***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*

***Miss Neha Narendra Ghonge***

*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 HOD’s Examiner’s***

***Signature Signature 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* ***Prof. Sangeeta Menon*** *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* ***Prof. Dhanashree Pawar*** *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:*

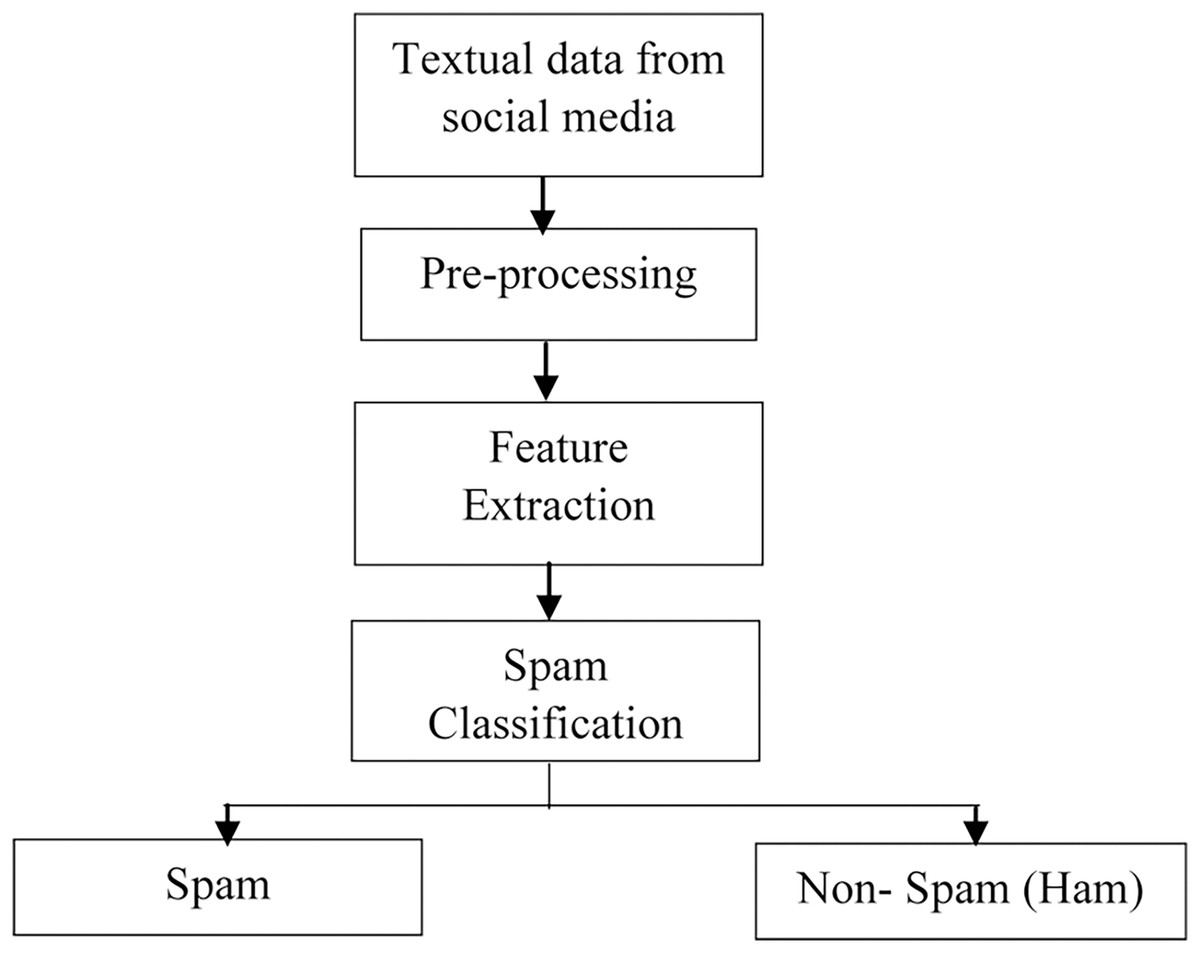
* C.P.U:- RMD Ryzen.
* R.A.M:- 8 Giga Bytes.
* Hard Disk:- 40 Giga Bytes.
* Type Of System : Single User

*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

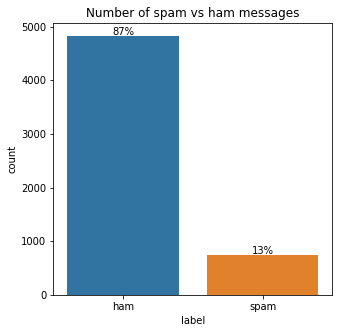
ax.annotate(percentage, (x, y), ha = 'center')

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\_len punct%

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')

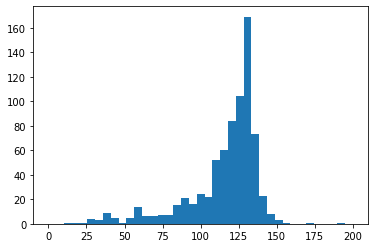
plt.legend(loc = 'best')

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')

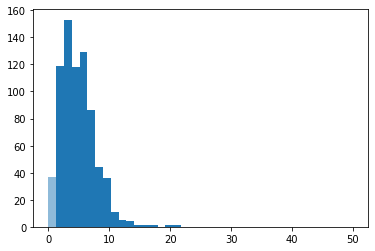
plt.legend(loc = 'best')

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

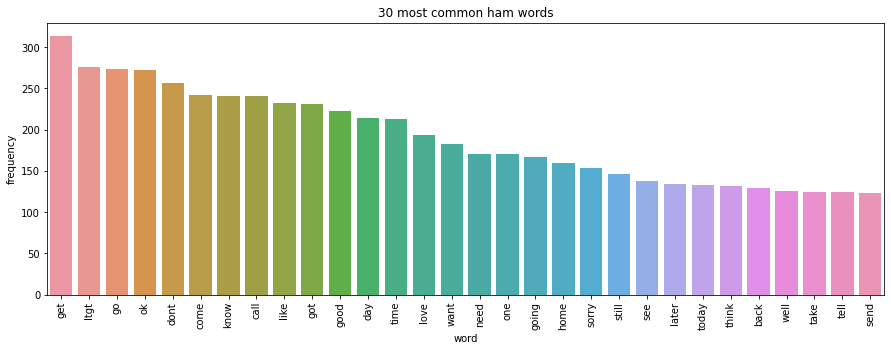
fig, ax = plt.subplots(figsize = (15, 5))

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))

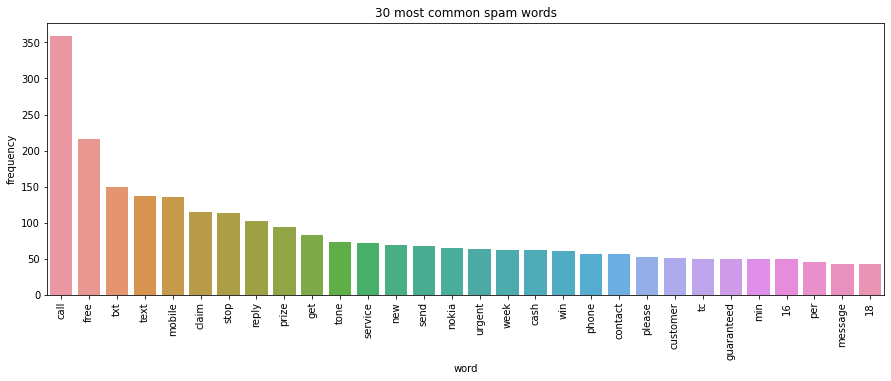
sns.barplot(x = 'word', y = 'frequency', data = spam\_words, ax = ax)

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:-**

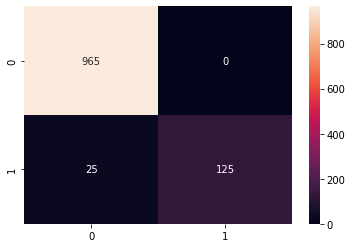
# Confusion matrix for RandomForestClassifier

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:-**

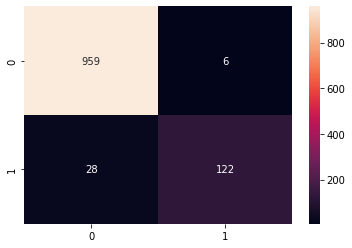
# Confusion matrix for GradientBoostingClassifier

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>