SPAM DETECTION CLASSIFIER PROJECT

Prepared by:

Sarthak Gupta

SME Name:

Mohd. Kashif

ACKNOWLEDGMENT

I would like to convey my heartfelt gratitude to Flip Robo Technologies for providing me with this wonderful opportunity to work on a Machine Learning project using NLP “Spam Detection Classifier Project” and also want to thank my SME “Mohd. Kashif” for providing the dataset and directions to complete this project. This project would not have been accomplished without their help and insights.

I would also like to thank my academic “Data Trained
Education” and their team who has helped me to learn Machine Learning and NLP.

Working on this project was an incredible experience as I learnt more from this Project during completion.

INTRODUCTION

1. Business Problem Framing

The SMS Spam Collection is a set of SMS tagged messages that have been collected for SMS Spam research. It contains one set of SMS messages in English of 5,574 messages, tagged according being ham (legitimate) or spam.

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.

2. Conceptual Background of the Domain Problem

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.

3. Review of Literature

Spam Detector is used to detect unwanted, malicious and virus infected texts and helps to separate them from the non-spam texts. It uses a binary type of classification containing the labels such as 'ham' (nonspam) and spam. Application of this can be seen in Google Mail (GMAIL) where it segregates the spam emails in order to prevent them from getting into the user's inbox.

4. Motivation for the Problem Undertaken

To build an application which can detect the spam by seeing the review.

Analytical Problem Framing

1. Mathematical/ Analytical Modelling of the Problem

1) Cleaned Data by removing irrelevant features
2) Pre-processing of text using NLP processing
3) Used Word Counts
4) Used Character Counts
5) Used Count Vectorizer
6) Split data into train and test
7) Built Model
8) Hyper parameter tunning

2. Data Sources and their formats

The data-set is in csv format: spam.csv. Features of this dataset are:

- v1- target column
- v2- containing messages
- Unnamed: 2- Containing Null Values
- Unnamed: 3- Containing Null Values
- Unnamed: 4- Containing Null Values

3. Data Pre-processing:

a) Checked Top 5 Rows of Dataset

```
spam_data.head()
```

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy.. Available only ...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN
4	ham	Nah I don't think he goes to usf, he lives aro...	NaN	NaN	NaN

b) Checked Total Numbers of Rows and Column

```
spam_data.shape
```

```
(5572, 5)
```

c) Checked All Column Name

```
spam_data.columns
```

```
Index(['v1', 'v2', 'Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'], dtype='object')
```

d) Checked Data Type of All Data

```
spam_data.dtypes
```

```
v1            object
v2            object
Unnamed: 2    object
Unnamed: 3    object
Unnamed: 4    object
```

e) Checked for Null Values

```
spam_data.isnull().sum()
```

```
v1               0
v2               0
Unnamed: 2    5522
Unnamed: 3    5560
Unnamed: 4    5566
```

There is null value in the dataset in all 3 columns except two.

f) Checked total number of unique values

```
spam_data['target'].nunique()
```

```
2
```

```
spam_data['target'].unique()
```

```
array(['ham', 'spam'], dtype=object)
```

```
spam_data['target'].value_counts()
```

```
ham     4825
spam     747
Name: target, dtype: int64
```

g) Data cleaning

- Dropped Column " Unnamed: 2, Unnamed: 3, Unnamed: 4 " as this column contains Null Values.

```
#droping irrelevant column
spam_data=spam_data.drop(columns=['Unnamed: 2','Unnamed: 3','Unnamed: 4'])
#renaming column v1 and v2 for better understanding
spam_data=spam_data.rename(columns={'v1' : 'target','v2' : 'message'})
spam_data
```

	target	message
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

- Checked and Dropped Duplicates Values.

```
spam_data.duplicated().sum()
```

```
403
```

```
spam_data.drop_duplicates(inplace = True)
```

h) Describing Data

```
spam_data.describe()
```

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

```
#describing data for all messages
spam_data[['length','num_words','num_sent']].describe()
```

	length	num_words	num_sent
count	5169.000000	5169.000000	5169.000000
mean	78.977945	18.455407	1.961308
std	58.236293	13.322448	1.432583
min	2.000000	1.000000	1.000000
25%	36.000000	9.000000	1.000000
50%	60.000000	15.000000	1.000000
75%	117.000000	26.000000	2.000000
max	910.000000	220.000000	38.000000

```
#describing data for ham messages
spam_data[spam_data['target'] == 0][['length','num_words','num_sent']].describe()
```

	length	num_words	num_sent
count	4516.000000	4516.000000	4516.000000
mean	70.459256	17.123339	1.815545
std	56.358207	13.491315	1.364098
min	2.000000	1.000000	1.000000
25%	34.000000	8.000000	1.000000
50%	52.000000	13.000000	1.000000
75%	90.000000	22.000000	2.000000
max	910.000000	220.000000	38.000000

```
#describing data for spam messages
spam_data[spam_data['target'] == 1][['length','num_words','num_sent']].describe()
```

	length	num_words	num_sent
count	653.000000	653.000000	653.000000
mean	137.891271	27.667688	2.969372
std	30.137753	7.008418	1.488910
min	13.000000	2.000000	1.000000
25%	132.000000	25.000000	2.000000
50%	149.000000	29.000000	3.000000
75%	157.000000	32.000000	4.000000
max	224.000000	46.000000	9.000000

4. Data Inputs- Logic- Output Relationships

I. Text Pre-Processing

```
# creating coloumn which will contain the no. of characters
spam_data['length'] = spam_data['message'].str.len()
```

```
# creating coloumn which will fetch no. of words
spam_data['num_words'] = spam_data['message'].apply(lambda x: len(nltk.word_tokenize(x)))
```

```
# creating coloumn which will fetch no. of sentences
spam_data['num_sent'] = spam_data['message'].apply(lambda x: len(nltk.sent_tokenize(x)))
```

```
spam_data.head()
```

	target	message	length	num_words	num_sent
0	ham	Go until jurong point, crazy.. Available only ...	111	24	2
1	ham	Ok lar... Joking wif u oni...	29	8	2
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	155	37	2
3	ham	U dun say so early hor... U c already then say...	49	13	1
4	ham	Nah I don't think he goes to usf, he lives aro...	61	15	1

```
# mapping labels to 1 and 0
spam_data['target'] = spam_data.target.map({'ham':0, 'spam':1})
```

```
spam_data.head()
```

	target	message	length	num_words	num_sent
0	0	Go until jurong point, crazy.. Available only ...	111	24	2
1	0	Ok lar... Joking wif u oni...	29	8	2

```
ps= PorterStemmer()
```

```
# defining function to apply functions: Lowercase, Tokenize, Alphanumeric, Stopwords, Stemming

def transform_text(message):
    message = message.lower()
    message = nltk.word_tokenize(message)
    y=[]
    for i in message:
        if i.isalnum():
            y.append(i)
    message = y[:]
    y.clear()

    for i in message:
        if i not in stopwords.words('english')and i not in string.punctuation:
            y.append(i)

    message = y[:]
    y.clear()

    for i in message:
        y.append(ps.stem(i))


    return " ".join(y)
```

```
spam_data["transformed_text"] = spam_data['message'].apply(transform_text)
```

Wordcloud

```
wc = WordCloud(width = 500, height = 300, min_font_size= 10, background_color= 'black')
```

```
#Generating Word Cloud for Spam Messages
spam_wordcloud = wc.generate(spam_data[spam_data['target']==1]['transformed_text'].str.cat(sep = " "))
plt.figure(figsize=(12,8))
plt.imshow(spam_wordcloud)
plt.show()
```

In the above Word Cloud, we can clearly see the words like text, free, reply, call, claim..etc. these words are generally seen in the SPAM Msgs.

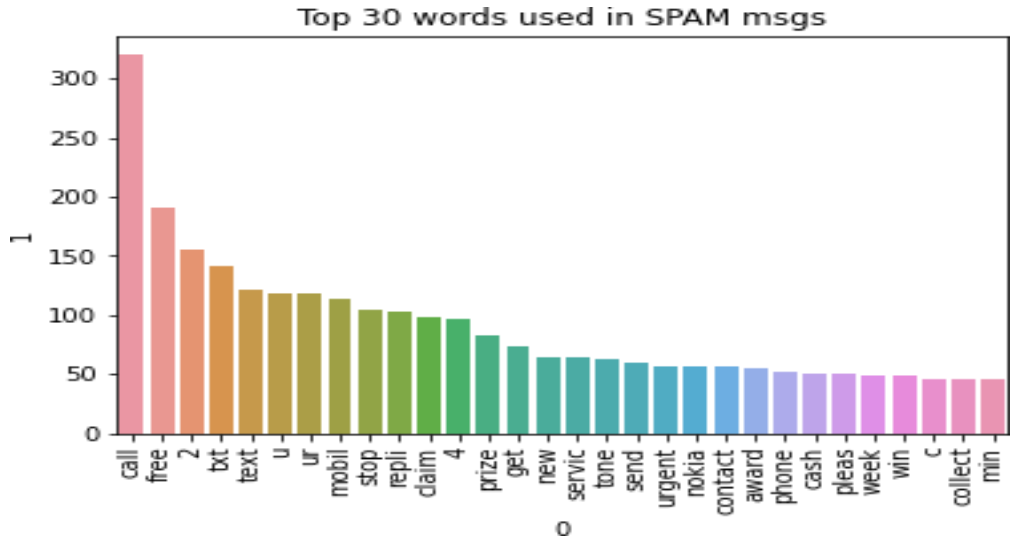
```
#Generating Word Cloud for Ham Messages
ham_wordcloud = wc.generate(spam_data[spam_data['target']==0]['transformed_text'].str.cat(sep = " "))
plt.figure(figsize=(12,8))
plt.imshow(ham_wordcloud)
plt.show()
```

In the above Word Cloud, we can clearly see the words like love, come, go, call, time..etc. these words are generally seen in the Ham Msgs.

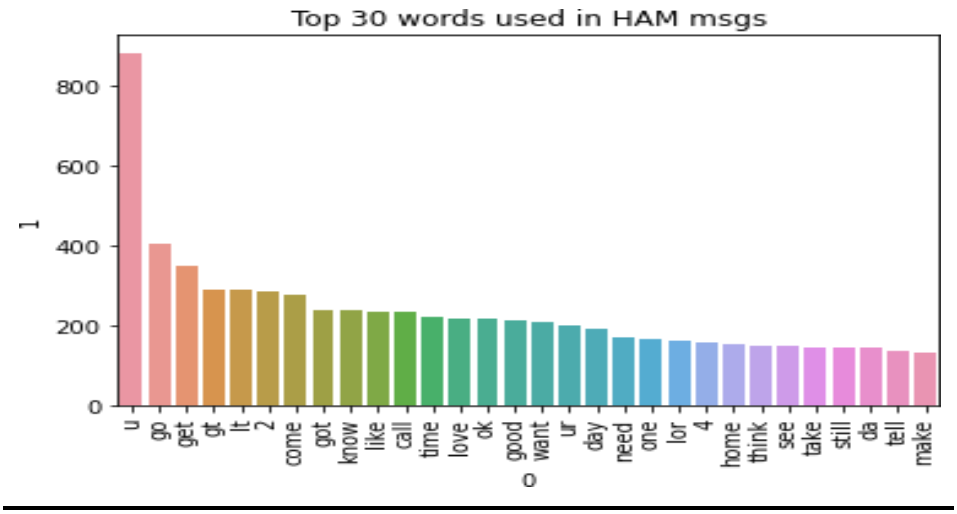
Top 30 words used in HAM and SPAM messaegs

```
spam_corpus = []
for msg in spam_data[spam_data['target']==1]['transformed_text'].tolist():
    for word in msg.split():
        spam_corpus.append(word)
```

```
from collections import Counter
sns.barplot(pd.DataFrame(Counter(spam_corpus).most_common(30))[0] , pd.DataFrame(Counter(spam_corpus).most_common(30))[1])
plt.title("Top 30 words used in SPAM msgs")
plt.xticks(rotation = "vertical")
plt.show
```



```
ham_corpus = []
for msg in spam_data[spam_data['target']==0]['transformed_text'].tolist():
    for word in msg.split():
        ham_corpus.append(word)

sns.barplot(pd.DataFrame(Counter(ham_corpus).most_common(30))[0] , pd.DataFrame(Counter(ham_corpus).most_common(30))[1])
plt.title("Top 30 words used in HAM msgs")
plt.xticks(rotation = "vertical")
plt.show
```



CountVectorizer

```
#Tokenization (a list of tokens), will be used as the analyzer
def process_text(text):

    #1 Remove Punctuation
    nopunc = [char for char in text if char not in string.punctuation]
    nopunc = ''.join(nopunc)

    #2 Remove Stop Words
    clean_words = [word for word in nopunc.split() if word.lower() not in stopwords.words('english')]

    #3 Return a list of clean words
    return clean_words
```

```
# Text Vectorization
cv = CountVectorizer()

messages_bow = CountVectorizer(analyzer=process_text).fit_transform(spam_data['message'])
```

5. State the set of assumptions (if any) related to the problem under consideration

- It was observed that there are two types of messages: ham and spam. So, have to detect which message is spam and this column is target column. And also have to rename column names.
- First column contains the type.

- Second column contains text which means these are messages and have detect these messages.
- Rest three columns contains Null Values, so, it is not relevant and have to be dropped.
- It was observed that in message column there are irrelevant values. So, we need to replace or pre-process those values.
- Also have to convert text (reviews) into vectors using counter- vectorize.
    - By looking into the Target Variable, it is assumed that it is a classification problem.

Model/s Development and Evaluation

1. Identification of possible problem-solving approaches (methods)

In this project, we want to differentiate between comments and its categories and for this we have used these approaches:

- Checked Total Numbers of Rows and Column
- Checked All Column Name
- Checked Data Type of All Data
- Checked for Null Values
- Checked total number of unique values
- Description of Data
- Dropped irrelevant Columns
- Replaced special characters and irrelevant data
- Checked all features through visualization.
- Removed unwanted punctuations and special characters
- Converted all messages to lower case
- Removed punctuations
- Removed StopWords
- Used Counter-Vectorization
- Used Word Counts
- Used Character Counts
- Checked loud word using WordCloud
- Converted text into vectors using Counter-Vectorize

2. Testing of Identified Approaches (Algorithms)

1. Logistic Regression
2. Linear Support Vector Classifier
3. Bernoulli NB
4. Multinomial NB
5. SGD Classifier
6. XGB Classifier

3. Run and evaluate selected models

```
X_train, X_test, y_train, y_test = train_test_split(messages_bow, spam_data['target'], random_state = 20)
```

```
# Defining the Classification Machine Learning Algorithms
lr = LogisticRegression(solver='lbfgs')
svc = LinearSVC()
bnb = BernoulliNB()
mnb = MultinomialNB()
xgb = XGBClassifier(verbosity=0)
sgd= SGDClassifier()

# Creating a function to train and test the model with evaluation metrics
def BuiltModel(model):
    print('*'*30+model.__class__.__name__+'*'*30)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_train)
    pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, pred)*100
    print(f"ACCURACY SCORE PERCENTAGE:", accuracy)
    # Confusion matrix and Classification report
    print(f"CLASSIFICATION REPORT: \n {classification_report(y_test, pred)}")
    print(f"CONFUSION MATRIX: \n {confusion_matrix(y_test, pred)}\n")
    print("-"*120)
    print("\n")
```

Training and testing of all the classification algorithms

```
for model in [lr,svc,bnb,mnb,xgb,sgd]:
    BuiltModel(model)
```

```
******************************LogisticRegression******************************
ACCURACY SCORE PERCENTAGE: 97.52513534416086
CLASSIFICATION REPORT:
              precision    recall  f1-score   support

           0       0.97      1.00      0.99      1139
           1       0.99      0.80      0.88       154

    accuracy                           0.98      1293
   macro avg       0.98      0.90      0.94      1293
weighted avg       0.98      0.98      0.97      1293

CONFUSION MATRIX:
 [[1138    1]
 [  31  123]]

------------------------------------------------------------------------------------------
```

```
*****************************LinearSVC*****************************
ACCURACY SCORE PERCENTAGE: 98.06651198762569
CLASSIFICATION REPORT:
              precision    recall  f1-score   support

           0       0.98      1.00      0.99      1129
           1       1.00      0.85      0.92       164

    accuracy                           0.98      1293
   macro avg       0.99      0.92      0.95      1293
weighted avg       0.98      0.98      0.98      1293

CONFUSION MATRIX:
 [[1129    0]
 [  25  139]]

------------------------------------------------------------------------


*****************************BernoulliNB*****************************
ACCURACY SCORE PERCENTAGE: 95.9783449342614
CLASSIFICATION REPORT:
              precision    recall  f1-score   support

           0       0.96      0.99      0.98      1129
           1       0.92      0.74      0.82       164

    accuracy                           0.96      1293
   macro avg       0.94      0.87      0.90      1293
weighted avg       0.96      0.96      0.96      1293

CONFUSION MATRIX:
 [[1119   10]
 [  42  122]]

*****************************MultinomialNB*****************************
ACCURACY SCORE PERCENTAGE: 96.13302397525135
CLASSIFICATION REPORT:
              precision    recall  f1-score   support

           0       0.99      0.97      0.98      1129
           1       0.80      0.92      0.86       164

    accuracy                           0.96      1293
   macro avg       0.90      0.94      0.92      1293
weighted avg       0.96      0.96      0.96      1293

CONFUSION MATRIX:
 [[1092   37]
 [  13  151]]

------------------------------------------------------------------------
```

```
******************************XGBClassifier******************************
ACCURACY SCORE PERCENTAGE: 96.51972157772622
CLASSIFICATION REPORT:
              precision    recall  f1-score   support

           0       0.96      1.00      0.98      1129
           1       0.97      0.75      0.85       164

    accuracy                           0.97      1293
   macro avg       0.97      0.87      0.91      1293
weighted avg       0.97      0.97      0.96      1293

CONFUSION MATRIX:
 [[1125    4]
 [  41  123]]

--------------------------------------------------------------------------------

 ******************************SGDClassifier******************************
 ACCURACY SCORE PERCENTAGE: 97.37045630317091
 CLASSIFICATION REPORT:
               precision    recall  f1-score   support

            0       0.97      1.00      0.99      1129
            1       0.99      0.80      0.89       164

     accuracy                           0.97      1293
    macro avg       0.98      0.90      0.94      1293
 weighted avg       0.97      0.97      0.97      1293

 CONFUSION MATRIX:
  [[1128    1]
  [  33  131]]

 --------------------------------------------------------------------------------
```

Cross validation score for best score models

```
def cross_val(model):
    print('*'*30+model.__class__.__name__+'*'*30)
    scores = cross_val_score(model,X,y, cv = 5).mean()
    print("Cross validation score:", scores)
    print("\n")

for model in [lr,svc,bnb,mnb,xgb,sgd]:
    cross_val(model)
```

LinearSVC is having best accuracy and cv score. So, we will use LinearSVC for model prediction.

HyperParameter Tuning

Linear SVC with GridSearchCV

```
# Lets select the different parameters for tuning our best model (Linear SVC)
grid_params =  {'C':(0.001, 0.01, 0.1, 1, 10),
                'penalty':('l1','l2'),
               'loss':('hinge','squared_hinge')}

# Train the model with given parameters using GridSearchCV
LSVC =  GridSearchCV(svc, grid_params, cv=3)
LSVC.fit(X_train, y_train)
```

```
GridSearchCV(cv=3, estimator=LinearSVC(),
             param_grid={'C': (0.001, 0.01, 0.1, 1, 10),
                         'loss': ('hinge', 'squared_hinge'),
                         'penalty': ('l1', 'l2')})
```

```
# Selecting the best parameters found by GridSearchCV
print(LSVC.best_params_)
print(LSVC.best_score_)
```

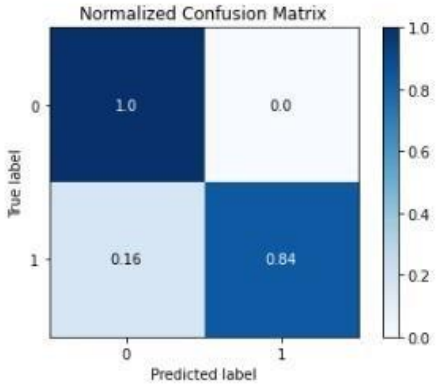
```
{'C': 1, 'loss': 'squared_hinge', 'penalty': 'l2'}
0.9731682146542827
```

```
# Final Model with the best chosen parameters list
best_model = LinearSVC(C= 1, loss= 'squared_hinge', penalty= 'l2')
best_model.fit(X_train,y_train) # fitting data to the best model
pred = best_model.predict(X_test)
accuracy = accuracy_score(y_test, pred)*100
# Printing the accuracy score
print("ACCURACY SCORE:", accuracy)
# Printing the classification report
print(f"\nCLASSIFICATION REPORT: \n {classification_report(y_test, pred)}")
# Printing the Confusion matrix
print(f"\nCONFUSION MATRIX: \n {confusion_matrix(y_test, pred)}")
```

```
ACCURACY SCORE: 98.06651198762569

CLASSIFICATION REPORT:
              precision    recall  f1-score   support

           0       0.98      1.00      0.99      1139
           1       1.00      0.84      0.91       154

    accuracy                           0.98      1293
   macro avg       0.99      0.92      0.95      1293
weighted avg       0.98      0.98      0.98      1293


CONFUSION MATRIX:
 [[1139    0]
 [  25  129]]
```

```
# Creating a normalized confusion matrix here
skplt.metrics.plot_confusion_matrix(y_test, pred, normalize=True)
```

```
<AxesSubplot:title={'center':'Normalized Confusion Matrix'}, xlabel='Predicted label', ylabel='True label'>
```



So the best accuracy score is 98.

- Saving The Predictive Model

```
joblib.dump(best_model, "Spam_Detection_Classifier.pkl")
```

```
['Spam_Detection_Classifier.pkl']
```

- Comparing Actual and Predicted

```
Model = joblib.load("Spam_Detection_Classifier.pkl")
# Predicting test data using loaded model
prediction = Model.predict(X_test)
# Analysing Predicted vs Actual results
Spam_Detection_Classifier = pd.DataFrame()
Spam_Detection_Classifier['Predicted Spam Messages Detection'] = prediction
Spam_Detection_Classifier['Actual Spam Messages Detection'] = y
Spam_Detection_Classifier
```

	Predicted Spam Messages Detection	Actual Spam Messages Detection
0	0	0.0
1	0	0.0
2	0	1.0
3	0	0.0
4	0	0.0
...	...	...
1288	0	0.0
1289	0	0.0
1290	0	0.0
1291	0	0.0
1292	0	0.0

1293 rows × 2 columns

- Saving the model in CSV format

```
# Converting the dataframe into CSV format and saving it
Spam_Detection_Classifier.to_csv('Spam_Detection_Classifier_Project.csv', index=False)
```

4. Key Metrics for success in solving problem under consideration
   - Accuracy Score, Precision Score, Recall Score, F1-Score and CV score are used for success. Also, confusion matrix is used for success.

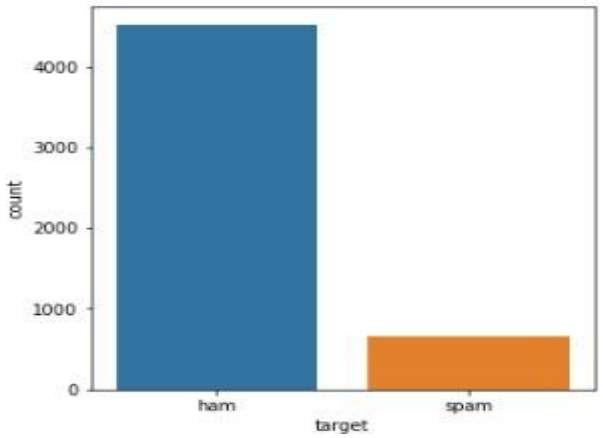
5. Visualization

Using Countplot

```
print(spam_data['target'].value_counts())
plt.figure(figsize=(5,5))
sns.countplot('target', data=spam_data)
```

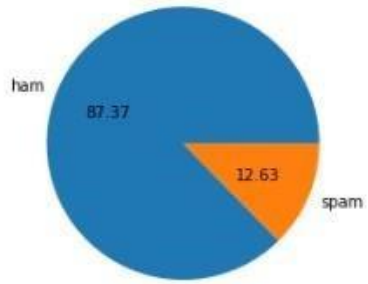
```
ham     4516
spam     653
Name: target, dtype: int64
```

```
<AxesSubplot:xlabel='target', ylabel='count'>
```



Using Pie-Plot

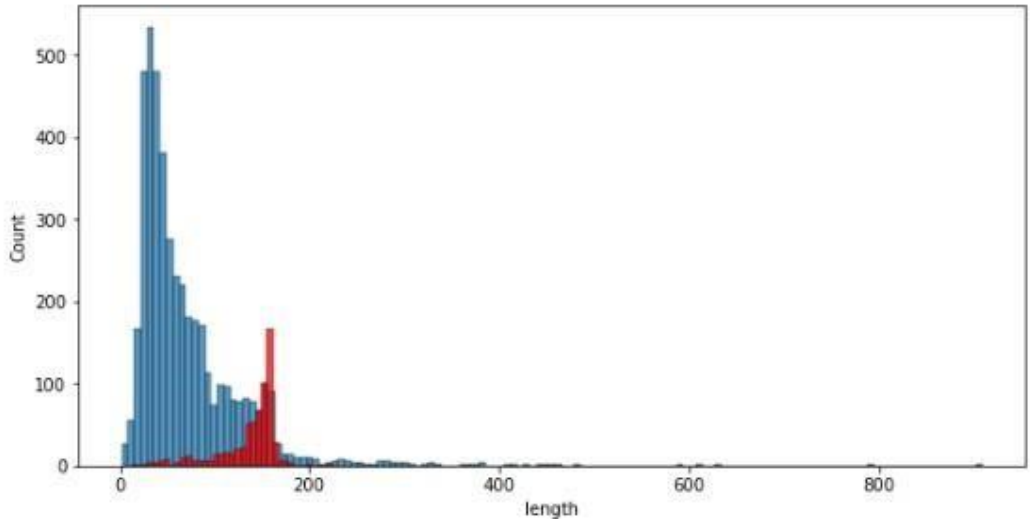
```
plt.pie(spam_data['target'].value_counts(), labels =['ham', 'spam'], autopct= "%.02f")
plt.show()
```



The above barplot and pie chart shows that our data is highly imbalanced as spam is 12.63% and ham is 87.37%.

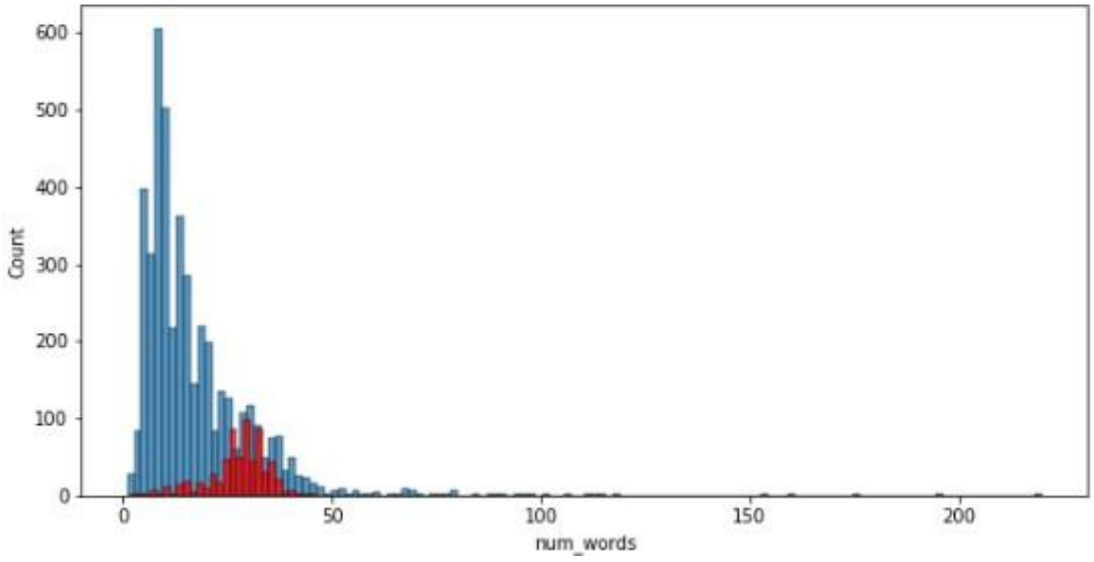
```
#plotting histogram
plt.figure(figsize=(10,5))
sns.histplot(spam_data[spam_data['target']==0]['length'])
sns.histplot(spam_data[spam_data['target']==1]['length'],color = 'red')
```

```
<AxesSubplot:xlabel='length', ylabel='Count'>
```



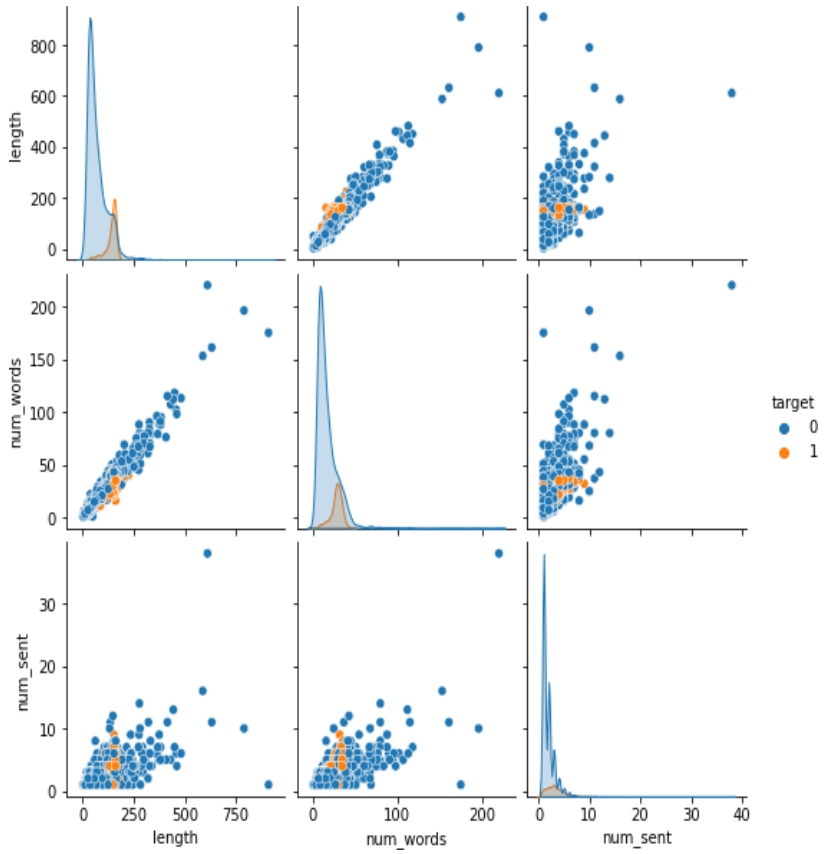

```
plt.figure(figsize=(10,5))
sns.histplot(spam_data[spam_data['target']==0]['num_words'])
sns.histplot(spam_data[spam_data['target']==1]['num_words'],color = 'red')
```

```
<AxesSubplot:xlabel='num_words', ylabel='Count'>
```



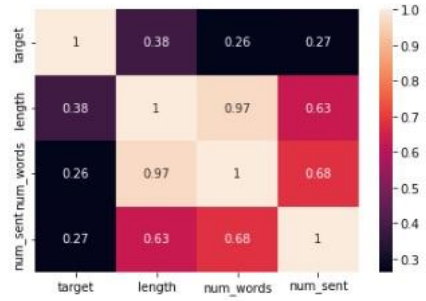
These histogram are for the number of characters and words in ham and spam messages and we can clearly see that spam messages have large number of charcters and words than ham messages.

```
#pairplot for relationship between each column
sns.pairplot(spam_data, hue = 'target')
```




```
#Using heatmap for corelations between columns
sns.heatmap(spam_data.corr(), annot = True)
```

```
<AxesSubplot:>
```



Chances of Multicolinearity will be high as there are Strong corelation between them, so we will take only one column i.e. length as it has maximum relation with the target coloumn.

6. Interpretation of the Results

- Through Pre-processing it is interpretated that all text is converted to lower case, removed Punctation, replaced extra space, removed stop-words, Calculated length of sentence, words and characters, converted text using Counter-Vectorize.

- Natural Language Processing and Machine Learning is used in this project.
- Used 6 Machine Learning Algorithms for choosing one best model which is giving best accuracy than others.
- By creating/building model we get best model: Linear SVC.

CONCLUSION

1. Key Findings and Conclusions of the Study

In this project we have detected spam and ham messages that have been collected for SMS Spam research. Then we have done different text process to eliminate problem of imbalance. By doing different EDA steps we have analyzed the text.

We have checked frequently occurring words in our data as well as rarely occurring words. After all these steps we have built function to train and test different algorithms and using various evaluation metrics we have selected Linear-SVC for our final model.

Finally, by doing hyperparameter tuning we got optimum parameters for our final model. And finally, we got improved accuracy score for our final model.

2. Learning Outcomes of the Study in respect of Data Science

- This project has demonstrated the importance of NLP.
- Through different powerful tools of visualization, we were able to analyses and interpret the huge data and with the help of pie plot, count plot & word cloud, I am able to see the distribution of spam and ham messages.
- Through data cleaning we were able to remove unnecessary columns, values, stop-words and punctuation from our dataset due to which our model would have suffered from overfitting or underfitting.

The few challenges while working on this project were: -

- Using NLP to find punctuations & stop words, it took time in giving the result.
- The data set took time to run some algorithms & to check the cross- validation score.

3. Limitations of this work and Scope for Future Work

As we know there are two types of messages to. So, it is difficult to detect with higher accuracies. Still, we can improve our accuracy by fetching more data and by doing extensive hyperparameter tuning.