Omer Ansari
Class: ISY6501X, Homework 1
5/23/18

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.

Answer:

I could use classification modeling to make a decision on hiring someone or not, based on the various attributes such as

- Interview scores
- LinkedIn feedback
- Prior works of candidate measured on a scoring model
- Years in prior roles

This would require access to our HR database which would have chronicled prior applicants and the go/no go decision that was made on them. This data would then be cut up in training, validation and test data and classification engines such as SVM or KKNN would be applied to it.

Then, when a new applicant was to apply, the same prior history would be extracted by social media (LinkedIn) as well as by their resume, and the large pool of candidates would then be down-selected for further, deeper review by the hiring manager.

This would serve as a smarter algorithm of sifting through the morass of jobs by crude methods used today such as keyword searches, flowery descriptions, recommendations, which a non-standard filtering method applied since each recruiter has their own subjective method and most of the time these recruiters are swamped anyway and tend to miss critical candidates amongst the mass of applicants.

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

Separate files for scripts of code (well commented)

Why (Goal):
The goal of this exercise is to create a Support Vector Machine model which is able to reasonably predict, based on numerous data on a credit card applicant, whether his/her application should be approved or not. It is based on using the "Credit Approval Data Set" from the UCI Machine learning Repository.

How (Process):

Pre-requisites:
This scripts requires the kernlab library, specifically uses the ksvm model[1].
The code needs to be in the same directory as all the data files.

High Level approach on writing the code:
I took the repetitive tasks and moved them out into their own functions. These are

- Creating an SVM model - CreateSVM()
- Finding coefficients – FindCoefficients()
- Find a0 intercept – Finda0()
- Finding the accuracy of the model – FindModelAccuracy_SVM()

In my main code, I read the csv file in using read.table, and passed my SVM model various values of C. C is a multiplier to modify the cost, according to help:

C	cost of constraints violation (default: 1) this is the 'C'-constant of the regularization term in the Lagrange formulation.

[1] I had prior R experience, but had to use all the 3 Office hours TA sessions to understand how to use the packages. Google searches did not yield friendly results.

According to a TA comments in piazza, and my own research[2], C amplifies the weight of the training error. Meaning, the bigger the value of C is, the more weight the model gives towards minimizing the training error and conversely the lesser the significance is to maximizing the margin.

$$\min P(\boldsymbol{w}, b) = \underbrace{\frac{1}{2}\|\boldsymbol{w}\|^2}_{\text{maximize margin}} + \underbrace{C\sum_i H_1[y_i f(\boldsymbol{x}_i)]}_{\text{minimize training error}}$$

And then I iterated using different values of C to find out the most optimum model.

Some hygiene work: I created and contained my solution in a project[3] , though I won't be able to upload the Rproj file. This helps in the overall R workflow and avoids injecting spurious environment variables and other issues (like incorrect directories).

What (Results and Conclusion):

While I have one R script which runs through all the sections, I will just paste the output of the relevant sections here..

```
> source("HW1-OmerAnsari-ISYE6501x-Summer2018-v0.1a.R")
[1] "====Question 2.2.1 : create a good classifier for this data. Show the
equation of your classifier, and how well it classifies  ===="
[1] "====starting iteration===="
[1] "C Value:0.001"
[1] "------"
 Setting default kernel parameters
 [1] "co-efficients:-0.00215977830767829" "co-efficients:0.0323381696036407"
 [3] "co-efficients:0.0466124485479392"   "co-efficients:0.111223161663807"
 [5] "co-efficients:0.375305335384774"    "co-efficients:-0.20202608114064"
 [7] "co-efficients:0.169560846581749"    "co-efficients:-
0.00492350086822834"
 [9] "co-efficients:-0.0252102660043933"  "co-efficients:0.0811897660661701"
[1] "a0 co-efficient:0.222615544845752"
[1] "model accuracy %:0.837920489296636"
[1] "====ending iteration===="
[1] "====starting iteration===="
[1] "C Value:0.1"
[1] "------"
 Setting default kernel parameters
 [1] "co-efficients:-0.00116089804745002"  "co-efficients:-
0.000636600228635587"
 [3] "co-efficients:-0.00152096785439533"  "co-efficients:0.003202063763689"
 [5] "co-efficients:1.00413387236201"      "co-efficients:-
0.00337736694810797"
 [7] "co-efficients:0.000242861640138644"  "co-efficients:-
0.000474702125811495"
```

[2] http://www.cs.columbia.edu/~kathy/cs4701/documents/jason_svm_tutorial.pdf
[3] https://www.tidyverse.org/articles/2017/12/workflow-vs-script/

```
 [9] "co-efficients:-0.00119319004716302"  "co-efficients:0.106445052697048"
[1] "a0 co-efficient:-0.0815522639500845"
[1] "model accuracy %:0.863914373088685"
[1] "====ending iteration===="
[1] "====starting iteration===="
[1] "C Value:10"
[1] "------"
 Setting default kernel parameters
 [1] "co-efficients:-0.00090336712808322"  "co-efficients:-
0.000789103574949435"
….
[1] "====starting iteration===="
[1] "C Value:100"
[1] "------"
 Setting default kernel parameters
 [1] "co-efficients:-0.00100653481057611"  "co-efficients:-
0.00117290480611665"
 [3] "co-efficients:-0.00162619672236963"  "co-efficients:0.0030064202649194"
 [5] "co-efficients:1.00494056410556"      "co-efficients:-
0.00282594323043472"
 [7] "co-efficients:0.000260029507016313"  "co-efficients:-
0.000534955143494997"
 [9] "co-efficients:-0.00122837582291523"  "co-efficients:0.106363399527188"
[1] "a0 co-efficient:-0.081584921659538"
[1] "model accuracy %:0.863914373088685"
[1] "====ending iteration===="
```

The key take away here is that with very small values of C, the accuracy of the model is much lower. (83%) but the accuracy maxes out when any value of C higher than 0.1

Note: The computational power to churn higher values of C was observed to be much longer than lower Cs.

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`.

Why and How:

The purpose and methodology is the same as 2.2.2 . Just that, in this question's case, we are evaluating multiple kernels. The way I did that was to pull out the code I have for 2.2.2 and run that independently as a script. This allowed me to easily change the model type and iterate.

The other kernels provided some really different answers…

Kernel = Rbfdot (Radial Basis Gaussian) shows CYCLICAL accuracy in model for increasing Cs…
A snippet of the output, shows that very really really small C (1e-04) the accuracy was 95.2%, it drops to 95.1% for C=0.001, and then goes up (but not by much) to 95.7%, and drops against for C >20.

```
[1] "C Value:1e-04"
[1] "model accuracy %:0.952599388379205"
[1] "====ending iteration===="
[1] "====starting iteration===="
[1] "C Value:0.001"
[1] "model accuracy %:0.951070336391437"
[1] "====ending iteration===="
[1] "====starting iteration===="
[1] "C Value:0.01"
[1] "model accuracy %:0.952599388379205"
[1] "====ending iteration===="
[1] "====starting iteration===="
[1] "C Value:0.1"
[1] "model accuracy %:0.957186544342508"
[1] "====ending iteration===="
[1] "====starting iteration===="
[1] "C Value:1"
[1] "model accuracy %:0.957186544342508"
[1] "====ending iteration===="
[1] "====starting iteration===="
[1] "C Value:1"
[1] "model accuracy %:0.957186544342508"
[1] "====ending iteration===="
[1] "====starting iteration===="
[1] "C Value:10"
[1] "model accuracy %:0.954128440366973"
…
```

Tanhdot (Hyperbolic tangent kernel) pegs the accuracy to 72%

```
[1] "C Value:75"
 Setting default kernel parameters
[1] "model accuracy %:0.7217125382263"
```

For a polynomial kernel (polydot), the accuracy remains the same at 86.3% no matter what value of C…

```
[1] "C Value:75"
 Setting default kernel parameters
[1] "model accuracy %:0.863914373088685"
```

In summary, different models work differently with the data, and apparently highest accuracy emerges from the radial rfbdot kernel.

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`).

Purpose:

The objective of this question is to use the k-nearest neighbor model for a set of credit card approval data, and identify the right value of k , which yields the maximum accuracy in the model

How:

The code for this is very straightforward, and similar to the prior sections. A string of K values:

```
kayValues = c(0.1, 1, 2, 5, 10, 100,300);
```

then I created a function FindModelAccuracy_KNN() which takes all but one row as training set, and tests against that one row, and iterated across all 654 values of KNN. I continued using SCALE=TRUE like all the other examples since the values within the data set didn't at all look normalized.

Output and results:

[1] "====Question 2.2.3 : knn model to find an optimum k value===="

```
[1] "knn model accuracy for k-value: 0.1: 0.547400611620795"
[1] "knn model accuracy for k-value: 1: 0.814984709480122"
[1] "knn model accuracy for k-value: 2: 0.814984709480122"
[1] "knn model accuracy for k-value: 5: 0.851681957186544"
[1] "knn model accuracy for k-value: 10: 0.850152905198777"
[1] "knn model accuracy for k-value: 100: 0.836391437308868"
[1] "knn model accuracy for k-value: 300: 0.828746177370031"
```

It is clear that for k = 5, the kknn (k-nearest) model yields the maximum accuracy.
0.1 is way too low (and entirely hypothetical, how can you have less than 1 neighbors ☺)
However the value is really low
In other words when a new applicant is compared to 5 nearest neighbors in terms of similar attributes , the model provides (max) 85.1% accuracy to be able to receive a new applicant and predict the approve/not approve vote for that applicant correctly.

Question 3.1

Using the same data set (`credit_card_data.txt` or `credit_card_data-headers.txt`) as in Question 2.2, use the `ksvm` or `kknn` function to find a good classifier:

(a) usingcross-validation(dothisforthek-nearest-neighborsmodel;SVMisoptional);and

Purpose:
The goal of this exercise is to evaluate using a cross validation model (cv.knn) the accuracy of prediction of the result of credit card applications

How:
For this question, I created a similar function FindModelAccuracy_CVKNN() which I passed a k-fold value as well as the credit card tabular data. The useful part of about cv.knn model as expressed by Margaret Bolton in her TA talk on 5/21 is that it encompasses within itself the actual kknn model, and it wraps on top that model by using the k-fold value and carving out the data in k to do cross validation. One area that I was not able to grasp was why this model did NOT ask for the "k" for the k nearest neighbor, and only asks the "k" in the k-fold cross validation.

Output and results:

```
[1] "Model accuracy based on K-fold nearest neighbor with kFold = 10 is
0.547400611620795"
```

This model clearly does not perform well at all. I suspect it has something to do with the fact that it doesn't allow k nearest neighbor as input and is likely using some small value (which we have seen in 2.2.3 to yield really poor model accuracy)

Q3.1.b:
.. splitting the data into training, validation, and test data sets (pick either KNN or SVM; the other is optional)

Purpose:
The goal of this exercise is to break apart the data into training, validation and test data, and then proceed to train the data with the former, find out the accuracy of the model via the validation data and then test the accuracy using the test data. This is done mainly to weed out any unfair lift the model is getting because of the same random patterns show up in the training and the test data.

How:
First of all, I set the seed as a constant (set.seed(1)), so I can have reproducible output. Then, I broke apart the overall data into training , validation and test with a ration of 70:15:15. This was done by the vector manipulation and reversing indices etc…e.g.
After the Training Data indices were identified through a random sample method, the rest of the data was carved out by inversing the indices

```
restOfData <- creditData2[-creditDataTrainingIndices, ]
```

after which, the restOfData was further randomly sampled and carved out.

A large range of C values was used:

```
CValues = c(0.0001, 0.001, 0.01, 0.1, 1, 1, 10, 20, 25, 75)
```

And the exact KSVM model as applied earlier in Q2.2.1 was used (with the same methodology of coding the "create model" function and iterating through it using the C values.
The output of the prediction (guess) was done by using the validation data against the model..

```
guess <- predict(modelOutput, creditDataValidation[ ,1:10])
```

The accuracy was measured against the validation data:

```
accuracy[countt] = sum(guess == creditDataValidation[ ,11]) /
nrow(creditDataValidation)

A separate model was created for each C value, and then the best model was
selected based on the highest accuracy.

This model was then used to test against the final tranche of test data and
the result was output.
```

Output and results:

The output snippet from the R script for this question is here:

```
[1] "### best accuracy measured in validation round: 0.826530612244898"
 [1] "### Finding the best C value: 0.01"
…
 [1] "the performance of best model on test data = 0.878787878787879"
>
>
```

It shows that the best C value which yielded the highest accuracy model was 0.01 , and this accuracy, 87.8% is clean of random errors, since we use the training, validation and test data tranches to remove higher percentages of fit (accuracy) because of the inherent random errors within the one data set.