529 % Accuracy
## Finished model with 11 nearest neightbors..... 85.1682 % Accuracy
## Finished model with 12 nearest neightbors..... 85.3211 % Accuracy
## Finished model with 13 nearest neightbors..... 85.1682 % Accuracy
## Finished model with 14 nearest neightbors..... 85.1682 % Accuracy
## Finished model with 15 nearest neightbors..... 85.3211 % Accuracy
## Finished model with 16 nearest neightbors..... 85.1682 % Accuracy
## Finished model with 17 nearest neightbors..... 85.1682 % Accuracy
## Finished model with 18 nearest neightbors..... 85.1682 % Accuracy
## Finished model with 19 nearest neightbors..... 85.01529 % Accuracy
## Finished model with 20 nearest neightbors..... 85.01529 % Accuracy
```

```
cat("\n\nBest performing model is model", which.max(accuracy_percentage),"with an accuracy score of", ma
```

```
##
##
## Best performing model is model 12 with an accuracy score of 0.853211
```

```
plot(accuracy_percentage, main = "KNN Accuracy", xlab = "K Nearest Neighbors", ylab = "Accuracy", col =
axis(1, seq(1,20,1), col = NA, col.ticks = 1)
axis(2, seq(0.80, 0.87, 0.005), col = NA, col.ticks = 1, las=2)
abline(h=seq(0.80, 0.87, 0.005),lty=2,col="black")
```



*__Conclusion__ It appears that having a K value of 12 is the best perfroming model, with an accuracy score 85.32%. Granted, this doesn't necessarily mean it is a good classifier, as no precesion or recall statistics have been calculated (confusion matrix) as done with SVM. But it appears to be a good starting point.