SMS classifier project

A project of creating an SMS spam classifier, using NLP techniques and machine learning algorithms

By: Juan Vicente Peluso

GitHub notebook

Table of contents

1. **Introduction**
    - SMS and spam
    - Problem description and data available
2. **Data quality check**
3. **Descriptive statistics**
    - Target feature distribution
    - Basic text features creation
    - Feature statistics
4. **Exploratory data analysis**
    - Features statistics per target label
    - Frequent words distribution per target label
    - Correlated features analysis
5. **Model development**
    - Train/Test split and baseline
    - NLP features creation, model selection and hyperparameter tuning
    - Model validation
    - Feature importance and predictions
6. **Conclusions**

1. Introduction

SMS and spam

Even in these times, when technology is advancing by leaps and bounds, SMS is still a widely-used communication platform. Only the recipient's phone number is needed, and we can send a message that arrives almost instantly, without using data. Spam is defined as unsolicited usually commercial messages (such as e-mails, text messages, or internet postings) sent to a large number of recipients or posted in a large number of places.

SMS spamming is an old marketing practice and is still a problem causing headaches and annoyances for consumers everywhere. Spam messages coming from SMS and messaging apps are becoming more widespread. According to the latest industry data, over half of text message users globally now receive at least one weekly spam message. Alarmingly, more than a quarter get spammed every day with unsolicited messages.

Spam can fill your inbox with no requested messages and sometimes is not only ads, sometimes are even a *security threat*.

Problem description

Keep the inbox without unwanted messages, avoiding unsolicited ads or messages that include threats that can either infect your phone or steal your private data. An application capable of filtering incoming SMS, discarding those that are spam, would avoid more than one trouble to the users. Using machine learning algorithms and techniques like ***NLP*** , a model can be developed for this purpose.

A critical aspect is that non-spam SMS reach the inbox always, because a non-spam SMS (ham) tagged as spam, will create suspicion in the process, unlike the occasional SMS spam that reaches the inbox.

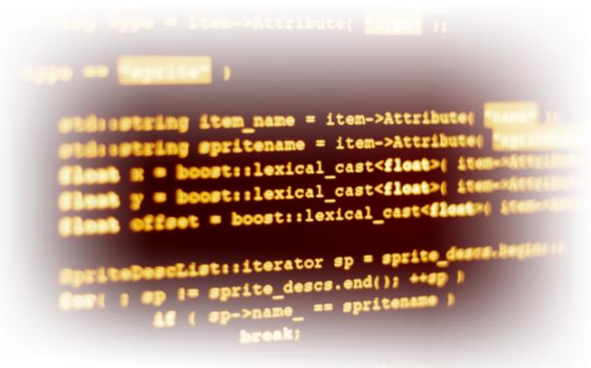
Using a *dataset* with 5000+ SMS gathered for mobile phone spam research, we will analyze the data and develop a machine learning model to classify the SMS. The dataset is a CSV file, and it has 5 columns; the first with the flag ***spam/ham***, which will be the target feature. And the remaining 4 with the text of the SMS.

As explained before, tagging a non-spam SMS as spam, is a big mistake; but also tagging too much spam SMS as a legitimate message, can't be good either. That's why we will measure the effectiveness of the model with the ***Precision, Recall, and F1 scores***.

2. Data quality check

The dataset has 5571 records (SMS) with no missing or NaN values in the flag or the corpus of the SMS (at least all rows have data on the first column). We found 403 duplicate rows, which we deleted.

To simplify the analysis and modeling process, we've merged the text of the 4 columns with the text of the SMS. Finally, we change the flag ***spam/ham*** flag to ***0/1*** values.

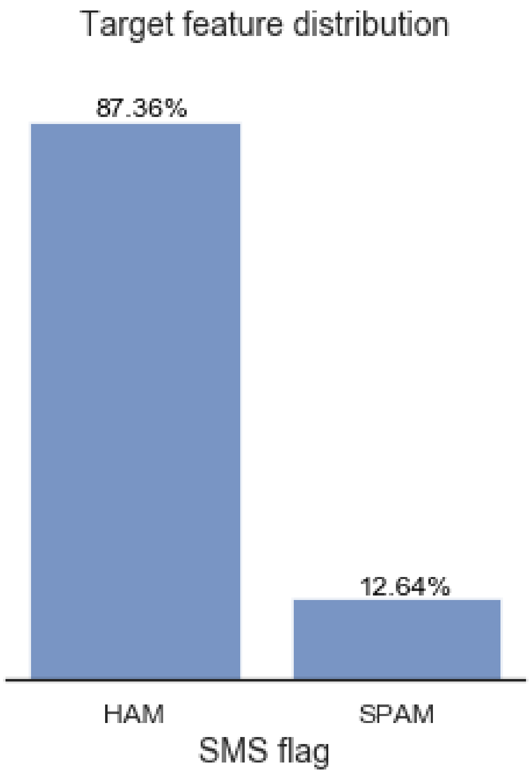


```
type = item->Attribute( "type" );
type == "sprite" )
std::string item_name = item->Attribute( "name" );
std::string spritename = item->Attribute( "sprite
float x = boost::lexical_cast<float>( item->Attri
float y = boost::lexical_cast<float>( item->Attri
float offset = boost::lexical_cast<float>( item->A
SpriteDescList::iterator sp = sprite_descs.begin();
for( ; sp != sprite_descs.end(); ++sp )
    if ( sp->name_ == spritename )
        break;
```

3. Descriptive statistics

Target feature distribution

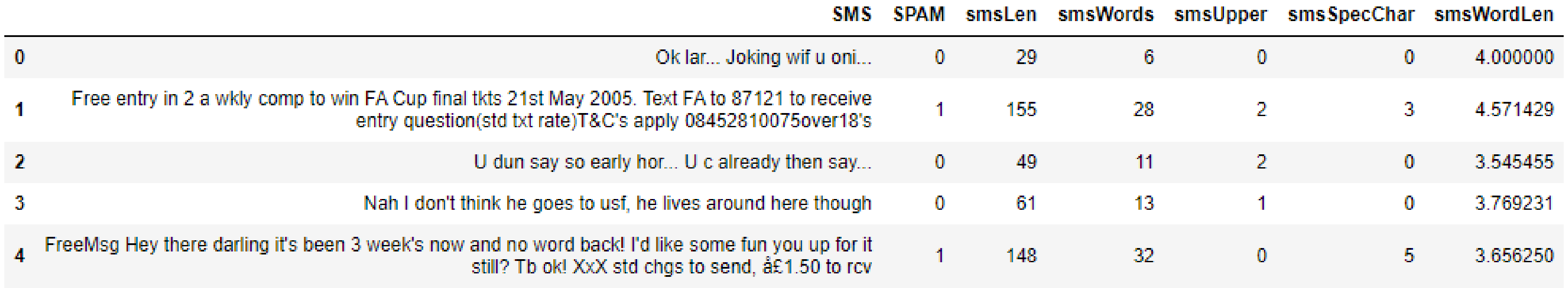
We are dealing with an imbalanced classification problem, having ***87.36%*** of the records are labeled as ***ham*** *(0)*, implying that only around 1 of each 7 or 8 messages is tagged as ***spam*** *(1)*.



Basic text feature creation

To start the analysis of the SMS content 5 statistical features were created:

- ***smsLen***: Length of the SMS.
- ***smsWords***: Number of words in the SMS.
- ***smsUpper***: Number of words in capital letters in the SMS.
- ***smsSpecChar***: Number of special characters in the SMS.
- ***smsWordLen***: Average length of the words of the SMS.

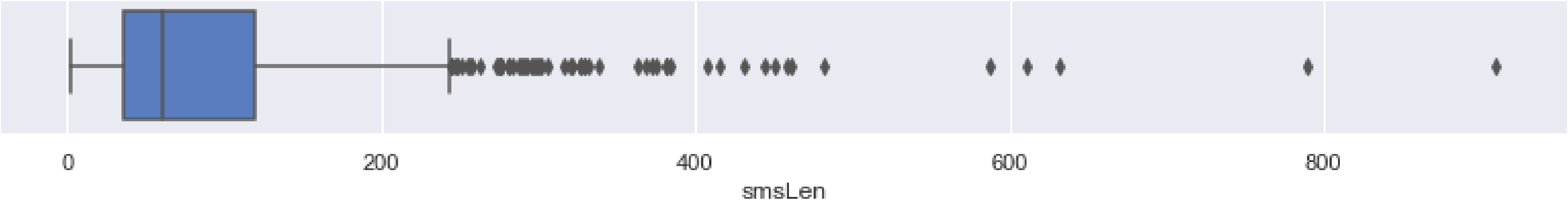


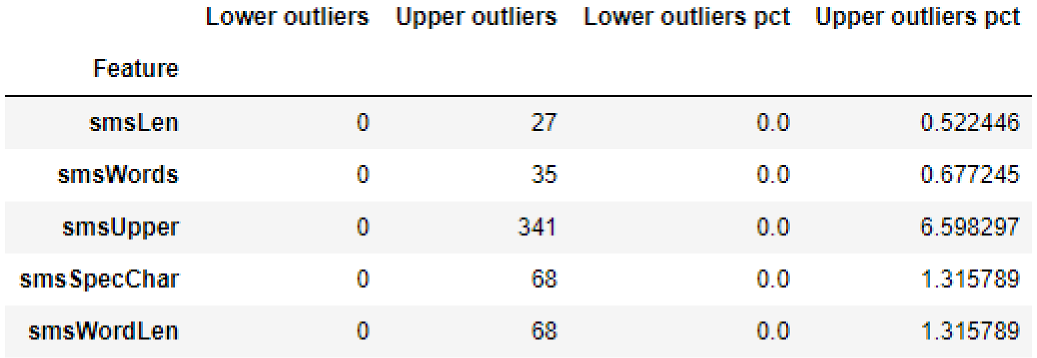
	SMS	SPAM	smsLen	smsWords	smsUpper	smsSpecChar	smsWordLen
0	Ok lar... Joking wif u oni...	0	29	6	0	0	4.000000
1	Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std txt rate)T&C's apply 08452810075over18's	1	155	28	2	3	4.571429
2	U dun say so early hor... U c already then say...	0	49	11	2	0	3.545455
3	Nah I don't think he goes to usf, he lives around here though	0	61	13	1	0	3.769231
4	FreeMsg Hey there darling it's been 3 week's now and no word back! I'd like some fun you up for it still? Tb ok! XxX std chgs to send, å£1.50 to rcv	1	148	32	0	5	3.656250

Features statistics

When we calculate the basic statistics of the newly created features, we see that there are several outliers on all of the features, for example, the ***smsLen*** feature has a mean of *79.5* characters long, and the max value has ***910***! Here, you can see the *box plot* of the ***smsLen*** feature, all the other plots can be seen in the notebook.

All features have upper range outliers but only the ***smsUpper*** has more than 2% of the records tagged as outliers, even expanding the whisker factor from 1.5 to 2.5 IQR. Even though they are actual records, they influence the group statistics and the analysis we can make of those results. So, for the EDA we'll remove those rows to get a more accurate insight of the data.





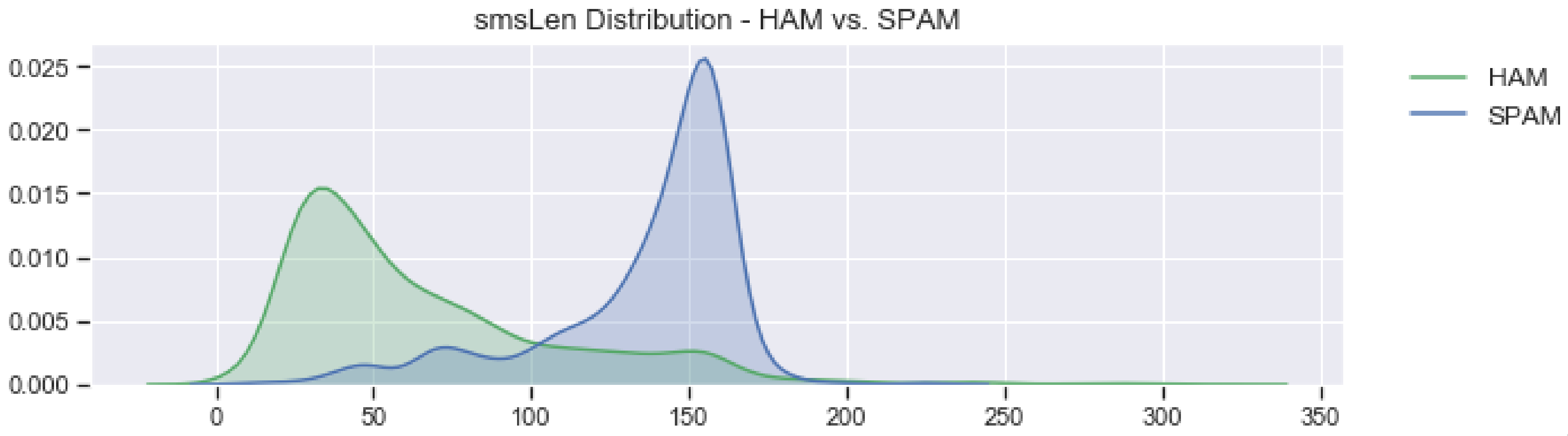
	Lower outliers	Upper outliers	Lower outliers pct	Upper outliers pct
Feature				
smsLen	0	27	0.0	0.522446
smsWords	0	35	0.0	0.677245
smsUpper	0	341	0.0	6.598297
smsSpecChar	0	68	0.0	1.315789
smsWordLen	0	68	0.0	1.315789

4. Exploratory data analysis

Feature statistics per target label

Once we've removed the outliers when we calculate the mean and the STD per label, we see important differences in the data. For example, a non-spam SMS is 66 characters long on average, while the mean for spam SMS is more than twice, reaching a length mean of 136.

In the KDE plot, we can see graphically how the density differs. In the notebook, you can see the plot of the other features

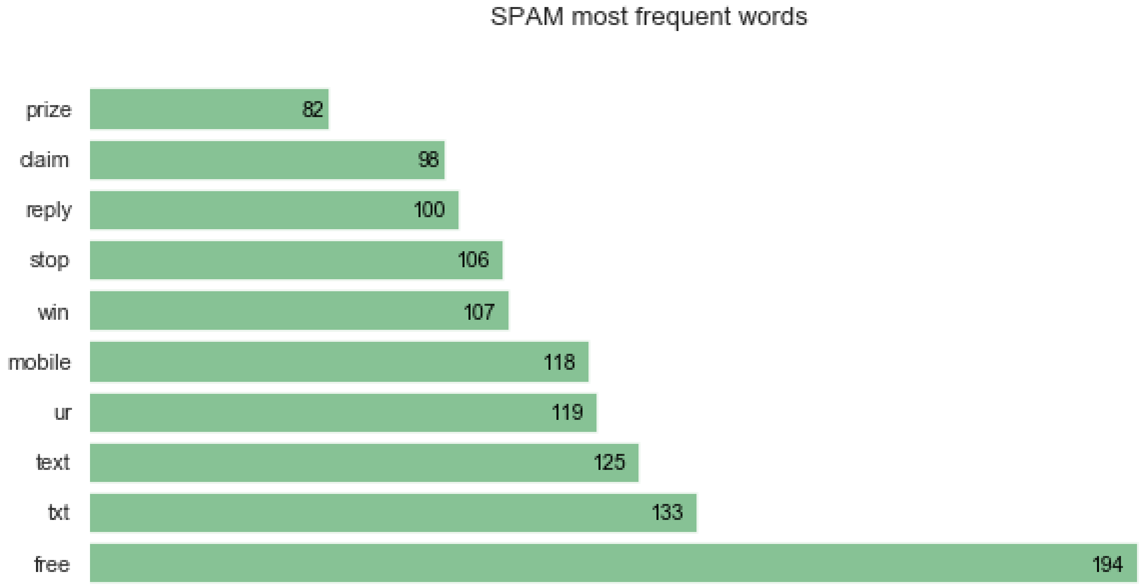


Frequent words distribution per target label

First, we normalize the SMS corpus applying the following actions:

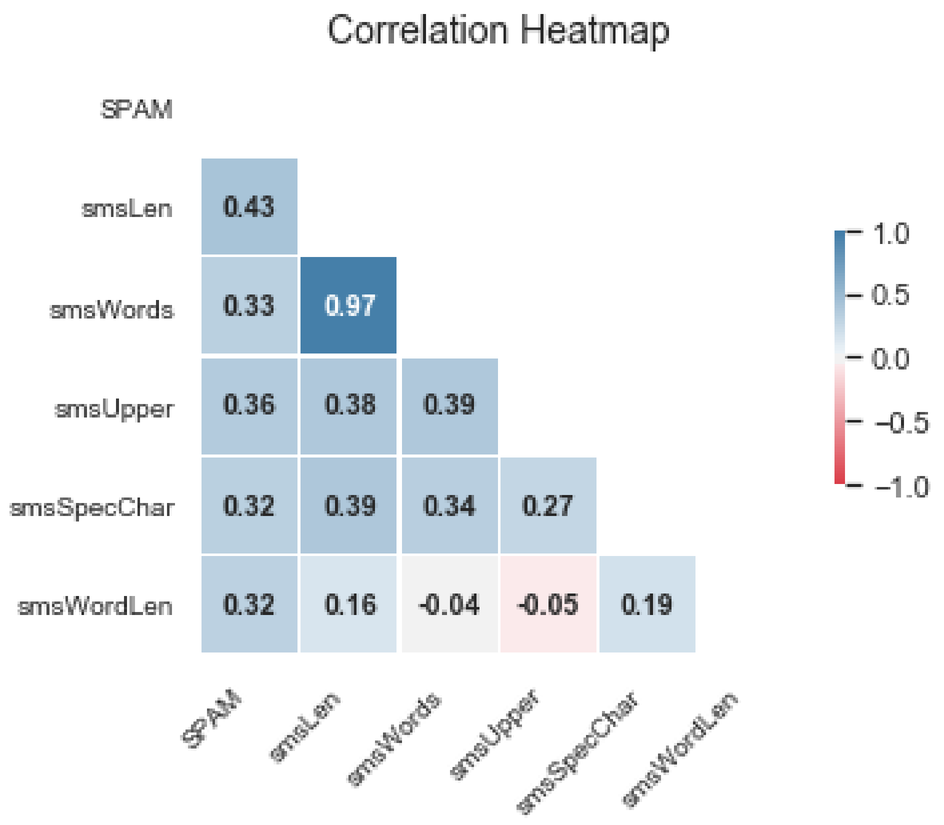
- Update to lowercase all words.
- Remove all punctuation characters.
- Remove stop words.
- Word lemmatisation.

Then, we create a **BoW** (bag-of-words) matrices per target label to see which words are the most frequent in each. In the spam SMS, we see words we'd expect in commercial or phishing messages like *free*, *reply*, *prize*, and *win*. The chart for non-spam SMS can be seen in the notebook.



Correlated features analysis

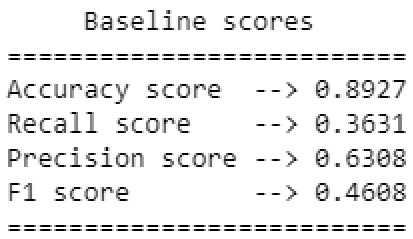
In general, the correlation between the spam label and the features is weak, being **0.43** the higher coefficient (*smsLen*). Among the other features is relevant and logic the high correlation between *smsWords* and *smsLen*, as more words present in the text, longer the SMS will be.


Correlation Heatmap

5. Model development

Train/Test split and baseline

The data is split into a 7:3 train/test ratio. Then, only the basic features created previously for the EDA phase, we create a basic Logistic Regression model and check its performance with a cross-validation score process, just to know where we stand before developing more tuned models. The results were the following:



```
     Baseline scores
==========================
Accuracy score  --> 0.8927
Recall score    --> 0.3631
Precision score --> 0.6308
F1 score        --> 0.4608
==========================
```

Accuracy will always be high in an unbalanced dataset, but the results of ***Precision***, ***Recall***, and consequently, of ***F1*** scores reflect that the model doesn't handle the minority class (spam) properly. Intending, we need to add more features to feed the model.

NLP features creation, model selection and hyperparameter tuning

First, we split the original dataset (only SMS corpus and spam flag) into the same proportions we did before, and create a pipeline with the following steps:

1. Creation of the matrix of features (basic/NLP) with *FeatureUnion*, the internal steps are:
   - Creation of basic features.
   - Creation of the *CountVectorizer* matrix, preprocessing the text (lowercase, stopwords, etc.) and limiting the number of features to 2500.
   - Creation of the *TF-IDF* matrix, preprocessing the text (lowercase, stopwords, etc.) and limiting the number of features to 2500.
2. Standardize the data with the *StandardScaler* method.
3. The classifier machine learning model.

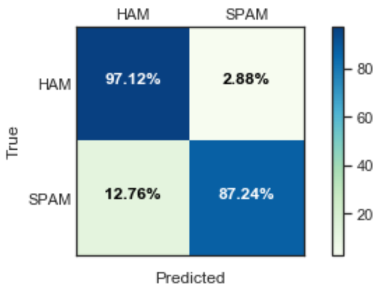
For each model, we run the GridSearchCV to find the best hyperparameters avoiding overfitting.

The machine learning algorithms selected are *Multinomial Naïve Bayes*, *Logistic Regression*, and *Linear SVM*. Mainly, because linear algorithms are very simple to train and the results are interpretable, as you can easily extract the most important coefficients from the model. The results of the whole process were:

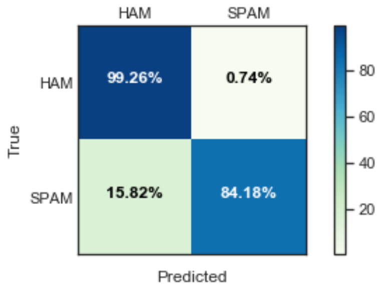
Classifier	Accuracy	Precision	Recall	F1 score	Best parameters
Multinomial NB	0.9547	0.7853	0.8906	0.8323	alpha: 0.003
Logistic Regression	0.9735	0.9767	0.8096	0.8846	C : 0.1
Linear SVM	0.9682	0.9279	0.8118	0.8655	C : 0.05 - gamma : 0.01

Model Validation

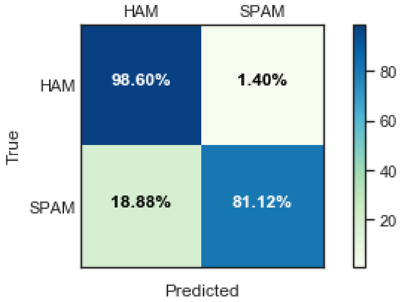
With the best hyperparameters found for each model, we fitted the models with the train data, and predict with the test data to validate the effectiveness of the models. In the confusion matrices, we can see the results:



```
== Naïve Bayes ==
Accuracy: 0.9587
Precision: 0.8143
Recall: 0.8724
F1: 0.8424
```



```
== Logistic Regression ==
Accuracy: 0.9736
Precision: 0.9429
Recall: 0.8418
F1: 0.8895
```



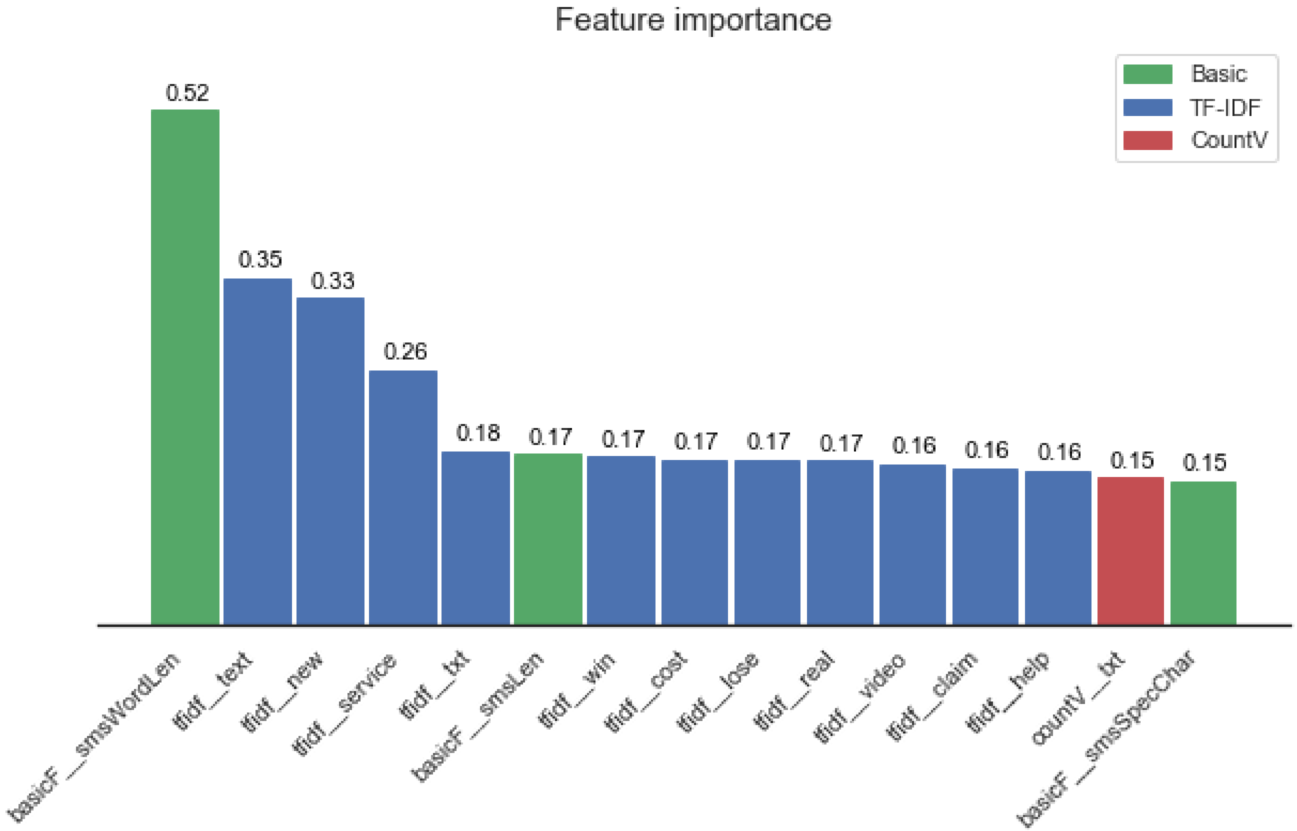
```
== SVM ==
Accuracy: 0.9639
Precision: 0.8933
Recall: 0.8112
F1: 0.8503
```

The results match with those obtained in the Cross-validation process. ***Logistic Regression*** is again the best model, obtaining a bit higher accuracy score, although the *Precision* score decreases, the *Recall* and *F1* scores increase, which we interpret the model generalizes better.

We see that the correct classification of the non-spam SMS is a ***99.26%***! Which is what we wanted to achieve in the first place. Also, only 15.82% of spam SMS was classified erroneously as non-spam.

Feature importance and predictions

The top-15 most important features of the model are the following. Oddly enough, the most important feature of the model is a basic statistic calculated over the SMS text (***smsWordLen***), and that 3 out of 5 basic statistics features are in the top-15, proving that sometimes the simpler, the better. Note too, that 11 features are TF-IDF weights and only 1 of the Count Vectorizer bag-of-words. The predictions were exported to an excel file.



6. Conclusions

We have shown that by using NLP techniques and calculating basic text statistics, we can develop a classifier to filter incoming SMS, avoiding ad or threatening messages in the inbox. Of course, the model performance can be improved, especially labeling the spam SMS.

Perhaps using a more advanced NLP technique like word embedding and developing a deep learning algorithm, the results would improve considerably, bearing in mind the increase in computational cost when training the model, and its complexity when explaining the results to a non-technical audience.