FLIP ROBO

Spam SMS Classification

Submitted by:

SANTOSH H. HULBUTTI

Table of Contents

ACKNOWLEDGMENT

This project is completed using knowledge/information available on internet.

Following are the websites & YouTube Channels, which were used to scape data & understand concepts related to ML, AI & Data Visualization.

Websites:

1. towardsdatascience.com
2. medium.com
3. analyticsvidya.com
4. DataTrained LMS Platform
5. Official documentation of ScikitLearn, Matplot library, AutoViz, Sweet Viz, Pandas Library & Seaborn library.
6. Stackoverflow.com
7. YouTube Channels:
    a. Krish Naik
    b. Nicholas Renotte
    c. Sidhdhardan
    d. Keith Galli

I would like to thank FlipRobo Technologies, for giving an opportunity to work as an intern during this project period. And also like to thank mentor Ms. Gulshana Chaudhary for assigning the project.

INTRODUCTION

Business Problem Framing

The development of the cell phone has prompted a sensational increment in SMS spam messages. Despite the fact that in many parts of the world, versatile informing channel is right now viewed as "spotless" and trusted, on the complexity ongoing reports obviously show that the volume of cell phone spam is drastically expanding step by step. Spam separating is a similarly late errand to arrangement such an issue. It acquires numerous worries and convenient solutions from SMS spam separating. This report moves to deal with the creation of machine learning (NLP) model for classification of messages as Ham or Spam from the dataset provided.

Conceptual Background of the Domain Problem

Short Message Services (SMS) is far more than just a technology for a chat. SMS technology evolved out of the global system for mobile communications standard, an internationally accepted. Spam is the abuse of electronic messaging systems to send unsolicited messages in bulk indiscriminately. While the most widely recognized form of spam is email spam, the term is applied to similar abuses in other media and mediums. SMS Spam in the context is very similar to email spams, typically, unsolicited bulk messaging with some business interest. SMS spam is used for commercial advertising and spreading phishing links. Commercial spammers use malware to send SMS spam because sending SMS spam is illegal in most countries. Sending spam from a compromised machine reduces the risk to the spammer because it obscures the provenance of the spam. SMS can have a limited number of characters, which includes alphabets, numbers, and a few symbols. A look through the messages shows a clear pattern. Almost all of the spam messages ask the users to call a number, reply by SMS, or visit some URL. The low price and the high bandwidth of the SMS network have attracted a large amount of SMS spam.

Review of Literature

- Understanding the words related to spamming in a given context of a message.
- Text segregation to classify as spam or ham.
- Understanding of the pattern of words in a message to be able to feed to machine learning model.
- Understanding of machine learning models best suited for spam classification.
- Understanding of Long Short-term memory (LSTM) – RNN based NLP models.

Motivation for the Problem Undertaken

The objective & motivation behind doing this project is to understand the various methods of NLP model building, pre-processing, text classification to gain knowledge related to the domain of spam text classification.

ANALYTICAL PROBLEM FRAMING

Mathematical/ Analytical Modeling of the Problem

- First, we check the basic information of the dataset that is its shape and its information using pandas library. The basic information tells the data type of our column and number of data present.
- Then we check the unique information of our data columns.
- After that we check for null values, if it present then we remove it because our data is text data and we cannot fill it.
- We check the length of each message to understand the distribution of data.
- We use seaborn library to plot the target data and using word cloud for getting the sense of loud words in Spam & Ham messages.
- In order to build a machine learning model for spam classification, it is necessary to first obtain a labelled dataset of SMS messages. This dataset should include both spam and non-spam messages, and should be representative of the types of messages that the model will be used to classify. Once the dataset has been obtained, it will be necessary to pre-process the data in order to extract features that can be used to train the model. This may involve techniques such as tokenization, stemming, and stop word removal.
- Once the data has been pre-processed, it can be split into training and testing sets. The training set is used to train the model, while the testing set is used to evaluate the model's performance. There are a variety of evaluation metrics that can be used to assess the performance of a spam classification model, including accuracy, precision, and recall.

Data Sources and their formats

- In this project the dataset was given in csv (comma separated values) format.
- The files contain one message per line. Each line is composed by two columns: v1 contains the label (ham or spam) and v2 contains the raw text.
- This corpus has been collected from free or free for research sources at the Internet:
    - A collection of 5573 rows SMS spam messages was manually extracted from the Grumble text Web site. This is a UK forum in which cell phone users make public claims about SMS spam messages, most of them without reporting the very spam message received. The identification of the text of spam messages in the claims is a very hard and time-consuming task, and it involved carefully scanning hundreds of web pages.
    - A subset of 3,375 SMS randomly chosen ham messages of the NUS SMS Corpus (NSC), which is a dataset of about 10,000 legitimate messages collected for research at the Department of Computer Science at the National University of Singapore. The messages largely originate from Singaporeans and mostly from students attending the University. These messages were collected from volunteers who were made aware that their contributions were going to be made publicly available.

Data Pre-processing Done

1. First, we check the information of the given dataset because it tells that how many rows and columns are present in our dataset and data type of the columns whether they are object, integer or float.
2. Dropping duplicates rows if present in dataset.
3. Then we check for the null values present in our dataset. If null values are present then drop it because we cannot able to fill the text data.
4. After that we check the summary statistics of our dataset. This part talks about the statistics of our dataset i.e., mean, median, max value, in values and also it tells whether outliers are present in our dataset or not.
5. In data cleaning we use mainly five steps using function:
    a. Removing HTML tags
    b. Removing special characters
    c. Converting everything to lowercase
    d. Removing stop words
    e. Using WordNet Lemmatization for lemmatization
    f. For vectorization we used SKLearn's CountVectorizer, TfidfVectorizer & TensorFlow's TextVectorization
    g. We checked for imbalanced data & found it. We applied Oversampling technique SMOTE to balance the dataset.
    h. We applied label encoder on our target feature
6. We create new column (clean text) after removing punctuations, stop words from input feature to check how much data is cleaned.

Data Inputs- Logic- Output Relationships

Data is fed in the form of a Pandas data frame to the model. The data is the vectorised meaningful words of the records. For the output/target we get the predicted label value of the record, that is whether the message is likely to be a spam or ham. The output results in a binary value either 1 or 0 respectively.

State the set of assumptions (if any) related to the problem under Consideration

- All stop words are present inside the stop words library
- The offensive words are always explicit and sarcasms are not considered

Hardware and Software Requirements and Tools Used

a. Software

i. ➔ Jupyter Notebook (Python 3.9)
ii. ➔ Microsoft Office

b. Hardware

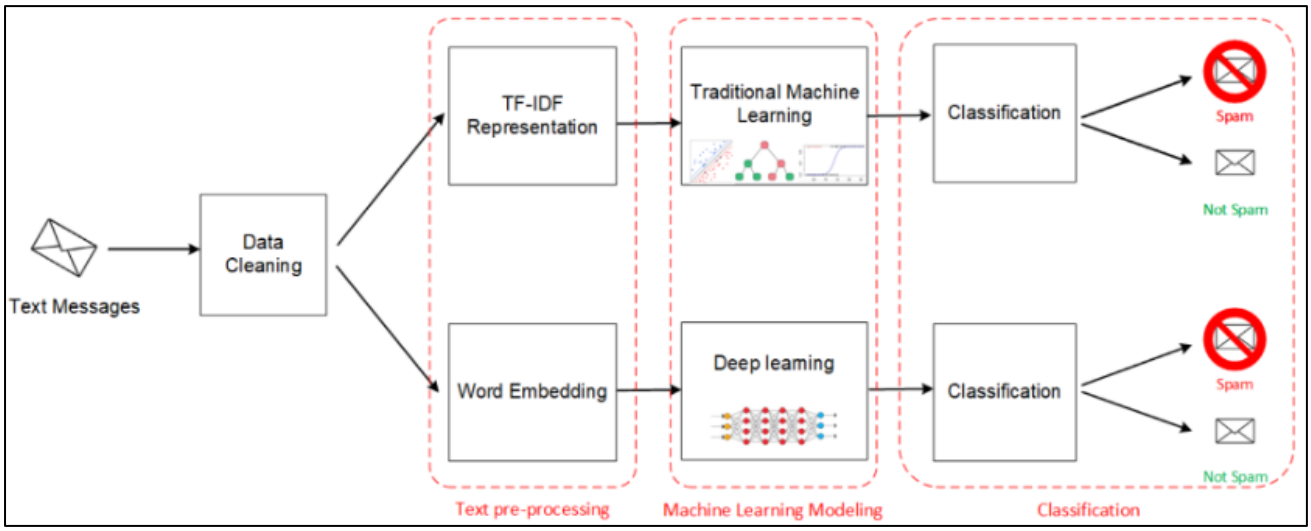
i. ➔ Processor – AMD Ryzen 5
ii. ➔ RAM - 8 GB
iii. ➔ Graphic Memory - 4Gb, Nvidia GEFORCE RTX1650

c. Python Libraries

i. ➔ Pandas
ii. ➔ Numpy
iii. ➔ Matplotlib
iv. ➔ Seaborn
v. ➔ Scipy
vi. ➔ Sklearn
vii. ➔ NLTK
viii. ➔ re

MODEL DEVELOPMENT AND EVALUATION

Identification of possible problem-solving approaches (methods)



Testing of Identified Approaches (Algorithms)

We have used 3 approaches as follows:

1. Using sklearn's **CountVectorizer** as a text vectorisation method & applying the Naïve bayes algorithms.

	algo	best random state	train_accuracy	test_accuracy	Score_diff	best cv fold	cross_val_score
1	Multinomial NB Classifier	129	88.19	88.18	0.28	5	87.898103
2	Bernoulli NB Classifier	74	97.42	97.34	0.64	11	96.701117
0	Gaussian NB Classifier	112	96.80	94.91	1.66	3	93.246296

2. Using sklearn's **TfidfVectorizeras** as a text vectorisation method & applying the Naïve bayes algorithms.

	algo	best random state	train_accuracy	test_accuracy	Score_diff	best cv fold	cross_val_score
1	Multinomial NB Classifier	74	97.74	97.74	0.00	10	97.741550
2	Bernoulli NB Classifier	111	98.85	98.85	0.05	10	98.804611
0	Gaussian NB Classifier	97	93.98	93.98	0.34	10	93.644881

3. Using TensorFlow's **TextVectorization** as a text vectorisation method & applying on LSTM neural network.

```
Precision: 0.9983525276184082, Recall:0.9105935096740723, Accuracy:0.5
```

Run and evaluate selected models

```
from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer
cv = CountVectorizer()
tfidf = TfidfVectorizer(max_features=3000)
```

```
X1 = cv.fit_transform(data3['text_lemmatize']).toarray()
X2 = tfidf.fit_transform(data3['text_lemmatize']).toarray()
y = data3["Target"]
```

```
from tensorflow.keras.layers import TextVectorization
```

```
X = data3['text_lemmatize']
y = data3["Target"]
```

```
MAX_FEATURES = 10000 # number of words in the vocab
```

```
vectorizer = TextVectorization(max_tokens=MAX_FEATURES,
                               output_sequence_length=2500,
                               output_mode='int')
```

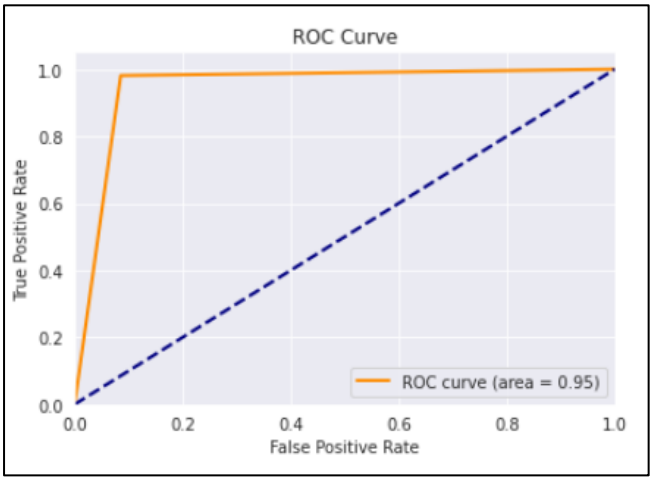
```
vectorizer.adapt(X.values)
vectorized_text = vectorizer(X.values)
```

1. Using sklearn's **CountVectorizer** as a text vectorisation method & applying the Naïve bayes algorithms.

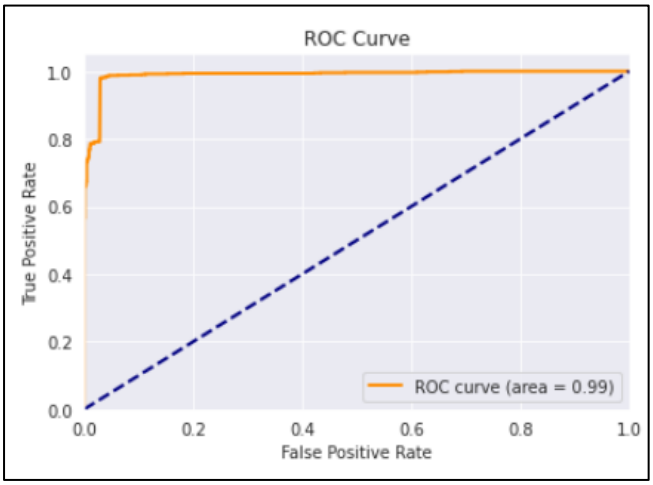
```
gnb = GaussianNB()
mnb = MultinomialNB()
bnb = BernoulliNB()
```

```
models = [gnb,mnb,bnb]
models_name = ['Gaussian NB Classifier',
               'Multinomial NB Classifier',
               'Bernoulli NB Classifier']
```

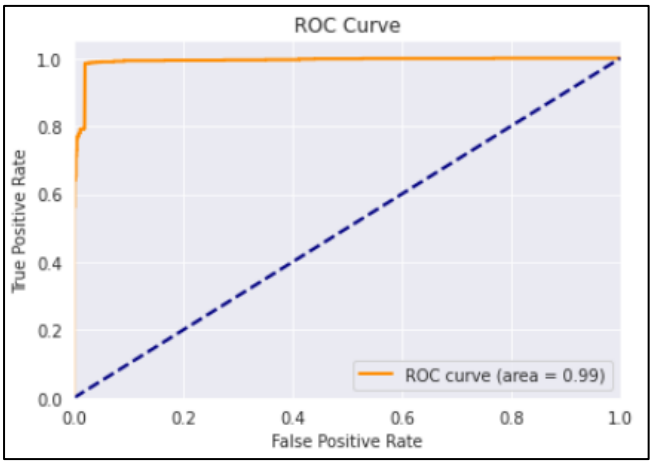
```
for Gaussian NB Classifier model..

Best Random_state number for splitting the data is:  112

Accuracy score for Train : 96.80%
Accuracy score for Test : 94.91%
====================
Confusion Matrix :

[[ 997   93]
 [  22 1146]]
====================


 Test Classification report
              precision    recall  f1-score   support

           0       0.98      0.91      0.95      1090
           1       0.92      0.98      0.95      1168

    accuracy                           0.95      2258
   macro avg       0.95      0.95      0.95      2258
weighted avg       0.95      0.95      0.95      2258

Cross Validation score at best cv=3 is : 93.25%
```



```
for Multinomial NB Classifier model..

Best Random_state number for splitting the data is:  129

Accuracy score for Train : 88.19%
Accuracy score for Test : 88.18%
====================
Confusion Matrix :

[[1094   32]
 [ 235  897]]
====================


 Test Classification report
              precision    recall  f1-score   support

           0       0.82      0.97      0.89      1126
           1       0.97      0.79      0.87      1132

    accuracy                           0.88      2258
   macro avg       0.89      0.88      0.88      2258
weighted avg       0.89      0.88      0.88      2258

Cross Validation score at best cv=5 is : 87.90%
```




```
for Bernoulli NB Classifier model..

Best Random_state number for splitting the data is:  74

Accuracy score for Train : 97.42%
Accuracy score for Test : 97.34%
====================
Confusion Matrix :

[[1087   46]
 [  14 1111]]
====================


 Test Classification report
              precision    recall  f1-score   support

           0       0.99      0.96      0.97      1133
           1       0.96      0.99      0.97      1125

    accuracy                           0.97      2258
   macro avg       0.97      0.97      0.97      2258
weighted avg       0.97      0.97      0.97      2258

Cross Validation score at best cv=11 is : 96.70%
```



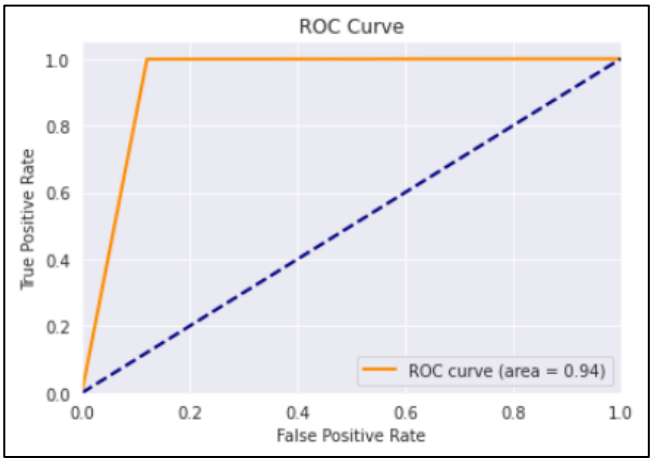
	algo	best random state	train_accuracy	test_accuracy	Score_diff	best cv fold	cross_val_score
1	Multinomial NB Classifier	129	88.19	88.18	0.28	5	87.898103
2	Bernoulli NB Classifier	74	97.42	97.34	0.64	11	96.701117
0	Gaussian NB Classifier	112	96.80	94.91	1.66	3	93.246296

2. Using sklearn's **TfidfVectorizeras** as a text vectorisation method & applying the Naïve bayes algorithms.

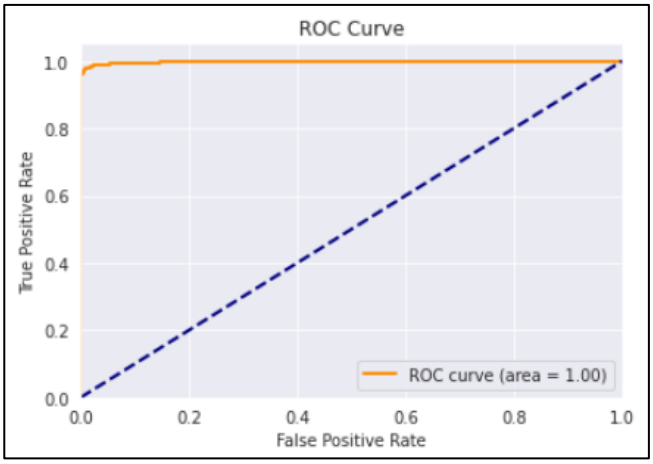
```
gnb2 = GaussianNB()
mnb2 = MultinomialNB()
bnb2 = BernoulliNB()
```

```
models = [gnb2,mnb2,bnb2]
models_name = ['Gaussian NB Classifier',
               'Multinomial NB Classifier',
               'Bernoulli NB Classifier']
```

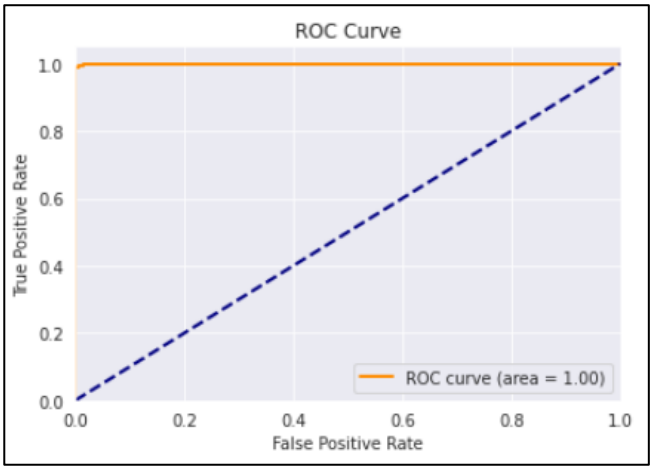
```
for Gaussian NB Classifier model..

Best Random_state number for splitting the data is:  97

Accuracy score for Train : 93.98%
Accuracy score for Test : 93.98%
====================
Confusion Matrix :

[[ 980  135]
 [   1 1142]]
====================


 Test Classification report
              precision    recall  f1-score   support

           0       1.00      0.88      0.94      1115
           1       0.89      1.00      0.94      1143

    accuracy                           0.94      2258
   macro avg       0.95      0.94      0.94      2258
weighted avg       0.95      0.94      0.94      2258

Cross Validation score at best cv=10 is : 93.64%
```




```
for Multinomial NB Classifier model..

Best Random_state number for splitting the data is:  74

Accuracy score for Train : 97.74%
Accuracy score for Test : 97.74%
====================
Confusion Matrix :

[[1095   38]
 [  13 1112]]
====================


 Test Classification report
              precision    recall  f1-score   support

           0       0.99      0.97      0.98      1133
           1       0.97      0.99      0.98      1125

    accuracy                           0.98      2258
   macro avg       0.98      0.98      0.98      2258
weighted avg       0.98      0.98      0.98      2258

Cross Validation score at best cv=10 is : 97.74%
```




```
for Bernoulli NB Classifier model..

Best Random_state number for splitting the data is:  111

Accuracy score for Train : 98.85%
Accuracy score for Test : 98.85%
====================
Confusion Matrix :

[[1104    0]
 [  26 1128]]
====================


 Test Classification report
              precision    recall  f1-score   support

           0       0.98      1.00      0.99      1104
           1       1.00      0.98      0.99      1154

    accuracy                           0.99      2258
   macro avg       0.99      0.99      0.99      2258
weighted avg       0.99      0.99      0.99      2258

Cross Validation score at best cv=10 is : 98.80%
```



	algo	best random state	train_accuracy	test_accuracy	Score_diff	best cv fold	cross_val_score
1	Multinomial NB Classifier	74	97.74	97.74	0.00	10	97.741550
2	Bernoulli NB Classifier	111	98.85	98.85	0.05	10	98.804611
0	Gaussian NB Classifier	97	93.98	93.98	0.34	10	93.644881

3. Using TensorFlow's **TextVectorization** as a text vectorisation method & applying on LSTM neural network.

```
dataset = tf.data.Dataset.from_tensor_slices((vectorized_text, Y))
dataset = dataset.cache()
dataset = dataset.shuffle(800)
dataset = dataset.batch(16)
dataset = dataset.prefetch(8) # helps bottlenecks
```

```
train = dataset.take(int(len(dataset)*.7))
val = dataset.skip(int(len(dataset)*.7)).take(int(len(dataset)*.15))
test = dataset.skip(int(len(dataset)*.85)).take(int(len(dataset)*.15))
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dropout, Bidirectional, Dense, Embedding
```

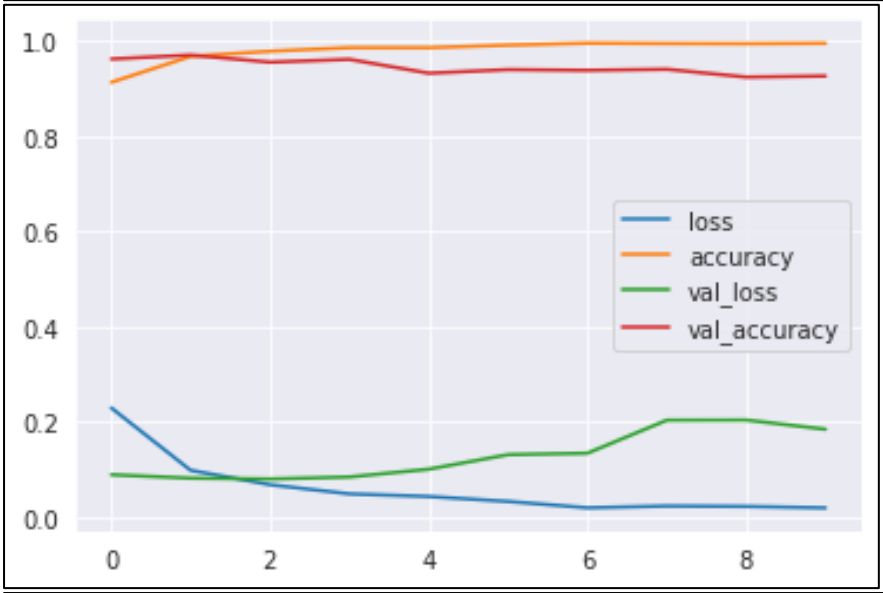
```
model = Sequential()
# Create the embedding layer
model.add(Embedding(MAX_FEATURES, 32))

# Bidirectional LSTM Layer
model.add(Bidirectional(LSTM(32, activation='tanh')))

# Feature extractor Fully connected layers with dropout layers
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))

# Final Layer
model.add(Dense(1, activation='sigmoid'))
```

```
opt = tf.keras.optimizers.Adam(learning_rate=0.001)
model.compile(loss='BinaryCrossentropy', optimizer=opt, metrics=['accuracy'])
```

```
Layer (type)                 Output Shape              Param #
=================================================================
embedding_4 (Embedding)      (None, None, 32)          320000

bidirectional_4 (Bidirectio  (None, 64)               16640
nal)

dense_15 (Dense)             (None, 128)               8320

dropout_6 (Dropout)          (None, 128)               0

dense_16 (Dense)             (None, 128)               16512

dropout_7 (Dropout)          (None, 128)               0

dense_17 (Dense)             (None, 64)                8256

dense_18 (Dense)             (None, 32)                2080

dense_19 (Dense)             (None, 1)                 33

=================================================================
Total params: 371,841
Trainable params: 371,841
Non-trainable params: 0
_________________________________________________________________
```

```
history = model.fit(train, epochs=10, batch_size = 32, validation_data=val,  callbacks=[callback])
```

```
Epoch 1/10
395/395 [==============================] - 53s 120ms/step - loss: 0.2298 - accuracy: 0.9123 - val_loss: 0.0897 - val_accuracy:
0.9613
Epoch 2/10
395/395 [==============================] - 45s 114ms/step - loss: 0.0989 - accuracy: 0.9668 - val_loss: 0.0824 - val_accuracy:
0.9695
Epoch 3/10
395/395 [==============================] - 46s 116ms/step - loss: 0.0687 - accuracy: 0.9775 - val_loss: 0.0807 - val_accuracy:
0.9546
Epoch 4/10
395/395 [==============================] - 46s 116ms/step - loss: 0.0495 - accuracy: 0.9851 - val_loss: 0.0849 - val_accuracy:
0.9606
Epoch 5/10
395/395 [==============================] - 46s 116ms/step - loss: 0.0440 - accuracy: 0.9851 - val_loss: 0.1012 - val_accuracy:
0.9315
Epoch 6/10
395/395 [==============================] - 46s 116ms/step - loss: 0.0336 - accuracy: 0.9903 - val_loss: 0.1317 - val_accuracy:
0.9390
Epoch 7/10
395/395 [==============================] - 46s 116ms/step - loss: 0.0203 - accuracy: 0.9945 - val_loss: 0.1348 - val_accuracy:
0.9375
Epoch 8/10
395/395 [==============================] - 48s 121ms/step - loss: 0.0240 - accuracy: 0.9937 - val_loss: 0.2040 - val_accuracy:
0.9397
Epoch 9/10
395/395 [==============================] - 47s 119ms/step - loss: 0.0232 - accuracy: 0.9934 - val_loss: 0.2043 - val_accuracy:
0.9234
Epoch 10/10
395/395 [==============================] - 47s 118ms/step - loss: 0.0202 - accuracy: 0.9941 - val_loss: 0.1850 - val_accuracy:
0.9256
```



X axis – No. of Epochs

Y axis - Loss

```
from tensorflow.keras.metrics import Precision, Recall, CategoricalAccuracy
pre = Precision()
re = Recall()
acc = CategoricalAccuracy()

for batch in test.as_numpy_iterator():
    # Unpack the batch
    X_true, y_true = batch
    # Make a prediction
    yhat = model.predict(X_true)

    # Flatten the predictions
    y_true = y_true.flatten()
    yhat = yhat.flatten()

    pre.update_state(y_true, yhat)
    re.update_state(y_true, yhat)
    acc.update_state(y_true, yhat)


print(f'Precision: {pre.result().numpy()}, Recall:{re.result().numpy()}, Accuracy:{acc.result().numpy()}')
```

```
Precision: 0.9983525276184082, Recall:0.9105935096740723, Accuracy:0.5
```

We selected ***sklearn's Tfidf Vectorizer as a text vectorisation method & Bernoulli NB Classifier*** for the following reasons:

- Maximum ACCURACY with high Precision value.
- Maximum Value of area under ROC curve

Key Metrics for success in solving problem under consideration

Following metrics used for evaluation:

- **Accuracy Score** - Dataset is balanced after oversampling method, so first we will look at best accuracy score of all the models.
- **F1 - Score** - in this data set the target will decide the message is spam or not, hence 0 and 1, and as both zero and one is important to us therefore recall and precision what will be our preferred metric and as we all know that, it combines precision and recall into one metric by calculating the harmonic mean between those two.
- **AUC ROC** - We can see a healthy ROC curve, pushed towards the top-left side both for positive and negative classes.
- **Cross Validation Score** - to check if our model is overfitting or not, we use cross validation score, higher the cross-validation scores higher the cross-validation score means the model is not overfitting.

Hyperparameter Tuning:

Hyper parameter tuning for Bernoulli NB Classifier on the TFIDF vectorised data

```
X2 = tfidf.fit_transform(data3['text_lemmatize']).toarray()
y = data3["Target"]

oversample_final = SMOTE()
X2, y = oversample.fit_resample(X2, y)
y.value_counts()

x_train, x_test, y_train, y_test = train_test_split(X2, y, test_size = 0.25, random_state = 111)
```

```
param_grid_bnb = {'alpha':[0.001, 0.002, 0.005, 0.05, 0.02, 0.01, 0.1, 0.2, 0.5, 1]}
```

```
bnb_grid = GridSearchCV(estimator = bnb2,
                        param_grid = param_grid_bnb,
                        verbose = 2,
                        scoring = 'accuracy')
```

```
bnb_grid.fit(x_train, y_train)
```

...

```
bnb_grid.best_score_
```

```
0.9898138629835339
```

```
bnb_grid.best_params_
```

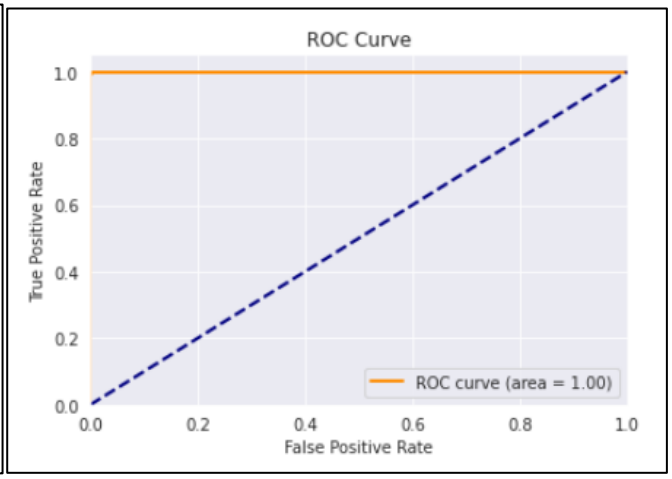
```
{'alpha': 0.02}
```

```
bnb2_final = BernoulliNB(alpha = 0.02)
```

```
bnb2_final.fit(x_train,y_train)
y_pred=bnb2_final.predict(x_test)
print('Accuracy Score: ', accuracy_score(y_test, y_pred))
```

```
Accuracy Score:  0.9902568644818424
```

```
====================
Confusion Matrix :

[[1104    0]
 [  22 1132]]
====================


 Test Classification report
              precision    recall  f1-score   support

           0       0.98      1.00      0.99      1104
           1       1.00      0.98      0.99      1154

    accuracy                           0.99      2258
   macro avg       0.99      0.99      0.99      2258
weighted avg       0.99      0.99      0.99      2258

Cross Validation score at best cv = 11 is : 99.06%
```



Saving & predictions of the model on Test data

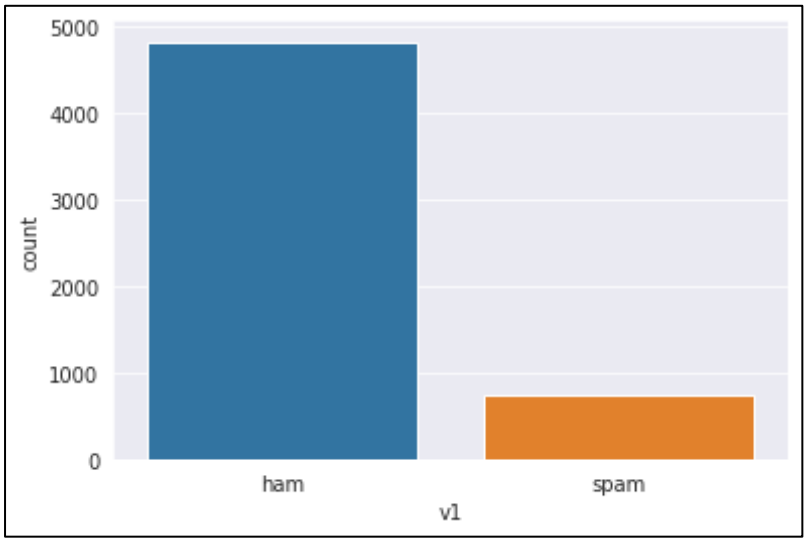
```
1  import pickle
```

```
1  filename='BernoulliNB_sms_spam_alpha-0.02.pkl'
2  pickle.dump(bnb2_final,open(filename,'wb'))
```

```
1  model = pickle.load(open('BernoulliNB_sms_spam_alpha-0.02.pkl','rb'))
2  pred =model.predict(x_test)
3  result = pd.DataFrame(list(zip(y_test, pred)), columns = ['Actual', 'Predicted'])
4  result
```

	Actual	Predicted
0	1	1
1	1	1
2	1	1
3	1	1
4	0	0
...	...	...
2253	0	0
2254	1	1
2255	0	0
2256	0	0
2257	1	1

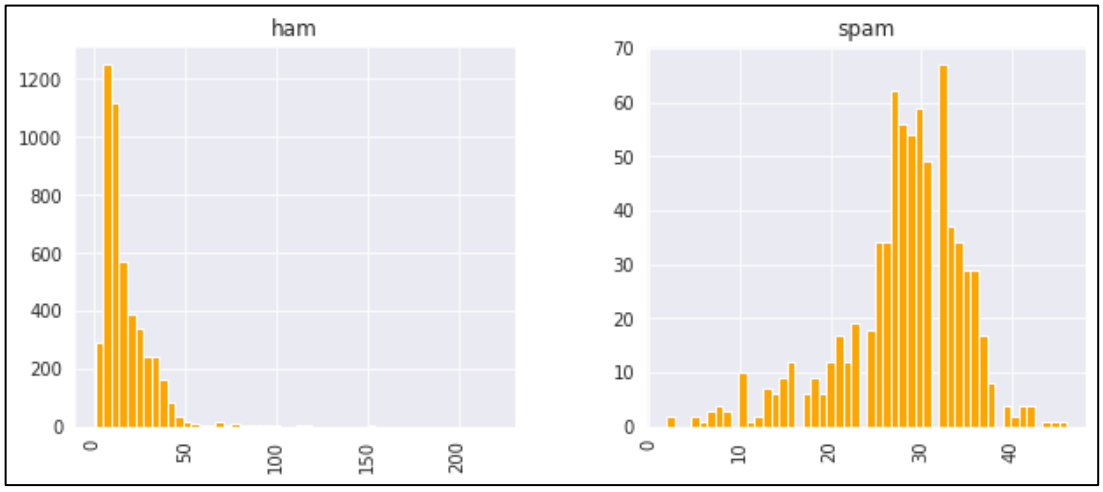
VISUALIZATIONS & EDA



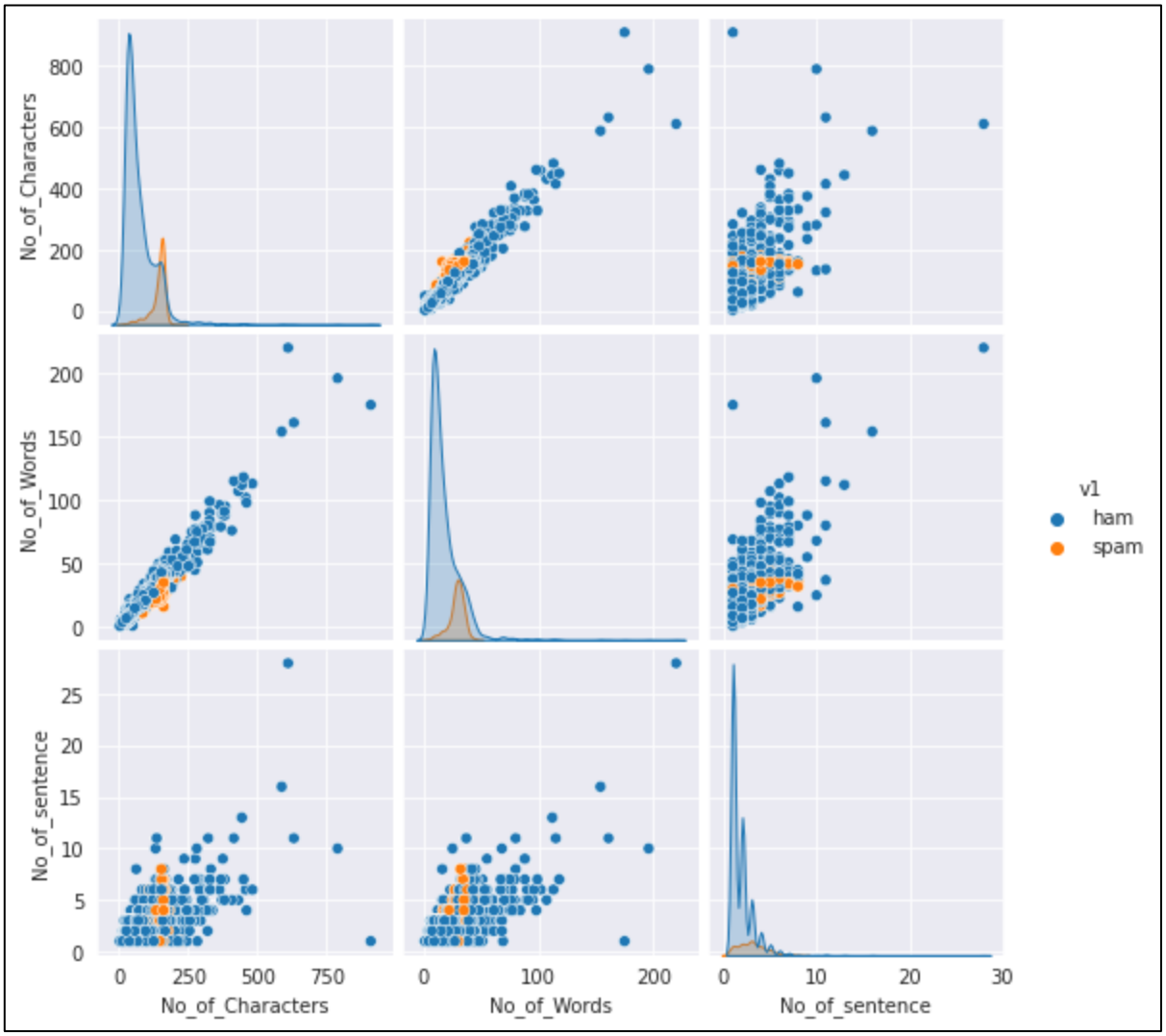
The Input data had data imbalance. We used oversampling technique (SMOTE) to balance the dataset.

	unique_entries	missing values	Dtypes
v1	2	0	object
v2	5169	0	object
Unnamed: 2	43	5522	object
Unnamed: 3	10	5560	object
Unnamed: 4	5	5566	object

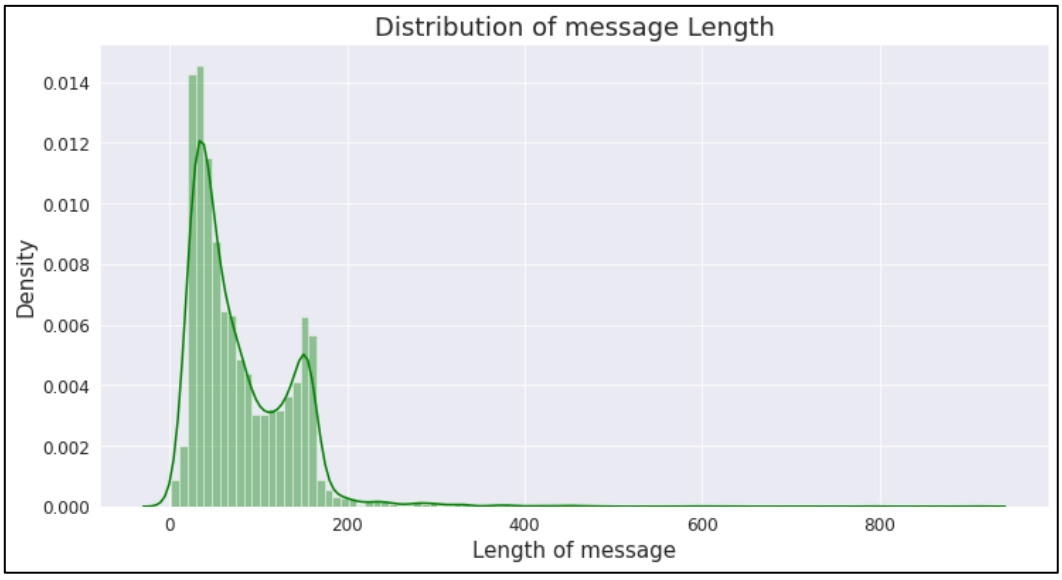
The Input data had 3 unnamed features with unique entries less than 45.
We checked the dataset & found out that those values were extended text of the same message, we concatenated those text & deleted those 3 features.



Messages with text length less than 100 were actually not spam.
But those texts with text length less than 40 were spam class.



From above plot we can say Lengthy text messages are not to be considered spam.



Overall, the dataset has text messages varying from 0 to 200 text length. Those above 200 are seem to be outliers.

Messages classified as Ham were having most common words like, ***[u, go, say, name]***

Messages classified as Spam were having most common words like, ***[free, mobile, pound, win]***

CONCLUSION

Key Findings and Conclusions of the Study

- Naïve bayes algorithms are best for spam classification problems.
- LSTM model does not perform better as compared with all 3 naïve bayes model used in the project.

Learning Outcomes of the Study in respect of Data Science

- In this project we learn how to build a machine learning model for classification-based problem. The pre processing part is always crucial for any type of machine learning / NLP project.
- The goal of any machine learning problem is to find a single model that will best predict our wanted outcome. So based on all the learning and outcomes our Bernoulli Naïve Bayes Classifier Model gives the best result so we save this model as our final model for future data/message predictions.

Limitations of this work and Scope for Future Work

- Data availability and quality: In order to build a reliable spam classification model, it is important to have access to a large and diverse dataset of labeled SMS messages. If the available dataset is small or not representative of the types of messages that the model will encounter in the real world, the model's performance may be limited.
- Preprocessing challenges: Preprocessing text data for NLP tasks can be challenging, particularly when working with SMS messages, which may contain a lot of spelling errors, abbreviations, and unconventional punctuation. These challenges can make it difficult to extract useful features from the data and can negatively impact the model's performance.
- Model complexity and overfitting: Choosing a machine learning model that is too complex for the available data can lead to overfitting, which means that the model will perform well on the training data but poorly on new, unseen data. On the other hand, choosing a model that is too simple may result in poor performance on both the training and test data.
- Limited generalizability: A spam classification model that is trained on a specific dataset may not generalize well to other types of SMS messages or to messages written in different languages. This can limit the model's usefulness in real-world applications.
- Ethical considerations: It is important to consider the ethical implications of building a spam classification model, particularly in terms of privacy and the potential for biased results.