K.T.S.P Mandal's

K.M.C College Khopoli

DEPARTMENT OF COMPUTER SCIENCE
KHOPOLI–410203

A Project Report

On

Spam Classification

Submitted To

University of Mumbai

By

<u>Miss Neha Narendra Ghonge</u>

Under Guidance Of

Prof. Sangeeta Menon

2021-2022

K.T.S.P MANDAL'S

KMC COLLEGE KHOPOLI

DEPARTMENT OF COMPUTER SCIENCE

CERTIFICATE

This is to certify that ***Neha Narendra Ghonge*** *has successfully completed the project on the topic of*

*"**Spam Classification**" in Sem-II.*

During the academic year 2021-2022 as per the guidelines issued by ***University of Mumbai***.

Teacher's Signature	***HOD's Signature***	***Examiner's Signature***

Date: ***Date:***

ACKNOWLEDGMENT

In the accomplishment of this project successfully, many people have best owned upon me their blessings and the heart pledged support, this time I am utilizing to thank all the people who have been concerned with this project.

Primarily, I would thank god for being able to complete this project with success. Then I would like to thank my principal **Prof,Dr.Pratap Patil** *and my project teacher* <u>**Prof. Sangeeta Menon**</u> *whose valuable guidance has been the ones that helped me patch this project and make it full proof success. Her suggestions and her instructions have served as the major contributor towards the completion of the project. I am also thankful to my head of department* <u>**Prof. Dhanashree Pawar**</u> *who encourage me and gave me moral support during my project.*

Technologies Used

Software requirement:

Software requirements for this system are as listed follows:

- Frontend : Python
- Software : Jupyter Notebook
- Operating System : Windows

Hardware requirement:

Minimum Hardware requirements for these system are listed below:

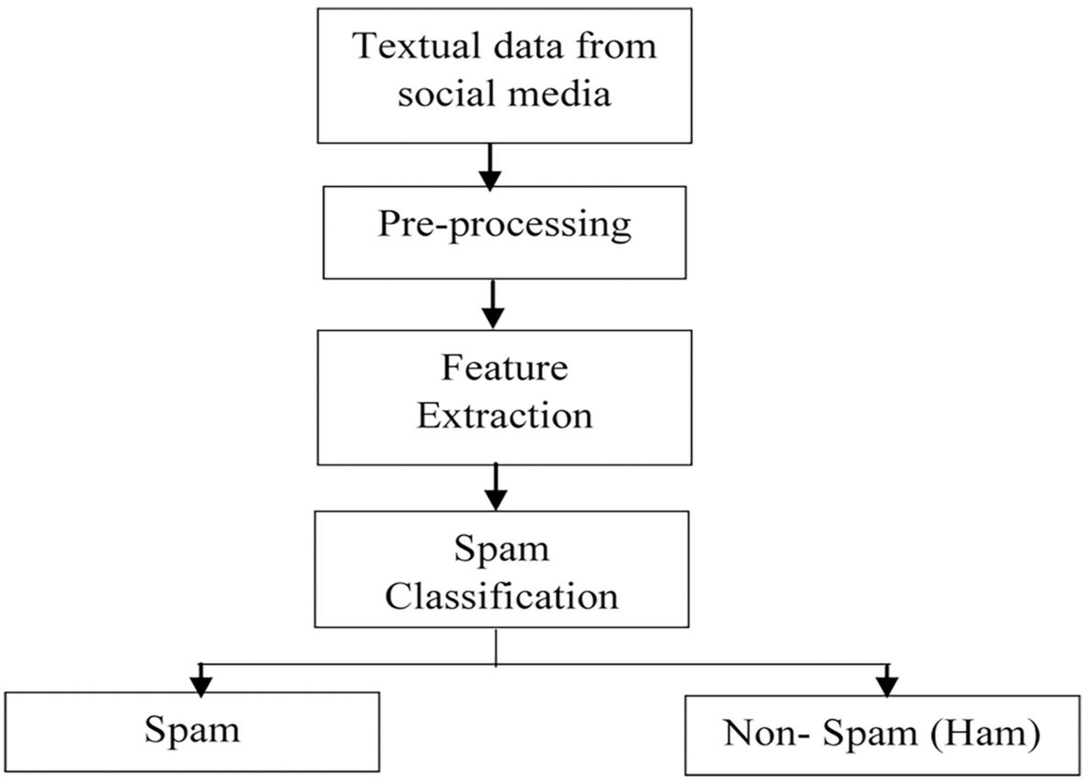
- C.P.U:- RMD Ryzen.
- R.A.M:- 8 Giga Bytes.
- Hard Disk:- 40 Giga Bytes.
- Type Of System : Single User

Description Of Project

What are spam messages?

Spam messages are messages sent to a large group of recipients without their prior consent, typically advertising for goods and services or business opportunities.
A spam message classification is a step towards building a tool for scam message identification and early scam detection.

Spam classification:-



Dataset

The dataset is from Kaggle, a collection of spam SMS messages, with 5572 messages, all classified as either 'ham' or 'spam' . The dataset contains 13.4% spam and 86.6% ham.

Methodology

The methodology is divided into
1.Import and read data
2.Exploratory data analysis (EDA)
3.Feature engineering
4.Cleaning text
5.Vectorization
6.Modelling (using RandomForestClassifier andGradientBoostingClassifier)

Exploratory data analysis (EDA):-

Exploratory data analysis is the process of performing initial investigations on data so as to discover patterns, to spot anomalies, to test hypothesis and to check assumptions with the help of summary statistics and graphical representations

Feature engineering:-
Feature engineering is the process of creating new features and/or transforming existing features to get the most out of your data.
In this section, I will create two new features:
- body_len (length of the body text excluding spaces)
- punct% (percentage of punctuation in the body text)

Cleaning text
In order to better manage our messy text messages, we will perform the following steps to clean up the input data:
- Turn words into lowercase letters only
- Remove punctuation
- Tokenize words
- Remove stopwords
- Stemming vs lemmatization (text normalization)

Turn words into lowercase letters only:-
Python does not see all characters as equal. Thus, we will need to convert all words into lowercase letters for consistency.

Remove punctuation:-
The rationale behind removing punctuation is that punctuation does not hold any meaning in a text. We want Python to only focus on the words in a given text and not worry about the punctuations that are involved.

Tokenize words:-
Tokenizing involves splitting a string or sentence into a list of characters and we can do so by utilising the regular expression (re) library in Python.

Remove stopwords:-
Stopwords are commonly used words in the English language like but, if and the that don't contribute much to the overall meaning of a sentence. For this reason, stopwords are usually removed in order to reduce the number of tokens Python needs to process when building our model.

Stemming vs lemmatization (word normalization):-
Stemming: The process of reducing inflection or derived words to their word stem or root by crudely chopping off the ends of a word to leave only the base.
Lemmatizing: The process of grouping together inflected forms of a word so they can be analyzed as a single term.
Broadly speaking, both stemming and lemmatizing serve the purpose of condensing the variations of the same word down to its root form. This is to prevent the computer from storing every single unique word it sees in a corpus of words but instead only take note of a word in its most basic form and correlate other words with similar meanings.

stemming takes a more crude approach than lemmatizing by simply chopping off the end of a word using heuristics, without any understanding of the context in which a word is used. As a result, stemming can sometimes not return an actual word in the dictionary unlike lemmatizing which will always return a dictionary word.
Lemmatizing, on the other hand, considers multiple factors before simplifying a given word and is generally considered more accurate compared to stemming. However, this comes at the expense of being slower and more computationally expensive than stemming.

Putting everything together into a single clean_text function:-
To summarise everything that we have learned about text cleaning into a single function that we can apply to our original text messages data.

Vectorization
Vectorizing is the process of encoding text as integers to create feature vectors.
How (CountVectorizer + TfidfTransformer) works?

CountVectorizer creates a document-term matrix where the entry of each cell will be a count of the number of times that word occurred in that document. TfidfTransformer is similar to that of a CountVectorizer but instead of the cells representing the count, the cells represent a weighting that is meant to identify how important a word is to an individual text message.

Modelling :-

Now that our data is ready, we can finally move on to modelling, that is building a binary classifier to classify a given text as ham or spam.
Here, we will consider two approaches: train-test-split and pipeline as well as two types of machine learning models, or more specifically, ensemble methods: random forest and gradient boosting.
Ensemble method is a technique that creates multiple models and then combine them to produce better results than any of the single models individually.

RandomForestClassifier:-

RandomForestClassifier is an ensemble learning method (bagging) that constructs a collection of decision trees and then aggregates the predictions of each tree to determine the final prediction.

The key hyperparameters to pay attention to are:

- **max_depth** (maximum depth of each decision tree)
- **n_estimators** (how many parallel decision trees to build)
- **random_state** (for reproducibility purpose)
- **n_jobs** (number of jobs to run in parallel)

GradientBoostingClassifier:-

GradientBoostingClassifier is an ensemble learning method (boosting) that takes an iterative approach to combine weak learners to create a strong learner by focusing on mistakes of prior iterations.
The key hyperparameters to pay attention to are:

- **learning_rate** (weight of each sequential tree on the final prediction)
- **max_depth** (maximum depth of each decision tree)
- **n_estimators** (number of sequential trees)
- **random_state** (for reproducibility purpose)

Coding And Output

IN 1:

```
import os
# Data wrangling and data visualistion
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# Processing text
import nltk
import re
import string
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer,
TfidfVectorizer
# Machine learning
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.compose import make_column_transformer
from sklearn.pipeline import make_pipeline
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_recall_fscore_support as score
# Others
import numpy as np
from collections import Counter
import time
from statistics import mean
data = pd.read_csv("D:\\archive\\spam.csv", encoding = "latin-1")
data = data.dropna(how = "any", axis = 1)
data.columns = ['label','body_text']
data.head()
```

Out 1:-

	label	body_text
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...

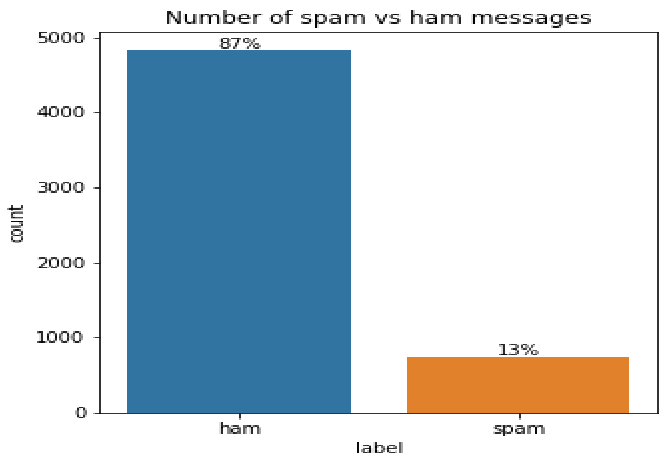
IN2:-

```
print(f"Input data has {len(data)} rows and {len(data.columns)} columns.")
print(f"Out of {len(data)} rows, {len(data[data.label == 'spam'])} are spam and
{len(data[data.label == 'ham'])} are ham.")
total = len(data)
plt.figure(figsize = (5, 5))
plt.title("Number of spam vs ham messages")
ax = sns.countplot(x = 'label', data = data)
```

```
for p in ax.patches:
    percentage = '{0:.0f}%'.format(p.get_height() / total * 100)
    x = p.get_x() + p.get_width() / 2
    y = p.get_height() + 20
    ax.annotate(percentage, (x, y), ha = 'center')
plt.show()
```

Out2:-

Input data has 5572 rows and 2 columns.
Out of 5572 rows, 747 are spam and 4825 are ham.



IN3:-

```
print(f"Number of null in label: {data.label.isnull().sum()}")
print(f"Number of null in text: {data.body_text.isnull().sum()}")
```

Out3:-

Number of null in label: 0
Number of null in text: 0

IN4:-

```
# body_len
data['body_len'] = data.body_text.apply(lambda x: len(x) - x.count(" "))
# punct%
def count_punct(text):
    count = sum([1 for char in text if char in string.punctuation])
    return round(count/(len(text) - text.count(" ")), 3) * 100
data['punct%'] = data.body_text.apply(lambda x: count_punct(x))

data.head()
```

Out4:-

label body_text body_lenpunct%
0 ham Go until jurong point, crazy.. Available only ...92 9.8
1 ham Ok lar... Joking wif u oni... 24 25.0

2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	128	4.7
3	ham	U dun say so early hor... U c already then say...	39	15.4
4	ham	Nah I don't think he goes to usf, he lives aro...	49	4.1

IN5:-

```
# Summary statistics
data[['body_len', 'punct%']].describe().transpose()
```

Out5:-

	count	mean	std	min	25%	50%	75%	max
body_len	5572.0	65.512024	48.629795	2.0	29.0	50.0	98.0	740.0
punct%	5572.0	7.202656	6.701062	0.0	3.3	5.6	9.2	100.0

IN6:-

```
# Text with maximum body_len
list(data.loc[data.body_len == 740, 'body_text'])
```

Out6:-

["For me the love should start with attraction.i should feel that I need her every time around me.she should be the first thing which comes in my thoughts.I would start the day and end it with her.she should be there every time I dream.love will be then when my every breath has her name.my life should happen around her.my life will be named to her.I would cry for her.will give all my happiness and take all her sorrows.I will be ready to fight with anyone for her.I will be in love when I will be doing the craziest things for her.love will be when I don't have to proove anyone that my girl is the most beautiful lady on the whole planet.I will always be singing praises for her.love will be when I start up making chicken curry and end up makiing sambar.life will be the most beautiful then.will get every morning and thank god for the day because she is with me.I would like to say a lot..will tell later.."]

IN7:-

```
# Text with maximum punct%
list(data.loc[data['punct%'] == 100, 'body_text'])
```

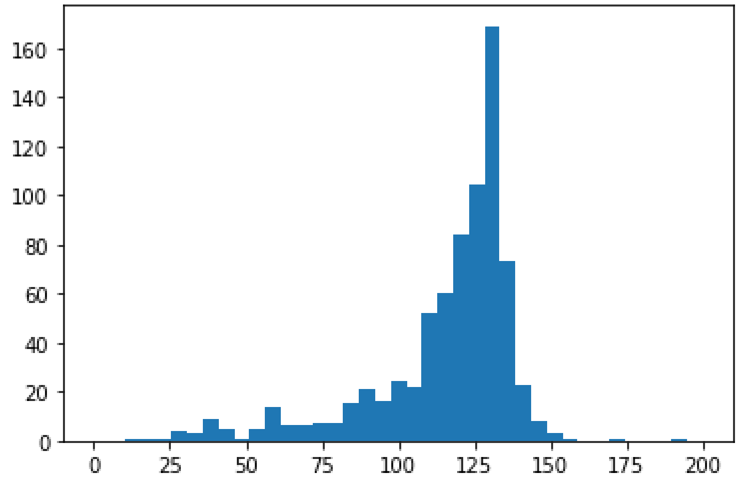
Out7:-

[':) ', ':-) :-)']

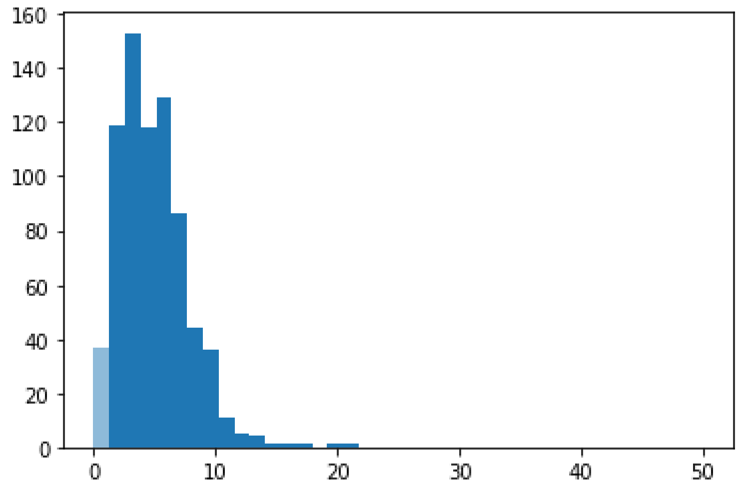
IN8:-

```
# Plot body_len distribution for ham and spam messages
bins = np.linspace(0, 200, 40)
data.loc[data.label == 'spam', 'body_len'].plot (kind = 'hist', bins = bins, alpha = 0.5, normed = True, label = 'spam')
data.loc[data.label == 'ham', 'body_len'].plot(kind = 'hist', bins = bins, alpha = 0.5, normed = True, label = 'ham')
plt.legend(loc = 'best')
plt.xlabel("body_len")
plt.title("Body length ham vs spam")
plt.show()
```

Out8:-



IN9:-
# Plot punct% for ham and spam messages
bins = np.linspace(0, 50, 40)
data.loc[data.label == 'spam', 'punct%'].plot(kind = 'hist', bins = bins, alpha = 0.5, normed = True, label = 'spam')
data.loc[data.label == 'ham', 'punct%'].plot(kind = 'hist', bins = bins, alpha = 0.5, normed = True, label = 'ham')
plt.legend(loc = 'best')
plt.xlabel("punct%")
plt.title("Punctuation percentage ham vs spam")
plt.show()



Out 9:-

IN10:-
"NLP" == "nlp"
Out10:-
False

IN11:-
"NLP".lower() == "nlp"
Out11:-
True

IN 12:-
"I love NLP" == "I love NLP."

Out 12:-
False

IN13:-
List of punctuations in the string library
string.punctuation

Out13:-
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'

IN 14:-
# Remove punctuation
text = 'OMG! Did you see what happened to her? I was so shocked when I heard the news. :('
print(text)
text = "".join([word for word in text if word not in string.punctuation])
print(text)

Out14:-
OMG! Did you see what happened to her? I was so shocked when I heard the news. :(
OMG Did you see what happened to her I was so shocked when I heard the news

IN 15:-
# Available commands in the re library
dir(re)
Out15:-
['A',
 'ASCII',
 'DEBUG',
 'DOTALL',
 'I',
 'IGNORECASE',
 'L',
 'LOCALE',
 'M',
 'MULTILINE',
 'Match',
 'Pattern',
 'RegexFlag',
 'S',
 'Scanner',
 'T',

```
 'TEMPLATE',
 'U',
 'UNICODE',
 'VERBOSE',
 'X',
 '_MAXCACHE',
 '__all__',
 '__builtins__',
 '__cached__',
 '__doc__',
 '__file__',
 '__loader__',
 '__name__',
 '__package__',
 '__spec__',
 '__version__',
 '_cache',
 '_compile',
 '_compile_repl',
 '_expand',
 '_locale',
 '_pickle',
 '_special_chars_map',
 '_subx',
 'compile',
 'copyreg',
 'enum',
 'error',
 'escape',
 'findall',
 'finditer',
 'fullmatch',
 'functools',
 'match',
 'purge',
 'search',
 'split',
 'sre_compile',
 'sre_parse',
 'sub',
 'subn',
 'template']
```

IN 16:-

```
messy_text = 'This-is-a-made/up.string*to>>>>test----2"""""different~regex-methods'
re.split('\W+', messy_text)
```

Out 16:-

```
['This',
 'is',
 'a',
```

```
 'made',
 'up',
 'string',
 'to',
 'test',
 '2',
 'different',
 'regex',
 'methods']
```

IN17:-

```
re.findall('\w+', messy_text)
```

Out 17:-

```
['This',
 'is',
 'a',
 'made',
 'up',
 'string',
 'to',
 'test',
 '2',
 'different',
 'regex',
 'methods']
```

IN18:-

```
# Examples of stopwords
stopwords = nltk.corpus.stopwords.words('english')
stopwords[0:500:25]
```

Out18:-

```
['i', 'herself', 'been', 'with', 'here', 'very', 'doesn', 'won']
```

IN19:-

```
print(text)
print(text.lower().split())
print([word for word in text.lower().split() if word not in stopwords])
```

Out19:-

```
OMG Did you see what happened to her I was so shocked when I heard the news
['omg', 'did', 'you', 'see', 'what', 'happened', 'to', 'her', 'i', 'was', 'so', 'shocked', 'when', 'i', 'heard', 'the', 'news']
['omg', 'see', 'happened', 'shocked', 'heard', 'news']
```

IN20:-

```
ps = nltk.PorterStemmer()
wn = nltk.WordNetLemmatizer()
print(ps.stem('goose'))
print(ps.stem('geese'))
```

Out20:-

goos
gees

IN21:-
```
print(wn.lemmatize('goose'))
print(wn.lemmatize('geese'))
```
Out21:-
```
goose
goose
```

IN22:-
```
# Create function for cleaning text
def clean_text(text):
    text = "".join([word.lower() for word in text if word not in string.punctuation])
    tokens = re.findall('\S+', text)
    # text = [ps.stem(word) for word in tokens if word not in stopwords]
    text = [wn.lemmatize(word) for word in tokens if word not in stopwords]
    return text
# Apply function to body_text
data['cleaned_text'] = data['body_text'].apply(lambda x: clean_text(x))
data[['body_text', 'cleaned_text']].head(10)
```
Out22:-
```
	body_text	cleaned_text
0	Go until jurong point, crazy.. Available only ...	[go, jurong, point, crazy, available, bugis, n...
1	Ok lar... Joking wif u oni...	[ok, lar, joking, wif, u, oni]
2	Free entry in 2 a wkly comp to win FA Cup fina...	[free, entry, 2, wkly, comp, win, fa, cup, fin...
3	U dun say so early hor... U c already then say...	[u, dun, say, early, hor, u, c, already, say]
4	Nah I don't think he goes to usf, he lives aro...	[nah, dont, think, go, usf, life, around, though]
5	FreeMsg Hey there darling it's been 3 week's n...	[freemsg, hey, darling, 3, week, word, back, i...
6	Even my brother is not like to speak with me. ...	[even, brother, like, speak, treat, like, aid,...
7	As per your request 'Melle Melle (Oru Minnamin...	[per, request, melle, melle, oru, minnaminungi...
8	WINNER!! As a valued network customer you have...	[winner, valued, network, customer, selected, ...
9	Had your mobile 11 months or more? U R entitle...	[mobile, 11, month, u, r, entitled, update, la...
```

IN23:-
```
# Collect ham words
ham_words = list(data.loc[data.label == 'ham', 'cleaned_text'])
# Flatten list of lists
ham_words = list(np.concatenate(ham_words).flat)
# Create dictionary to store word frequency
ham_words = Counter(ham_words)
```

```
pd.DataFrame(ham_words.most_common(50), columns = ['word', 'frequency'])
```
Out23:-

	word	frequency
0	u	1027
1	im	449
2	get	314
3	2	305
4	ltgt	276
5	go	273
6	ok	272
7	dont	257
8	come	242
9	know	241
10	call	241
11	ur	240
12	ill	236
13	like	232
14	got	231
15	good	223
16	day	214
17	time	213
18	love	193
19	want	183
20	need	171
21	one	170
22	4	168
23	going	167
24	home	160
25	lor	160
26	sorry	153
27	still	146
28	r	141
29	see	138
30	n	134
31	later	134
32	today	133
33	think	132
34	da	132
35	back	129
36	well	126
37	take	124
38	tell	124
39	send	123
40	say	118
41	cant	118
42	ì	117
43	hi	117
44	much	112
45	oh	111
46	make	111

47	thing	111
48	night	110
49	hey	106

IN24:-

```
# Collect spam words
spam_words = list(data.loc[data.label == 'spam', 'cleaned_text'])
# Flatten list of lists
spam_words = list(np.concatenate(spam_words).flat)
# Create dictionary to store word frequency
spam_words = Counter(spam_words)
pd.DataFrame(spam_words.most_common(50), columns = ['word', 'frequency'])
```

Out24:-

	word	frequency
0	call	359
1	free	216
2	2	173
3	u	155
4	txt	150
5	ur	144
6	text	137
7	mobile	135
8	4	119
9	claim	115
10	stop	113
11	reply	102
12	prize	94
13	get	83
14	tone	73
15	service	72
16	new	69
17	send	67
18	nokia	65
19	urgent	63
20	week	62
21	cash	62
22	win	61
23	phone	57
24	contact	56
25	please	52
26	customer	51
27	tc	50
28	guaranteed	50
29	min	50
30	16	49
31	per	46
32	message	43
33	18	43
34	chat	42

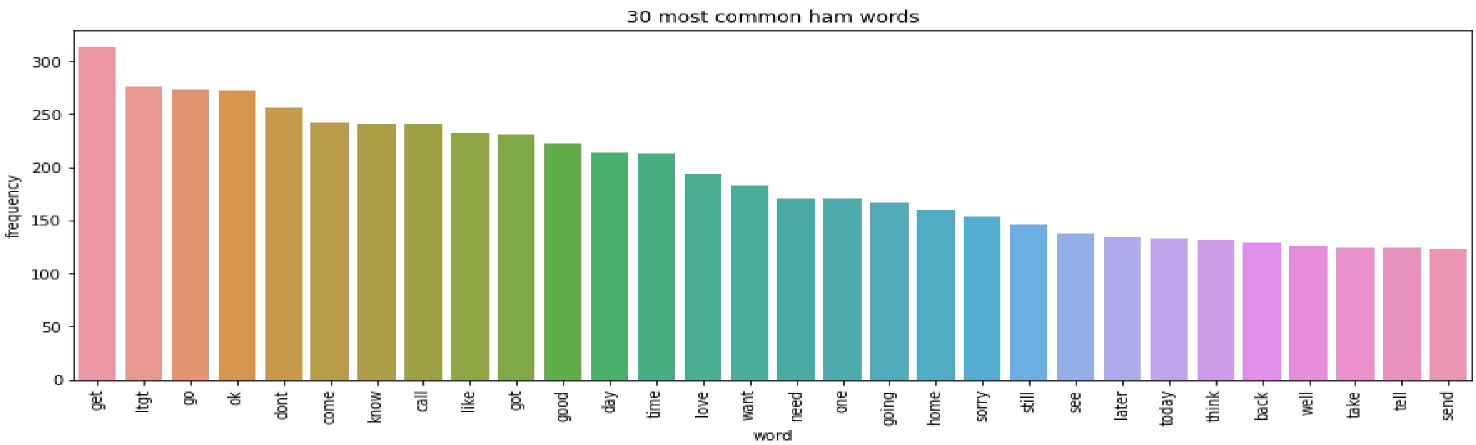
35	draw	39
36	number	39
37	awarded	38
38	latest	37
39	offer	37
40	line	37
41	today	36
42	voucher	36
43	å£1000	35
44	show	35
45	150ppm	34
46	landline	34
47	receive	33
48	camera	33
49	1	33

IN25:-

```
# Define extra stopwords
extra_stopwords = ['u', 'im', '2', 'ur', 'ill', '4', 'lor', 'r', 'n', 'da', 'oh']
# Remove extra stopwords
data['cleaned_text'] = data['cleaned_text'].apply(lambda x: [word for word in x if word not in extra_stopwords])
# Organise ham words data
ham_words = list(data.loc[data.label == 'ham', 'cleaned_text'])
ham_words = list(np.concatenate(ham_words).flat)
ham_words = Counter(ham_words)
ham_words = pd.DataFrame(ham_words.most_common(30), columns = ['word', 'frequency'])
# Plot most common harm words
fig, ax = plt.subplots(figsize = (15, 5))
sns.barplot(x = 'word', y = 'frequency', data = ham_words, ax = ax)
plt.xticks(rotation = '90')
plt.title("30 most common ham words")
```

Out25:-

Text(0.5, 1.0, '30 most common ham words')

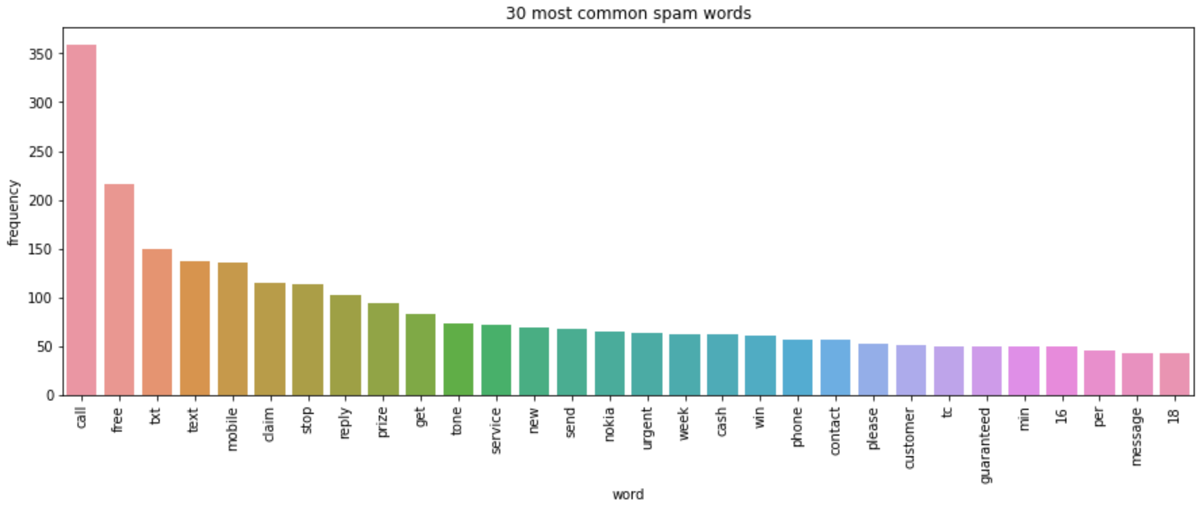


IN26:-

```
# Organise spam words data
spam_words = list(data.loc[data.label == 'spam', 'cleaned_text'])
spam_words = list(np.concatenate(spam_words).flat)
spam_words = Counter(spam_words)
spam_words = pd.DataFrame(spam_words.most_common(30), columns = ['word',
'frequency'])
# Plot most common harm words
fig, ax = plt.subplots(figsize = (15, 5))
sns.barplot(x = 'word', y = 'frequency', data = spam_words, ax = ax)
plt.xticks(rotation = '90')
plt.title("30 most common spam words")
```

Out26:-

Text(0.5, 1.0, '30 most common spam words')



IN27:-

```
# CountVectorizer
corpus = ['I love bananas', 'Bananas are so amazing!', 'Bananas go so well with pancakes']
count_vect = CountVectorizer()
corpus = count_vect.fit_transform(corpus)
count_vect.get_feature_names()
```

Out27:-

['amazing', 'are', 'bananas', 'go', 'love', 'pancakes', 'so', 'well', 'with']

IN28:-

```
pd.DataFrame(corpus.toarray(), columns = count_vect.get_feature_names())
```

Out28:-

	amazing	are	bananas	go	love	pancakes	so	well	with
0	0	0	1	0	1	0	0	0	0
1	1	1	1	0	0	0	1	0	0
2	0	0	1	1	0	1	1	1	1

IN29:-

```
# TfidfTransformer
tfidf_transformer = TfidfTransformer()
corpus = tfidf_transformer.fit_transform(corpus)
pd.DataFrame(corpus.toarray(), columns = count_vect.get_feature_names())
```

Out29:-

	amazing	are	bananas	go	love	pancakes	so	well	with
0	0.000000	0.000000	0.508542	0.000000	0.861037	0.000000	0.000000	0.000000	0.000000
1	0.584483	0.584483	0.345205	0.000000	0.000000	0.000000	0.444514	0.000000	0.000000
2	0.000000	0.000000	0.266075	0.450504	0.000000	0.450504	0.342620	0.450504	0.450504

IN30:-

```
# TfidfVectorizer
corpus = ['I love bananas', 'Bananas are so amazing!', 'Bananas go so well with pancakes']
tfidf_vect = TfidfVectorizer()
corpus = tfidf_vect.fit_transform(corpus)
pd.DataFrame(corpus.toarray(), columns = tfidf_vect.get_feature_names())
```

Out30:-

	amazing	are	bananas	go	love	pancakes	so	well	with
0	0.000000	0.000000	0.508542	0.000000	0.861037	0.000000	0.000000	0.000000	0.000000
1	0.584483	0.584483	0.345205	0.000000	0.000000	0.000000	0.444514	0.000000	0.000000
2	0.000000	0.000000	0.266075	0.450504	0.000000	0.450504	0.342620	0.450504	0.450504

IN31:-

```
data.head()
```

Out31:-

	label	body_text	body_len	punct%	cleaned_text
0	ham	Go until jurong point, crazy.. Available only ...	92	9.8	[go, jurong, point, crazy, available, bugis, g...
1	ham	Ok lar... Joking wif u oni...	24	25.0	[ok, lar, joking, wif, oni]
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	128	4.7	[free, entry, wkly, comp, win, fa, cup, final,...
3	ham	U dun say so early hor... U c already then say...	39	15.4	[dun, say, early, hor, c, already, say]
4	ham	Nah I don't think he goes to usf, he lives aro...	49	4.1	[nah, dont, think, go, usf, life, around, though]

IN32:-

```
# Train test split
X_train, X_test, Y_train, Y_test = train_test_split(data[['body_text', 'body_len', 'punct%']],
data.label, random_state = 42, test_size = 0.2)
# Check shape
print(f"X_train shape: {X_train.shape}")
print(f"Y_train shape: {Y_train.shape}")
print(f"X_test shape: {X_test.shape}")
print(f"Y_test shape: {Y_test.shape}")
X_trainshape: (4457, 3)
Y_trainshape: (4457,)
X_testshape: (1115,3 )
Y_testshape: (1115,)
```

Out32:-

```
X_train shape: (4457, 3)
Y_train shape: (4457,)
X_test shape: (1115, 3)
Y_test shape: (1115,)
```

IN33:-

```
# Instantiate and fit TfidfVectorizer
tfidf_vect = TfidfVectorizer(analyzer = clean_text)
tfidf_vect_fit = tfidf_vect.fit(X_train['body_text'])
# Use fitted TfidfVectorizer to transform body text in X_train and X_test
tfidf_train = tfidf_vect.transform(X_train['body_text'])
tfidf_test = tfidf_vect.transform(X_test['body_text'])
# Recombine transformed body text with body_len and punct% features
X_train = pd.concat([X_train[['body_len', 'punct%']].reset_index(drop = True),
pd.DataFrame(tfidf_train.toarray())], axis = 1)
X_test = pd.concat([X_test[['body_len', 'punct%']].reset_index(drop = True),
pd.DataFrame(tfidf_test.toarray())], axis = 1)
# Check shape
print(f"X_train shape: {X_train.shape}")
print(f"Y_train shape: {Y_train.shape}")
print(f"X_test shape: {X_test.shape}")
print(f"Y_test shape: {Y_test.shape}")
```

Out33:-

```
X_train shape: (4457, 7865)
Y_train shape: (4457,)
X_test shape: (1115, 7865)
Y_test shape: (1115,)
```

IN34:-

```
# Default random forest
print(RandomForestClassifier())
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0,
```

```
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, n_estimators=100,
                     n_jobs=None, oob_score=False, random_state=None,
                     verbose=0, warm_start=False)
```

Out34:-

```
RandomForestClassifier()
RandomForestClassifier()
```

IN 35:-

```
# Manual grid search for random forest
def explore_rf_params(n_est, depth):
    rf = RandomForestClassifier(n_estimators = n_est, max_depth = depth, n_jobs = -1,
random_state = 42)
    rf_model = rf.fit(X_train, Y_train)
    Y_pred = rf_model.predict(X_test)
    precision, recall, fscore, support = score(Y_test, Y_pred, pos_label = 'spam', average =
'binary')
    print(f"Est: {n_est} / Depth: {depth} ---- Precision: {round(precision, 3)} / Recall:
{round(recall, 3)} / Accuracy: {round((Y_pred==Y_test).sum() / len(Y_pred), 3)}")

for n_est in [50, 100, 150]:
    for depth in [10, 20, 30, None]:
        explore_rf_params(n_est, depth)
```

Out35:-

```
Est: 50 / Depth: 10 ---- Precision: 1.0 / Recall: 0.253 / Accuracy: 0.9
Est: 50 / Depth: 20 ---- Precision: 1.0 / Recall: 0.547 / Accuracy: 0.939
Est: 50 / Depth: 30 ---- Precision: 1.0 / Recall: 0.687 / Accuracy: 0.958
Est: 50 / Depth: None ---- Precision: 1.0 / Recall: 0.847 / Accuracy: 0.979
Est: 100 / Depth: 10 ---- Precision: 1.0 / Recall: 0.28 / Accuracy: 0.903
Est: 100 / Depth: 20 ---- Precision: 1.0 / Recall: 0.553 / Accuracy: 0.94
Est: 100 / Depth: 30 ---- Precision: 1.0 / Recall: 0.693 / Accuracy: 0.959
Est: 100 / Depth: None ---- Precision: 1.0 / Recall: 0.833 / Accuracy: 0.978
Est: 150 / Depth: 10 ---- Precision: 1.0 / Recall: 0.253 / Accuracy: 0.9
Est: 150 / Depth: 20 ---- Precision: 1.0 / Recall: 0.527 / Accuracy: 0.936
Est: 150 / Depth: 30 ---- Precision: 1.0 / Recall: 0.693 / Accuracy: 0.959
Est: 150 / Depth: None ---- Precision: 1.0 / Recall: 0.827 / Accuracy: 0.977
```

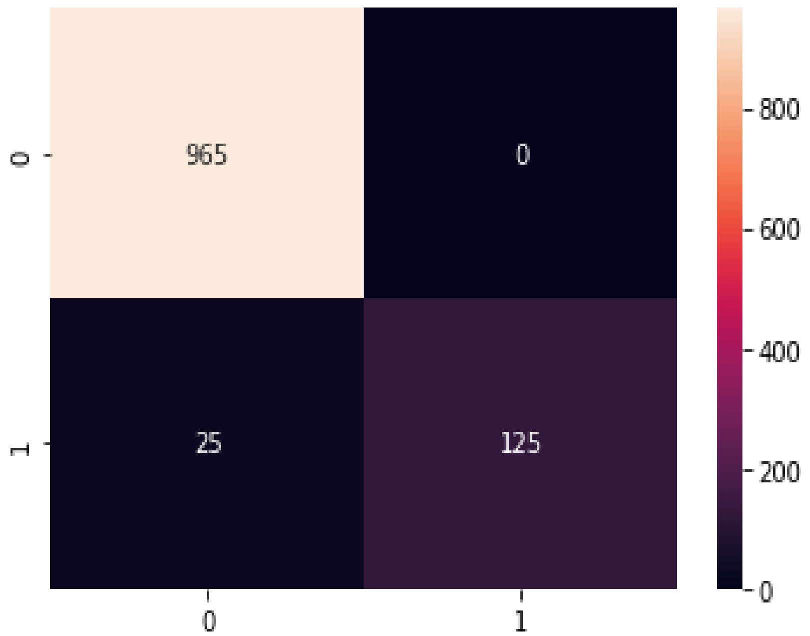
IN36:-

```
# Instantiate RandomForestClassifier with optimal set of hyperparameters
rf = RandomForestClassifier(n_estimators = 100, max_depth = None, random_state = 42,
n_jobs = -1)
# Fit model
start = time.time()
rf_model = rf.fit(X_train, Y_train)
end = time.time()
fit_time = end - start
# Predict
```

start = time.time()
Y_pred = rf_model.predict(X_test)
end = time.time()
pred_time = end - start
# Time and prediction results
precision, recall, fscore, support = score(Y_test, Y_pred, pos_label = 'spam', average = 'binary')
print(f"Fit time: {round(fit_time, 3)} / Predict time: {round(pred_time, 3)}")
print(f"Precision: {round(precision, 3)} / Recall: {round(recall, 3)} / Accuracy: {round((Y_pred==Y_test).sum() / len(Y_pred), 3)}")

Out36:-
Fit time: 4.795 / Predict time: 0.214
Precision: 1.0 / Recall: 0.833 / Accuracy: 0.978

IN37:-
# Confusion matrix for RandomForestClassifier
matrix = confusion_matrix(Y_test, Y_pred)
sns.heatmap(matrix, annot = True, fmt = 'd')
Out37:-
<AxesSubplot:>



IN38:-
#Default gradient boosting
print(GradientBoostingClassifier())
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
learning_rate=0.1, loss='deviance', max_depth=3,
max_features=None, max_leaf_nodes=None,

min_impurity_decrease=0.0,
min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, n_estimators=100,
n_iter_no_change=None,
random_state=None, subsample=1.0, tol=0.0001,
validation_fraction=0.1, verbose=0,
warm_start=False)

Out38:-

GradientBoostingClassifier()
GradientBoostingClassifier()

IN39:-

```
# Instantiate GradientBoostingClassifier
gb = GradientBoostingClassifier(random_state = 42)
# Fit model
start = time.time()
gb_model = gb.fit(X_train, Y_train)
end = time.time()
fit_time = end - start
# Predict
start = time.time()
Y_pred = gb_model.predict(X_test)
end = time.time()
pred_time = end - start
# Time and prediction results
precision, recall, fscore, support = score(Y_test, Y_pred, pos_label = 'spam', average =
'binary')
print(f"Fit time: {round(fit_time, 3)} / Predict time: {round(pred_time, 3)}")
print(f"Precision: {round(precision, 3)} / Recall: {round(recall, 3)} / Accuracy:
{round((Y_pred==Y_test).sum() / len(Y_pred), 3)}")
```

Out39:-

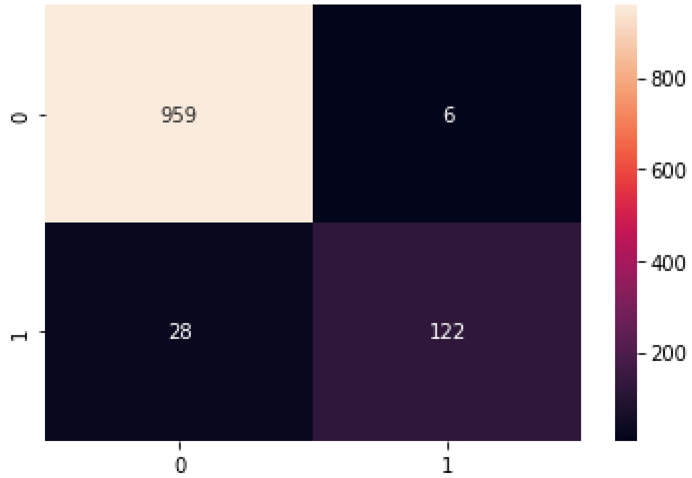
Fit time: 10162.323 / Predict time: 0.155
Precision: 0.953 / Recall: 0.813 / Accuracy: 0.97

IN40:-

```
# Confusion matrix for GradientBoostingClassifier
matrix = confusion_matrix(Y_test, Y_pred)
sns.heatmap(matrix, annot = True, fmt = 'd')
```

Out40:-

<AxesSubplot:>



IN41:-

```
# Instantiate TfidfVectorizer, RandomForestClassifier and GradientBoostingClassifier
tfidf_vect = TfidfVectorizer(analyzer = clean_text)
rf = RandomForestClassifier(random_state = 42, n_jobs = -1)
gb = GradientBoostingClassifier(random_state = 42)
# Make columns transformer
transformer = make_column_transformer((tfidf_vect, 'body_text'), remainder = 'passthrough')
# Build two separate pipelines for RandomForestClassifier and GradientBoostingClassifier
rf_pipeline = make_pipeline(transformer, rf)
gb_pipeline = make_pipeline(transformer, gb)
# Perform 5-fold cross validation and compute mean score
rf_score = cross_val_score(rf_pipeline, data[['body_text', 'body_len', 'punct%']], data.label, cv = 5, scoring = 'accuracy', n_jobs = -1)
gb_score = cross_val_score(gb_pipeline, data[['body_text', 'body_len', 'punct%']], data.label, cv = 5, scoring = 'accuracy', n_jobs = -1)
print(f"Random forest score: {round(mean(rf_score), 3)}")
print(f"Gradient boosting score: {round(mean(gb_score), 3)}")
```

Out41:-

```
Random forest score: 0.973
Gradient boosting score: 0.962
```

Conclusion And Future Use

Conclusion:-

To wrap up, we have successfully completed an end-to-end natural language processing (NLP) project which involves building a binary classifier capable of classifying a given text message as spam or ham.

We started off the project by exploring the dataset, followed by feature engineering where we created two new features: body_len and punct%. We then moved on to performing some preprocessing steps that are specific to the NLP pipeline such as removing punctuations and stopwords, tokenizing and stemming / lemmatization. After that, we performed vectorization using TfidfVectorizer in order to encode text and turn them into feature vectors for machine learning. Finally, we were able to build two separate prediction models: **RandomForestClassifier** and **GradientBoostingClassifier** as well as compare their accuracy and overall performance.

Future Use:-

- For classification of email spam.
- Applied in social media (Twitter).

References

- https://towardsdatascience.com/spam-messages-classification-3a7ede4f8ba1#:~:text=A%20spam%20message%20classification%20is,by%20Markus%20Winkler%20on%20Unsplash.&text=The%20dataset%20is%20from%20Kaggle,ham'%20or%20'spam'%20.
- https://towardsdatascience.com/how-to-build-your-first-spam-classifier-in-10-steps-fdbf5b1b3870
- https://www.kaggle.com/uciml/sms-spam-collection-dataset