Northeastern University

ASSIGNMENT FRONT SHEET

Course Name: **ALY6040 Data Mining Applications**

Professor Name: **Nagadeepa Shanmuganathan**

Student Name: **Dong Quoc Tuong (Lukas)**

Student Class: **Fall 2019 CPS** **Term: Winter 2021**

Module 5: SVM Classifier with Heart
Completion Date: Feb 28t **Due Time:12:00am**

Statement of Authorship

I confirm that this work is my own. Additionally, I confirm that no part of this coursework, except where clearly quoted and referenced, has been copied from material belonging to any other person e.g. from a book, handout, another student. I am aware that it is a breach of Northeastern University's regulations to copy the work of another without clear acknowledgement and that attempting to do so renders me liable to disciplinary procedures. To this effect, I have uploaded my work onto Turnitin and have ensured that I have made any relevant corrections to my work prior to submission.

☒ **Tick here** to confirm that your paper version is identical to the version submitted through Turnitin

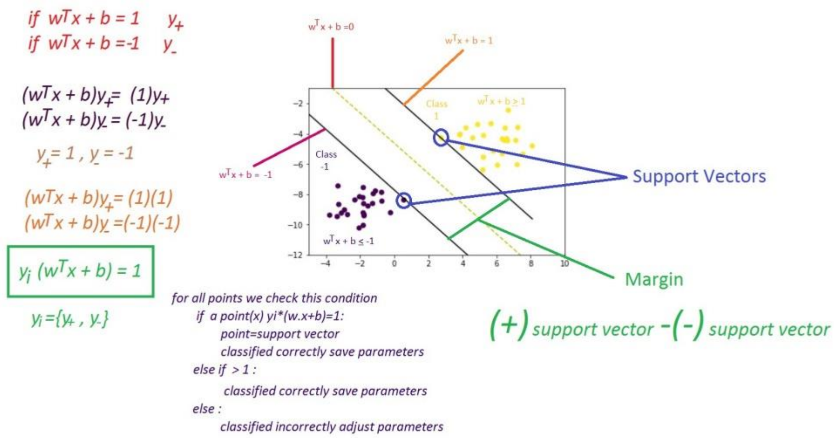
Data Import

The dataset we will be working on is Heart disease which consists of 300 inputs across 14 numeric attributes. The first 13 variables are independent variables that are used to predict dependent variable at the 14th index. Here is a look at the first few lines of the dataset after we loaded. From a respective point of view, it seems like 14th variable only include binary outcome of 0 and 1, meaning that this is likely a Classification exercise. So we are going to build the model of to classify whether a patient is suffering from any heart diseases or not

```
> head(heart_df)
  V1 V2 V3  V4  V5 V6 V7  V8 V9 V10 V11 V12 V13 V14
1 63  1  1 145 233  1  2 150  0 2.3   3   0   6   0
2 67  1  4 160 286  0  2 108  1 1.5   2   3   3   1
3 67  1  4 120 229  0  2 129  1 2.6   2   2   7   1
4 37  1  3 130 250  0  0 187  0 3.5   3   0   3   0
5 41  0  2 130 204  0  2 172  0 1.4   1   0   3   0
6 56  1  2 120 236  0  0 178  0 0.8   1   0   3   0
> 
```

Packages loading

Now we will load the "caret" package first to use the SVM model later on. Support Vector Machine (SVM) is a supervised machine learning algorithm popular for classification and regression problems By using Kernel trick, SVM is able to find optimal boundary between the possible outputs that you defined after transforming the data. SVM is the go-to classification algorithm like this project thanks to the non-linear kernel approach that does not require the algorithm to calculate a straight line. Thus capturing much more complex relationships between data points without the need to perform difficulty transformation on your own. (Gandhi, 2018) However, such cutting edge aspect means more computational power and training time. Here is a picture of how SVM makes it prediction:



Data Slicing

Then proceed to slice the dataset according to the 0.7/0.3 ratio with createDataPartition() for the train and test set. But one must remember to set.seed() so that your work is replicable and yield identical result when you rerun it again in another platform. This is the dimension of the train (210) and test (90) set

```
> dim(training); dim(testing);
[1] 210  14
[1] 90 14
```

Preprocessing & Training

In real life, dataset is often dirty and not as clean as one might hope, and thus, we need to clean the dataset first before training it in the model. We received NA after applying the anyNA() method because it indicates the dataset does not have any missing data .

```
> anyNA(heart_df)
[1] FALSE
```

Next we will standardize the data so that it does not have any issue with scaling later on. The we can do this by scaling it to 0,1 and convert the to the dependent variable into categorical using factors()

Training the SVM model and test set prediction

Now we get to the training part. But before constructing the training model, we need to use trainControl() to control the computational nuances of train() method with 3 parameters inside. We set the “method” as repeated cross validation (repeatedcv), number =10, repeats =3. The outcome is a list that we will pass through the train() after setting up the seed.

For the training model, we set the parameter as follow:

- method = "svmLinear"
- trControl=trctrl
- preProcess = c("center", "scale")
- tuneLength = 10

Below is the result of the trained SVM model. It is a linear model which is why it just tested at value “C”=1

```
Support Vector Machines with Linear Kernel

210 samples
 13 predictor
  2 classes: '0', '1'

Pre-processing: centered (13), scaled (13)
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 189, 189, 188, 189, 189, 190, ...
Resampling results:

  Accuracy   Kappa
  0.7677345  0.5318105

Tuning parameter 'C' was held constant at a value of 1
>
```

The next step is to use predict() to pass the test set through the model, and this is the prediction outcome for the test set

```
> test_pred <- predict(svm_Linear, newdata = testing)
> test_pred
 [1] 1 1 0 1 1 1 0 0 1 1 0 0 0 1 1 0 1 1 1 0 0 1 1 0 0 1 0 1 0 0 0
[32] 0 0 1 0 1 1 0 1 1 1 0 0 0 1 0 0 1 1 0 0 1 1 1 0 1 0 1 1 0 0 0
[63] 0 1 1 0 0 1 0 0 0 0 1 1 1 0 0 0 1 1 0 0 1 1 0 1 1 1 0 1
Levels: 0 1
```

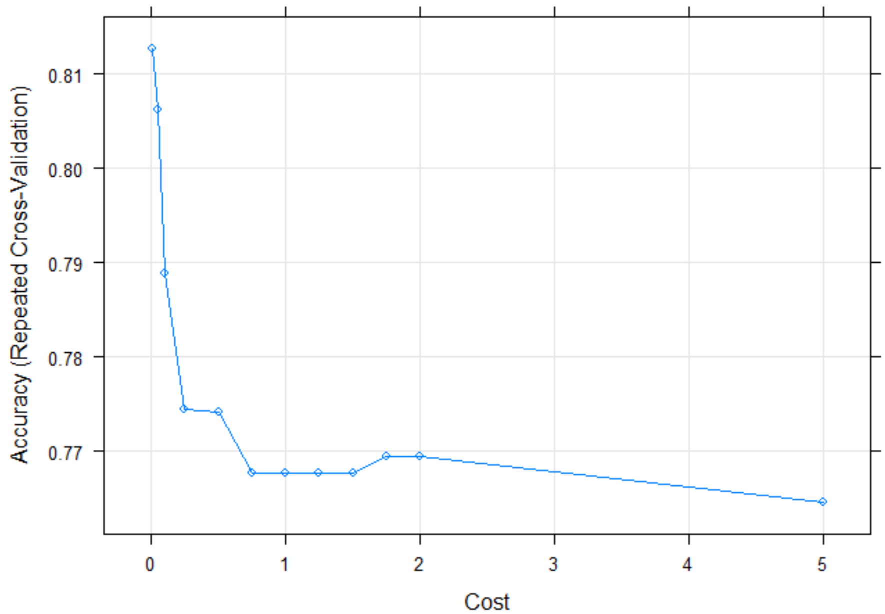
<u>Model's accuracy</u>

The confusion matrix indicates that the accuracy rate is about 83.3%

```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 40  5
         1 10 35

               Accuracy : 0.8333
                 95% CI : (0.74, 0.9036)
    No Information Rate : 0.5556
    P-Value [Acc > NIR] : 2.25e-08

                  Kappa : 0.6667

 Mcnemar's Test P-Value : 0.3017

            Sensitivity : 0.8000
            Specificity : 0.8750
         Pos Pred Value : 0.8889
         Neg Pred Value : 0.7778
             Prevalence : 0.5556
         Detection Rate : 0.4444
   Detection Prevalence : 0.5000
      Balanced Accuracy : 0.8375

       'Positive' Class : 0
```

We can also do some customizations for selecting C value (cost) in Linear Classifier by inserting values in grid search with expand.grid(). The plot below demonstrated that the one with the best accuracy is C=0.01, which yields accuracy rate of 0.812.

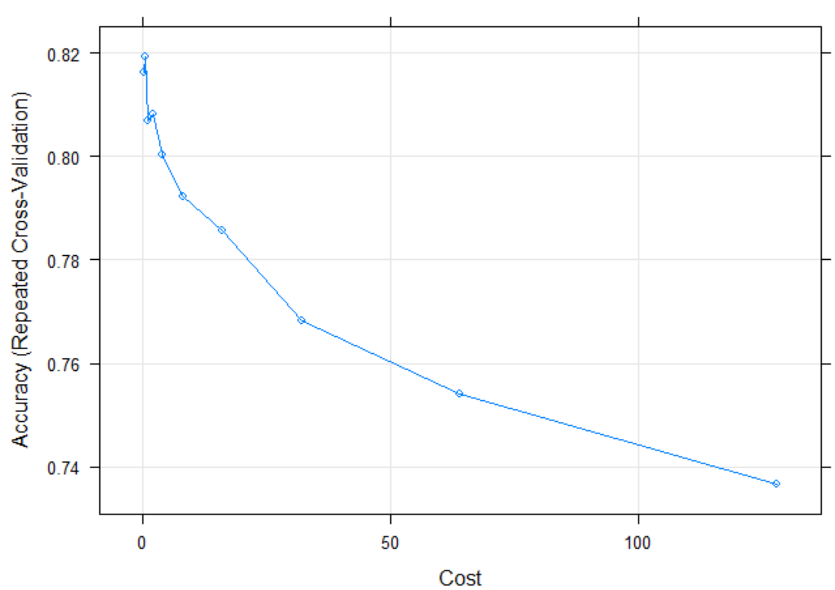
```
Support Vector Machines with Linear Kernel

210 samples
 13 predictor
  2 classes: '0', '1'

Pre-processing: centered (13), scaled (13)
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 189, 189, 188, 189, 189, 190, ...
Resampling results across tuning parameters:

  C     Accuracy   Kappa
  0.00        NaN        NaN
  0.01  0.8126768  0.6197440
  0.05  0.8062554  0.6074071
  0.10  0.7888528  0.5732929
  0.25  0.7744733  0.5453734
  0.50  0.7741631  0.5452208
  0.75  0.7677345  0.5317827
  1.00  0.7677345  0.5318105
  1.25  0.7677345  0.5318105
  1.50  0.7677345  0.5318105
  1.75  0.7694012  0.5350138
  2.00  0.7694012  0.5350138
  5.00  0.7646320  0.5254978

Accuracy was used to select the optimal model using the
 largest value.
The final value used for the model was C = 0.01.
> 
```



SVM Classifier using Non-Linear Kernel

Using the Non-linear Kernel, we can build a Radial Basis Function. (Raschka, 2020) In Radial Kernel, we need to select proper value cost of Cost “C” parameter and “sigma” parameter. From the table, the final sigma value is 0.057 with C= 0.5

```
Support Vector Machines with Radial Basis Function Kernel

210 samples
 13 predictor
  2 classes: '0', '1'

Pre-processing: centered (13), scaled (13)
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 189, 189, 189, 189, 189, 190, ...
Resampling results across tuning parameters:

  C       Accuracy   Kappa
    0.25  0.8163203  0.6289507
    0.50  0.8192713  0.6341953
    1.00  0.8067965  0.6096803
    2.00  0.8081602  0.6134678
    4.00  0.8002309  0.5984699
    8.00  0.7922078  0.5820086
   16.00  0.7857792  0.5694564
   32.00  0.7683117  0.5347809
   64.00  0.7540115  0.5058719
  128.00  0.7366739  0.4712769

Tuning parameter 'sigma' was held constant at a value of 0.05705594
Accuracy was used to select the optimal model using the
 largest value.
The final values used for the model were sigma = 0.05705594 and
 C = 0.5.
```

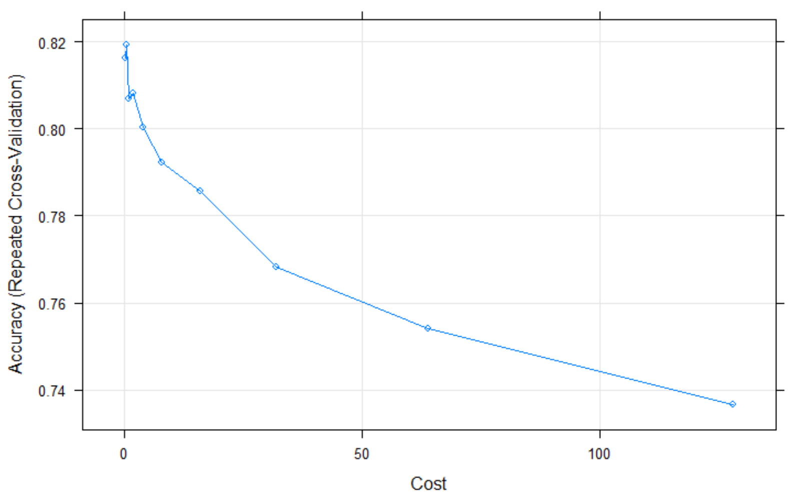


For predicting, we will use predict() with model's parameters as svm_Radial & newdata= testing. We are getting an accuracy of 86.67%. So, in this case with values of C=0.5 & sigma= 0.057, we are getting good results.

```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 42  4
         1  8 36

               Accuracy : 0.8667
                 95% CI : (0.7787, 0.9292)
    No Information Rate : 0.5556
    P-Value [Acc > NIR] : 2.452e-10

                  Kappa : 0.7327

 Mcnemar's Test P-Value : 0.3865

            Sensitivity : 0.8400
            Specificity : 0.9000
         Pos Pred Value : 0.9130
         Neg Pred Value : 0.8182
             Prevalence : 0.5556
         Detection Rate : 0.4667
   Detection Prevalence : 0.5111
      Balanced Accuracy : 0.8700

       'Positive' Class : 0
```

Let 's do it one last time again with different values of C and sigma. We will use gridsearch() once agaiun and have the best values of sigma =0.025 & C=0.1

```
Accuracy was used to select the optimal model using the largest value.
The final values used for the model were sigma = 0.025 and C = 0.1.
>
```



The final model with svm_Radial_Grid classifier yields 88.89%. So, it shows Radial classifier is giving better results as compared to Linear classifier even after tuning it. That is what exactly what we are aiming for.

```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 46  6
         1  4 34

               Accuracy : 0.8889
                 95% CI : (0.8051, 0.9454)
    No Information Rate : 0.5556
    P-Value [Acc > NIR] : 7.675e-12

                  Kappa : 0.7739

 Mcnemar's Test P-Value : 0.7518

            Sensitivity : 0.9200
            Specificity : 0.8500
         Pos Pred Value : 0.8846
         Neg Pred Value : 0.8947
             Prevalence : 0.5556
         Detection Rate : 0.5111
   Detection Prevalence : 0.5778
      Balanced Accuracy : 0.8850

       'Positive' Class : 0
```

References

Gandhi, R. (2018, July 5). *Support Vector Machine — Introduction to Machine Learning Algorithms*. Towards Data Science. https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

Raschka, S. (2020, June). *How to Select Support Vector Machine Kernels*. KDnuggets. https://www.kdnuggets.com/2016/06/select-support-vector-machine-kernels.html