Credit Card Classification Models

2024-01-18

Support Vector Machine Model for Classification

```
# call library 'kernlab' for use.
library(kernlab)

# load 'credit_card_data.txt' file into a table without headers.
myData <- read.table("credit_card_data.txt", stringsAsFactors = FALSE, header = FALSE)
head(myData)
```

```
##   V1    V2    V3   V4 V5 V6 V7 V8  V9 V10 V11
## 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
```

```
# call ksvm function, vanilladot is a simple linear kernel.
myModel <- ksvm(V11~V1+V2+V3+V4+V5+V6+V7+V8+V9+V10,
                data = myData,
                type = "C-svc",
                kernel = "vanilladot",
                C = 100,
                scaled = TRUE)
```

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

```
# call model
myModel
```

```
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 100
##
## Linear (vanilla) kernel function.
##
## Number of Support Vectors : 189
##
## Objective Function Value : -17887.92
## Training error : 0.136086
```

```
#Getting the coefficients of the equation of the classifier & the intercept.
a <- colSums(myModel@xmatrix[[1]]*myModel@coef[[1]])
a0 <- -myModel@b
a
```

```
##            V1            V2            V3            V4            V5
## -0.0010065348 -0.0011729048 -0.0016261967  0.0030064203  1.0049405641
##            V6            V7            V8            V9           V10
## -0.0028259432  0.0002600295 -0.0005349551 -0.0012283758  0.1063633995
```

```
a0
```

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

```
# see what the model predicts.
pred <- predict(myModel,myData[,1:10])
pred
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0
##  [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [112] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [149] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [186] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1
## [223] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1
## [260] 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
## [297] 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
## [334] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [371] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [408] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [445] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [482] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [519] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1
## [556] 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
## [593] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
## [630] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
# see what fraction of the model's predictions match the actual classification.
sum(pred == myData[,11]) / nrow(myData)
```

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

For this model, I have found the best C value parameter for the SVM. I have tried different values for C, starting at 10, and choosing magnitudes to replace the value of C. I have tried every power from 10^-4 to 10^6 and found that the fraction of the model's predictions that match the actual classification does not change when C is 1,10, or 100, but when the value of C is 100, it seems to have the least amount of support vectors attached, and hence is less prone to overfitting. The model's prediction accuracy with the optimal C value of 100 is 86.4%.

Also, below will be the mathematical formula for the equation of the classifier based on the coefficients of the weighted predictors found in the model...

Formula for the equation of the classifier:

$$0 = a1V1 + a2V2 + \ldots + a10V10 + a0$$

$$0 = -0.0010065348\,V1 + -0.0011729048\,V2 + -0.0016261967\,V3 + 0.0030064203\,V4 +$$

$$1.0049405641\,V5 + -0.0028259432\,V6 + 0.0002600295\,V7 + -0.0005349551\,V8 +$$

$$-0.0012283758\,V9 + 0.1063633995\,V10 + 0.08158492$$

K-Nearest Neighbor Model for Classification

```
# Call library 'kknn' for use.
library(kknn)

# Initialize a variable to store the count of the matches.
matches <- 0

# Iterate through values of i for all data points.
for (i in 1:654){
  # Create the kknn model to iterate through all 654 data points.
  myModel_knn = kknn(V11~.,
                     myData[-i,],
                     myData[i,],
                     k = 12,
                     distance = 2,
                     kernel = "optimal",
                     scale = TRUE)

  # Round the fitted values of the model to 0 or 1, to see if they match the data from myData.
  rounded_fitted_values <- round(fitted.values(myModel_knn))

  # Check if the rounded values match the data and add 1 to the matches if so.
  if (rounded_fitted_values == myData[i, 11]){
    matches <- matches + 1
  }
}

# Calculate the percentage of matches, (matches) / (total data points) * 100.
accuracy <- (matches / 654) * 100
accuracy
```

```
## [1] 85.3211
```

In this model, I wish to find the best k value for the knn model in order to have the highest percentage of matches from rounded fitted values of the model to the data's response. I have plugged in values for k from 1 to 100, and found that when k is very low around 1,2, or 3, the accuracy is at it's lowest, and when k is very high such as above 50, the accuracy is also at it's lowest. However, I have found two values for which the accuracy is at it's very best, and that is when the value of k is 12 and 15. At these values of k, the accuracy is a whopping 85.3211%.

Whilst using these 2 machine learning models to find the highest percentage of accuracy, I believe the Support Vector Machine did best with it's optimal C value, it was able to perform well and predict at a percentage of 86.4% which is only narrowly above the K-Nearest Neighbor model's percentage of 85.3%.