Homework 1

8/23/2020

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.

Discussion

In the oil and gas industry is very important to predict if a well is going to be commercial before starting the drilling process. I can see how support vector machines could be helpful for these type of problems because we can classify wells as “commercial” and “non commercial” and set a decision boundary based on what the data is telling us. With enough observations, we could train a model that could be applicable to a specific field and even to a basin.

In my opinion predictors could be: permeability, porosity, depth, API degrees and rate of penetration of the area.

Question 2.2.1

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 (https://archive.ics.uci.edu/ml/datasets/Credit+Approval)) without the categorical variables and without data points that have missing values.

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. (Don’t worry about test/validation data yet; we’ll cover that topic soon.)

Analysis and procedure

First, I loaded the data and displayed the first rows. I noticed 11 columns and per the indications in the lectures the last column is the response and the first 10 columns are the predictors.

```
# file.choose()  to select the directory
route <-"C:\\Users\\adri_\\Documents\\Gatech\\ISYE6501\\week 1_Introduction_classification\\Fall
2020hw1\\data 2.2\\credit_card_data-headers.txt"

credit_data <-read.table(route,sep="\t", header=TRUE)
head(credit_data)
```

```
##   A1    A2    A3   A8 A9 A10 A11 A12 A14 A15 R1
## 1  1 30.83 0.000 1.25  1   0   1   1 202   0  1
## 2  0 58.67 4.460 3.04  1   0   6   1  43 560  1
## 3  0 24.50 0.500 1.50  1   1   0   1 280 824  1
## 4  1 27.83 1.540 3.75  1   0   5   0 100   3  1
## 5  1 20.17 5.625 1.71  1   1   0   1 120   0  1
## 6  1 32.08 4.000 2.50  1   1   0   0 360   0  1
```

Then I set the seed to ensure that I get the same result always.

```
set.seed(1)
```

Because I did not have the package “kernlab” I had to install it. This package is useful for machine learning methods including classification through support vector machines. After installing it I activated the whole library.

```
#install.packages("kernlab")
library(kernlab)
```

With all the library available, I created the model using the “ksvm” tool. In order to avoid future problems with the data I converted the predictors observations into a matrix of values, and the response observations into a factor.

The ksvm tool has many different kernels, these kernels can be very simple for example when we can linearly separate the data but can also be more mathematically robust for example when we need to convert from one vector space into a different one to make the data separable. I trained this model using a linear kernel called “vanilladot” and scaled the features. I also set the hyperparameter C to 100, to see how it works.

```
ksvm_model<-ksvm(x = as.matrix(credit_data[,1:10]),
                 y = as.factor(credit_data[,11]),
                 type = "C-svc",
                 kernel= "vanilladot",
                 C = 100, scaled = TRUE)
```

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

After that,I calculated the parameters a1…am of the equation and the intercept a0.

```
# calculate a1...am
a <- colSums(ksvm_model@xmatrix[[1]] * ksvm_model@coef[[1]])
a
```

```
##            A1            A2            A3            A8            A9
## -0.0010065348 -0.0011729048 -0.0016261967  0.0030064203  1.0049405641
##           A10           A11           A12           A14           A15
## -0.0028259432  0.0002600295 -0.0005349551 -0.0012283758  0.1063633995
```

```
# calculate a0. negating the model to represent the final equation as "....a0 = 0" instead of
 "...... = -a0"
a0 <- - ksvm_model@b
a0
```

```
## [1] 0.08158492
```

This means, If we were to represent our equation,it would be:

(-0.0010065348)*A1 + (-0.0011729048)A2 + (-0.0016261967)*A3 + (0.0030064203)*A8 + (1.0049405641)* A9 + (-0.0028259432)* A10 + (0.0002600295)* A11 + (-0.0005349551)*A12 + (-0.0012283758)*A14 +(0.1063633995) * A15 + (0.08158492) = 0

Now, that I have a model trained, I will make some predictions using my SVM classifier. I can apply this in any data and see how it works. In this case, I will use it to predict on the credit data of the homework:

```
preds<- predict(ksvm_model,newdata=credit_data[,1:10])
```

In this step I will test the performance of the classifier with a confusion matrix. For this, I had to install the package "Caret" and activate the library.

```
#install.packages("caret")
library(caret)
```

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

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

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

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

```
confusionMatrix(factor(credit_data[,11],levels = c(1,0)),preds)
```

```
## Warning in confusionMatrix.default(factor(credit_data[, 11], levels = c(1, :
## Levels are not in the same order for reference and data. Refactoring data to
## match.
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 286  72
##          1  17 279
##
##                Accuracy : 0.8639
##                  95% CI : (0.8352, 0.8893)
##     No Information Rate : 0.5367
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7297
##
##  Mcnemar's Test P-Value : 1.041e-08
##
##             Sensitivity : 0.9439
##             Specificity : 0.7949
##          Pos Pred Value : 0.7989
##          Neg Pred Value : 0.9426
##              Prevalence : 0.4633
##          Detection Rate : 0.4373
##    Detection Prevalence : 0.5474
##       Balanced Accuracy : 0.8694
##
##        'Positive' Class : 0
##
```

Our confusion matrix shows that from the 654 observations, if they were to be perfectly predicted:

I would have 565 observations from the class 0. This calculation comes from (286+279), and 89 observations should have been from the class 1. This second calculation comes from (17+72).

Since the prediction is not perfect, as shown by the accuracy of the whole model by 86,39% then we can see how many true positives, true negatives, false positives and false negatives we have in our model.

The confusion matrix shows that 286 observations from the class “0” were accurately predicted while 72 were misclassified as to be from the same class. Similarly, 279 observations were accurately predicted from class “1” and 17 were false positives, this means they were misclassified to be of the class 0.

Positive predicted values comes from: 286 / (286 + 72) = 79,89 % and Negative predicted values comes from: 279 / (279 + 17) = 94.25 %

Sensitivity is the probability of getting a true positive.In this case from 286 / (286+17) = 94.39% and Specificity is the probability of getting a true negative.In this case from 279 / (72+279) = 79.49 %

Now, I want to see what fraction of the model’s predictions match the actual classification. This value was also calculated by the confusion matrix.

```
sum(preds == credit_data[,11]) / nrow(credit_data)
```

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

Final thoughts:

Using this linear kernel gives me an 86.39% of match between the actual prediction and the data analyzed.

Additional steps on Question 2.1

I tested the model assuming other C values, with the intention to change “the wide of the street” and see how this would impact the accuracy of the model.For this, I created a list and iterated over it. The results shows a big impact on the case of C= 1e-4, where there was a dramatic drop in the accuracy which makes sense because the smaller the C is, the wider the margin, therefore the bigger the chances to have more errors.

```
# Testing other C values
c_list <- list(1e-4, 1e-3, 1e-2, 0.1, 1, 10, 100, 1000)
acc_list <-list()



for (i in c_list) {
  ksvm_model<-ksvm(x = as.matrix(credit_data[,1:10]),
                   y = as.factor(credit_data[,11]),
                   type = "C-svc",
                   kernel= "vanilladot",
                   C = i, scaled = TRUE)
  preds<- predict(ksvm_model,newdata=credit_data[,1:10])
  acc_val <- (sum(preds == credit_data[,11]) / nrow(credit_data))
  acc_list <- append(acc_list, acc_val)

}
```

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

```
what_if_results <- do.call(rbind, Map(data.frame, C=c_list, Accuracy=acc_list))
what_if_results
```

```
##       C  Accuracy
## 1 1e-04 0.5474006
## 2 1e-03 0.8379205
## 3 1e-02 0.8639144
## 4 1e-01 0.8639144
## 5 1e+00 0.8639144
## 6 1e+01 0.8639144
## 7 1e+02 0.8639144
## 8 1e+03 0.8623853
```

Optional 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.

I went ahead and created 3 other models changing the kernels: A polynomial kernel, a radial basis kernel, and a hyperbolic tangent kernel. I assumed C = 0.01 in each model.

Polynomial kernel:

```
# Trying a polynomial kernel:
ksvm_model<-ksvm(x = as.matrix(credit_data[,1:10]),
                 y = as.factor(credit_data[,11]),
                 type = "C-svc",
                 kernel= "polydot", kpar=list(degree=3),
                 C = 0.01, scaled = TRUE)

preds<- predict(ksvm_model,newdata=credit_data[,1:10])
confusionMatrix(factor(credit_data[,1],levels = c(1,0)),preds)
```

```
## Warning in confusionMatrix.default(factor(credit_data[, 1], levels = c(1, :
## Levels are not in the same order for reference and data. Refactoring data to
## match.
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 104  99
##          1 251 200
##
##                Accuracy : 0.4648
##                  95% CI : (0.4261, 0.5039)
##     No Information Rate : 0.5428
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : -0.0367
##
##  Mcnemar's Test P-Value : 6.956e-16
##
##             Sensitivity : 0.2930
##             Specificity : 0.6689
##          Pos Pred Value : 0.5123
##          Neg Pred Value : 0.4435
##              Prevalence : 0.5428
##          Detection Rate : 0.1590
##    Detection Prevalence : 0.3104
##       Balanced Accuracy : 0.4809
##
##        'Positive' Class : 0
##
```

Gaussian kernel:

```
# Trying the radial basis kernel "Gaussian":
ksvm_model<-ksvm(x = as.matrix(credit_data[,1:10]),
                 y = as.factor(credit_data[,11]),
                 type = "C-svc",
                 kernel= "rbfdot",
                 C = 0.01, scaled = TRUE)

preds<- predict(ksvm_model,newdata=credit_data[,1:10])
confusionMatrix(factor(credit_data[,11],levels = c(1,0)),preds)
```

```
## Warning in confusionMatrix.default(factor(credit_data[, 11], levels = c(1, :
## Levels are not in the same order for reference and data. Refactoring data to
## match.
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 358   0
##          1 283  13
##
##                Accuracy : 0.5673
##                  95% CI : (0.5283, 0.6056)
##     No Information Rate : 0.9801
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0479
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.55850
##             Specificity : 1.00000
##          Pos Pred Value : 1.00000
##          Neg Pred Value : 0.04392
##              Prevalence : 0.98012
##          Detection Rate : 0.54740
##    Detection Prevalence : 0.54740
##       Balanced Accuracy : 0.77925
##
##        'Positive' Class : 0
##
```

Hyperbolic tangent kernel:

```
# Trying the Hyperbolic tangent kernel:
ksvm_model<-ksvm(x = as.matrix(credit_data[,1:10]),
                 y = as.factor(credit_data[,11]),
                 type = "C-svc",
                 kernel= "tanhdot",
                 C = 0.01, scaled = TRUE)
```

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

```
preds<- predict(ksvm_model,newdata=credit_data[,1:10])
confusionMatrix(factor(credit_data[,11],levels = c(1,0)),preds)
```

```
## Warning in confusionMatrix.default(factor(credit_data[, 11], levels = c(1, :
## Levels are not in the same order for reference and data. Refactoring data to
## match.
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 285  73
##          1  17 279
##
##                Accuracy : 0.8624
##                  95% CI : (0.8336, 0.8879)
##     No Information Rate : 0.5382
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7268
##
##  Mcnemar's Test P-Value : 6.731e-09
##
##             Sensitivity : 0.9437
##             Specificity : 0.7926
##          Pos Pred Value : 0.7961
##          Neg Pred Value : 0.9426
##              Prevalence : 0.4618
##          Detection Rate : 0.4358
##    Detection Prevalence : 0.5474
##       Balanced Accuracy : 0.8682
##
##        'Positive' Class : 0
##
```

Final thoughts on varying the kernel

In this case, the best scenario is the Hyperbolic tangent kernel as it has the best accuracy 86,24% and the best results for true positives (285 values) and true negatives (279 values).

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).

Analysis and procedure

I started cleaning the environment and the loaded the data again.

```
# Clear environment
rm(list = ls())

# Loading the data again
route <-"C:\\Users\\adri_\\Documents\\Gatech\\ISYE6501\\week 1_Introduction_classification\\Fall
2020hw1\\data 2.2\\credit_card_data-headers.txt"
credit_data <- read.table(route, stringsAsFactors = FALSE, header = TRUE)
head(credit_data)
```

```
##   A1    A2    A3   A8 A9 A10 A11 A12 A14 A15 R1
## 1  1 30.83 0.000 1.25  1   0   1   1 202   0  1
## 2  0 58.67 4.460 3.04  1   0   6   1  43 560  1
## 3  0 24.50 0.500 1.50  1   1   0   1 280 824  1
## 4  1 27.83 1.540 3.75  1   0   5   0 100   3  1
## 5  1 20.17 5.625 1.71  1   1   0   1 120   0  1
## 6  1 32.08 4.000 2.50  1   1   0   0 360   0  1
```

I proceeded to install the “kknn” package and activated the library.

```
#install.packages("kknn")
library(kknn)
```

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

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

Then I proceeded to create the model and iterated over a loop for k = 1 to k=15. In this loop I was careful not to use the “i th” point itself.

```
#Creating model and iterating 15 values of k.

accuracy <- vector("numeric", 15)
for (k in 1:15) {
  error_sum <- 0
  for (i in 1:nrow(credit_data)) {
    data.learn <- credit_data[-i, ]
    data.valid <- credit_data[i,]
    data.kknn <- kknn(R1~., data.learn, data.valid, k = k, scale = TRUE)
    data.fit <- round(fitted(data.kknn), digits = 0)
    error_sum <- error_sum + abs(data.fit - credit_data[i, 11])
  }
  accuracy[k] <- 1- error_sum/nrow(credit_data)
}
accuracy
```

```
##  [1] 0.8149847 0.8149847 0.8149847 0.8149847 0.8516820 0.8455657 0.8470948
##  [8] 0.8486239 0.8470948 0.8501529 0.8516820 0.8532110 0.8516820 0.8516820
## [15] 0.8532110
```

Final thoughts

When k= 12 the accuracy of the model is 85.32%. This also happens when k= 15, making these two classification models the best of all the options that I iterated.