Machine Learning Capstone - SMS Spam Classifier

Enlik Tjioe

Updated: March 29, 2019

Introduction

This SMS dataset is collected from real SMS dataset with a spam/ham label for every messages. In this capstone project, we are going to build a classification model to predict spam from sms texts.

I will using *Naive Bayes* and *Random Forest* model for this project.

Load Library

We are using `pacman` library for easier install/load library using `p_load()` function

```
# Easy Install/Load Library
library(pacman)

# Data Manipulation
p_load(dplyr)

# Data Visualization
p_load(ggplot2)
p_load(plotly)

# Text Mining and Wordcloud
p_load(tm)
p_load(e1071)
p_load(SnowballC)
p_load(wordcloud)

# Machine Learning
p_load(caret)
p_load(ROCR)
p_load(partykit)
p_load(ranger)

# Functional Programming
p_load(purrr)
```

Pre-processing Data

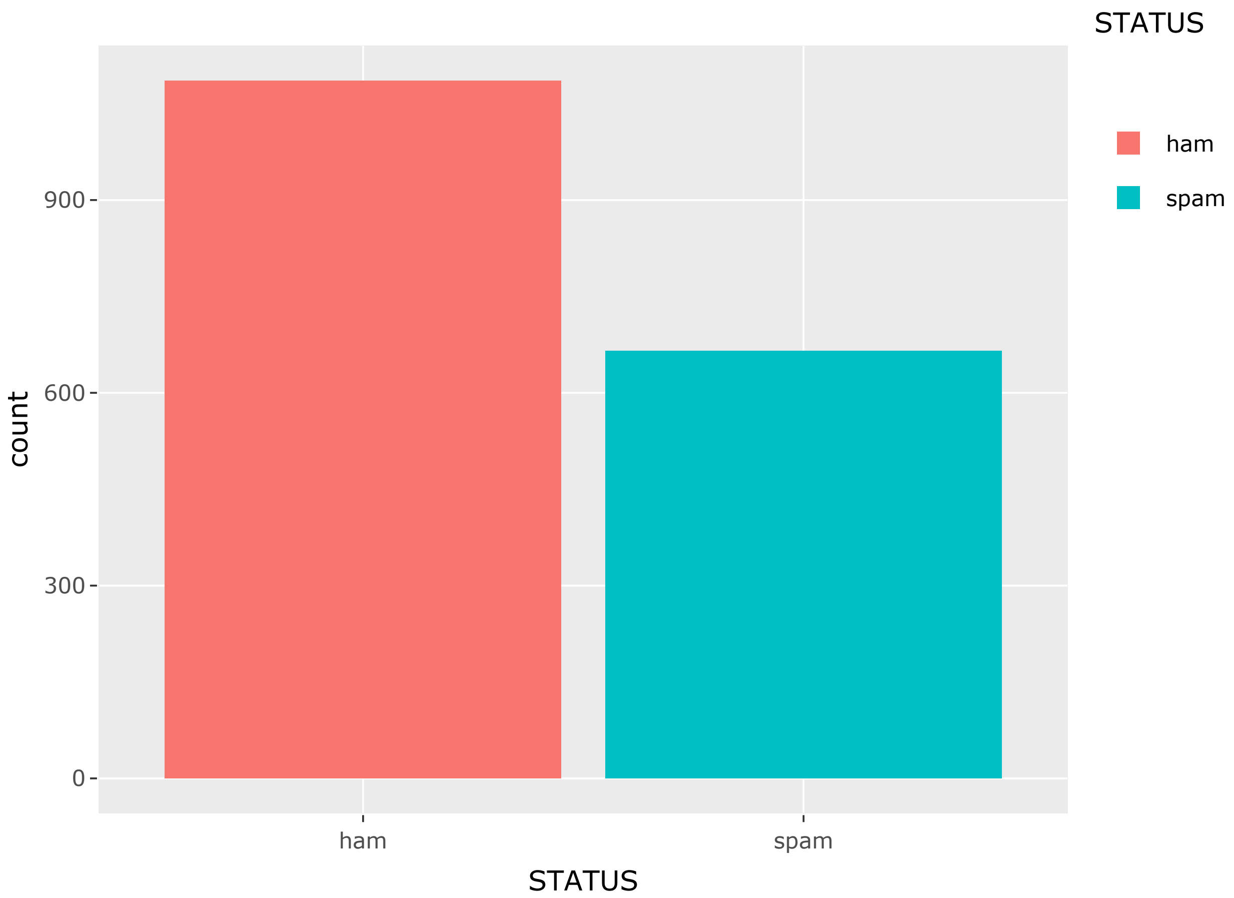
```
sms <- read.csv("datasets/SMS/sms.csv")
glimpse(sms)
```

```
## Observations: 1,751
## Variables: 3
## $ STATUS  <fct> ham, spam, ham, spam, spam, spam, spam, spam, spam, spam…
## $ CONTAIN <fct> "Sy wa ga sampe2 soalnya", "Km baru saja akses Apps Seha…
## $ DATE    <fct> 2018-02-28 11:43:00, 2018-02-28 01:52:00, 2018-02-27 15:…
```

Proportion of Ham or Spam Count

Using `ggplotly` , we will visualize proportion of ham and spam count from our sms dataset

```
ggplotly(ggplot(sms, aes(x = STATUS, fill = STATUS)) +
  geom_bar(stat = "count"))
```



Text Mining Process

Using some *text-mining* package, we will transform our sms text into corpus format and then clean it using `tm_map()` function.

```
corpus <- VCorpus(VectorSource(sms$CONTAIN))

# Custom function for transform corpus
transformer <- content_transformer(function(x, pattern){
  gsub(pattern, " ", x)
})

# stopword for Indonesian language
stopwords.id <- readLines("datasets/SMS/stopwords-id.txt")

# Cleaning Corpus Process
corpus <- tm_map(corpus, content_transformer(tolower))
corpus <- tm_map(corpus, transformer, "\\n")
corpus <- tm_map(corpus, removePunctuation)
corpus <- tm_map(corpus, removeNumbers)
corpus <- tm_map(corpus, stripWhitespace)
corpus <- tm_map(corpus, stemDocument)
corpus <- tm_map(corpus, removeWords, stopwords.id)

corpus[[1]]$content
```

```
## [1] "sy wa ga samp "
```

Create Document Term Matrix

We will create *document term matrix* using cleaned corpus data above

```
sms.dtm <- DocumentTermMatrix(corpus)
freqTerms <- findFreqTerms(sms.dtm, 5)
length(freqTerms)
```

```
## [1] 587
```

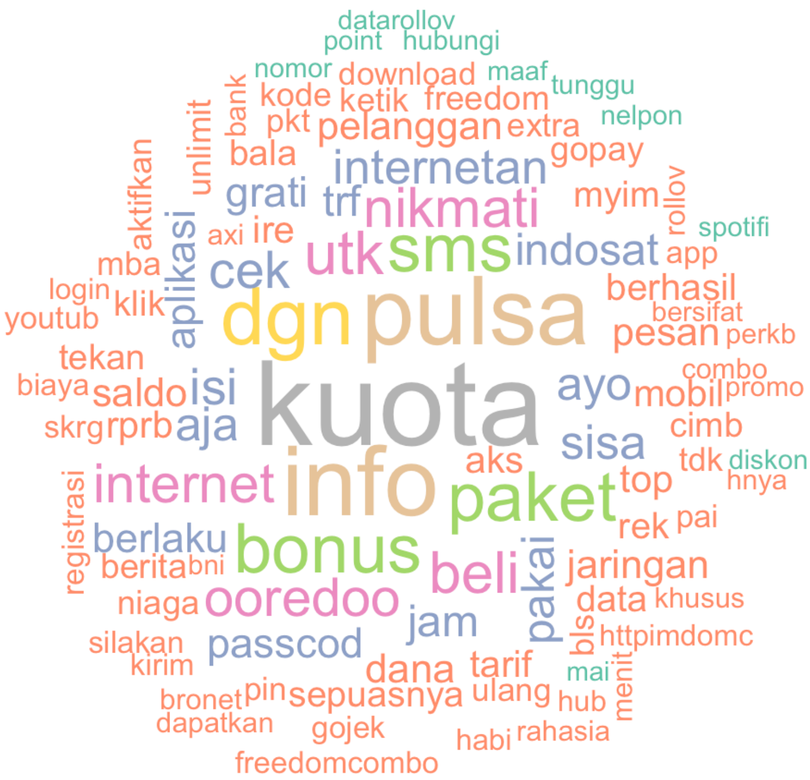
```
freqTerms[1:10]
```

```
##  [1] "add"      "admin"    "aja"      "aks"      "aktif"    "aktifkan"
##  [7] "akun"     "all"      "ambil"    "andb"
```

Make a wordcloud

Explore most frequent word to appear using `wordcloud()`

```
wordcloud(corpus,
          min.freq = 5,
          max.words = 100,
          random.order = FALSE,
          colors = brewer.pal(8, "Set2"))
```



From wordcloud above, we can see some of most frequent words such as: kuota, pulsa, bonus, paket, etc

Split train and test dataset

We will split our sms data into train and test, and we will use that to train our model using data train and evaluate usin data test.

```
data.intrain <- sample(nrow(sms.dtm), nrow(sms.dtm)*0.8)
sms.dtm.train <- sms.dtm[data.intrain, ]
sms.dtm.test <- sms.dtm[-data.intrain, ]

corpus.train <- corpus[data.intrain]
corpus.test <- corpus[-data.intrain]

sms.status.train <- sms[data.intrain, ]$STATUS
sms.status.test <- sms[-data.intrain, ]$STATUS
```

```
prop.table(table(sms.status.train))
```

```
## sms.status.train
##   ham  spam
## 0.625 0.375
```

```
prop.table(table(sms.status.test))
```

```
## sms.status.test
##       ham      spam
## 0.6011396 0.3988604
```

```
dtm_train <- sms.dtm.train[, freqTerms]
dim(dtm_train)
```

```
## [1] 1400  587
```

```
dtm_test <- sms.dtm.test[, freqTerms]
dim(dtm_test)
```

```
## [1] 351 587
```

We will create function to classify numeric value into *ham* or *spam* class

```
convert_count <-  function(x) {
  y <- ifelse(x > 0, "spam", "ham")
  y
}
```

Implement our own function `convert_count()` into data train and test

```
dtm_train <- apply(dtm_train, 2, convert_count)
dtm_test <- apply(dtm_test, 2, convert_count)

dtm_test[1:10, 500:510]
```

```
##     Terms
## Docs tarif tarik tcash tdk   tea   teh   tekan  telkomsel telp  telpon
##   1  "ham" "ham" "ham" "ham" "ham" "ham" "ham"  "ham"     "ham" "ham"
##   5  "ham" "ham" "ham" "ham" "ham" "ham" "spam" "ham"     "ham" "ham"
##   14 "ham" "ham" "ham" "ham" "ham" "ham" "ham"  "ham"     "ham" "ham"
##   20 "ham" "ham" "ham" "ham" "ham" "ham" "ham"  "ham"     "ham" "ham"
##   21 "ham" "ham" "ham" "ham" "ham" "ham" "spam" "ham"     "ham" "ham"
##   26 "ham" "ham" "ham" "ham" "ham" "ham" "ham"  "ham"     "ham" "ham"
##   35 "ham" "ham" "ham" "ham" "ham" "ham" "ham"  "ham"     "ham" "ham"
##   43 "ham" "ham" "ham" "ham" "ham" "ham" "ham"  "ham"     "ham" "ham"
##   48 "ham" "ham" "ham" "ham" "ham" "ham" "ham"  "ham"     "ham" "ham"
##   49 "ham" "ham" "ham" "ham" "ham" "ham" "ham"  "ham"     "ham" "ham"
##     Terms
## Docs temukan
##   1  "ham"
##   5  "ham"
##   14 "ham"
##   20 "ham"
##   21 "ham"
##   26 "ham"
##   35 "ham"
##   43 "ham"
##   48 "ham"
##   49 "ham"
```

Naive Bayes Model

In machine learning, naive Bayes classifiers are a family of simple “probabilistic classifiers” based on applying Bayes’ theorem with strong (naive) independence assumptions between the features.

Naive Bayes classifiers are highly scalable, requiring a number of parameters linear in the number of variables (features/predictors) in a learning problem.

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes` theorem with the *naive* assumption.

Train Our Model

Using `naiveBayes` function we will train our Naive Bayes model using `sms.status.train` data

```
set.seed(151)
modelNB <- naiveBayes(dtm_train, sms.status.train)
```

Make Prediction

Make our prediction based on our model

```
pred <- predict(modelNB, dtm_test)
dim(dtm_test)
```

```
## [1] 351 587
```

Create Confusion Matrix

We will check result of confusion matrix from our Naive Bayes model

```
conf <- confusionMatrix(pred, sms.status.test)
conf
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction ham spam
##       ham  201   10
##       spam  10  130
##
##                Accuracy : 0.943
##                  95% CI : (0.9134, 0.9649)
##     No Information Rate : 0.6011
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8812
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9526
##             Specificity : 0.9286
##          Pos Pred Value : 0.9526
##          Neg Pred Value : 0.9286
##              Prevalence : 0.6011
##          Detection Rate : 0.5726
##    Detection Prevalence : 0.6011
##       Balanced Accuracy : 0.9406
##
##        'Positive' Class : ham
##
```

```
dim(sms.status.test)
```

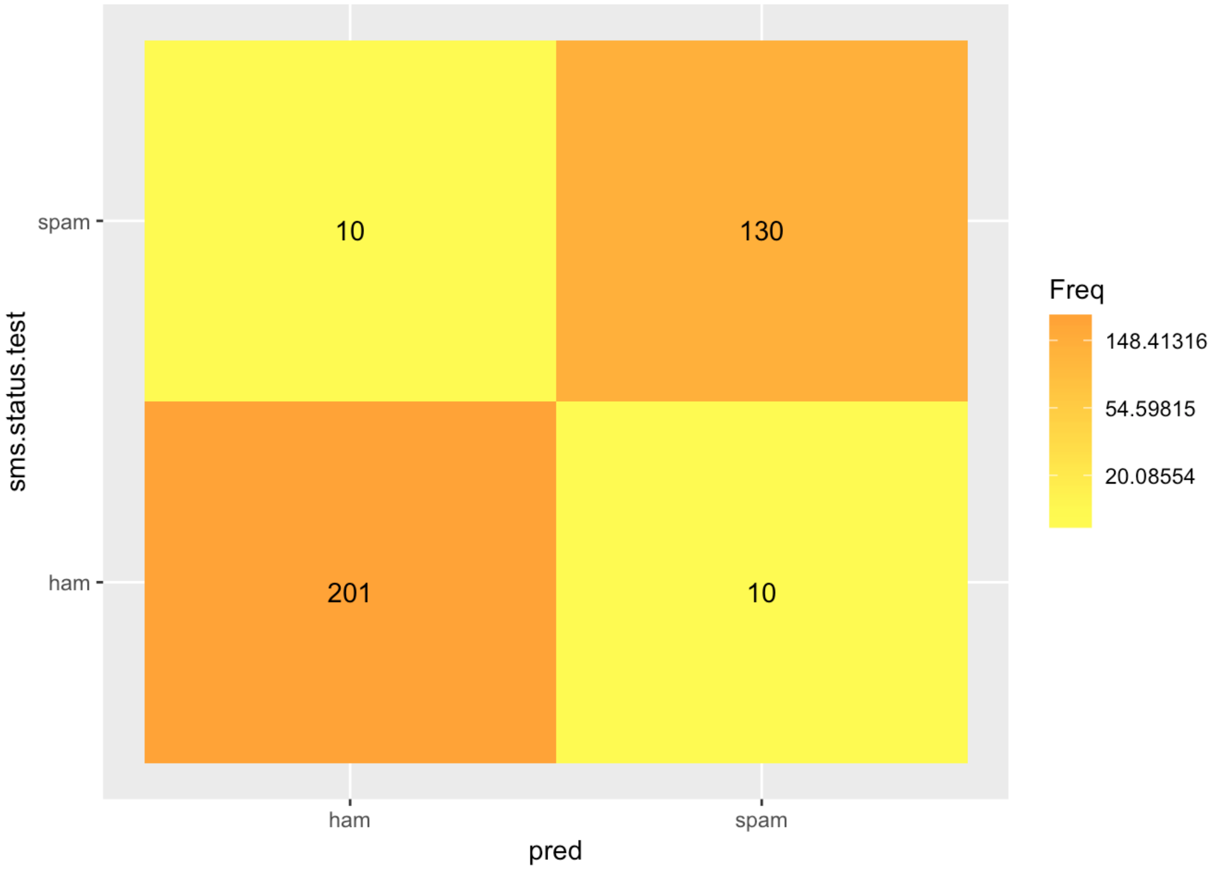
```
## NULL
```

Visualize Confusion Matrix

We will visualize our confusion matrix into a graph

```
conf_matrix <- as.data.frame(table(pred, sms.status.test))

ggplot(data = conf_matrix, aes(x = pred, y = sms.status.test)) +
  geom_tile(aes(fill = Freq)) +
  geom_text(aes(label = sprintf("%1.0f", Freq)), vjust = 1) +
  scale_fill_gradient(low = "yellow",
                      high = "orange",
                      trans = "log")
```



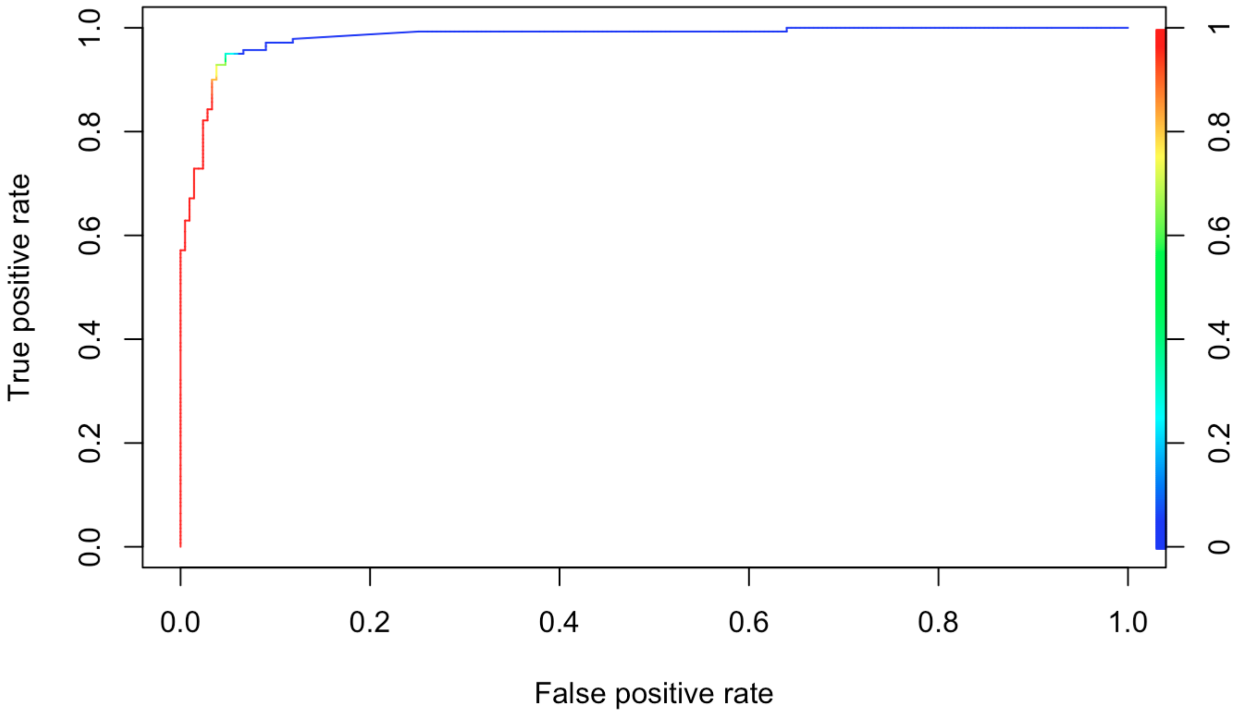
ROC Curve

ROC is the alternative method to check our model performance, inform us about how much our model *TRUE POSITIVE* and *FALSE POSITIVE* value. We will how good our performance from the curve called AUC.

AUC value range from 0 to 1

```
probs <- predict(modelNB, dtm_test, type = "raw")

pred <- prediction(probs[, "spam"], sms.status.test)
plot(performance(pred, measure = "tpr", x.measure = "fpr"), colorize = TRUE)
```




```
auc_value <- performance(pred, measure = "auc")
auc_value@y.values[[1]]
```

```
## [1] 0.981889
```

AUC: Area Under the Curve = 0.98. That means this model was good enough to predict our *POSITIVE CLASS* and *NEGATIVE CLASS*. It's suitable when our data has unbalance label.

Submission Test

We will try to predict *spam or ham* class for our `submissionSMS.csv` data.

```
sub.sms <- read.csv("datasets/SMS/submissionSMS.csv")
glimpse(sub.sms)
```

```
## Observations: 321
## Variables: 3
## $ DATE    <fct> 2018-04-25 14:20:00, 2018-04-25 14:13:00, 2018-04-25 12:…
## $ CONTAIN <fct> "ELITE RELOAD PULSA:Kami ingin menawarkan anda menjadi a…
## $ STATUS  <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, …
```

Text Mining on SubmissionSMS Data

Same as before, we need to transform our data first using text mining method

```r
sub.corpus <- VCorpus(VectorSource(sub.sms$CONTAIN))

# Custom function for transform corpus
transformer <- content_transformer(function(x, pattern){
  gsub(pattern, " ", x)
})

# stopword for Indonesian language
stopwords.id <- readLines("datasets/SMS/stopwords-id.txt")

# Cleaning Corpus Process
sub.corpus <- tm_map(sub.corpus, content_transformer(tolower))
sub.corpus <- tm_map(sub.corpus, transformer, "\\n")
sub.corpus <- tm_map(sub.corpus, removePunctuation)
sub.corpus <- tm_map(sub.corpus, removeNumbers)
sub.corpus <- tm_map(sub.corpus, stripWhitespace)
sub.corpus <- tm_map(sub.corpus, stemDocument)
sub.corpus <- tm_map(sub.corpus, removeWords, stopwords.id)

sub.corpus[[1]]$content
```

```
## [1] "elit reload pulsakami  menawarkan   agen pulsa all oper harga vvvminat in
vit bbm da wa"
```

```r
sub.dtm <- DocumentTermMatrix(sub.corpus)
sub.freqTerms <- findFreqTerms(sub.dtm, 5)
length(sub.freqTerms)
```

```
## [1] 182
```

```r
sub.freqTerms[1:10]
```

```
##  [1] "abaikan"  "aja"      "aks"      "aktif"    "aktifkan" "andatlp"
##  [7] "aplikasi" "app"      "asyik"    "axi"
```

```r
convert_count <-  function(x) {
  y <- ifelse(x > 0, "spam", "ham")
  y
}
```

```r
sub.test <- apply(sub.dtm, 2, convert_count)
dim(sub.test)
```

```
## [1] 321 848
```

Predict Using Our Model

We will using our `modelNB` Naive Bayes model that we've created before, and try to predict class from submission sms dataset.

```
sub.pred <- predict(modelNB, sub.test)
sub.sms$STATUS <- sub.pred

# write_csv(sub.sms, "enlik_spam_classification.csv")
```

Conclusion

Our Naive Bayes classification model gave around 94% accuracy value based on our training model

References

Takes too long to create a random forest model for text data (https://community.rstudio.com/t/takes-too-long-to-create-a-random-forest-model-for-text-data/20747/2)

Wikipedia Naive Bayes classifier (https://en.wikipedia.org/wiki/Naive_Bayes_classifier)