Spam Detection in SMS Text Messages

IBM Advanced Data Science Capstone

Eman Mousavinejad

March 2022

Introduction

Spam is an unwanted message sent electronically whose content may be malicious. The spam messages are dangerous in many ways such as

- Exposure of private and personal information through undesired advertisement
- Falling victim to a fraud or financial scheme
- Being wrongly attracted into malware and phishing websites
- Being involuntary exposed to inappropriate online content

Our Purpose

The main purpose of this project is to build a spam detection model. Particularly, our focus lies on building a binary classification model to detect whether the text messages are spam or ham (not spam).

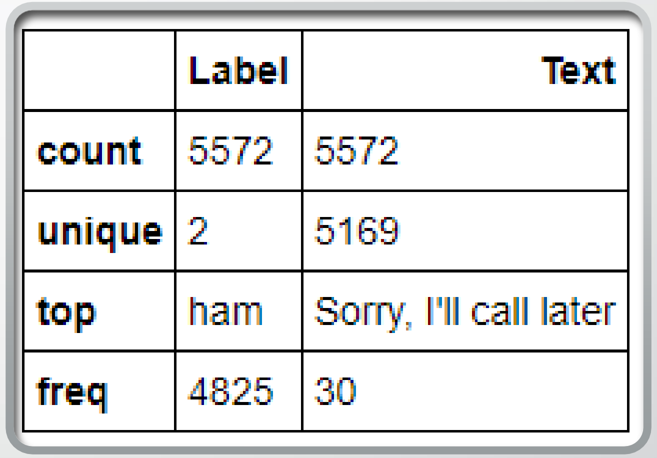
Data Collection

Data source is collected from UCI Machine Learning Repository as a CSV file with the total of 5572 SMS text messages categorized with ham and spam labels.

Exploratory Data Analysis (EDA)

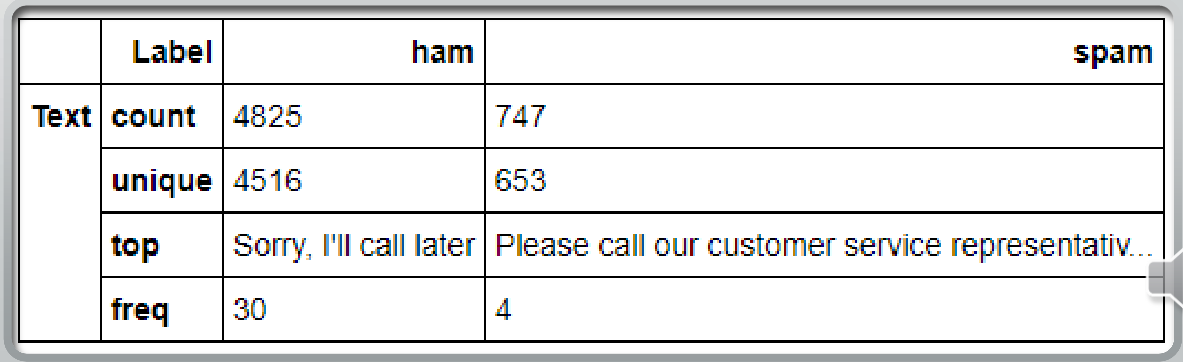
Describe method:

- There are 5572 labels and messages
- There are two unique labels "ham" and "spam"
- There are 5169 unique messages and 403 duplicated messages
- The top label is "ham" and the top message is "Sorry, I'll call you later"



	Label	Text
count	5572	5572
unique	2	5169
top	ham	Sorry, I'll call later
freq	4825	30

Group by labels:

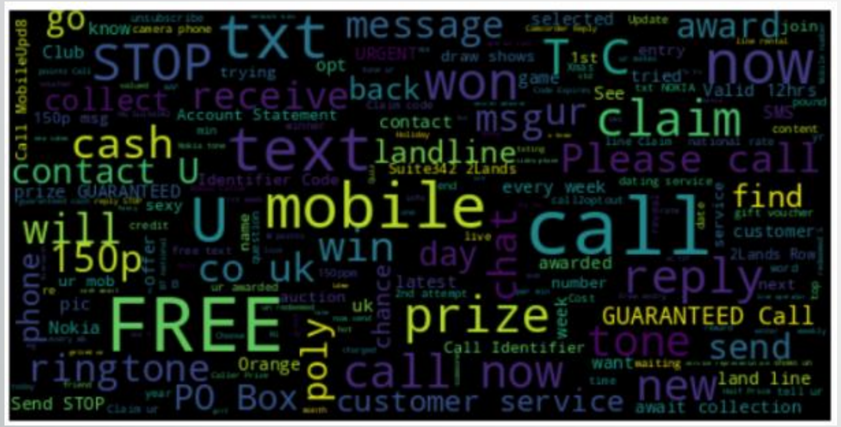


	Label	ham	spam
Text	count	4825	747
	unique	4516	653
	top	Sorry, I'll call later	Please call our customer service representativ...
	freq	30	4

EDA (Cont.)

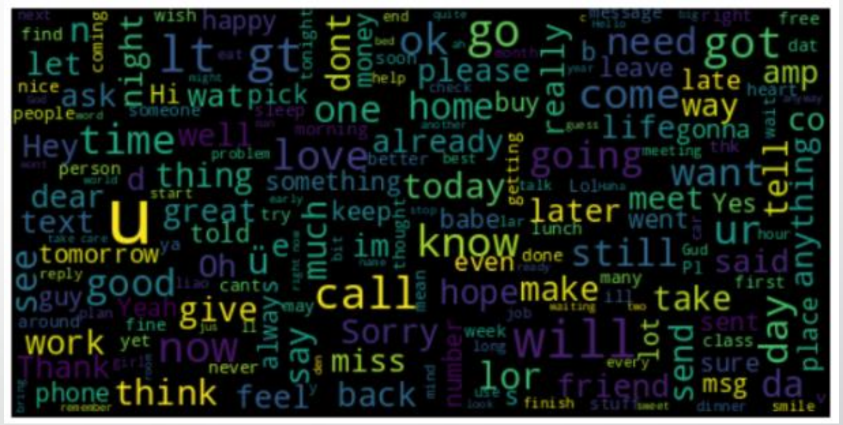
WordCloud – Visualizing the most frequent words in the text messages.

WordCloud for Spam Messages



The most frequent words: free, call, text, claim, reply

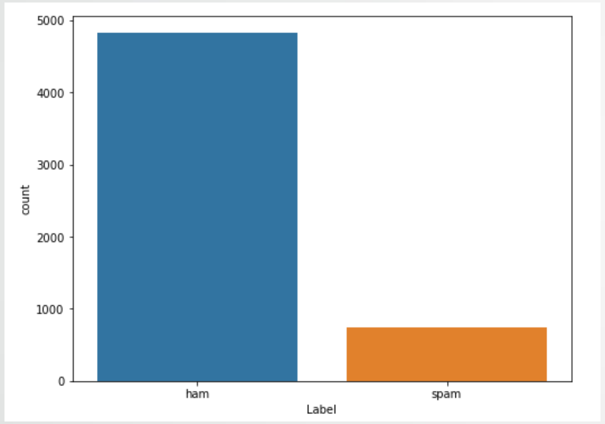
WordCloud for Ham Messages



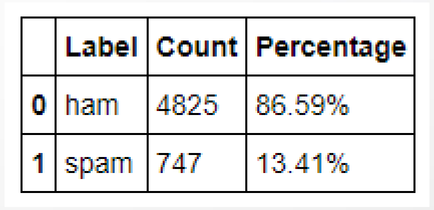
The most frequent words: now, work, how, Ok, sorry

EDA (Cont.)

Bar Chart – Demonstrating the distribution of ham and spam SMS messages.

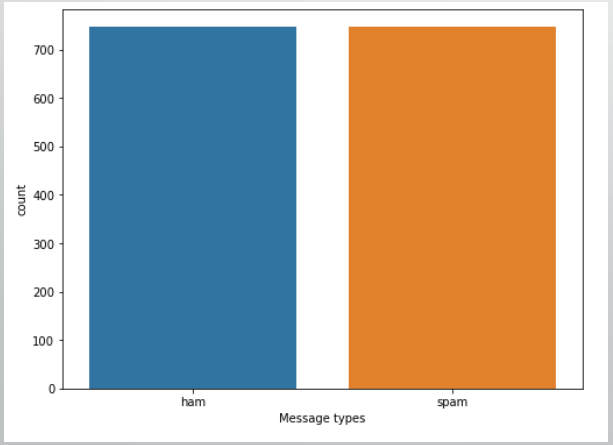


The data is imbalanced



	Label	Count	Percentage
0	ham	4825	86.59%
1	spam	747	13.41%

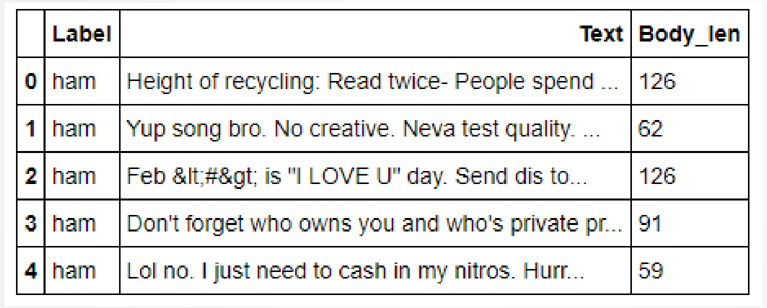
Bar Chart – Demonstrating the distribution of ham and spam SMS messages after downsampling.



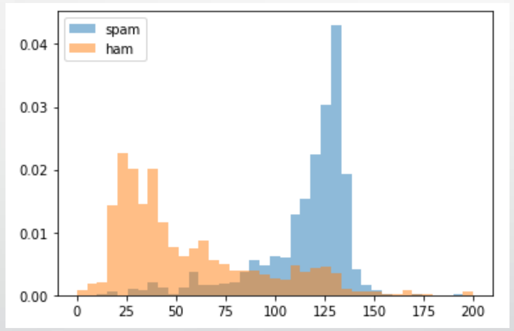
There are 747 spam and ham messages

Feature Creation

We assume that spam messages tend to be longer than ham text messages. We calculate the length of each message and add a new column as **Body_len** to collect this information for each row.



	Label	Text	Body_len
0	ham	Height of recycling: Read twice- People spend ...	126
1	ham	Yup song bro. No creative. Neva test quality. ...	62
2	ham	Feb &lt;#&gt; is "I LOVE U" day. Send dis to...	126
3	ham	Don't forget who owns you and who's private pr...	91
4	ham	Lol no. I just need to cash in my nitros. Hurr...	59



Body length is very different for ham versus spam. Spam text messages seem to be quite a bit longer than ham text messages. So, we can conclude that this extra feature can be really helpful for the model to distinguish ham from spam.

We save our new dataframe in the IBM Cloud Object Storage with Parquet format

Data Processing

We'll use text processing which include:

- Remove punctuations which are !"#$%&'()*+,-./:;<=>?@[]^_`{|}~ (regex_replace function from pyspark.sql.functions)
- Tokenization: splits sentences into words and convert all words to lower case (Tokenizer in Spark Mllib)
- Remove stop words which do not add much meaning to a sentence, such as the, he, she, have, … (StopWordsRemover in Spark MLlib)
- Convert categorical labels into numeric ones which will be later used for binary classification (StringIndexer in Spark Mllib)

Vectorization and Feature Engineering

The process that converts text to a form that machine can understand is called **vectorizing**. The most popular vectorization methods are

- Count vectorizing
- N-gram vectorizing
- Term Frequency-Inverse Document Frequency (TF-IDF) vectorizing

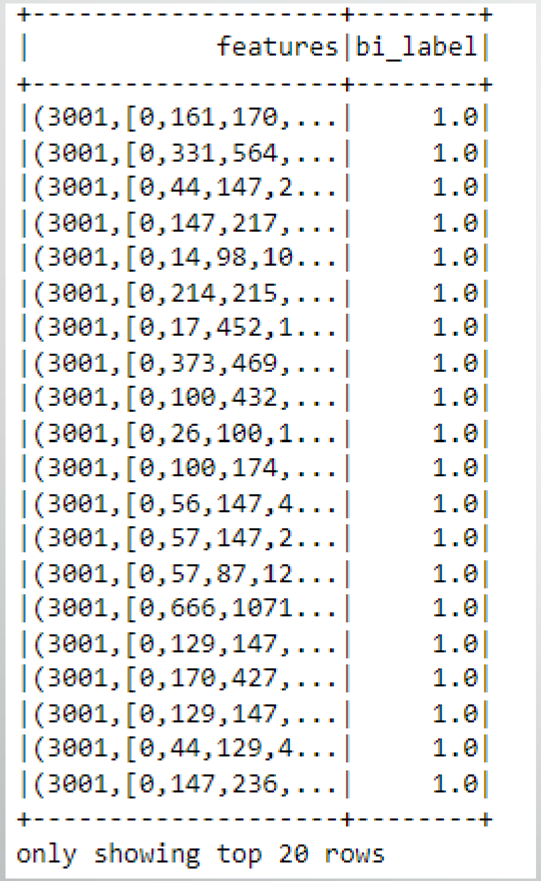
In this project, we use (TF-IDF) vectorizer from Spark Mllib.

Furthermore, we combine the features generated by TF-IDF vectorizer and Body-len through VectorAssembler transformer from Spark Mllib.

Finally, we create a pipeline so that we can apply all these steps into our dataset, except for the punctuation removal, which has been done before applying the pipeline.

Vectorization and Feature Engineering

Finally, we just keep the feature and label columns and drop the unnecessary columns.



```
+--------------------+--------+
|            features|bi_label|
+--------------------+--------+
|(3001,[0,161,170,...|     1.0|
|(3001,[0,331,564,...|     1.0|
|(3001,[0,44,147,2...|     1.0|
|(3001,[0,147,217,...|     1.0|
|(3001,[0,14,98,10...|     1.0|
|(3001,[0,214,215,...|     1.0|
|(3001,[0,17,452,1...|     1.0|
|(3001,[0,373,469,...|     1.0|
|(3001,[0,100,432,...|     1.0|
|(3001,[0,26,100,1...|     1.0|
|(3001,[0,100,174,...|     1.0|
|(3001,[0,56,147,4...|     1.0|
|(3001,[0,57,147,2...|     1.0|
|(3001,[0,57,87,12...|     1.0|
|(3001,[0,666,1071...|     1.0|
|(3001,[0,129,147,...|     1.0|
|(3001,[0,170,427,...|     1.0|
|(3001,[0,129,147,...|     1.0|
|(3001,[0,44,129,4...|     1.0|
|(3001,[0,147,236,...|     1.0|
+--------------------+--------+
only showing top 20 rows
```

We save this dataframe in the IBM Cloud Object Storage with Parquet format so as to use it for ML and DL training and testing datasets.

Choice of model for ML and DL

ML Algorithms

- Logistic Regression Classifier
- Naive Bayes Classifier
- Random Forest Classifier

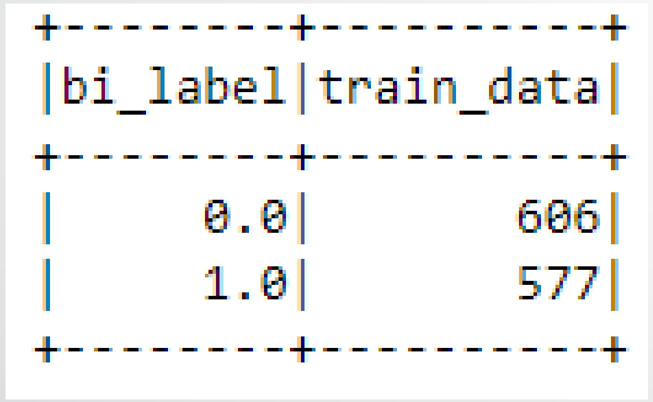
DL Algorithm

Feed Forward Neural Network/Multi-Layer Perceptron for our model

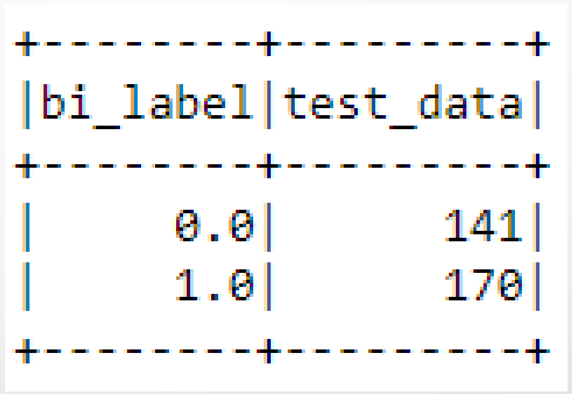
Data Splitting

we split the data into training and test datasets - 80% training data and 20% test data.

The following tables show how the class labels are split between the training and test data sets:

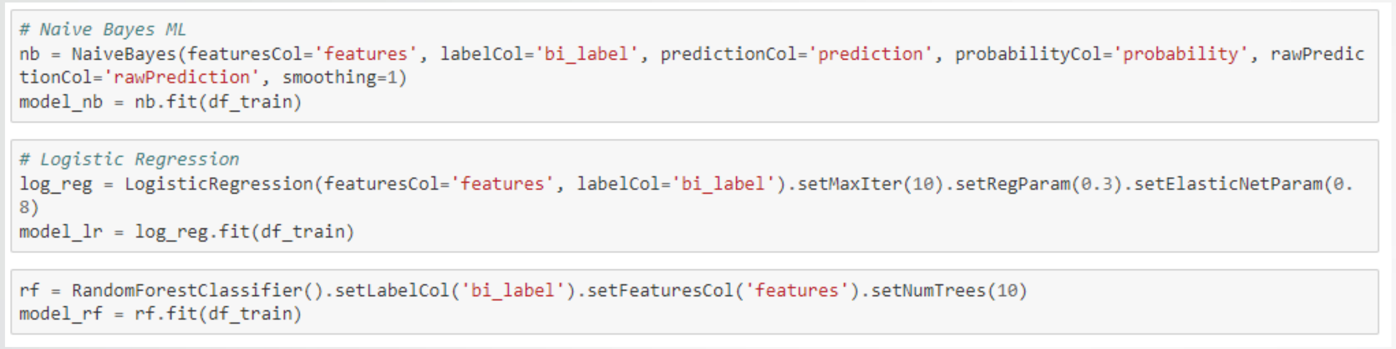


```
+--------+----------+
|bi_label|train_data|
+--------+----------+
|     0.0|       606|
|     1.0|       577|
+--------+----------+
```



```
+--------+---------+
|bi_label|test_data|
+--------+---------+
|     0.0|      141|
|     1.0|      170|
+--------+---------+
```

ML Models Definitions

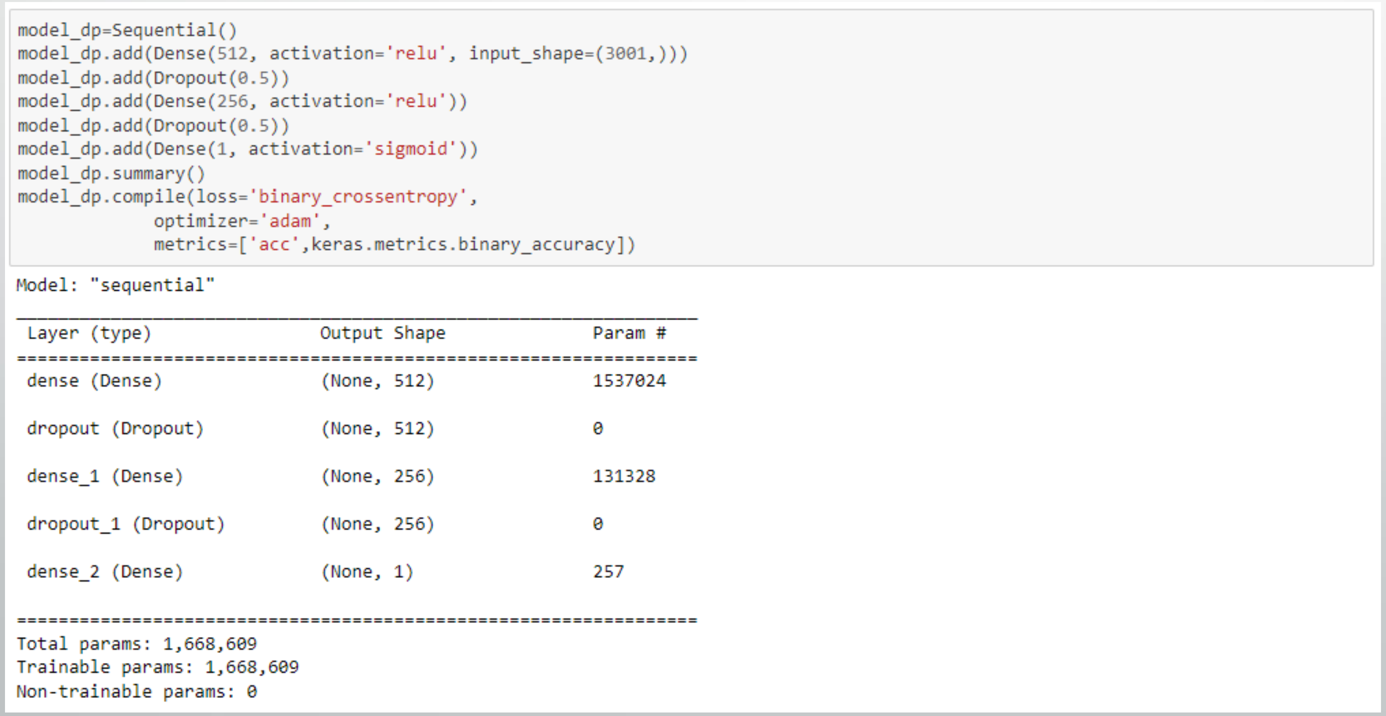


```
# Naive Bayes ML
nb = NaiveBayes(featuresCol='features', labelCol='bi_label', predictionCol='prediction', probabilityCol='probability', rawPredic
tionCol='rawPrediction', smoothing=1)
model_nb = nb.fit(df_train)
```

```
# Logistic Regression
log_reg = LogisticRegression(featuresCol='features', labelCol='bi_label').setMaxIter(10).setRegParam(0.3).setElasticNetParam(0.
8)
model_lr = log_reg.fit(df_train)
```

```
rf = RandomForestClassifier().setLabelCol('bi_label').setFeaturesCol('features').setNumTrees(10)
model_rf = rf.fit(df_train)
```

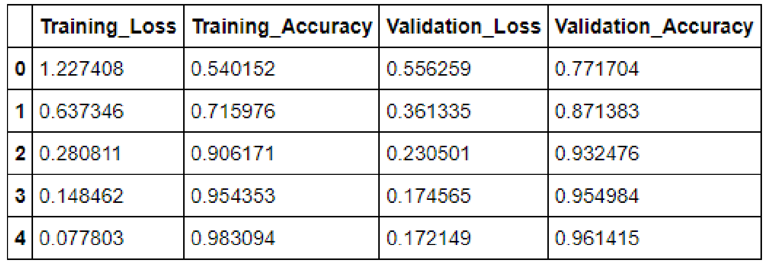
DL Model Definition



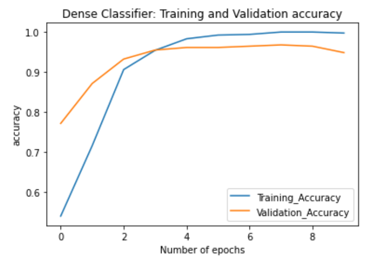
```
model_dp=Sequential()
model_dp.add(Dense(512, activation='relu', input_shape=(3001,)))
model_dp.add(Dropout(0.5))
model_dp.add(Dense(256, activation='relu'))
model_dp.add(Dropout(0.5))
model_dp.add(Dense(1, activation='sigmoid'))
model_dp.summary()
model_dp.compile(loss='binary_crossentropy',
             optimizer='adam',
             metrics=['acc',keras.metrics.binary_accuracy])
```

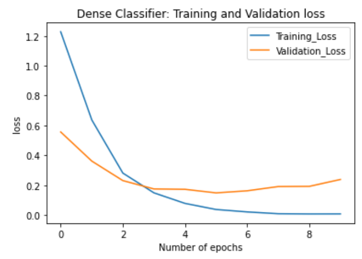
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 512)               1537024

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 256)               131328

 dropout_1 (Dropout)         (None, 256)               0

 dense_2 (Dense)             (None, 1)                 257

=================================================================
Total params: 1,668,609
Trainable params: 1,668,609
Non-trainable params: 0
```

DL model loss and accuracy

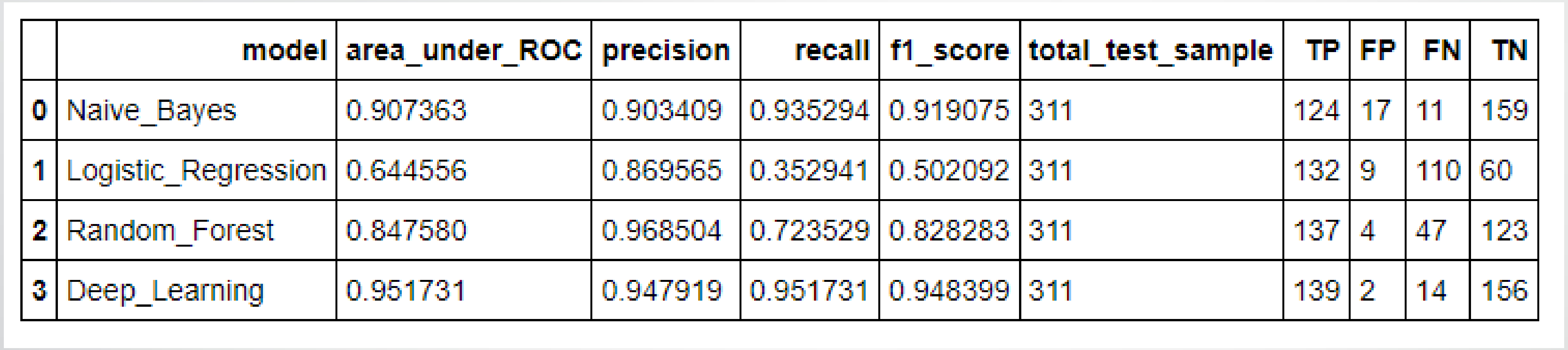


	Training_Loss	Training_Accuracy	Validation_Loss	Validation_Accuracy
0	1.227408	0.540152	0.556259	0.771704
1	0.637346	0.715976	0.361335	0.871383
2	0.280811	0.906171	0.230501	0.932476
3	0.148462	0.954353	0.174565	0.954984
4	0.077803	0.983094	0.172149	0.961415





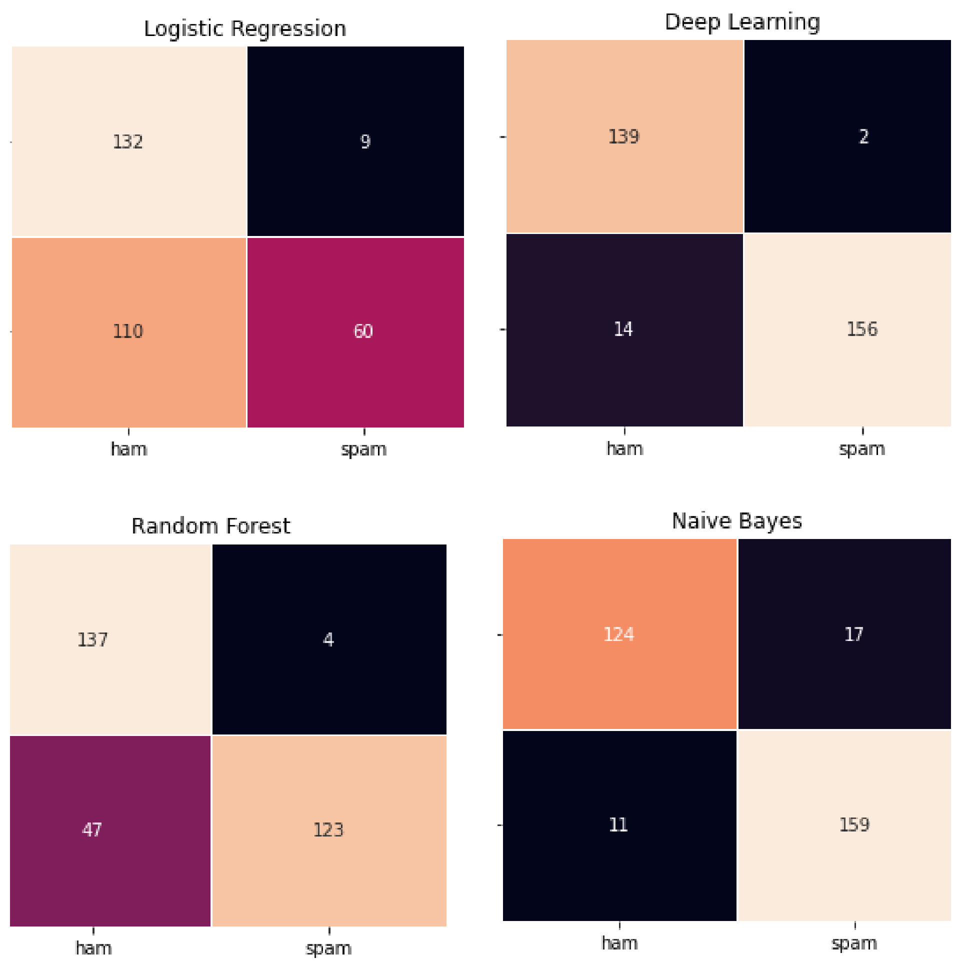
Model Evaluation



	model	area_under_ROC	precision	recall	f1_score	total_test_sample	TP	FP	FN	TN
0	Naive_Bayes	0.907363	0.903409	0.935294	0.919075	311	124	17	11	159
1	Logistic_Regression	0.644556	0.869565	0.352941	0.502092	311	132	9	110	60
2	Random_Forest	0.847580	0.968504	0.723529	0.828283	311	137	4	47	123
3	Deep_Learning	0.951731	0.947919	0.951731	0.948399	311	139	2	14	156

The above table shows that the Naive Bayes Classifier has better performance among the other Machine Learning models that we employed in this project. Furthermore, our Deep Learning model performs better than our machine learning models.

Confusion Matrix for ML and DL models



Thank You