K-SVM Classification Using R

Kent Ng

May 20, 2018

Let's begin by loading the kernlab R package (already installed). Setting a seed value as best practice.

```
library(kernlab)
set.seed(42)
```

Next, load "2.2credit_Card_data-headersSummer2018.txt" into a dataframe

```
data_2_1<-read.table("2.2credit_card_data-headersSummer2018.txt", header = T, sep='\t')
```

View first 6 rows of the data to ensure the data was loaded properly

```
head(data_2_1)
```

```
##   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
```

Just for fun, let's take a look at how the data is structured and distributed.

```
str(data_2_1)
```

```
## 'data.frame':    654 obs. of  11 variables:
##  $ A1 : int  1 0 0 1 1 1 1 0 1 1 ...
##  $ A2 : num  30.8 58.7 24.5 27.8 20.2 ...
##  $ A3 : num  0 4.46 0.5 1.54 5.62 ...
##  $ A8 : num  1.25 3.04 1.5 3.75 1.71 ...
##  $ A9 : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ A10: int  0 0 1 0 1 1 1 1 1 1 ...
##  $ A11: int  1 6 0 5 0 0 0 0 0 0 ...
##  $ A12: int  1 1 1 0 1 0 0 1 1 0 ...
##  $ A14: int  202 43 280 100 120 360 164 80 180 52 ...
##  $ A15: int  0 560 824 3 0 0 31285 1349 314 1442 ...
##  $ R1 : int  1 1 1 1 1 1 1 1 1 1 ...
```

```
summary(data_2_1)
```

```
##        A1               A2              A3               A8
##  Min.   :0.0000   Min.   :13.75   Min.   : 0.000   Min.   : 0.000
##  1st Qu.:0.0000   1st Qu.:22.58   1st Qu.: 1.040   1st Qu.: 0.165
##  Median :1.0000   Median :28.46   Median : 2.855   Median : 1.000
##  Mean   :0.6896   Mean   :31.58   Mean   : 4.831   Mean   : 2.242
##  3rd Qu.:1.0000   3rd Qu.:38.25   3rd Qu.: 7.438   3rd Qu.: 2.615
##  Max.   :1.0000   Max.   :80.25   Max.   :28.000   Max.   :28.500
##        A9              A10              A11              A12
##  Min.   :0.0000   Min.   :0.0000   Min.   : 0.000   Min.   :0.0000
##  1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.: 0.000   1st Qu.:0.0000
##  Median :1.0000   Median :1.0000   Median : 0.000   Median :1.0000
##  Mean   :0.5352   Mean   :0.5612   Mean   : 2.498   Mean   :0.5382
##  3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.: 3.000   3rd Qu.:1.0000
##  Max.   :1.0000   Max.   :1.0000   Max.   :67.000   Max.   :1.0000
##       A14               A15               R1
##  Min.   :   0.00   Min.   :     0   Min.   :0.0000
##  1st Qu.:  70.75   1st Qu.:     0   1st Qu.:0.0000
##  Median : 160.00   Median :     5   Median :0.0000
##  Mean   : 180.08   Mean   :  1013   Mean   :0.4526
##  3rd Qu.: 271.00   3rd Qu.:   399   3rd Qu.:1.0000
##  Max.   :2000.00   Max.   :100000   Max.   :1.0000
```

Now lets call KSVM and see the coefficient values of A1 to A15

```
model<-ksvm(as.matrix(data_2_1[,1:10]),as.factor(data_2_1[,11]),type="C-svc",kernel="vanilladot"
,C=100,scaled=TRUE)
```

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

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

We can also find value of A0 from the ksvm model

```
A0<--model@b
A0
```

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

Therefore, the equation of the classifier is:

0.08158492 - 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

Let’s see what the model predicts

```
pred<-predict(model,data_2_1[,1:10])
pred
```

```
##   [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
##  [36] 1 1 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
##  [71] 1 1 1 0 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
## [106] 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
## [141] 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 1 1 1 1 1 1 1 1
## [176] 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
## [211] 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 1 1 1 1
## [246] 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
## [281] 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 0 0 0
## [316] 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 0 0 0 0 0 0 0
## [351] 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
## [386] 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
## [421] 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
## [456] 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [491] 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
## [526] 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 1 1 1 1 1
## [561] 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 0 0 0
## [596] 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
## [631] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## Levels: 0 1
```

Compare model predictions with actual classification (a.k.a model accuracy). For the purpose of this project, we will not validate our model accuracy using a test data set; therefore, our model will likely be overfitted and the model accuracy will be overstated. We will do so in later projects.

```
sum(pred == data_2_1[,11]) / nrow(data_2_1)
```

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

We can explore other values of C to try and see if we can get a better accuracy. First, let’s try to vary the k value from 100-105.

```
for(i in 100:105){
  model_new<-ksvm(as.matrix(data_2_1[,1:10]),as.factor(data_2_1[,11]),type="C-svc",kernel="vanil
ladot",C=i,scaled=TRUE)
  pred_new<-predict(model_new,data_2_1[,1:10])
  print(i)
  print (sum(pred_new == data_2_1[,11]) / nrow(data_2_1))
}
```

```
##  Setting default kernel parameters
## [1] 100
## [1] 0.8639144
##  Setting default kernel parameters
## [1] 101
## [1] 0.8639144
##  Setting default kernel parameters
## [1] 102
## [1] 0.8639144
##  Setting default kernel parameters
## [1] 103
## [1] 0.8639144
##  Setting default kernel parameters
## [1] 104
## [1] 0.8639144
##  Setting default kernel parameters
## [1] 105
## [1] 0.8639144
```

We can see that varying the C value by 1 to 10 has minimal effect to the model accuracy. Let's try changes in larger magnitude.

```
for(i in c(0.00001,0.0001,0.001,0.01,0.1,1,10,100,200,300,1000,10000)){
  model_new<-ksvm(as.matrix(data_2_1[,1:10]),as.factor(data_2_1[,11]),type="C-svc",kernel="vanil
ladot",C=i,scaled=TRUE)
  pred_new<-predict(model_new,data_2_1[,1:10])
  print(i)
  print (sum(pred_new == data_2_1[,11]) / nrow(data_2_1))
}
```

```
##  Setting default kernel parameters
## [1] 1e-05
## [1] 0.5474006
##  Setting default kernel parameters
## [1] 1e-04
## [1] 0.5474006
##  Setting default kernel parameters
## [1] 0.001
## [1] 0.8379205
##  Setting default kernel parameters
## [1] 0.01
## [1] 0.8639144
##  Setting default kernel parameters
## [1] 0.1
## [1] 0.8639144
##  Setting default kernel parameters
## [1] 1
## [1] 0.8639144
##  Setting default kernel parameters
## [1] 10
## [1] 0.8639144
##  Setting default kernel parameters
## [1] 100
## [1] 0.8639144
##  Setting default kernel parameters
## [1] 200
## [1] 0.8639144
##  Setting default kernel parameters
## [1] 300
## [1] 0.8623853
##  Setting default kernel parameters
## [1] 1000
## [1] 0.8623853
##  Setting default kernel parameters
## [1] 10000
## [1] 0.8623853
```

We can see that the model accuracy is at its highest when C is in the 0.01 to 200, after which the model accuracy begins to drop. Therefore, our C=100 value is a good (enough) choice.

Out of curiosity, let's try a different kernel ("Gaussian") in our ksvm model to see if it would increase our model accuracy

```
model_new<-ksvm(as.matrix(data_2_1[,1:10]),as.factor(data_2_1[,11]),type="C-svc",kernel="rbfdot"
,C=100,scaled=TRUE)
pred_newkern<-predict(model_new,data_2_1[,1:10])
sum(pred_newkern == data_2_1[,11]) / nrow(data_2_1)
```

```
## [1] 0.9525994
```

As we can see, using a different kernel of the ksvm can drastically change the performance of the model.