Spotlight: SMS Spam Classifier

Swagat Malla

Introduction

Every day we get a lot of messages on our phones. Most of the times they are genuine messages from our friends, family, and other contacts, but we also do receive unwanted spam messages.The concept of spam is rather subjective; however, we can agree that most of the times these are advertisements, scams, money making schemes, pornography, and so on.

In this project, we use the *SMS Spam Collection Data Set*– made available by the University of California, Irvine Machine Learning Repository– that contains 5574 SMS messages that are classified as *spam* or *ham* (genuine message). A quick look at the documentation tells us that the texts were collected from people in the UK and Singapore.

Using the texts, we will generate features, which we will use to train and evaluate our classification model. We will first use Lasso regularization technique for our binomial logistic regression.

Data Tidying

To begin with, we have five columns in total. The first two are straight forward- `v1` is the classification column and `v2`contains the texts. The last three columns, however, contain a lot of blank entries, they do not seem to add a lot of value to the data set. So, we get rid of them and rename the rest of the attributes appropriately. There also seem to be duplicated instance where the texts are repeated, so we leave them out. Furthermore, we convert our target variable `type` into a factor.

Feature Engineering

To make a machine learning model, we will need attributes that describe each observation of SMS text, so let's generate some features from the raw texts. To achieve this, we can use different kinds of counts. For instance, every text is composed of characters such as alphabets, numbers, signs, etc; we can count them! Here I have included counts of characters, words, sentences, numbers, and upper case letters. Going through the raw texts, we can see few instances where users have used emoticons (special characters to represent facial characters such as :-)). There are also texts that include currency symbols like `£`, which makes sense given that a lot of the users were British. Thus, it is reasonable to guess that there might be some texts with dollar signs `$` as some users were from Singapore. For emoticons and currency symbols, we can use regular expressions to find the case we are looking for. For emoticons, `qdapRegex` library (which contains a dictionary of regular expressions) comes in handy. The process for creating few of the variables is shown below.

```
#give a unique id for each text
msg_tbl <- msg_tbl %>%
  mutate(id = 1:nrow(msg_tbl))

#count of total characters, numbers, uppercase characters, and sentences
msg_tbl <- msg_tbl %>%
```

```
  mutate(char_count  = nchar(text),
         num_count = str_count(text, "[0-9]"),
         upperCase_count = str_count(text, "[A-Z]"))

sentence_tbl <- msg_tbl %>%
  unnest_tokens(word, text, token = "sentences") %>%
  group_by(id) %>%
  summarize(sentences_count = n())

msg_tbl <- msg_tbl %>%
  left_join(sentence_tbl, by = "id")
```

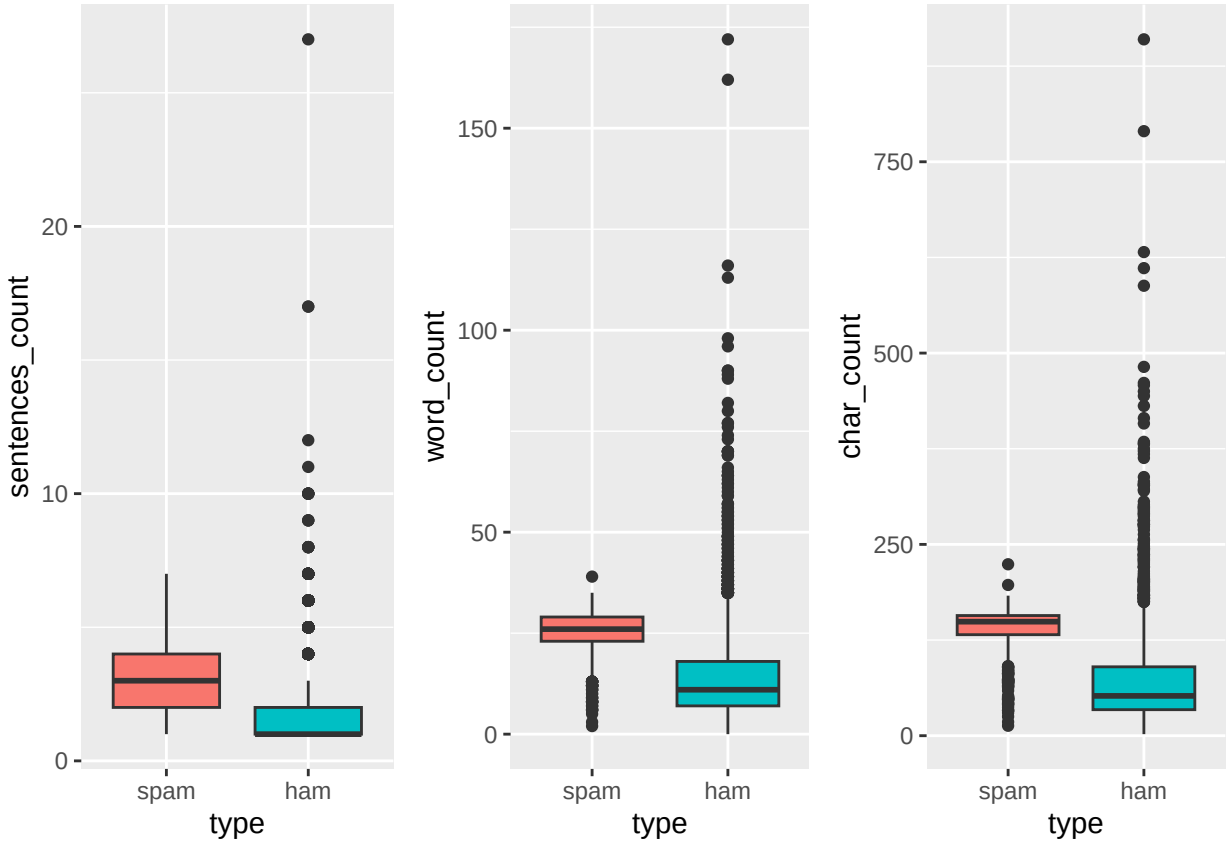
Exploring Features

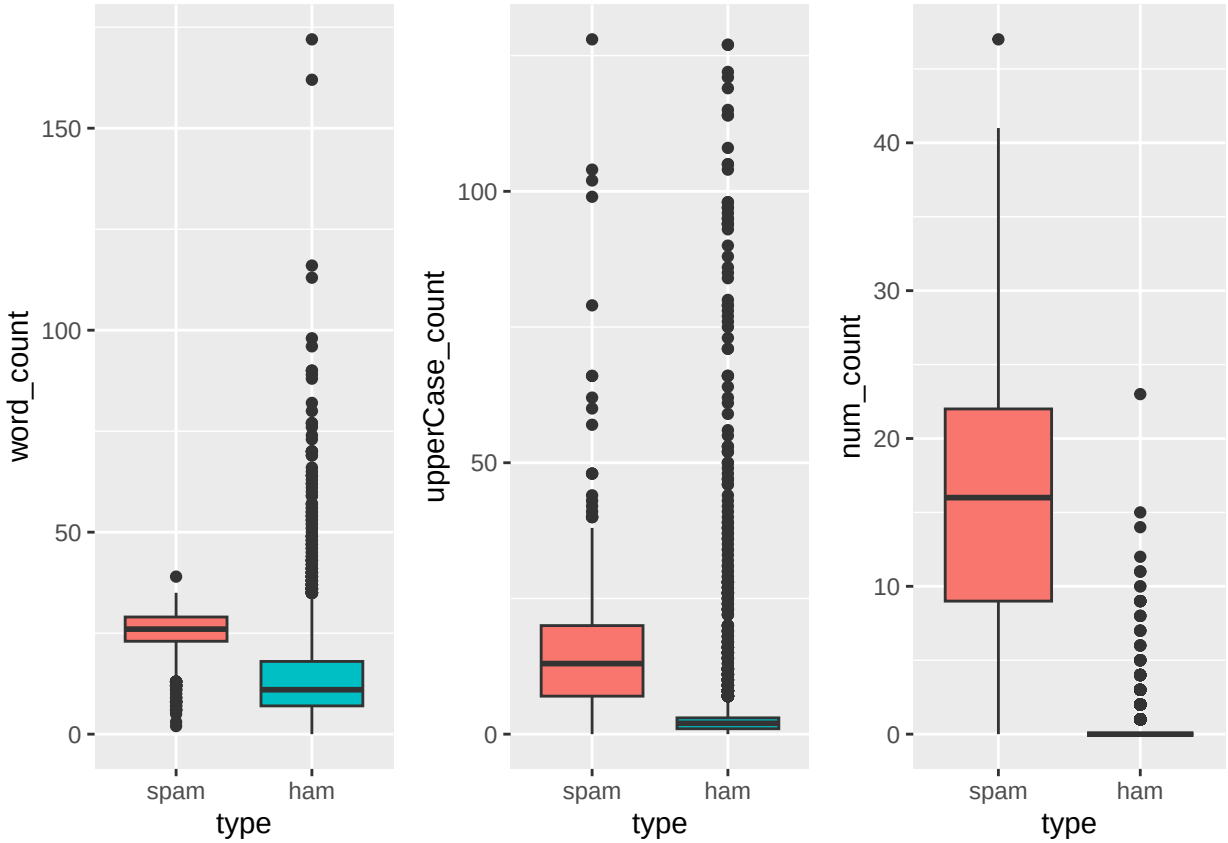
Going over the summary of our data set, we see that there is only one level (0) for `has_dollar`, meaning none of the texts had a dollar sign for some reason. It is safe to just remove it.

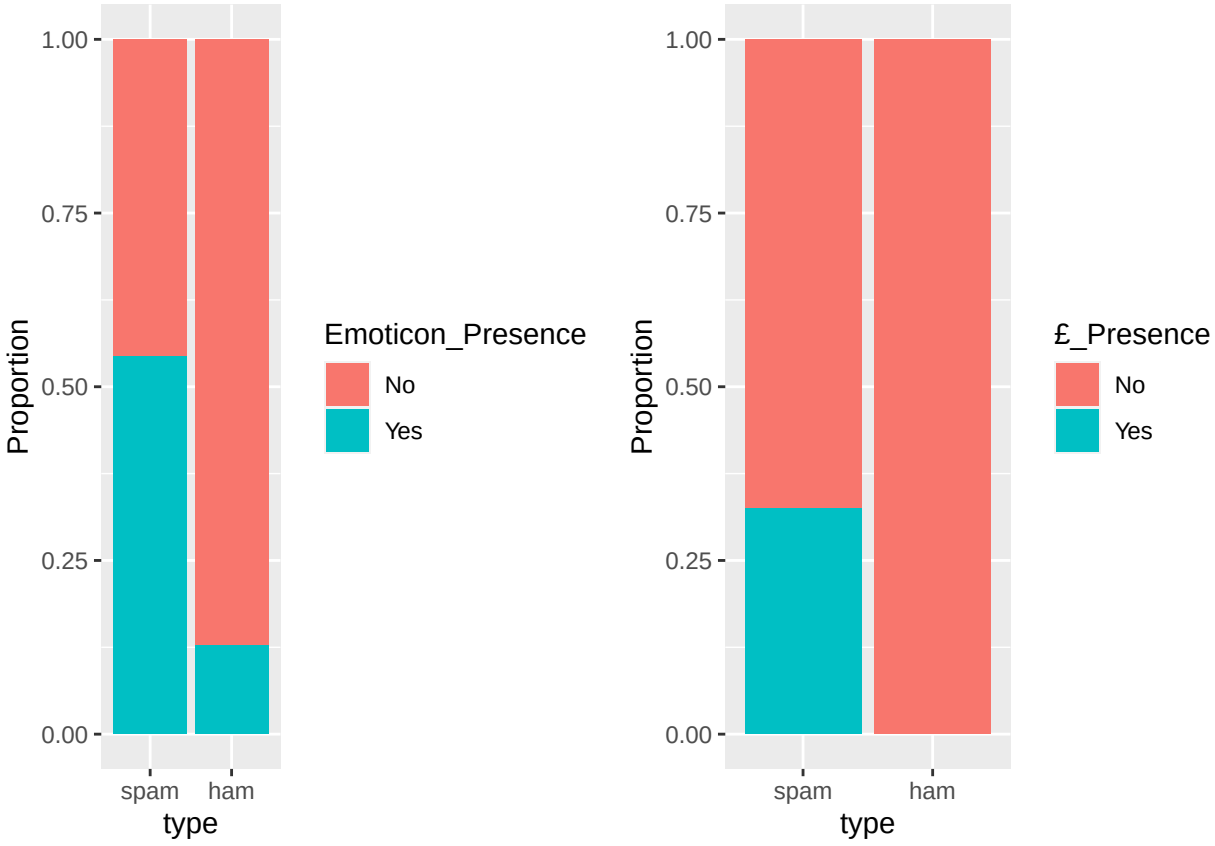
Calculating the proportions of each type of texts, we find that our data set is unbalanced. There are more observations of ham texts than spam ones; 87.4% of texts are ham, and 12.6% of them are spam. We need to be keep this in mind when we evaluate the accuracy of the model; we need to make sure the sensitivity and specificity are balanced.

```
## # A tibble: 2 x 3
##   type      n proportion
##   <fct> <int>      <dbl>
## 1 spam    653      0.126
## 2 ham    4516      0.874
```

Looking at our plots for the counts, we can easily see that all of the median counts are higher for spam texts although are outliers for ham texts. For our binary variables, spam texts tend to have emoticons and `£` more so than real texts.


sentences_count
20
10
0
spam
ham
type
word_count
150
100
50
0
spam
ham
type
char_count
750
500
250
0
spam
ham
type


word_count
150
100
50
0
spam
ham
type
upperCase_count
100
50
0
spam
ham
type
num_count
40
30
20
10
0
spam
ham
type



Using Lasso Regularization Technique

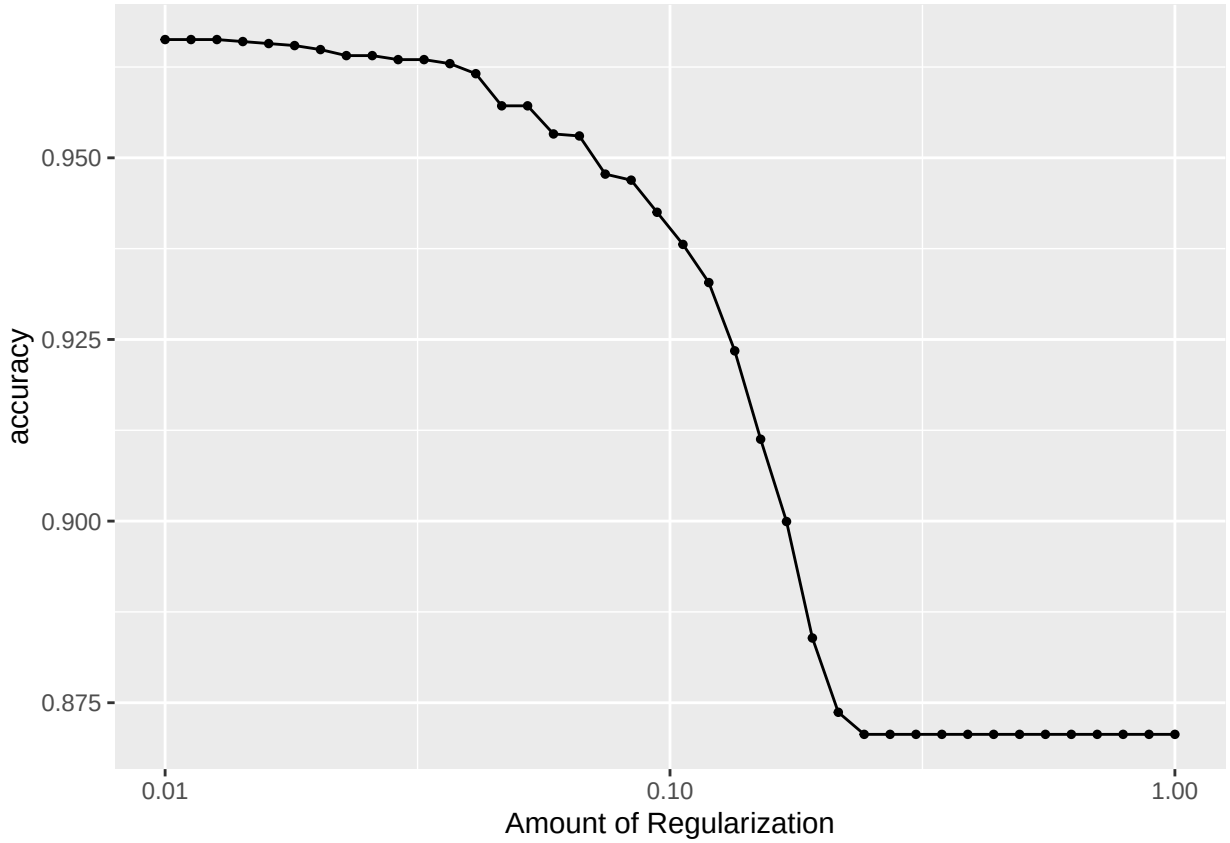
We created seven different predictors for text type in the previous section. Even though we saw that the texts do differ based on these attributes, we do not really know if all predictors are equally important.To this end, we can make use of a regularization technique to select the variables that are more important.

In Lasso regularization, we minimize the sum of λ times the absolute value of all of our coefficients. Here, λ is the penalty term that we will tune by using cross-validation with 10 folds and varying *lamda* from 0.01 to 1.

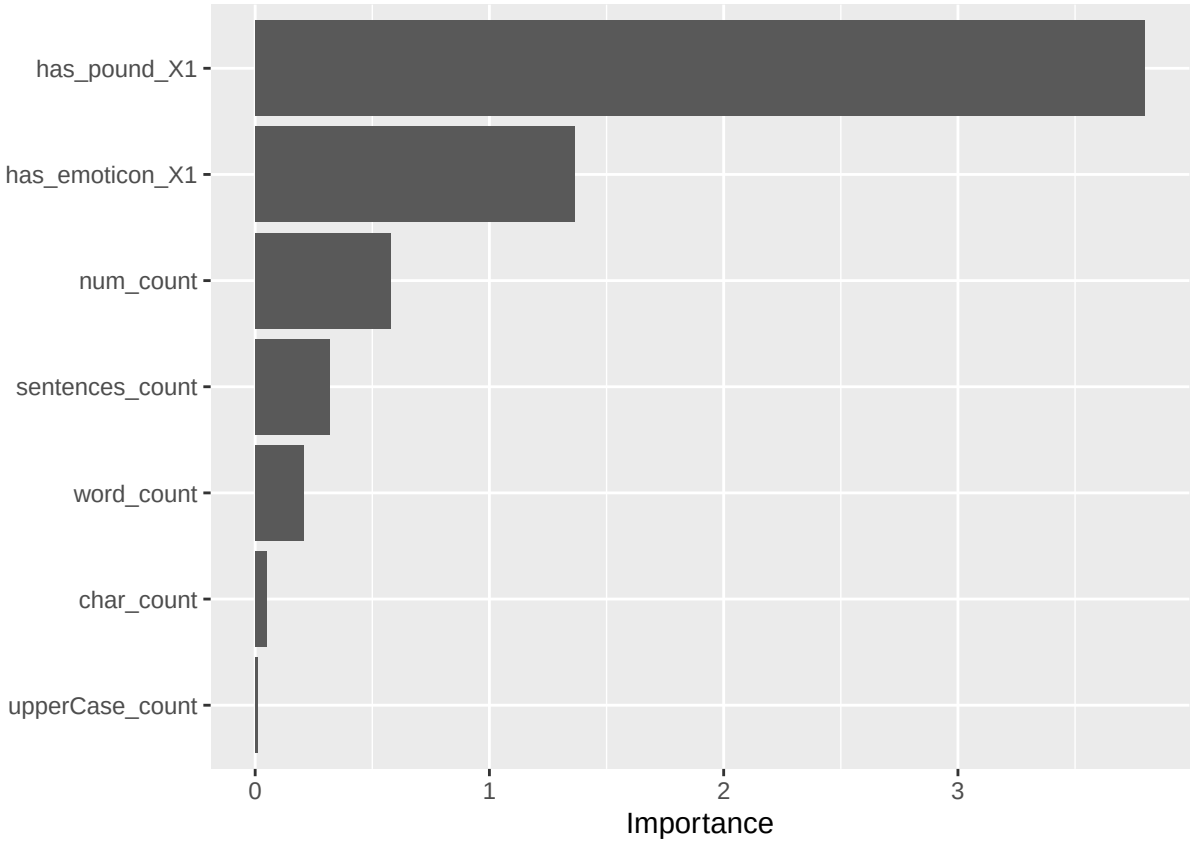
The plot shows us how increasing the penalty affects the accuracy of the model. It seems the smallest penalty offers the best accuracy. Thus, we pick 0.01 as our penalty.

```
#cross validation
set.seed(1234)
spam_fold <- vfold_cv(spam_train_tbl, v = 10)

penalty_grid <-
  grid_regular(penalty(range = c(-2,0)), levels = 40)

tune_result <- tune_grid(
  spam_wf,
  resamples = spam_fold,
  grid = penalty_grid
)
```

```
autoplot(tune_result, metric = "accuracy")
```



The bar plot for the most important variables shows that presence/absence of pound and emoticon are very important to the model.



Similarly, looking at the coefficients, we see that `word_count` and `sentence_count` both are eliminated.

```
## # A tibble: 5 x 3
##   term            estimate penalty
##   <chr>              <dbl>   <dbl>
## 1 has_pound_X1    -1.31       0.01
## 2 has_emoticon_X1 -0.967      0.01
## 3 num_count       -0.407      0.01
## 4 upperCase_count -0.00486    0.01
## 5 char_count      -0.00349    0.01
```

The accuracy and the specificity of the model are 0.977 and 0.997, which is very high. However, the sensitivity is considerably low at 0.832, which means our model is worse at identifying spams.

```
##            Truth
## Prediction spam  ham
##       spam  154    4
##       ham    31 1362
```

```
## # A tibble: 3 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.977
## 2 sens     binary         0.832
## 3 spec     binary         0.997
```

To balance out the sensitivity and specificity, we can change the default threshold of 0.5. To do this, we create a tibble `roc_tbl` with all possible thresholds and the resulting sensitivity and specificity.

Then we can look for a threshold that gives a sensitivity of around 0.90. Note that we do not go for higher values because we do not want the specificity to be affected significantly, which could cause the classifier to flag real texts as spam.

```
#balancing sensitivity and specificity
roc_tbl <- roc_curve(spam_test_pred, type,
.pred_spam)
```

```
## # A tibble: 10 x 3
##    .threshold specificity sensitivity
##         <dbl>       <dbl>       <dbl>
##  1      0.144       0.983       0.908
##  2      0.147       0.984       0.908
##  3      0.151       0.985       0.908
##  4      0.151       0.985       0.908
##  5      0.153       0.986       0.908
##  6      0.172       0.987       0.908
##  7      0.178       0.987       0.903
##  8      0.179       0.988       0.903
##  9      0.196       0.988       0.903
## 10      0.197       0.989       0.903
```

With the new threshold 0.144, we get an improved sensitivity of 0.908 without reducing the specificity by a whole lot.

```
##            Truth
## Prediction spam  ham
##       spam  168   23
##       ham    17 1343
```

```
## # A tibble: 3 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.974
## 2 sens     binary         0.908
## 3 spec     binary         0.983
```

Switching to Random Forest (Ensemble Method)

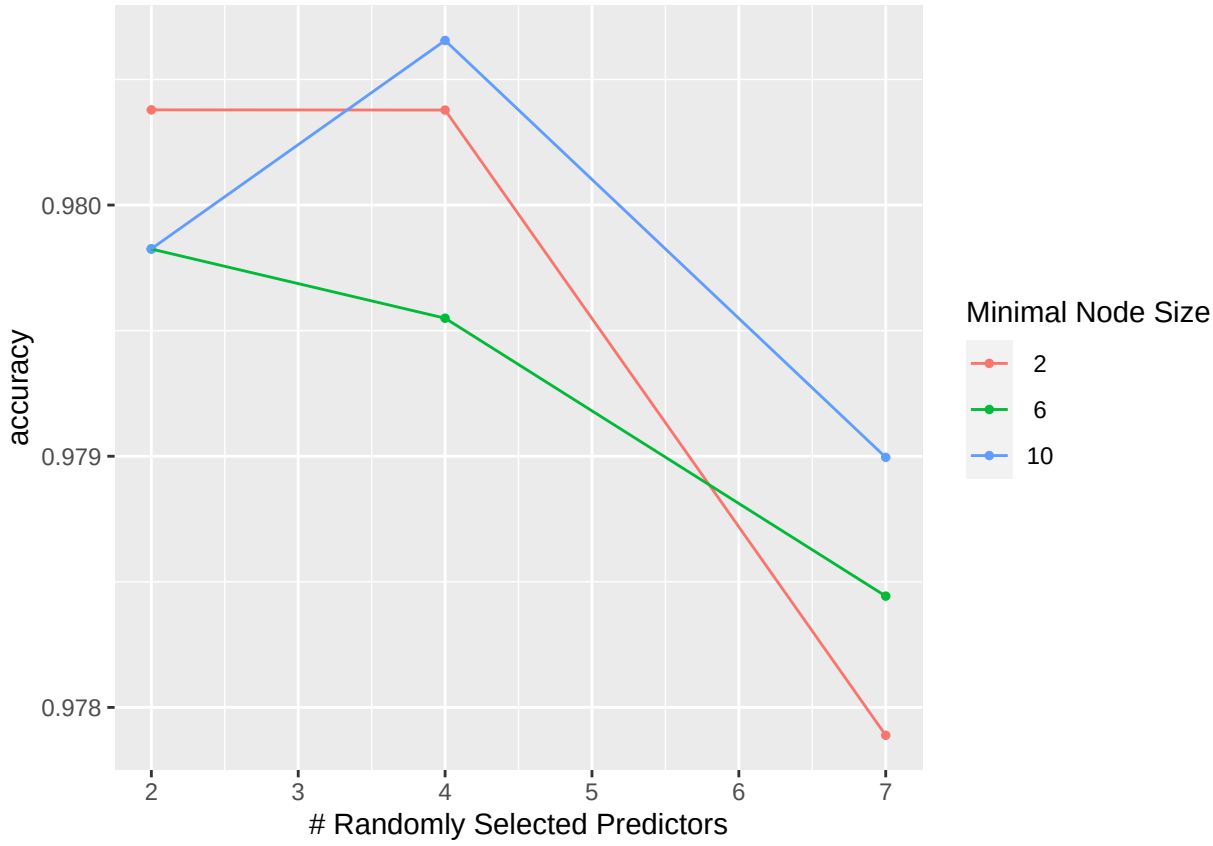
Recursive partitioning trees, also known as decision trees,is a machine learning algorithm that can be used for both classification and regression tasks. Basically, we recursively split the data into subsets based on the values of one or more input variables such that we end up creating a tree-like structure with a certain depth.

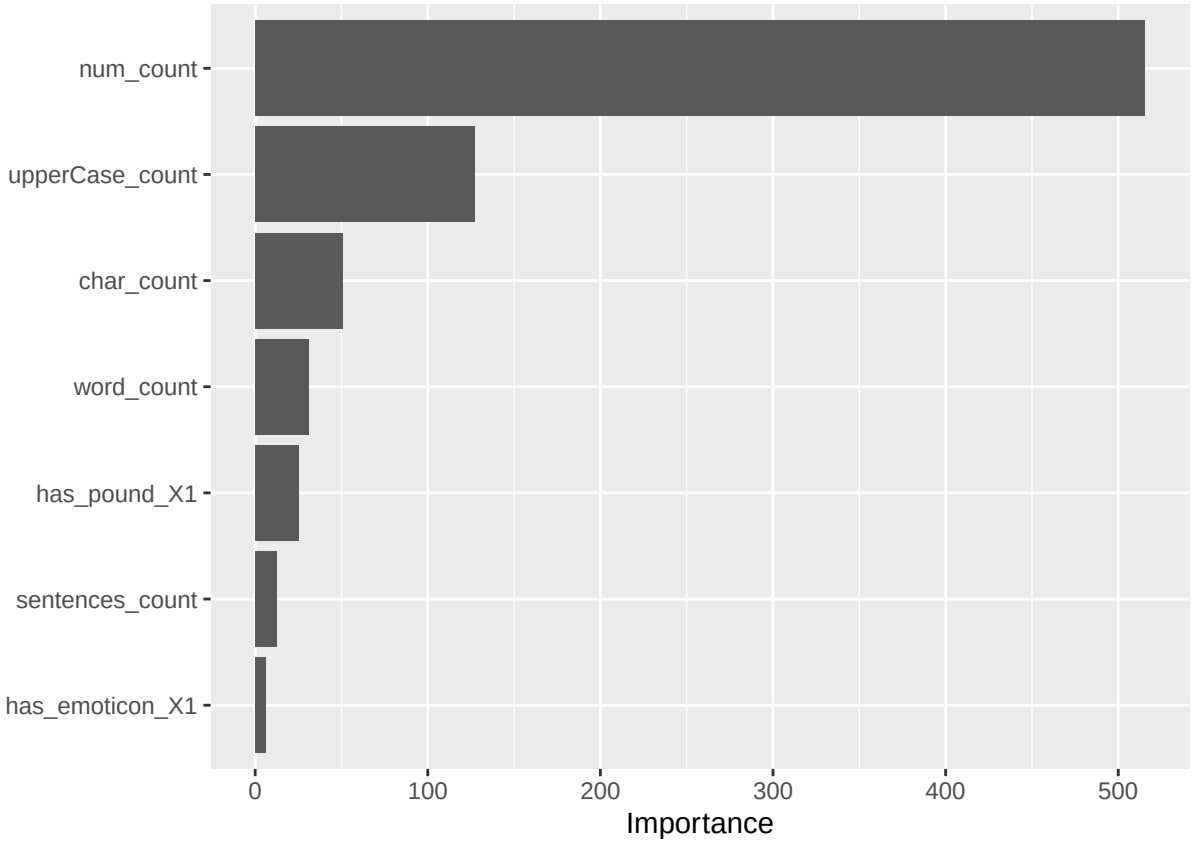
In the random forest approach for classification, we use bootstrapping to create multiple subsets of our training dataset. Then for each subset, we randomly choose a fixed number of predictors and build a tree; using a subset of our predictors decorrelates our trees. Finally, we classify an input according to the majority classification.

Here, we generate 1000 trees and tune parameter `min_n()` to control the minimum number of observations in each node and `m_try()` to control the number of predictors chosen for each tree in the forest.

```
## # A tibble: 1 x 3
##    mtry min_n .config
##   <int> <int> <chr>
## 1     4    10 Preprocessor1_Model8
```

From the plot below, we can see that a minimal node size of 10 and 4 randomly selected predictors give us the best accuracy. Similarly, we the bar plot for variable importance shows us that `num_count` is considerably more important in this model.





The random forest model gives us an accuracy of 0.987, sensitivity of 0.914, and specificity of 0.997. The confusion matrix shows that 4 ham and 16 spam messages were misclassified.

```
## # A tibble: 3 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.987
## 2 sens     binary         0.914
## 3 spec     binary         0.997
```

```
##            Truth
## Prediction spam  ham
##       spam  169    4
##       ham    16 1362
```

Conclusion

Both models perform relatively well on this data. The logistic model using lasso regularization is worse at identifying spams at the default threshold. After reducing the threshold to 0.144, however, we got an accuracy of 0.974, sensitivity of 0.908, and specificity of 0.983. The model uses 5 out of 7 variables which (in the decreasing order of importance) are `has_pound`, `has_emoticon`, `num_count`, `upperCase_count`, and `char_count`.

Random Forest gives a slightly better accuracy of 0.987. Sensitivity and specificity are are also improved at 0.914 and 0.997 respectively. The top 4 important variable are`num_count`, `upperCase_count`, `char_count`,

and `word_count`. Presence of pound and emoticon are apparently not so important in this model. A high accuracy like this could be too good to be in the real world. For instance, in this data ham texts used emoticons to a lesser extent, which might not hold true for today's context. To build a more robust model, we should train it on a larger and newer corpus of texts that includes users from all around the world, not just from the U.K. and Singapore.