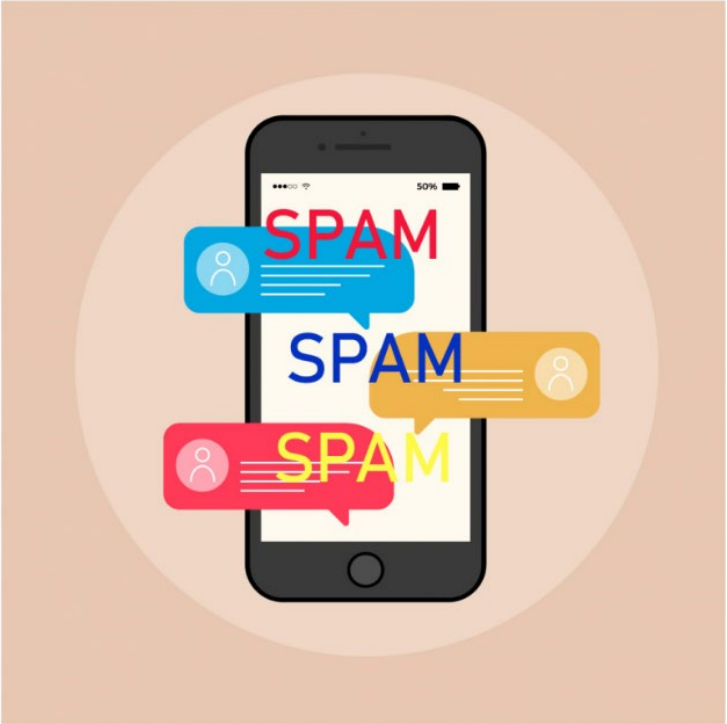
SMS spam classification using NLP: Methods, approaches, and applications

By Anisha Agarwal



Introduction

The easy accessibility and simplicity of SMS have made it attractive to malicious users thereby incurring unnecessary costing on the mobile users and also the Secure Mobile Message Communication is jeopardized.

Thus, this article is to identify and review existing state-of-the-art methodology for SMS spam classification based on certain metrics: ML and AI methods and techniques, approaches, and deployed environment.

Approach

Area	Sub-area	Pros	Cons
Approaches	Content-based	• Simplest and most frequently used approach for SMS classification • Classification is done based on users' preference ratings.	• Limited keywords for effective classification. • It can generate false positive if legitimate users' sends a message with some blacklisted keywords. • It requires a lot of effort for routine updates for new spam keywords.
	Non-content based	The privacy of users' messages content is preserved.	• Increase network traffic (flooding) thus increasing overhead. • Not adaptive to spammer's drift. • The high false positive rate • Time-consuming.
	Hybrid	• Allows more features for spam classification thus improving accuracy when compared with the traditional content-based filter.	• Time complexity

Implementation

1. Import the required Libraries.
2. Data Preprocessing.
3. Bag of Words.
4. Adding new Feature. Like- Length of the text, Profanity of the text, Parts of Speech(POS).
5. EDA of the dataset.
6. Word Tokenization.
7. Implementing different ML classifying models. Like- LogisticRegression, MultinomialNB, RandomForestClassifier, LinearSVC, SGDClassifier, GradientBoostingClassifier. And compare these to find which Model is best for this classification.

Libraries

```
#importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import re
import nltk
import string
from gensim.test.utils import common_texts, get_tmpfile
from gensim.models import Word2Vec
from collections import Counter
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split

%matplotlib inline
import warnings

warnings.filterwarnings('ignore')

%config InlineBackend.figure_format = 'retina'
```

Data Preprocessing

1. Removing unnecessary columns and renaming features name.

```
#drop unwanted columns and name change
data = data.drop(["Unnamed: 2", "Unnamed: 3", "Unnamed: 4"], axis=1)
data = data.rename(columns={"v1":"label", "v2":"text"})
```

Data Preprocessing:

2. Numericalizing categorical feature which is our label (ham or sam).

```
[ ]  # convert label to a numerical variable
     data['label_cat'] = data.label.map({'ham':0, 'spam':1})
```

```
[ ]  data.head()
```

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

Data Preprocessing:

3. Generating corpus from raw sms messages (stopwords,lowering,stemming).

```
[ ]
corpus = []
for i in range(0, len(data)):
    review = re.sub('[^a-zA-Z]', ' ', data['text'][i])
    review = review.lower()
    review = review.split()
    ps = PorterStemmer()
    nltk.download('stopwords')
    review = [ps.stem(word) for word in review if not word in set(stopwords.words('english'))]
    review = ' '.join(review)
    corpus.append(review)
```

Data Preprocessing:

Adding Clean Text to Dataset

```python

def text_process(mess):
    """
    Takes in a string of text, then performs the following:
    1. Remove all punctuation
    2. Remove all stopwords
    3. Returns a list of the cleaned text
    """
    STOPWORDS = stopwords.words('english') + ['u', 'ü', 'ur', '4', '2', 'im', 'dont', 'doin', 'ure']
    # Check characters to see if they are in punctuation
    nopunc = [char for char in mess if char not in string.punctuation]

    # Join the characters again to form the string.
    nopunc = ''.join(nopunc)

    # Now just remove any stopwords
    return ' '.join([word for word in nopunc.split() if word.lower() not in STOPWORDS])
```

```
[ ]
```

```python
data['clean_text'] = data.text.apply(text_process)
data.head()
data['clean_text'].fillna("unknown", inplace=True)
```

Data Preprocessing:

4. Creating bag of words model using CountVectorizer.

```
[ ]
# Creating the Bag of Words model
cv = CountVectorizer()
X = cv.fit_transform(corpus).toarray()
y = data.iloc[:, 0].values
```

```
[ ]
#showing first and last 20 features names
print(cv.get_feature_names()[0:20])
print(cv.get_feature_names()[-20:])
```

```
['aa', 'aah', 'aaniy', 'aaoooright', 'aathi', 'ab', 'abbey', 'abdomen', 'abeg', 'abel', 'aberdeen', 'abi', 'abil', 'abiola', 'abj', 'abl', 'abnorm', 'abouta', 'abroad', 'absenc']
['yunni', 'yuo', 'yuou', 'yup', 'yupz', 'zac', 'zaher', 'zealand', 'zebra', 'zed', 'zero', 'zf', 'zhong', 'zindgi', 'zoe', 'zogtoriu', 'zoom', 'zouk', 'zs', 'zyada']
```

Bag of Words: Code to Generate Bag of Words

```
[ ]
#Visualisations
from wordcloud import WordCloud
```

```
[ ]
ham_words = ''
spam_words = ''
spam = data[data.label_cat == 1]
ham = data[data.label_cat == 0]
```

```
[ ]
for val in spam.text:
    text = re.sub('[^a-zA-Z]', ' ', val)
    text = text.lower()
    text = text.split()
    ps = PorterStemmer()
    text = [ps.stem(word) for word in text if not word in set(stopwords.words('english'))]
    for words in text:
        spam_words = spam_words + words + ' '

for val in ham.text:
    text = re.sub('[^a-zA-Z]', ' ', val)
    text = text.lower()
    text = text.split()
    ps = PorterStemmer()
    text = [ps.stem(word) for word in text if not word in set(stopwords.words('english'))]
    for words in text:
        ham_words = ham_words + words + ' '
```

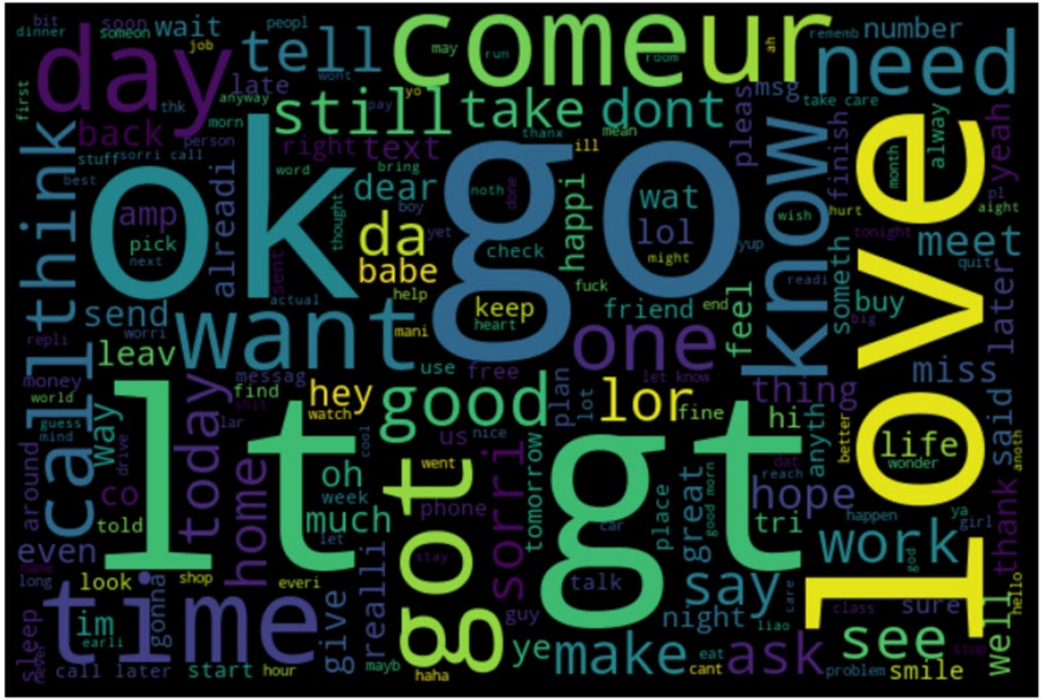
```
[ ]
# Generate a word cloud image
spam_wordcloud = WordCloud(width=600, height=400).generate(spam_words)
ham_wordcloud = WordCloud(width=600, height=400).generate(ham_words)
```

Code to plot Word of Cloud Spam Words

```
#Spam Word cloud
plt.figure( figsize=(10,8), facecolor='k')
plt.imshow(spam_wordcloud)
plt.axis("off")
plt.tight_layout(pad=0)
plt.show()
```

Code to plot Word of Cloud Ham Words

```
#Ham word cloud
plt.figure( figsize=(10,8), facecolor='k')
plt.imshow(ham_wordcloud)
plt.axis("off")
plt.tight_layout(pad=0)
plt.show()
```



New Features added: Length of Text

```
[ ]  data['text_len'] = data['text'].apply(len)
     data.head()
```

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

New Features added: Profanity Check

```
!pip install profanity-check
```

```
# import joblib
from profanity_check import predict, predict_prob

def check_profanity(df):
  data['contains_profanity'] = predict(data['text'])
  return data

data = check_profanity(data)
data
```

	label	text	label_cat	text_len	clean_text	contains_profanity
0	ham	Go until jurong point, crazy.. Available only ...	0	111	Go jurong point crazy Available bugis n great ...	0
1	ham	Ok lar... Joking wif u oni...	0	29	Ok lar Joking wif oni	0
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	1	155	Free entry wkly comp win FA Cup final tkts 21s...	0
3	ham	U dun say so early hor... U c already then say...	0	49	dun say early hor c already say	0
4	ham	Nah I don't think he goes to usf, he lives aro...	0	61	Nah think goes usf lives around though	0
...	...	...	...	...	...	...
5567	spam	This is the 2nd time we have tried 2 contact u...	1	161	2nd time tried contact â£750 Pound prize claim...	0
5568	ham	Will Ì_ b going to esplanade fr home?	0	37	Ì b going esplanade fr home	0
5569	ham	Pity, * was in mood for that. So...any other s...	0	57	Pity mood Soany suggestions	0
5570	ham	The guy did some bitching but I acted like i'd...	0	125	guy bitching acted like id interested buying s...	0
5571	ham	Rofl. Its true to its name	0	26	Rofl true name	0

5572 rows × 6 columns

New Features added: Readability Score

```
!pip install readability
```

```
import readability
count=0
readability_list = []
final_list = []
txt = data['clean_text']
for i in txt:
  if not i.strip():
    final_list.append(0)
  else:
    results = readability.getmeasures(i, lang='en')
    readability_list = results['readability grades']['FleschReadingEase']
    final_list.append(readability_list)
data['readability_score'] = final_list
data.describe()
```

	label_cat	text_len	contains_profanity	readability_score
count	5572.000000	5572.000000	5572.000000	5572.000000
mean	0.134063	80.118808	0.027997	84.414829
std	0.340751	59.690841	0.164979	41.101875
min	0.000000	2.000000	0.000000	-555.580000
25%	0.000000	36.000000	0.000000	62.790000
50%	0.000000	61.000000	0.000000	84.900000
75%	0.000000	121.000000	0.000000	104.980000
max	1.000000	910.000000	1.000000	205.820000

New Features added: Parts of Speech (POS)

```
!python -m textblob.download_corpora
```

```python
from textblob import TextBlob
from collections import Counter
txt = data['clean_text']
count = 0
adj_list = []
adv_list = []
final_list_adj = []
final_list_adv = []
def textblob_adj(text):
  blobed = TextBlob(text)
  # counts = Counter(tag for word,tag in blobed.tags)
  adj_list = []
  adv_list = []
  adj_tag_list = ['JJ','JJR','JJS']
  adv_tag_list = ['RB','RBR','RBS']
  for (a, b) in blobed.tags:
      if b in adj_tag_list:
          adj_list.append(a)
      elif b in adv_tag_list:
          adv_list.append(a)
      else:
          pass
  return adj_list, adv_list

for i in txt:
  adj_list, adv_list = textblob_adj(i)
  if not adj_list:
    final_list_adj.append(0)
  else:
    final_list_adj.append(1)
  if not adv_list:
    final_list_adv.append(0)
  else:
    final_list_adv.append(1)
  # final_list_adj.append(adj_list)
  # final_list_adv.append(adv_list)

data['adjective'] = final_list_adj
data['adverb'] = final_list_adv
data
```

	label	text	label_cat	text_len	readability_cat	clean_text	contains_profanity	readability_score	adjective	adverb
0	ham	Go until jurong point, crazy.. Available only ...	0	111	0	Go jurong point crazy Available bugis n great ...	0	79.557500	[jurong, great, buffet, wat]	[n, amore]
1	ham	Ok lar... Joking wif u oni...	0	29	1	Ok lar Joking wif oni	0	100.240000	[lar]	[]
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	1	155	1	Free entry wkly comp win FA Cup final tkts 21s...	0	96.059545	[Free, final, receive]	[]
3	ham	U dun say so early hor... U c already then say...	0	49	1	dun say early hor c already say	0	90.958571	[early]	[already]

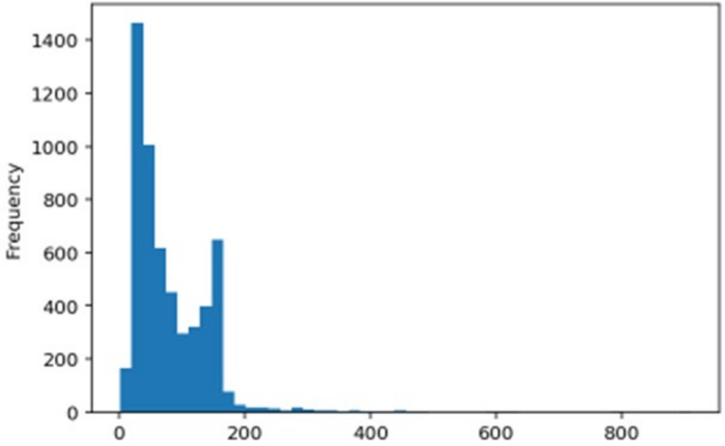
Exploratory Data Analysis:

```
data.groupby('label').describe().T
```

	label	ham	spam
label_cat	count	4825.000000	747.000000
	mean	0.000000	1.000000
	std	0.000000	0.000000
	min	0.000000	1.000000
	25%	0.000000	1.000000
	50%	0.000000	1.000000
	75%	0.000000	1.000000
	max	0.000000	1.000000
text_len	count	4825.000000	747.000000
	mean	71.023627	138.866131
	std	58.016023	29.183082
	min	2.000000	13.000000
	25%	33.000000	132.500000
	50%	52.000000	149.000000
	75%	92.000000	157.000000
	max	910.000000	224.000000

Maximum Length of the Text Plotted

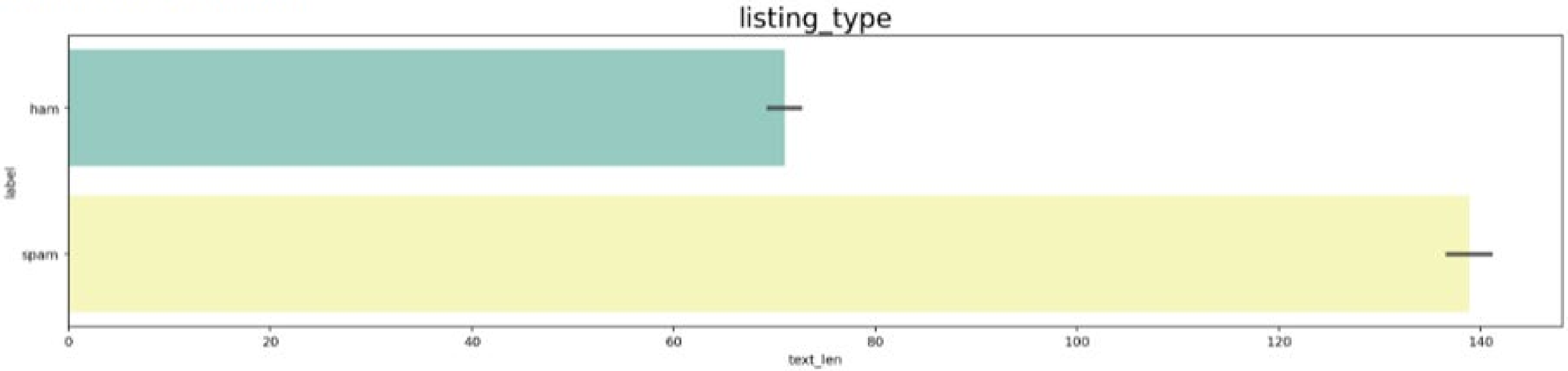
```
[23]  1  data['text_len'].plot(kind='hist',bins=50)
      2  plt.show()
```



Spam and Ham Text against the Length

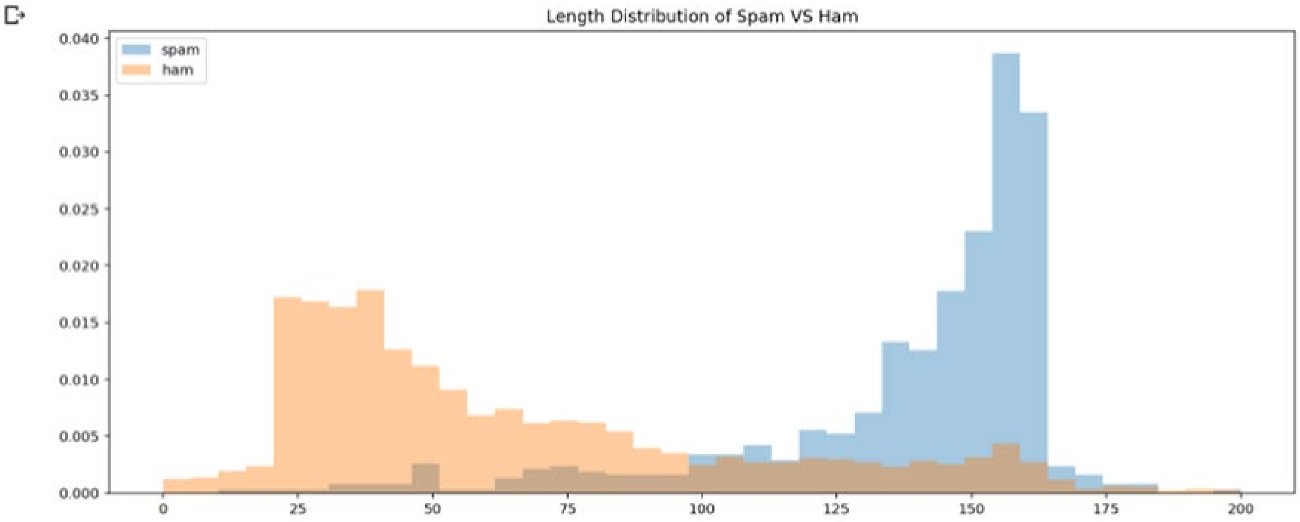
[24]
```
# sns.barplot(data['Mes_len'],data['label'])
# plt.show()
fig_dims = (20, 4)
fig, ax = plt.subplots(figsize=fig_dims)
sns.barplot(x='text_len',y='label',ax=ax,data=data,palette='Set3')
plt.title("listing_type",fontsize=20)
```

```
Text(0.5, 1.0, 'listing_type')
```



Distribution of text length

```python
# Plot the distribution of text length (spam vs ham)
plt.figure(figsize=(15,6))
plt.hist(data[data['label_cat']==1]['text_len'],bins = np.linspace(0,200,num=40),alpha=0.4,label='spam',density=True)
plt.hist(data[data['label_cat']==0]['text_len'],bins = np.linspace(0,200,num=40),alpha =0.4,label ='ham', density=True)
plt.legend(loc ='upper left')
plt.title('Length Distribution of Spam VS Ham')
plt.show()
```



It means usually shorter messages are hams and longer messages are spams. Hence, classifiers such as Naive Bayes might turnout to be a success over here

Ham Tokenization for first 50 Words:

```
words = data[data.label=='ham'].clean_text.apply(lambda x: [word.lower() for word in x.split()])
ham_words = Counter()

for msg in words:
    ham_words.update(msg)

ham_words.most_common(50)
```

OutPut

```
[('get', 303),
 ('ltgt', 276),
 ('ok', 272),
 ('go', 247),
 ('ill', 236),
 ('know', 232),
 ('got', 231),
 ('like', 229),
 ('call', 229),
 ('come', 224),
 ('good', 222),
 ('time', 189),
```

Spam Tokenization for first 50 Words:

```
words = data[data.label=='spam'].clean_text.apply(lambda x: [word.lower() for word in x.split()])
spam_words = Counter()

for msg in words:
    spam_words.update(msg)

spam_words.most_common(50)
```

OutPut

```
[('call', 347),
 ('free', 216),
 ('txt', 150),
 ('mobile', 123),
 ('text', 120),
 ('claim', 113),
 ('stop', 113),
 ('reply', 101),
 ('prize', 92),
 ('get', 83),
 ('new', 69),
 ('send', 67),
```

Classification Model Data Preparation:

Classification Model

```
X = data['clean_text']
y = data['label_cat']
```

Vectorize the data

```
[49]
vect = CountVectorizer(min_df=5, ngram_range=(1,2)).fit(X)

X_vec = vect.transform(X)

len(vect.get_feature_names())
```

```
2256
```

Train and Split the Data

```
[50]
X_train, X_test, y_train, y_test = train_test_split(X_vec, y, test_size=0.1, random_state = 0)

print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(5014, 2256)
(558, 2256)
(5014,)
(558,)
```

Logistic Regression:

```
from sklearn.pipeline import Pipeline
from sklearn.pipeline import make_pipeline
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.linear_model import LogisticRegression

model_lr = Pipeline([('tfidf', TfidfTransformer()),
                     ('model',LogisticRegression()),
                     ])

model_lr.fit(X_train,y_train)

ytest = np.array(y_test)
pred_y = model_lr.predict(X_test)
```

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

print('accuracy %s' % accuracy_score(pred_y, y_test))
print(classification_report(ytest, pred_y))
```

```
accuracy 0.953405017921147
              precision    recall  f1-score   support

           0       0.95      1.00      0.97       465
           1       0.99      0.73      0.84        93

    accuracy                           0.95       558
   macro avg       0.97      0.86      0.91       558
weighted avg       0.95      0.95      0.95       558
```

MultinomialNB:

```
[53]
from sklearn.naive_bayes import MultinomialNB

model_nb = Pipeline([('tfidf', TfidfTransformer()),
                     ('model',MultinomialNB()),
                     ])

model_nb.fit(X_train,y_train)

ytest = np.array(y_test)
pred = model_nb.predict(X_test)
```

```
[54]
print('accuracy %s' % accuracy_score(pred, y_test))
print(classification_report(ytest, pred))
```

```
accuracy 0.96415770609319
              precision    recall  f1-score   support

           0       0.96      1.00      0.98       465
           1       0.99      0.80      0.88        93

    accuracy                           0.96       558
   macro avg       0.97      0.90      0.93       558
weighted avg       0.96      0.96      0.96       558
```

Random Forest Classifier:

```
[55]
from sklearn.ensemble import RandomForestClassifier

model_rf = Pipeline([('tfidf', TfidfTransformer()),
                    ('model',RandomForestClassifier(n_estimators=50)),
                    ])

model_rf.fit(X_train,y_train)

ytest = np.array(y_test)
preds = model_rf.predict(X_test)
```

```
[56]
print('accuracy %s' % accuracy_score(preds, y_test))
print(classification_report(ytest, preds))
```

```
accuracy 0.9695340501792115
              precision    recall  f1-score   support

           0       0.97      1.00      0.98       465
           1       0.99      0.83      0.90        93

    accuracy                           0.97       558
   macro avg       0.98      0.91      0.94       558
weighted avg       0.97      0.97      0.97       558
```

Linear SVC:

[57]

```python
from sklearn.svm import LinearSVC, SVC

model_svc = Pipeline([('tfidf', TfidfTransformer()),
                      ('model',LinearSVC()),
                      ])

model_svc.fit(X_train,y_train)

ytest = np.array(y_test)
predict = model_svc.predict(X_test)
```

[58]

```python
print('accuracy %s' % accuracy_score(predict, y_test))
print(classification_report(ytest, predict))
```

```
accuracy 0.982078853046595
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       465
           1       0.99      0.90      0.94        93

    accuracy                           0.98       558
   macro avg       0.98      0.95      0.97       558
weighted avg       0.98      0.98      0.98       558
```

SGD Classifier:

[59]

```
>>> from sklearn.linear_model import SGDClassifier

model_sg = Pipeline([('tfidf', TfidfTransformer()),
                     ('model',SGDClassifier()),
                     ])

model_sg.fit(X_train,y_train)

ytest = np.array(y_test)
predicted = model_sg.predict(X_test)
```

[60]

```
print('accuracy %s' % accuracy_score(predicted, y_test))
print(classification_report(ytest, predicted))
```

```
accuracy 0.978494623655914
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       465
           1       0.98      0.89      0.93        93

    accuracy                           0.98       558
   macro avg       0.98      0.94      0.96       558
weighted avg       0.98      0.98      0.98       558
```

Gradient Boosting Classifier:

```
[61]
from sklearn.ensemble import GradientBoostingClassifier

model_gb = Pipeline([('tfidf', TfidfTransformer()),
                     ('model', GradientBoostingClassifier(random_state=100, n_estimators=150,min_samples_split=100, max_depth=6)),
                     ])

model_gb.fit(X_train,y_train)

ytest = np.array(y_test)
y_pred = model_gb.predict(X_test)
```

```
[62]
print('accuracy %s' % accuracy_score(y_pred, y_test))
print(classification_report(ytest, y_pred))
```

```
accuracy 0.9695340501792115
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       465
           1       0.95      0.86      0.90        93

    accuracy                           0.97       558
   macro avg       0.96      0.93      0.94       558
weighted avg       0.97      0.97      0.97       558
```

Compare Models:

```
log_acc = accuracy_score(pred_y, y_test)
nb_acc = accuracy_score(pred, y_test)
rf_acc = accuracy_score(preds, y_test)
gb_acc = accuracy_score(y_pred, y_test)
svm_acc = accuracy_score(predict, y_test)
sg_acc = accuracy_score(predicted, y_test)
```

[64]

```
models = pd.DataFrame({
                    'Model': ['Logistic Regression', 'Naive Bayes', 'Random Forest', 'Gradient Boosting', 'SVM', 'SGD'],
                    'Score': [log_acc, nb_acc, rf_acc, gb_acc, svm_acc, sg_acc]})
models.sort_values(by='Score', ascending=False)
```

	Model	Score
4	SVM	0.982079
5	SGD	0.978495
2	Random Forest	0.969534
3	Gradient Boosting	0.969534
1	Naive Bayes	0.964158
0	Logistic Regression	0.953405

Inference

1. We provided the text and refined the text (removal of stopwords, punctuations, and performed lemmatization). This helped in improving the Accuracy.
2. We have used different Model Pipeline containing TfidfVectorizer, where SVM model gives the best accuracy score of 98%.
3. The top Spam Tokenized words are- Call, Txt, Claim, Prize, Stop etc. These words gives an indication that it is either an commercial SMS or Spam SMS which is not used in regular life.
4. Most likely spam SMS's have longer length in text as compared to Non Spam SMS.
5. Readability score is less or negative in Spam SMS as compared to Non Spam SMS.
6. Parts of speech that is adjective and adverbs, we can see that adjectives are used most frequently in Spam SMS as compared to Non Spam SMS.

Thank You!!!