Homework 1

5/21/2020

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 work for a coalition loyalty program that rewards members for their purchases made on a credit card. I could use a classification model to predict if a member if going attrite and close their card in the next month.

Key indicators would include:

- Number of transactions in the last 3 months and 12 months
- Spend on the credit card in the last 3 months and 12 months
- Number of points earned in the last 3 and 12 months
- Number of items redeemed in the last 3 and 12 months

If there is a significant decrease in these predictors between the last 3 months and 12 months, then it's possible that the member is no longer engaged with the loyalty program and will likely attrite.

Question 2.2.1

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.

Loading the libraries and data:

```
library(kknn)
library(kernlab)
library(readr)
library(caret)
data=read.delim('credit_card_data_headers.txt')
x = data[,1:10]
y = data[,11]
set.seed(2184)
```

Using ksvm with the vanilladot kernal and testing different C values:

```
for (c_exp in c(-5:5))
{
  ksvm_model <- ksvm(as.matrix(x),as.factor(y),type='C-svc',kernel='vanilladot',C=10^c_exp,scaled=TRUE)
  pred <- predict(ksvm_model, x)
  accuracy <- sum(pred == y) / nrow(data)

  print(paste0("C value: ", 10^c_exp, " Exponent: ", c_exp, " Accuracy: ", accuracy))
}
```

```
##  Setting default kernel parameters
## [1] "C value: 1e-05 Exponent: -5 Accuracy: 0.547400611620795"
##  Setting default kernel parameters
## [1] "C value: 1e-04 Exponent: -4 Accuracy: 0.547400611620795"
##  Setting default kernel parameters
## [1] "C value: 0.001 Exponent: -3 Accuracy: 0.837920489296636"
##  Setting default kernel parameters
## [1] "C value: 0.01 Exponent: -2 Accuracy: 0.863914373088685"
##  Setting default kernel parameters
## [1] "C value: 0.1 Exponent: -1 Accuracy: 0.863914373088685"
##  Setting default kernel parameters
## [1] "C value: 1 Exponent: 0 Accuracy: 0.863914373088685"
##  Setting default kernel parameters
## [1] "C value: 10 Exponent: 1 Accuracy: 0.863914373088685"
##  Setting default kernel parameters
## [1] "C value: 100 Exponent: 2 Accuracy: 0.863914373088685"
##  Setting default kernel parameters
## [1] "C value: 1000 Exponent: 3 Accuracy: 0.862385321100917"
##  Setting default kernel parameters
## [1] "C value: 10000 Exponent: 4 Accuracy: 0.862385321100917"
##  Setting default kernel parameters
## [1] "C value: 1e+05 Exponent: 5 Accuracy: 0.863914373088685"
```

Since C= 10^-2 gives the best accuracy and larger C values didn't generate a better accuracy, I'll use 0.01 as my C value.

```
final_ksvm_model <- ksvm(as.matrix(x),as.factor(y),type='C-svc',kernel='vanilladot',C=0.01,scaled=TRUE)
```

```
##  Setting default kernel parameters
```

```
final_ksvm_model
```

```
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 0.01
##
## Linear (vanilla) kernel function.
##
## Number of Support Vectors : 288
##
## Objective Function Value : -2.2926
## Training error : 0.136086
```

Generating coefficients and intercept:

```
a <- colSums(final_ksvm_model@xmatrix[[1]] * final_ksvm_model@coef[[1]])
a0 <- final_ksvm_model@b
a
```

```
##            A1            A2            A3            A8            A9
## -0.0001500738 -0.0014818294  0.0014083130  0.0072863886  0.9916470037
##           A10           A11           A12           A14           A15
## -0.0044661236  0.0071482899 -0.0005468386 -0.0016930578  0.1054824270
```

```
a0
```

```
## [1] -0.08198854
```

Calculating accuracy:

```
pred <- predict(final_ksvm_model, x)
accuracy <- sum(pred == y) / nrow(data)
accuracy
```

```
## [1] 0.8639144
```

Results

The SVM classifer with the vanilladot kernal and C = 0.01 is:
-0.0001500738 * A1 - 0.0014818294 * A2 + 0.0014083130 * A3 + 0.0072863886 * A8 + 0.9916470037 * A9 - 0.0044661236 * A10 + 0.0071482899 * A11 - 0.0005468386 * A12 - 0.0016930578 * A14 + 0.1054824270 * A12 - 0.08198854 = 0

This classifier yields an accuracy of 86.4%.

Question 2.2.2

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.

Fixing C = 100 and testing different kernals:

```
for (kern in c('vanilladot', 'rbfdot', 'polydot', 'tanhdot', 'laplacedot', 'besseldot', 'anovadot', 'sp
{
  ksvm_model <- ksvm(as.matrix(x),as.factor(y),type='C-svc',kernel=kern,C=100,scaled=TRUE)
  pred <- predict(ksvm_model, x)
  accuracy <- sum(pred == y) / nrow(data)

  print(paste0("Kernal: ", kern, " Accuracy: ", accuracy))
}
```

```
##  Setting default kernel parameters
## [1] "Kernal: vanilladot Accuracy: 0.863914373088685"
## [1] "Kernal: rbfdot Accuracy: 0.957186544342508"
##  Setting default kernel parameters
## [1] "Kernal: polydot Accuracy: 0.863914373088685"
##  Setting default kernel parameters
## [1] "Kernal: tanhdot Accuracy: 0.7217125382263"
## [1] "Kernal: laplacedot Accuracy: 1"
##  Setting default kernel parameters
## [1] "Kernal: besseldot Accuracy: 0.925076452599388"
##  Setting default kernel parameters
## [1] "Kernal: anovadot Accuracy: 0.906727828746177"
##  Setting default kernel parameters
## [1] "Kernal: splinedot Accuracy: 0.978593272171254"
```

splinedot (Spline Kernal) yields the highest accuracy of 97.9%.
polydot (Polynomial Kernal) yields the exact same accuracy as the vanilladot.

Testing C = 0.01 and testing different kernals:

```
for (kern in c('vanilladot', 'rbfdot', 'polydot', 'tanhdot', 'laplacedot', 'besseldot', 'anovadot', 'sp
{
  ksvm_model <- ksvm(as.matrix(x),as.factor(y),type='C-svc',kernel=kern,C=0.01,scaled=TRUE)
  pred <- predict(ksvm_model, x)
  accuracy <- sum(pred == y) / nrow(data)

  print(paste0("Kernal: ", kern, " Accuracy: ", accuracy))
}
```

```
##  Setting default kernel parameters
## [1] "Kernal: vanilladot Accuracy: 0.863914373088685"
## [1] "Kernal: rbfdot Accuracy: 0.565749235474006"
##  Setting default kernel parameters
## [1] "Kernal: polydot Accuracy: 0.863914373088685"
##  Setting default kernel parameters
## [1] "Kernal: tanhdot Accuracy: 0.862385321100917"
## [1] "Kernal: laplacedot Accuracy: 0.547400611620795"
##  Setting default kernel parameters
## [1] "Kernal: besseldot Accuracy: 0.678899082568807"
##  Setting default kernel parameters
## [1] "Kernal: anovadot Accuracy: 0.862385321100917"
##  Setting default kernel parameters
## [1] "Kernal: splinedot Accuracy: 0.81039755351682"
```

vanilladot and polydot accuracy stays the same when C is changed and they both give the best accuracy out of all kernals in this scenario.

Question 2.2.3

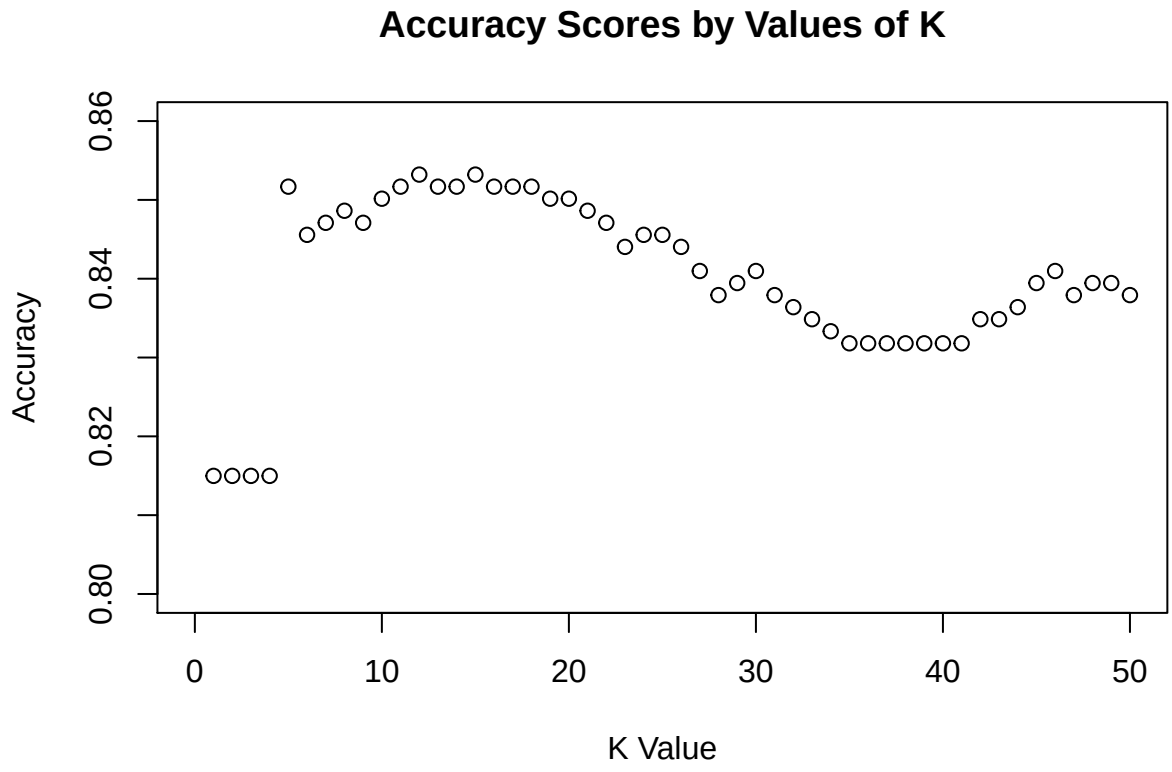
Using the k-nearest-neighbors classification function kknn contained in the R kknn package, suggest a good value of k, and show how well it classifies that data points in the full data set. Don't forget to scale the data (scale=TRUE in kknn).

For different values of k, I want to set up the knn model and calculate each k value's accuracy score. Once I have the accuracy score for values of k, I can choose the optimal k based on the accuracy score.

```
accuracies <- rep(0,50)

for (K in 1:50)
{
  # Initialize vector that will store the predicted values of model generated with each value of k
  prediction <- rep(0,(nrow(data)))

  for (i in 1:nrow(data)) # Iterate through each data point for leave one out validation.
    {
    model <- kknn(R1~.,data[-i,],data[i,], k=K, scale = TRUE)

    # Result of model is continuous, round to nearest 0 or 1
```

```
    prediction[i] <- round(fitted(model), digits=0)
    }
  # Calculate accuracy for each calculated prediction vector
  k_accuracy <- sum(prediction == data[,11]) / nrow(data)

  # Assign accuracy value to corresponding K index in the accuracies vector
  accuracies[K] <- k_accuracy
}
```

```
plot=plot(accuracies, main='Accuracy Scores by Values of K',xlab='K Value',ylab='Accuracy', xlim=c(0,50)
```




```
max(accuracies)
```

```
## [1] 0.853211
```

```
which.max(accuracies)
```

```
## [1] 12
```

Results

The optimal k is k = 12 and generates an accuracy of 85.3%.

Question 3.1

Using the same data set Question 2.2, use the ksvm or kknn function to find a good classifier:

(a) using cross-validation (do this for the k-nearest-neighbors model; SVM is optional) and

I will use kknn to find a good classifier. For cross validation, I want to separate the credit card data into two sets: one for training (80%) and one for testing (20%).

```
set.seed(0)
training_data <- createDataPartition(y=data[,11], p=0.8, list = FALSE)
train <- data[training_data,]
train[["R1"]] = factor(train[["R1"]])
test <- data[-training_data,]
```

Checking dimensions of my data:

```
dim(data)
```

```
## [1] 654  11
```

```
dim(train)
```

```
## [1] 524  11
```

```
dim(test)
```

```
## [1] 130  11
```

Use 10 folds for cross validation:

```
set.seed(0)
k_fold <- trainControl(method='cv', number = 10)
knn_model_10f <- train(R1~., data = train, method = "knn",
                  trControl = k_fold,
                  preProcess = c("center", "scale"),
                  tuneLength = 10)
knn_model_10f
```

```
## k-Nearest Neighbors
##
## 524 samples
##  10 predictor
##   2 classes: '0', '1'
##
## Pre-processing: centered (10), scaled (10)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 472, 471, 472, 471, 471, 472, ...
## Resampling results across tuning parameters:
##
```

```
##   k   Accuracy   Kappa
##    5  0.8509797  0.7000272
##    7  0.8548258  0.7084582
##    9  0.8490929  0.6968824
##   11  0.8338171  0.6649552
##   13  0.8357402  0.6688569
##   15  0.8433237  0.6841556
##   17  0.8453193  0.6879225
##   19  0.8510160  0.6993384
##   21  0.8414731  0.6778881
##   23  0.8414369  0.6776527
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 7.
```

Using 10 fold cross validation, the optimal k is 7. This is different from the optimal I calculated in the previous question of 12.

Testing out 5 folds for cross validation:

```
set.seed(0)
k_fold <- trainControl(method='cv', number = 5)
knn_model_5f <- train(R1~., data = train, method = "knn",
                  trControl = k_fold,
                  preProcess = c("center", "scale"), tuneLength = 10)
knn_model_5f
```

```
## k-Nearest Neighbors
##
## 524 samples
##  10 predictor
##   2 classes: '0', '1'
##
## Pre-processing: centered (10), scaled (10)
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 420, 419, 419, 419, 419
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    5  0.8473443  0.6919897
##    7  0.8510989  0.6998805
##    9  0.8606410  0.7187490
##   11  0.8549451  0.7079689
##   13  0.8588095  0.7151335
##   15  0.8664286  0.7300428
##   17  0.8568864  0.7101967
##   19  0.8568315  0.7088531
##   21  0.8568498  0.7089197
##   23  0.8492308  0.6934782
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 15.
```

With less folds used in cross validation, the optimal k is now 15.

Using the 10 Fold model on the test set and checking the accuracy:

```
res_10f <- predict(knn_model_10f, newdata = test)
sum(res_10f == test$R1) / nrow(test)
```

```
## [1] 0.8615385
```

Using the 5 Fold model on the test set and checking the accuracy:

```
res_5f <- predict(knn_model_5f, newdata = test)
sum(res_5f == test$R1) / nrow(test)
```

```
## [1] 0.8461538
```

Comparing the results from the 10 fold and 5 fold cross validated models on the test data, the accuracy for the 10 fold model is better at 86.2% while the 5 fold model is not that far behind at 84.6%

Since the accuracy in the 10 fold model is better, I will take k = 7 as the optimal k.

(b) splitting the data into training, validation, and test data sets (pick either KNN or SVM; the other

is optional).

Splitting the data into three parts: training (80%), testing (10%), validation (10%)

```
set.seed(8010)
split1 <- createDataPartition(y=data[,11], p=0.8,list = FALSE)
training <- data[split1,]
data2 <- data[-split1,]

split2 <- createDataPartition(y = data2[,11], p = 0.5, list = FALSE)
validate <- data2[split2,]
testing <- data2[-split2,]
```

Checking the dimesions of my data sets:

```
dim(data)
```

```
## [1] 654  11
```

```
dim(validate)
```

```
## [1] 65 11
```

```
dim(testing)
```

```
## [1] 65 11
```

```r
dim(training)
```

```
## [1] 524  11
```

Now that I have the training and validating data sets, I will use the training set to create different models with different values of k. Once I create each model, I will calculate accuracy to choose the optimal value for k.

```r
for (k in 1:40)
  {
  knn_model <- kknn(R1~., training, validate, k=k, scale = TRUE)
  predict <- round(fitted(knn_model))
  accuracy <- sum(predict == validate$R1) / nrow(validate)
  print(paste0("K: ", k, " Accuracy: ", accuracy))
  }
```

```
## [1] "K: 1 Accuracy: 0.876923076923077"
## [1] "K: 2 Accuracy: 0.876923076923077"
## [1] "K: 3 Accuracy: 0.876923076923077"
## [1] "K: 4 Accuracy: 0.876923076923077"
## [1] "K: 5 Accuracy: 0.907692307692308"
## [1] "K: 6 Accuracy: 0.907692307692308"
## [1] "K: 7 Accuracy: 0.907692307692308"
## [1] "K: 8 Accuracy: 0.892307692307692"
## [1] "K: 9 Accuracy: 0.892307692307692"
## [1] "K: 10 Accuracy: 0.892307692307692"
## [1] "K: 11 Accuracy: 0.892307692307692"
## [1] "K: 12 Accuracy: 0.892307692307692"
## [1] "K: 13 Accuracy: 0.907692307692308"
## [1] "K: 14 Accuracy: 0.907692307692308"
## [1] "K: 15 Accuracy: 0.907692307692308"
## [1] "K: 16 Accuracy: 0.907692307692308"
## [1] "K: 17 Accuracy: 0.907692307692308"
## [1] "K: 18 Accuracy: 0.907692307692308"
## [1] "K: 19 Accuracy: 0.907692307692308"
## [1] "K: 20 Accuracy: 0.907692307692308"
## [1] "K: 21 Accuracy: 0.892307692307692"
## [1] "K: 22 Accuracy: 0.876923076923077"
## [1] "K: 23 Accuracy: 0.861538461538462"
## [1] "K: 24 Accuracy: 0.861538461538462"
## [1] "K: 25 Accuracy: 0.861538461538462"
## [1] "K: 26 Accuracy: 0.861538461538462"
## [1] "K: 27 Accuracy: 0.861538461538462"
## [1] "K: 28 Accuracy: 0.861538461538462"
## [1] "K: 29 Accuracy: 0.861538461538462"
## [1] "K: 30 Accuracy: 0.861538461538462"
## [1] "K: 31 Accuracy: 0.861538461538462"
## [1] "K: 32 Accuracy: 0.861538461538462"
## [1] "K: 33 Accuracy: 0.861538461538462"
## [1] "K: 34 Accuracy: 0.861538461538462"
## [1] "K: 35 Accuracy: 0.861538461538462"
## [1] "K: 36 Accuracy: 0.861538461538462"
```

```
## [1] "K: 37 Accuracy: 0.861538461538462"
## [1] "K: 38 Accuracy: 0.861538461538462"
## [1] "K: 39 Accuracy: 0.861538461538462"
## [1] "K: 40 Accuracy: 0.861538461538462"
```

Based on these results, the highest accuracy occurs when k is between 5 and 7 or 13 and 20. I will use k = 7 and calculate the accuracy with the test set.

```
k = 7
knn_model_test <- kknn(R1~., training, testing, k=k, scale = TRUE)
predict <- round(fitted(knn_model_test))
accuracy <- sum(predict == testing$R1) / nrow(testing)
print(paste0("K: ", k, " Accuracy: ", accuracy))
```

```
## [1] "K: 7 Accuracy: 0.815384615384615"
```

Comparing the validation set and the test set, the accuracy score decreases from 90.8% to 81.5% as expected.