Text Classification and Support Vector Machine

Promothesh Chatterjee*

Support Vector Machines

SVM is among the more popular supervised learning algorithm that is used for classification and prediction. SVM is a popular choice for many data science competitions.

Support Vector machines try to find the optimal hyperplane (which is simply a line in two-dimensional space, a plane in three-dimensional space and a hyperplane in multidimensional spaces) that maximizes the margin between the two classes.In simple language, we want to find an optimal separating boundary that can separate elements to different classes. Take a look at figure 1 (from Nguyen 2017)

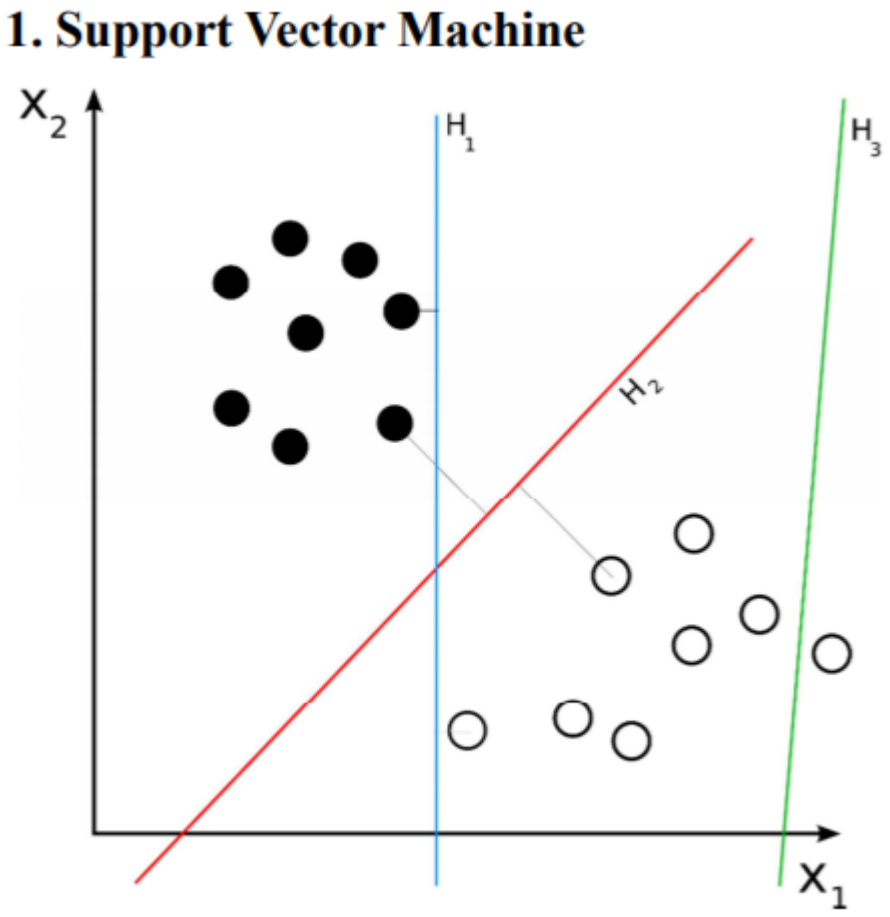
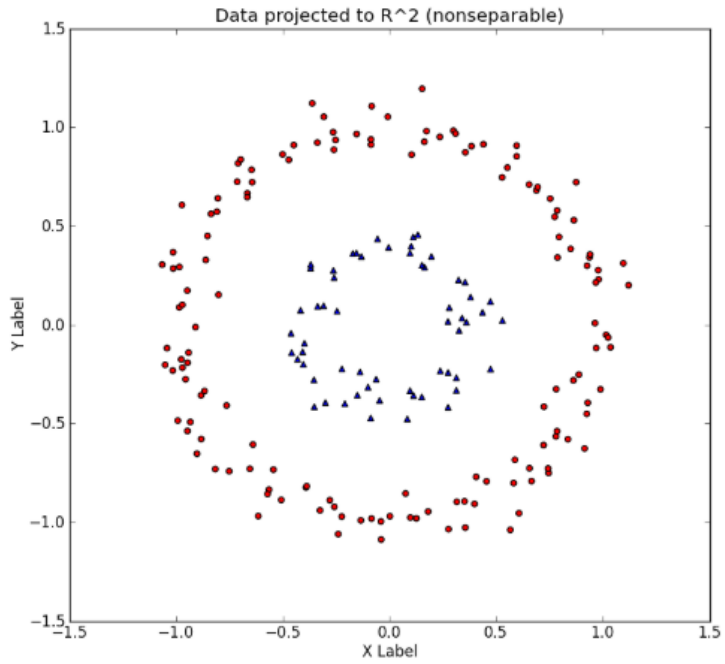


Figure 1. *Separating hyperplanes.*

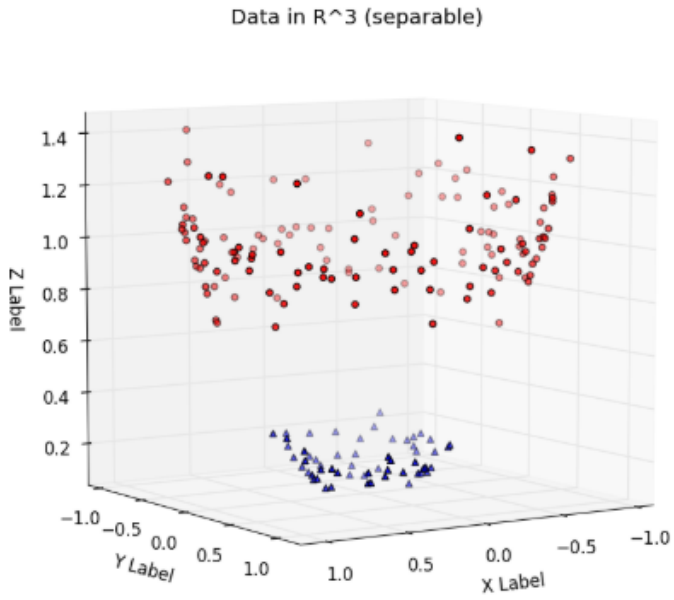
Essentially, one begins with a random hyperplane and tests for the distance of the nearest datapoints with the hyperplane. These nearest datapoint observations that support hyperplane on either sides are called as support vectors. The distance between the hyperplane and the support vectors is known as the margin. Classification of data in SVM is done such that the distance between the hyperplane and the support vectors (margin) is maximized.

In the case above, a linear separation between the classes was easy to see and implement, however in real life the cases are not so straightforward. For those complex non-linear instances we use the kernel trick. We utilize kernel trick to transform data into another dimension whereby one can easily draw the hyperplane between different classes. For visualization of some popular kernel tricks, see https://datafreakankur.com/machine-learning-kernel-functions-3d-visualization/

Let's take an example to understand this better. Assume there is complex classification case as shown below:



Obviously, a linear hyperplane is not going to work in this case. Kernel trick allows us to project this to an n-dimensional space which allows for appropriate hyperplane to separate the classes.



Resolving a situation like this involves grappling with two issues. First, in terms of computational resources it is very expensive to map each datapoint. Second, it is not easy to figure out which is the appropriate function that will make the data linearly separable as there are plethora of possibilities. How can we resolve these two issues? The way SVM is formulated mathematically, we don't need to know the mapping function

but as long as we can obtain a dot product between data points, we would be able to generate a separating hyperplane in a very high dimensional space. The functions that provide the value of dot products are called as kernel functions. The kernel functions allow us compute a dot product in infinite dimensional feature space without actually mapping points to that space. This property of kernels is sometimes referred to as the kernel trick. The use of kernels therefore makes SVMs computationally efficient because each data point doesn't need to be mapped to a higher dimensional space. As users of SVM, we do not need to know the function that will make data linearly separable. All that we require is a set of pre-existing kernels that can be used in the analysis.

R Implementation of SVM

So broadly speaking, we can fit two kinds of kernels when implementing SVM- linear and non-linear.

For linear kernel, the only hyperparameter that we have to provide is Cost (or C). Cost refers to tolerance for wrong classification. For high values of C, the algorithm will select a hyperplane with smaller margin (in SVM we are trying to maximize margins and minimize classification errors) though classifies data points accurately. On the other hand, if the value of C is kept very low, SVM will choose a hyperplane with bigger margins but have many more misclassification errors.

Linear SVM Let's look at a linear classifier on our hotel reviews dataset, then in the next section we will look at a non-linear classifier.

Let's first read in the dataset and do some preprocessing.

```
library(tidyverse)
library(e1071)
library(caret)
library(quanteda)
library(irlba)
library(randomForest)

# Load up the .CSV data and explore in RStudio.
hotel_raw <- read_csv("Data/hotel-reviews.csv")
set.seed(1234)
hotel_raw<-hotel_raw[sample(nrow(hotel_raw), 5000), ]# take a small sample

summary(hotel_raw)
```

```
##    User_ID          Description        Browser_Used       Device_Used
##  Length:5000        Length:5000        Length:5000        Length:5000
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##  Is_Response
##  Length:5000
##  Class :character
##  Mode  :character
```

Let's use library caret (by the way, that's an acronym for Classification And Regression Training) to create our training and test datasets.

```
set.seed(12345)
hotel_record <- createDataPartition(hotel_raw$Is_Response, times = 1,
                                    p = 0.7, list = FALSE)
hotel_train <- hotel_raw[hotel_record,]
hotel_test <- hotel_raw[-hotel_record,]
```

Run the usual quanteda code for preprocessing and creating DTM

```
library(quanteda)
hotel_train_token2 <- tokens(hotel_train$Description, what = "word",
                         remove_numbers = TRUE, remove_punct = TRUE,
                         remove_symbols = TRUE) %>%
                         tokens_tolower()

hotel_train_dfm<-hotel_train_token2 %>%
  tokens_remove(stopwords(source = "smart")) %>%
  tokens_wordstem() %>%
  dfm() %>%
    dfm_trim( min_termfreq = 10, min_docfreq = 2) %>%
    dfm_tfidf()
```

Let's perform LSA to reduce the dimensions of the dataset, append the feature data frame with class labels and add a variable for length of review.

```
# Perform SVD. Specifically, reduce dimensionality down to 300 columns
# for our latent semantic analysis (LSA).
library(irlba)
train_lsa <- irlba(t(hotel_train_dfm), nv = 300, maxit = 600)

train_svd <- data.frame(Label = hotel_train$Is_Response, ReviewLength= nchar(hotel_train$Description),t
```

Cross-validation steps

```
# Use caret to create stratified folds for 10-fold cross validation repeated
# 2 times (i.e., create 20 random stratified samples)
set.seed(48743)
cv.folds <- createMultiFolds(train_svd$Label, k = 10, times = 2)
# basically this will create 20 random stratified samples

cv.cntrl <- trainControl(method = "repeatedcv", number = 10,
                         repeats = 2, index = cv.folds)
```

Let's train our model now:

```
library(doSNOW)

# Time the code execution
start.time <- Sys.time()

# Create a cluster to work on 4 logical cores.
cl <- makeCluster(3, type = "SOCK") # Simplistically, we are asking computer to run
# 3 instances of Rstudio for processing. I have total of 4 cores in my computer
registerDoSNOW(cl) # register the instance

SVM_Linear <- train(Label ~ ., data = train_svd, method = "svmLinear",
                    preProcess = c("center", "scale"),
                    trControl = cv.cntrl, tuneLength = 7)


# Processing is done, stop cluster.
on.exit(stopCluster(cl))
# Total time of execution
total.time <- Sys.time() - start.time
total.time
```

```
## Time difference of 4.212443 mins
```

Let's look at the predictions of our training dataset for linear kernel.

```
train_svd$Label<-as.factor(train_svd$Label)
preds <- predict(SVM_Linear, train_svd)
confusionMatrix(preds, train_svd$Label)
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  happy not happy
##   happy      2335       227
##   not happy    95       844
##
##                Accuracy : 0.908
##                  95% CI : (0.898, 0.9174)
##     No Information Rate : 0.6941
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7757
##
##  Mcnemar's Test P-Value : 2.87e-13
##
```

```
##             Sensitivity : 0.9609
##             Specificity : 0.7880
##          Pos Pred Value : 0.9114
##          Neg Pred Value : 0.8988
##              Prevalence : 0.6941
##          Detection Rate : 0.6670
##    Detection Prevalence : 0.7318
##       Balanced Accuracy : 0.8745
##
##        'Positive' Class : happy
##
```

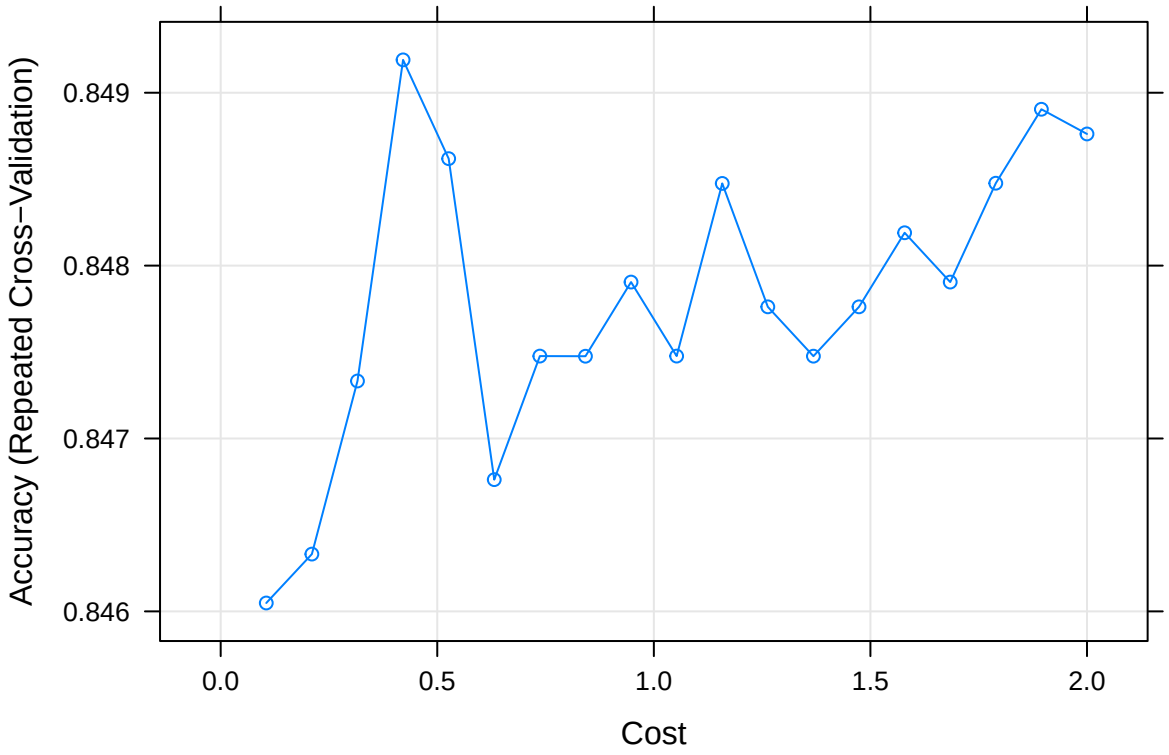
The default value of C is 1 when we build the SVM linear classifier using caret. We can compute SVM models using different values of C and choose the one that gives best accuracy for predictions. You can try the code below:

```
# Time the code execution
start.time <- Sys.time()

# Create a cluster to work on 4 logical cores.
cl <- makeCluster(3, type = "SOCK") # Simplistically, we are asking computer to run
# 3 instances of Rstudio for processing. I have total of 4 cores in my computer
registerDoSNOW(cl) # register the instance

set.seed(12345)
model <- train(
  Label ~., data = train_svd, method = "svmLinear",
  trControl = cv.cntrl,
  tuneGrid = expand.grid(C = seq(0, 2, length = 20)),
  preProcess = c("center","scale")
  )


# Processing is done, stop cluster.
on.exit(stopCluster(cl))
# Total time of execution
total.time <- Sys.time() - start.time
total.time
```

```
## Time difference of 54.18905 mins
```

Plot model accuracy vs different values of Cost

```
plot(model)
```



Non-linear SVM

We can build non-linear SVM classifiers with caret package too (some popular non-linear SVMs are polynomial kernel or radial kernel). Caret automatically chooses the appropriate model tuning parameters and maximizes the model accuracy.

Let's run the code for radial basis kernel (again, without getting into the underlying math, we let caret generate the optimal sigma and C values). We can easily use polynomial kernel by specifying method=svmPoly in the code below.

```
# Time the code execution
start.time <- Sys.time()

# Create a cluster to work on 4 logical cores.
cl <- makeCluster(3, type = "SOCK") # Simplistically, we are asking computer to run
# 3 instances of Rstudio for processing. I have total of 4 cores in my computer
registerDoSNOW(cl) # register the instance

set.seed(12345)
SVM_RB <- train(
```

```
  Label ~., data = train_svd, method = "svmRadial",
  trControl = cv.cntrl,
  preProcess = c("center","scale"),
  tuneLength = 7
  )

# Processing is done, stop cluster.
on.exit(stopCluster(cl))
# Total time of execution
total.time <- Sys.time() - start.time
total.time
```

```
## Time difference of 9.474481 mins
```

Print the best tuning parameter sigma and C that maximizes model accuracy

```
SVM_RB$bestTune
```

```
##         sigma C
## 4 0.004609185 2
```

Let's look at the predictions of our training dataset for non-linear kernel.

```
preds <- predict(SVM_RB, train_svd)
confusionMatrix(preds, train_svd$Label)
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  happy not happy
##   happy      2419       119
##   not happy    11       952
##
##                Accuracy : 0.9629
##                  95% CI : (0.9561, 0.9689)
##     No Information Rate : 0.6941
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.91
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9955
##             Specificity : 0.8889
##          Pos Pred Value : 0.9531
##          Neg Pred Value : 0.9886
##              Prevalence : 0.6941
```

```
##          Detection Rate : 0.6909
##    Detection Prevalence : 0.7249
##       Balanced Accuracy : 0.9422
##
##        'Positive' Class : happy
##
```

Testing the SVM Model Now let's verify the predictive ability of the model in the test holdout data. Let's first preprocess the test data.

```
hotel_test_token <- tokens(hotel_test$Description, what = "word",
                       remove_numbers = TRUE, remove_punct = TRUE,
                       remove_symbols = TRUE, remove_hyphens = TRUE) %>%
                       tokens_tolower()

hotel_test_dfm<-hotel_test_token %>%
  tokens_remove(stopwords(source = "smart")) %>%
  tokens_wordstem() %>%
  dfm() %>%
    dfm_trim( min_termfreq = 10, min_docfreq = 2) %>%
    dfm_tfidf()
```

We need to ensure that our test data has the exact same format as our training data, we can use quanteda function dfm_match for that.

```
hotel_test_dfm <- dfm_match(hotel_test_dfm, features = featnames(hotel_train_dfm))

hotel_test_matrix <- as.matrix(hotel_test_dfm)
hotel_test_dfm
```

```
## Document-feature matrix of: 1,499 documents, 2,211 features (98.00% sparse) and 0 docvars.
##        features
## docs        hotel      rate romant  boston agre delux suit bathtub shower half
##   text1 0.6434684 0.9882809      0 0          0     0    0       0      0    0
##   text2 0         0              0 0          0     0    0       0      0    0
##   text3 0         0              0 0          0     0    0       0      0    0
##   text4 0.5362237 0              0 0          0     0    0       0      0    0
##   text5 0         0              0 1.49456    0     0    0       0      0    0
##   text6 0.1072447 0              0 0          0     0    0       0      0    0
## [ reached max_ndoc ... 1,493 more documents, reached max_nfeat ... 2,201 more features ]
```

As with TF-IDF, we will need to project new data (e.g., the test data) into the SVD semantic space. The following code illustrates how to do this. This is based of "Learning Analytics in R with SNA, LSA, and MPIA by Fridolin Wild", I will be happy to share the book chapter if you are interested.

```
sigma.inverse <- 1 / train_lsa$d # taking the tranpose of the singular matrix is # same as calculating
u.transpose <- t(train_lsa$u)#transpose of the term matrix
test_svd <- t(sigma.inverse * u.transpose %*% t(hotel_test_dfm))
test_svd<-as.matrix(test_svd)
```

Lastly, we can now build the test data frame to feed into our trained machine learning model for predictions. Let's add Label and ReviewLength.

```
test_svd <- data.frame(Label = hotel_test$Is_Response, ReviewLength= nchar(hotel_test$Description), tes
```

Now we can make predictions on the test data set using our trained SVM.

```
test_svd$Label<-as.factor(test_svd$Label)
preds <- predict(SVM_RB, test_svd)
confusionMatrix(preds, test_svd$Label)
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  happy not happy
##   happy       984       177
##   not happy    57       281
##
##                Accuracy : 0.8439
##                  95% CI : (0.8245, 0.8619)
##     No Information Rate : 0.6945
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.603
##
##  Mcnemar's Test P-Value : 7.294e-15
##
##             Sensitivity : 0.9452
##             Specificity : 0.6135
##          Pos Pred Value : 0.8475
##          Neg Pred Value : 0.8314
##              Prevalence : 0.6945
##          Detection Rate : 0.6564
##    Detection Prevalence : 0.7745
##       Balanced Accuracy : 0.7794
##
##        'Positive' Class : happy
##
```

As you can see, there seems to be some model performance mismatch. The training set vastly outperforms test set for the non-linear kernel. Let's also quickly see how the test set performs on linear kernel.

```
test_svd$Label<-as.factor(test_svd$Label)
preds <- predict(SVM_Linear, test_svd)
confusionMatrix(preds, test_svd$Label)
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  happy not happy
##   happy       959       148
##   not happy    82       310
##
##                Accuracy : 0.8466
##                  95% CI : (0.8273, 0.8645)
##     No Information Rate : 0.6945
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6232
##
##  Mcnemar's Test P-Value : 1.819e-05
##
##             Sensitivity : 0.9212
##             Specificity : 0.6769
##          Pos Pred Value : 0.8663
##          Neg Pred Value : 0.7908
##              Prevalence : 0.6945
##          Detection Rate : 0.6398
##    Detection Prevalence : 0.7385
##       Balanced Accuracy : 0.7990
##
##        'Positive' Class : happy
##
```

The performance is definitely not as good in test set for both linear as well non-linear kernel. Some of the common causes of this mismatch are:

1. Model Overfitting
2. Unrepresentative Data Sample
3. Stochastic Nature of the Algorithm

See, this web article for further details: https://machinelearningmastery.com/the-model-performance-mismatch-problem/