SPAM SMS DETECTION

August 9, 2023

Name of the Intern : Tejas Vidyadhar Kudalkar

Internship Project Name - SMS SPAM DETECTION Using Machine Learning

Company Name - CodSoft

[1]:
```
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
```

[2]:
```
data = pd.read_csv("C://Users//Dell//Downloads//spam.csv", encoding='latin-1')
```

[3]:
```
data
```

[3]:
```
        v1                                                 v2 Unnamed: 2  \
0      ham  Go until jurong point, crazy.. Available only …         NaN
1      ham                      Ok lar… Joking wif u oni…        NaN
2     spam  Free entry in 2 a wkly comp to win FA Cup fina…         NaN
3      ham  U dun say so early hor… U c already then say…        NaN
4      ham  Nah I don't think he goes to usf, he lives aro…         NaN
…    …                                                  …        …
5567  spam  This is the 2nd time we have tried 2 contact u…         NaN
5568   ham              Will Ì_ b going to esplanade fr home?          NaN
5569   ham  Pity, * was in mood for that. So…any other s…        NaN
5570   ham  The guy did some bitching but I acted like i'd…         NaN
5571   ham                        Rofl. Its true to its name          NaN

     Unnamed: 3 Unnamed: 4
0           NaN        NaN
1           NaN        NaN
2           NaN        NaN
3           NaN        NaN
4           NaN        NaN
```

```
…          …         …
5567        NaN        NaN
5568        NaN        NaN
5569        NaN        NaN
5570        NaN        NaN
5571        NaN        NaN

[5572 rows x 5 columns]
```

[4]:
```
data.shape
```

[4]:
```
(5572, 5)
```

[5]:
```
data.head()
```

[5]:
```
     v1                                                 v2 Unnamed: 2  \
0   ham  Go until jurong point, crazy.. Available only …         NaN
1   ham                      Ok lar… Joking wif u oni…        NaN
2  spam  Free entry in 2 a wkly comp to win FA Cup fina…         NaN
3   ham  U dun say so early hor… U c already then say…        NaN
4   ham  Nah I don't think he goes to usf, he lives aro…         NaN

  Unnamed: 3 Unnamed: 4
0        NaN        NaN
1        NaN        NaN
2        NaN        NaN
3        NaN        NaN
4        NaN        NaN
```

[6]:
```
data.tail()
```

[6]:
```
        v1                                                 v2 Unnamed: 2  \
5567  spam  This is the 2nd time we have tried 2 contact u…         NaN
5568   ham              Will Ì_ b going to esplanade fr home?         NaN
5569   ham  Pity, * was in mood for that. So…any other s…        NaN
5570   ham  The guy did some bitching but I acted like i'd…         NaN
5571   ham                         Rofl. Its true to its name         NaN

     Unnamed: 3 Unnamed: 4
5567        NaN        NaN
5568        NaN        NaN
5569        NaN        NaN
5570        NaN        NaN
5571        NaN        NaN
```

1 Data Cleaning

[7]:
```
data.isnull().sum()
```

[7]:
```
v1              0
v2              0
Unnamed: 2    5522
Unnamed: 3    5560
Unnamed: 4    5566
dtype: int64
```

Here Column v1 and v2 we have zero missing value and another three column has maximum missing values found so you can drop it.

[8]:
```
#Drop 3 Columns
data.drop(["Unnamed: 2","Unnamed: 3","Unnamed: 4"], axis = 1, inplace= True)
```

[9]:
```
data
```

[9]:
```
        v1                                                  v2
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…
…    …                                                   …
5567  spam  This is the 2nd time we have tried 2 contact u…
5568   ham              Will Ì_ b going to esplanade fr home?
5569   ham  Pity, * was in mood for that. So…any other s…
5570   ham  The guy did some bitching but I acted like i'd…
5571   ham                        Rofl. Its true to its name

[5572 rows x 2 columns]
```

[10]:
```
data.columns
```

[10]:
```
Index(['v1', 'v2'], dtype='object')
```

[11]:
```
# rename column names
data.columns = ['Category_Types','SMS']
```

[12]:
```
data.columns
```

[12]:
```
Index(['Category_Types', 'SMS'], dtype='object')
```

In above side we are rename the coumns names

[13]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5572 entries, 0 to 5571
Data columns (total 2 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   Category_Types  5572 non-null   object
 1   SMS             5572 non-null   object
dtypes: object(2)
memory usage: 87.2+ KB
```

[14]:
```
data.isnull().sum()
```

[14]:
```
Category_Types    0
SMS               0
dtype: int64
```

[15]:
```
data.head(10)
```

[15]:

	Category_Types	SMS
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…
5	spam	FreeMsg Hey there darling it's been 3 week's n…
6	ham	Even my brother is not like to speak with me. …
7	ham	As per your request 'Melle Melle (Oru Minnamin…
8	spam	WINNER!! As a valued network customer you have…
9	spam	Had your mobile 11 months or more? U R entitle…

[16]:
```
# #turn ham/spam into numerical data ,creating a new column called Spam.
data['Spam']=data['Category_Types'].apply(lambda x:1 if x=='spam' else 0)
```

We have to create a new column 'Spam' because machine learning cannot work on category data means text. so here column Category_Types which has 'ham' and 'spam' data . so we have to compare to categorical data into numerical data so as you see, so i compare ham or spam compare to 0 and 1.

[17]:
```
data.columns
```

[17]:
```
Index(['Category_Types', 'SMS', 'Spam'], dtype='object')
```

[18]:
```
data
```

[18]:

	Category_Types	SMS	Spam
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
…               …                                                  …    …
5567           spam  This is the 2nd time we have tried 2 contact u…     1
5568            ham              Will Ì_ b going to esplanade fr home?     0
5569            ham  Pity, * was in mood for that. So…any other s…     0
5570            ham  The guy did some bitching but I acted like i'd…     0
5571            ham                         Rofl. Its true to its name     0

[5572 rows x 3 columns]
```

The line of code you provided is using the pandas library to create a new column called 'Spam' in the DataFrame 'data' based on the values in the 'Category_Types' column.

data['Category_Types']: This accesses the 'Category_Types' column in the DataFrame 'data'. Assuming 'data' is a pandas DataFrame, it contains multiple columns, and this code focuses on the 'Category_Types' column.

.apply(lambda x: 1 if x == 'spam' else 0): The apply() function is used to apply a function to each element of the 'Category_Types' column. In this case, a lambda function is used. A lambda function is an anonymous function that takes an argument (in this case, represented as 'x') and returns a value based on a condition.

The lambda function checks if the value 'x' in the 'Category_Types' column is equal to the string 'spam'. If it is, it returns 1; otherwise, it returns 0. In other words, the lambda function is creating a new binary column where 'spam' is represented as 1, and all other values are represented as 0.

data['Spam'] = …: This assigns the results of the lambda function (1 or 0) to a new column called 'Spam' in the DataFrame 'data'.

This is a common technique used to convert categorical data into a binary representation for further analysis or machine learning tasks.

[19]:
```
# check duplicate values
data.duplicated().sum()
```

[19]:
```
403
```

[20]:
```
# remove duplicate values
data = data.drop_duplicates(keep='first')
```

[21]:
```
data.duplicated().sum()
```

[21]:
```
0
```

2 EDA

[22]:
```
data.shape
```

[22]:
```
(5169, 3)
```

[23]:
```
data.head()
```

[23]:

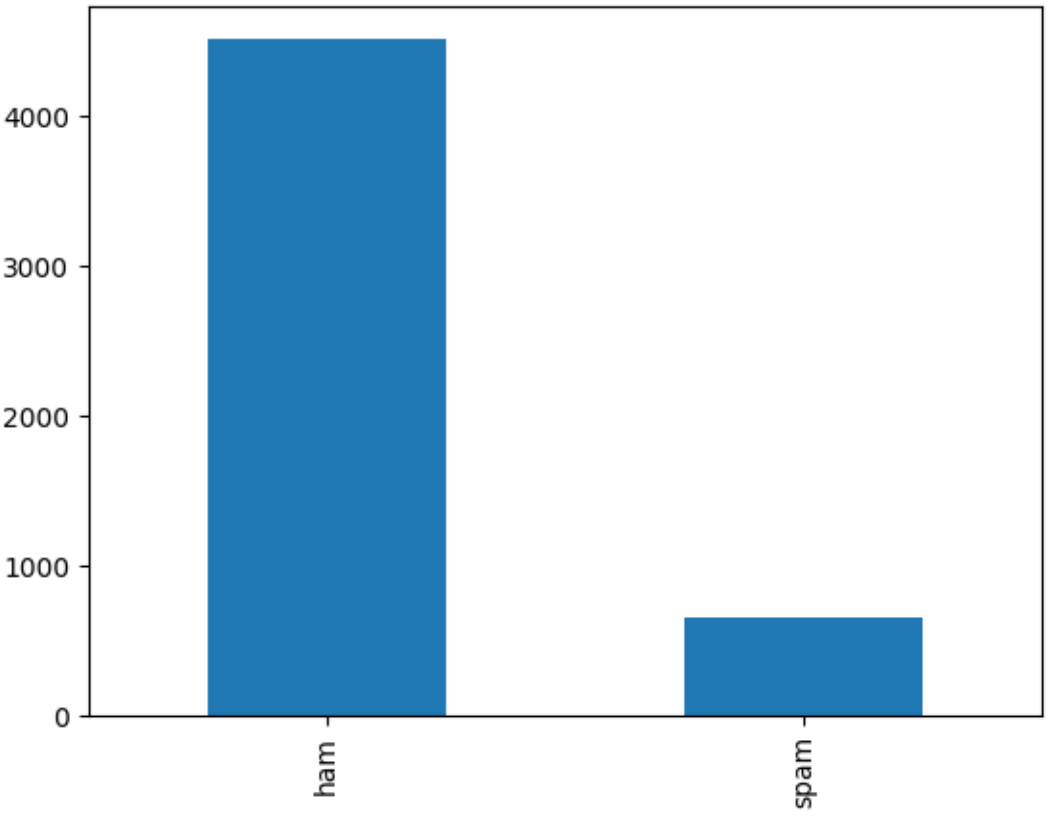
	Category_Types	SMS	Spam
0	ham	Go until jurong point, crazy.. Available only …	0
1	ham	Ok lar… Joking wif u oni…	0
2	spam	Free entry in 2 a wkly comp to win FA Cup fina…	1
3	ham	U dun say so early hor… U c already then say…	0
4	ham	Nah I don't think he goes to usf, he lives aro…	0

[24]:
```
data['Category_Types'].value_counts()
```

[24]:
```
ham     4516
spam     653
Name: Category_Types, dtype: int64
```

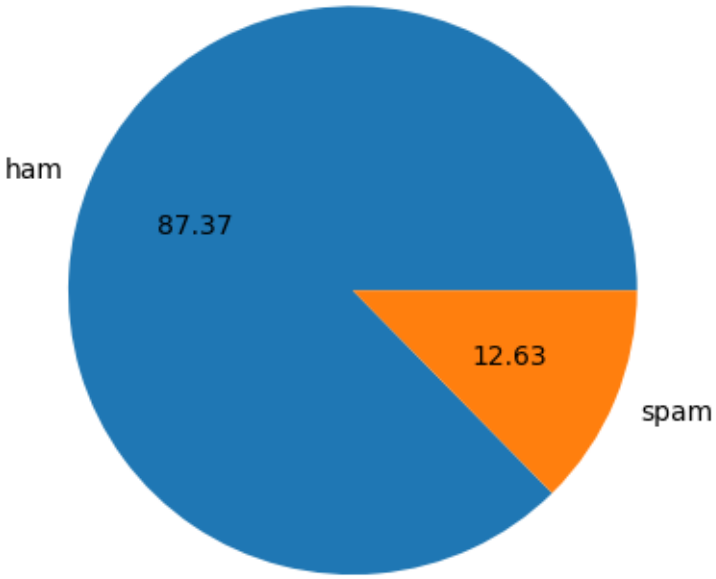
[25]:
```
data['Category_Types'].value_counts().plot(kind ='bar')
```

[25]:
```
<AxesSubplot: >
```



[26]:
```
import matplotlib.pyplot as plt
```

```
plt.pie(data['Category_Types'].value_counts(),labels =['ham','spam'],autopct␣
  ↪='%0.2f')
plt.show()
```



[27]:
```
# data is imbalanced
```

[28]:
```
#convert to X and Y
X = data.SMS
Y = data.Spam
print(X.shape)
print(Y.shape)
```

```
(5169,)
(5169,)
```

[29]:
```
#split the data into training and testing sets using the 'train_test_split'␣
  ↪function from sklearn.
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,random_state = 1)
```

[30]:
```
X_train.head()
```

[30]:
```
1                            Ok lar… Joking wif u oni…
5362    I'm in inside office..still filling forms.don …
3468    All day working day:)except saturday and sunday..
2865                         Smith waste da.i wanna gayle.
3781                      How r Ì_ going to send it to me?
Name: SMS, dtype: object
```

[31]:
```
Y_train.head()
```

[31]:
```
1       0
5362    0
3468    0
2865    0
3781    0
Name: Spam, dtype: int64
```

[32]:
```
# Used the 'TfidfVectorizer' from sklearn to convert the text data into␣
 ↪numerical vectors, considering the stop words in English.
# vectorizing the sentences,removing stop words
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer(stop_words='english')
```

[33]:
```
tfidf.fit(X_train)
```

[33]:
```
TfidfVectorizer(stop_words='english')
```

[34]:
```
#printing the vocabulary
tfidf.vocabulary_
```

[34]:
```
{'ok': 4508,
 'lar': 3691,
 'joking': 3534,
 'wif': 6878,
 'oni': 4527,
 'inside': 3389,
 'office': 4496,
 'filling': 2619,
 'forms': 2723,
 'don': 2235,
 'know': 3640,
 'leave': 3729,
 'day': 2027,
 'working': 6953,
 'saturday': 5439,
 'sunday': 6067,
 'smith': 5747,
 'waste': 6783,
 'da': 1988,
```

'wanna': 6767,
'gayle': 2849,
'ì_': 7099,
'going': 2924,
'send': 5521,
'freemsg': 2754,
'hi': 3145,
'baby': 1097,
'wow': 6971,
'just': 3562,
'got': 2947,
'new': 4372,
'cam': 1496,
'moby': 4182,
'hot': 3222,
'pic': 4758,
'fancy': 2559,
'chat': 1613,
'im': 3315,
'w8in': 6735,
'4utxt': 495,
'rply': 5354,
'82242': 615,
'hlp': 3167,
'08712317606': 96,
'msg150p': 4242,
'2rcv': 385,
'yup': 7080,
'noe': 4411,
'leh': 3741,
'xy': 7022,
'lunch': 3920,
'good': 2934,
'time': 6334,
'nice': 4384,
'bit': 1256,
'different': 2153,
'weekends': 6825,
'change': 1594,
'ya': 7024,
'soon': 5803,
'problem': 4977,
'sch': 5456,
'rem': 5206,
'correctly': 1863,
'blimey': 1280,
'exercise': 2489,

```
'yeah': 7035,
'kinda': 3623,
'remember': 5209,
'wot': 6966,
'hmm': 3170,
'haha': 3035,
'heard': 3102,
'text': 6244,
'common': 1771,
'hearin': 3103,
'wat': 6786,
'doing': 2228,
'ur': 6605,
'let': 3753,
'ask': 1005,
'did': 2139,
'smile': 5742,
'today': 6370,
'gud': 3008,
'evng': 2469,
'care': 1523,
'understand': 6555,
'ride': 5298,
'equally': 2425,
'uneventful': 6562,
'pesky': 4728,
'cyclists': 1985,
'night': 4391,
'replying': 5239,
'boye': 1352,
'changed': 1595,
'phone': 4745,
'number': 4457,
'probably': 4976,
'money': 4200,
'worries': 6960,
'things': 6287,
'coming': 1766,
'outstanding': 4591,
'invoices': 3424,
'work': 6950,
'months': 4212,
'ago': 815,
'course': 1886,
'tease': 6198,
'simply': 5665,
'grins': 2990,
```

'posted': 4884,
'prey': 4951,
'loving': 3896,
'devouring': 2124,
'kiss': 3630,
'whats': 6857,
'hill': 3153,
'monster': 4208,
'hope': 3204,
'great': 2979,
'fine': 2632,
'busy': 1449,
'respond': 5259,
'imma': 3321,
'assume': 1019,
'asleep': 1012,
'start': 5921,
'calling': 1491,
'shit': 5595,
'happy': 3072,
'year': 7036,
'man': 3983,
'oh': 4504,
'wasted': 6784,
'den': 2083,
'muz': 4286,
'chiong': 1658,
'sat': 5432,
'sun': 6065,
'liao': 3760,
'finished': 2636,
'eating': 2331,
'plate': 4796,
'leftovers': 3736,
'gone': 2930,
'info': 3369,
'bt': 1413,
'dont': 2238,
'water': 6792,
'logging': 3840,
'desert': 2103,
'geoenvironmental': 2872,
'implications': 3327,
'drop': 2280,
'tank': 6163,
'weekdays': 6823,
'special': 5845,

'price': 4952,
'haiz': 3040,
'eat': 2328,
'cut': 1976,
'nails': 4303,
'oso': 4573,
'wait': 6746,
'finish': 2635,
'drivin': 2276,
'morning': 4218,
'reach': 5118,
'home': 3192,
'safe': 5396,
'sound': 5820,
'feel': 2588,
'jus': 3561,
'way': 6799,
'lor': 3868,
'tot': 6416,
'dun': 2301,
'wan': 6765,
'stay': 5932,
'loverboy': 3892,
'does': 2214,
'keeps': 3595,
'queen': 5056,
'hmmm': 3171,
'doesn': 2216,
'ache': 744,
'speak': 5843,
'miss': 4144,
'desparately': 2108,
'unni': 6583,
'thank': 6259,
'dear': 2038,
'recharge': 5158,
'rakhesh': 5092,
'life': 3769,
'face': 2527,
'choices': 1664,
'toss': 6415,
'coin': 1740,
'becoz': 1184,
'settle': 5547,
'question': 5059,
'air': 828,
'heart': 3104,

```
'hoping': 3211,
'gudni8': 3009,
'ax': 1080,
'yuou': 7079,
'getting': 2883,
'pc': 4690,
'mom': 4194,
'spot': 5884,
'need': 4343,
'lt': 3909,
'gt': 3004,
'want': 6769,
'video': 6680,
'phone750': 4746,
'anytime': 921,
'network': 4365,
'mins': 4132,
'150': 285,
'pounds': 4893,
'week': 6822,
'08000776320': 49,
'reply': 5238,
'delivery': 2078,
'tomorrow': 6387,
'yar': 7029,
'used': 6620,
'dat': 2018,
'route': 5347,
'workin': 6952,
'overtime': 4597,
'nigpun': 4395,
'random': 5098,
'dude': 2295,
'sheets': 5577,
'party': 4661,
'study': 6008,
'thought': 6303,
'trip': 6461,
'looooooool': 3863,
'makes': 3976,
'sense': 5527,
'sofa': 5776,
'reference': 5174,
'sleep': 5712,
'couch': 1874,
'link': 3797,
'sent': 5531,
```

```
'wasn': 6780,
'went': 6842,
'didn': 2141,
'babe': 1094,
'celebration': 1575,
'rents': 5227,
'look': 3857,
'building': 1427,
'coat': 1730,
'sick': 5645,
'hurry': 3265,
'wear': 6806,
'gym': 3027,
'urgent': 6608,
'09066612661': 223,
'landline': 3680,
'complementary': 1782,
'tenerife': 6223,
'holiday': 3187,
'10': 242,
'000': 1,
'cash': 1545,
'await': 1070,
'collection': 1749,
'sae': 5394,
'cs': 1938,
'po': 4821,
'box': 1338,
'wa14': 6737,
'2px': 384,
'150ppm': 290,
'18': 301,
'sender': 5522,
'hol': 3181,
'offer': 4493,
'nights': 4394,
'nt': 4450,
'staying': 5935,
'port': 4867,
'step': 5943,
'ex': 2475,
'like': 3778,
'buff': 1422,
'wind': 6890,
'uhhhhrmm': 6531,
'isnt': 3451,
'having': 3088,
```

```
'tb': 6183,
'test': 6238,
'bad': 1102,
'youre': 7063,
'sorry': 5812,
'delay': 2070,
'yes': 7045,
'masters': 4020,
'pa': 4612,
'knw': 3644,
'ru': 5361,
'princess': 4959,
'pls': 4812,
'company': 1775,
'saibaba': 5400,
'colany': 1743,
'happening': 3067,
'til': 6331,
'custom': 1971,
'officer': 4497,
'discount': 2185,
'try': 6478,
'weight': 6829,
'tear': 6197,
'comes': 1762,
'falls': 2550,
'eyes': 2523,
'stupid': 6017,
'friend': 2769,
'share': 5573,
'bslvyl': 1410,
'hey': 3140,
'mate': 4024,
'hows': 3235,
'honey': 3196,
'ave': 1063,
'gimmi': 2894,
'goss': 2945,
'lost': 3873,
'okie': 4512,
'scared': 5453,
'say': 5446,
'fat': 2571,
'giving': 2903,
'problems': 4979,
'mayb': 4043,
'll': 3823,
```

```
'uncle': 6545,
'movies': 4233,
'guide': 3015,
'plus': 4816,
'torrents': 6412,
'particularly': 4657,
'legal': 3738,
'slowing': 5731,
'gr8': 2961,
'started': 5922,
'cos': 1866,
'meet': 4067,
'online': 4529,
'moon': 4214,
'sday': 5480,
'joined': 3528,
'training': 6434,
'callon': 1492,
'friday': 2765,
'won': 6934,
'lol': 3846,
'hungry': 3260,
'make': 3975,
'sure': 6090,
'alex': 850,
'knows': 3643,
'birthday': 1254,
'minutes': 4136,
'far': 2564,
'concerned': 1798,
'accidentally': 734,
'deleted': 2072,
'message': 4094,
'resend': 5249,
'greatest': 2980,
'courage': 1885,
'earth': 2321,
'bear': 1176,
'defeat': 2061,
'losing': 3871,
'gn': 2911,
'tc': 6185,
'shall': 5568,
'told': 6378,
'asking': 1009,
'wats': 6795,
'matter': 4034,
```

```
'worry': 6961,
'important': 3328,
'place': 4784,
'poorly': 4861,
'punishment': 5029,
'worst': 6963,
'thing': 6286,
'happened': 3065,
'brb': 1364,
'gonna': 2932,
'kill': 3617,
'collect': 1746,
'valentine': 6640,
'weekend': 6824,
'paris': 4650,
'flight': 2662,
'hotel': 3223,
'200': 323,
'prize': 4971,
'guaranteed': 3006,
'69101': 550,
'www': 6996,
'rtf': 5358,
'sphosting': 5859,
'com': 1755,
'happen': 3064,
'silent': 5657,
'tensed': 6224,
'juz': 3566,
'receive': 5151,
'sunshine': 6072,
'hols': 3190,
'claim': 1683,
'med': 4062,
'stamped': 5910,
'self': 5510,
'address': 768,
'envelope': 2420,
'drinks': 2272,
'uk': 6533,
'113': 257,
'bray': 1363,
'wicklow': 6875,
'eire': 2360,
'quiz': 5069,
'starts': 5924,
'unsub': 6588,
```

```
'stop': 5964,
'come': 1760,
'mum': 4265,
'repent': 5232,
'ttyl': 6486,
'house': 3230,
'right': 5299,
'okay': 4509,
'thanks': 6260,
'voucher': 6721,
'holder': 3184,
'weeks': 6827,
'http': 3245,
'tlp': 6355,
'reward': 5289,
'ts': 6480,
'apply': 942,
'free': 2748,
'msg': 4241,
'single': 5672,
'partner': 4659,
'area': 965,
'1000s': 246,
'real': 5129,
'people': 4705,
'waiting': 6749,
'62220cncl': 539,
'stopcs': 5969,
'08717890890å': 126,
'50': 501,
'room': 5335,
'û_': 7102,
'hadn': 3033,
'clocks': 1706,
'shouted': 5623,
'realised': 5131,
'wahay': 6741,
'hour': 3227,
'bed': 1186,
'09066649731from': 224,
'complimentary': 1787,
'ibiza': 3283,
'434': 457,
'sk3': 5693,
'8wp': 677,
'glad': 2904,
'11': 255,
```

```
'plenty': 4809,
'claire': 1686,
'goes': 2919,
'car': 1519,
'half': 3041,
'apeshit': 927,
'enjoy': 2401,
'urself': 6616,
'tmr': 6359,
'brought': 1403,
'shiny': 5589,
'warming': 6776,
'putting': 5044,
'constant': 1823,
'making': 3979,
'loved': 3888,
'cared': 1525,
'09061213237': 178,
'5000': 503,
'luxury': 3929,
'canary': 1504,
'islands': 3449,
'177': 300,
'm227xy': 3936,
'16': 296,
'dint': 2169,
'touch': 6420,
'received': 5153,
'mobile': 4175,
'content': 1828,
'boo': 1311,
'little': 3815,
'bored': 1322,
'affidavit': 798,
'says': 5449,
'twiggs': 6500,
'st': 5902,
'division': 2197,
'courtroom': 1887,
'double': 2248,
'check': 1620,
'pure': 5036,
'hearted': 3105,
'person': 4721,
'wonderful': 6938,
'enemies': 2392,
'guilty': 3018,
```

```
'enemy': 2393,
'catch': 1554,
'world': 6956,
'goodmorning': 2938,
'amp': 878,
'smiley': 5745,
'bears': 1177,
'nick': 4386,
'tom': 6382,
'pete': 4729,
'dick': 2136,
'fact': 2530,
'types': 6520,
'gay': 2848,
'photo': 4751,
'upload': 6598,
'08718730666': 138,
'10p': 252,
'min': 4118,
'texts': 6254,
'08712460324': 107,
'luck': 3913,
'catches': 1555,
'think': 6288,
'near': 4336,
'campus': 1500,
'printer': 4962,
'cool': 1850,
'mean': 4052,
'groovy': 2995,
'wine': 6895,
'groovying': 2996,
'ring': 5304,
'guys': 3024,
'costumes': 1872,
'gift': 2890,
'future': 2811,
'yowifes': 7069,
'hint': 3156,
'congrats': 1812,
'3g': 430,
'videophones': 6682,
'09063458130': 196,
'videochat': 6681,
'wid': 6876,
'mates': 4025,
'play': 4798,
```

```
'java': 3489,
'games': 2828,
'dload': 2203,
'polyph': 4850,
'music': 4279,
'noline': 4420,
'rentl': 5226,
'field': 2606,
'quickly': 5063,
'administrator': 773,
'help': 3122,
'swoop': 6125,
'picking': 4761,
'birds': 1249,
'meeting': 4069,
'hmv': 3174,
'bonus': 1310,
'500': 502,
'genuine': 2870,
'vouchers': 6722,
'answer': 905,
'easy': 2327,
'questions': 5061,
'86688': 644,
'100percent': 247,
'shag': 5562,
'interested': 3403,
'sextextuk': 5553,
'txt': 6505,
'xxuk': 7013,
'suzy': 6104,
'69876': 556,
'txts': 6513,
'cost': 1867,
'tncs': 6363,
'website': 6814,
'rounder': 5345,
'required': 5245,
'prin': 4956,
'4goten': 483,
'bout': 1335,
'scammers': 5452,
'smart': 5735,
'regular': 5194,
'vodafone': 6710,
'prem': 4925,
'rate': 5105,
```

'subscription': 6032,
'nos': 4436,
'beware': 1226,
'missing': 4148,
'pray': 4915,
'inshah': 3388,
'allah': 857,
'darren': 2016,
'saying': 5448,
'ge': 2858,
'dinner': 2167,
'later': 3702,
'awkward': 1077,
'needs': 4347,
'fucking': 2793,
'dealing': 2036,
'conacted': 1794,
'dating': 2023,
'service': 5541,
'entered': 2408,
'09111030116': 238,
'pobox12n146tf15': 4826,
'lobby': 3830,
'love': 3887,
'covers': 1891,
'kisses': 3631,
'actually': 759,
'decided': 2048,
'haven': 3084,
'left': 3735,
'early': 2319,
'project': 4992,
'quite': 5066,
'late': 3699,
'ard': 964,
'12': 261,
'wun': 6995,
'dad': 1991,
'gets': 2879,
'crazy': 1908,
'buy': 1453,
'maggi': 3962,
'mee': 4066,
'sen': 5520,
'kind': 3622,
'advice': 788,
'textbuddy': 6247,

```
'horny': 3214,
'25p': 347,
'search': 5484,
'postcode': 4883,
'gaytextbuddy': 2851,
'89693': 669,
'08715500022': 119,
'rpl': 5353,
'cnl': 1725,
'swollen': 6124,
'glands': 2905,
'throat': 6310,
'end': 2387,
'loosu': 3867,
'hospital': 3217,
'careless': 1530,
'bruce': 1407,
'downs': 2256,
'fletcher': 2659,
'exact': 2476,
'intentions': 3402,
'feb': 2585,
'gotta': 2953,
'lei': 3742,
'wanted': 6770,
'tell': 6211,
'score': 5463,
'relax': 5199,
'motivating': 4224,
'sharing': 5575,
'xmas': 7007,
'years': 7037,
'eve': 2454,
'tickets': 6325,
'sale': 5406,
'club': 1714,
'10am': 250,
'till': 6332,
'8pm': 675,
'thurs': 6319,
'fri': 2764,
'selling': 5515,
'fast': 2568,
'living': 3820,
'simple': 5663,
'laughing': 3707,
'winning': 6900,
```

```
'tooo': 6402,
'difficult': 2154,
'long': 3853,
'applebees': 940,
'ans': 903,
'huh': 3253,
'handed': 3051,
'dare': 2008,
'lolnice': 3847,
'fish': 2645,
'rofl': 5326,
'true': 6470,
'ola': 4515,
'maybe': 4044,
've': 6659,
'direct': 2172,
'cars': 1541,
'bids': 1234,
'arrange': 984,
'shipping': 5592,
'partnership': 4660,
'invest': 3417,
'takes': 6148,
'rest': 5264,
'wud': 6991,
'reliant': 5202,
'dnt': 2205,
'carlos': 1534,
'took': 6399,
'minute': 4135,
'sweet': 6114,
'likes': 3781,
'dislikes': 2189,
'chance': 1593,
'win': 6889,
'250': 344,
'wk': 6918,
'action': 753,
'80608': 608,
'movietrivia': 4234,
'tv': 6498,
'custcare': 1970,
'08712405022': 106,
'1x150p': 320,
'olol': 4520,
'printed': 4961,
'forum': 2726,
```

```
'post': 4880,
'guy': 3023,
'prob': 4975,
'fixed': 2649,
'gpu': 2960,
'replacement': 5234,
'hopefully': 3207,
'ignore': 3305,
'technical': 6201,
'support': 6083,
'providing': 5013,
'assistance': 1017,
'customer': 1972,
'email': 2376,
'jamster': 3479,
'videosound': 6683,
'gold': 2925,
'credits': 1917,
'videosounds': 6684,
'logos': 3844,
'musicnews': 4280,
'fun': 2801,
'09701213186': 240,
'vikky': 6688,
'sms': 5753,
'ac': 724,
'sptv': 5892,
'jersey': 3508,
'devils': 2123,
'detroit': 2117,
'red': 5169,
'wings': 6896,
'ice': 3289,
'hockey': 3177,
'correct': 1861,
'incorrect': 3350,
'especially': 2440,
'talk': 6152,
'boston': 1328,
'personal': 4723,
'statement': 5928,
'woulda': 6968,
'realized': 5135,
'said': 5401,
'nyc': 4469,
'explicit': 2509,
'sex': 5552,
```

```
'30': 397,
'secs': 5495,
'02073162414': 11,
'costs': 1870,
'20p': 331,
'gsex': 3003,
'pobox': 4823,
'2667': 349,
'wc1n': 6802,
'3xx': 443,
'wet': 6852,
'sooo': 5806,
'barely': 1133,
'stand': 5912,
'wonder': 6937,
'internet': 3408,
'connection': 1815,
'slow': 5730,
'tomo': 6385,
'fishrman': 2646,
'woke': 6930,
'mrng': 4238,
'dark': 2009,
'waited': 6747,
'sack': 5390,
'ful': 2799,
'stones': 5963,
'strtd': 5999,
'throwin': 6312,
'thm': 6297,
'in2': 3338,
'sea': 5483,
'pass': 4664,
'atlast': 1031,
'1stone': 316,
'rose': 5341,
'tht': 6316,
'diamonds': 2133,
'moral': 4215,
'wake': 6750,
'eerie': 2348,
'nokia': 4416,
'tones': 6390,
'4u': 494,
'tone': 6389,
'title': 6347,
'8007': 600,
```

```
'dracula': 2261,
'titles': 6348,
'ghost': 2887,
'addamsfa': 763,
'munsters': 4272,
'exorcist': 2495,
'twilight': 6501,
'getzed': 2884,
'pobox36504w45wq': 4830,
'150p': 287,
'effects': 2351,
'gorgeous': 2943,
'isn': 3450,
'brighten': 1383,
'thk': 6295,
'quit': 5065,
'jazz': 3494,
'yogasana': 7056,
'em': 2375,
'lessons': 3752,
'shanil': 5572,
'exchanged': 2481,
'uncut': 6553,
'diamond': 2132,
'stuff': 6011,
'leaving': 3731,
'excellent': 2480,
'dino': 2168,
'swing': 6121,
'hello': 3119,
'moving': 4235,
'flat': 2656,
'pick': 4759,
'lamp': 3676,
'caroline': 1537,
'hen': 3131,
'wah': 6739,
'def': 2060,
'wont': 6941,
'terms': 6229,
'conditions': 1802,
'wil': 6884,
'pull': 5025,
'case': 1544,
'plan': 4789,
'spending': 5857,
'confidence': 1805,
```

```
'derek': 2101,
'taylor': 6181,
'management': 3987,
'announcement': 899,
'recently': 5156,
'tried': 6460,
'unable': 6542,
'07090298926': 25,
'schedule': 5457,
'ref': 5173,
'9307622': 684,
'joys': 3544,
'lifeis': 3771,
'waking': 6751,
'daywith': 2030,
'thoughts': 6304,
'somewheresomeone': 5793,
'cares': 1531,
'tosend': 6413,
'warm': 6775,
'greeting': 2986,
'semester': 5518,
'apart': 924,
'yesterday': 7047,
'responsibilities': 5262,
'530': 516,
'dunno': 2302,
'appt': 956,
'fault': 2577,
'listen': 3810,
'twice': 6499,
'reboot': 5144,
'ym': 7052,
'seen': 5505,
'buzz': 1458,
'tuition': 6489,
'330': 416,
'hm': 3168,
'1120': 256,
'1205': 262,
'mind': 4121,
'1st': 313,
'no1': 4409,
'8077': 609,
'txting': 6511,
'36504': 420,
'w45wq': 6734,
```

```
 'norm150p': 4431,
 'ba': 1090,
 'dao': 2007,
 'pm': 4819,
 'ah': 817,
 'boytoy': 1356,
 'geeee': 2860,
 'tm': 6356,
 'remind': 5213,
 'thinking': 6291,
 'ill': 3312,
 'eatin': 2330,
 'significance': 5652,
 'normally': 4433,
 'mail': 3970,
 'worse': 6962,
 'uses': 6623,
 'stops': 5971,
 'better': 1222,
 'complete': 1783,
 'castor': 1551,
 'idea': 3294,
 'converted': 1840,
 'live': 3816,
 'available': 1059,
 'washob': 6779,
 'nobbing': 4410,
 'nickey': 4387,
 'platt': 4797,
 'instead': 3395,
 'unless': 6578,
 'situation': 5688,
 'gurl': 3021,
 'appropriate': 951,
 '44': 458,
 '7732584351': 583,
 '3510i': 418,
 'colour': 1752,
 'deliveredtomorrow': 2077,
 '300': 398,
 '100': 243,
 …}
```

[35]:
```
#vocab size
len(tfidf.vocabulary_.keys())
```

[35]: 7110

[36]:
```
# transforming the train and test datasets
X_train_transformed = tfidf.transform(X_train)
X_test_transformed = tfidf.transform(X_test)
```

[37]:
```
# note that the type is transformed (sparse) matrix
print(type(X_train_transformed))
print(X_train_transformed)
```

```
<class 'scipy.sparse._csr.csr_matrix'>
  (0, 6878)     0.4368017329157393
  (0, 4527)     0.5290929480381518
  (0, 4508)     0.28135732783089035
  (0, 3691)     0.412508914897925
  (0, 3534)     0.5290929480381518
  (1, 4496)     0.34630073157977126
  (1, 3729)     0.31445144618484844
  (1, 3640)     0.2369250505998017
  (1, 3389)     0.4251985856084617
  (1, 2723)     0.4877236556198847
  (1, 2619)     0.4877236556198847
  (1, 2235)     0.2617633396638459
  (2, 6953)     0.419904970793644
  (2, 6067)     0.504934288678524
  (2, 5439)     0.4677385415465359
  (2, 2027)     0.5915588190293688
  (3, 6783)     0.46068783822965204
  (3, 6767)     0.38310948096450137
  (3, 5747)     0.5140354933133167
  (3, 2849)     0.5395644291776539
  (3, 1988)     0.29262874035666403
  (4, 7099)     0.6266510037666704
  (4, 5521)     0.5480575885963683
  (4, 2924)     0.5540229228652536
  (5, 6971)     0.20230705808532568
  :     :
  (3871, 2027)  0.198883029342835
  (3871, 1702)  0.3788357992135742
  (3871, 1672)  0.3333233151527864
  (3872, 6746)  0.3335841421800436
  (3872, 5521)  0.27289935130558607
  (3872, 4132)  0.3448645414054676
  (3872, 3909)  0.2539034353468356
  (3872, 3310)  0.43715457453423917
  (3872, 3004)  0.25461871385998336
  (3872, 1643)  0.4015562757448001
  (3872, 805)   0.4622410666192576
  (3873, 2358)  1.0
  (3874, 6605)  0.17416802884883723
```

```
  (3874, 6292)  0.31118598335242004
  (3874, 5845)  0.2679878620862202
  (3874, 5491)  0.31637343656281564
  (3874, 5283)  0.3370735469126734
  (3874, 3975)  0.22548333701116602
  (3874, 3860)  0.2831754812345934
  (3874, 1825)  0.24030239795948263
  (3874, 774)   0.3370735469126734
  (3874, 462)   0.377429233724679
  (3874, 169)   0.377429233724679
  (3875, 3868)  0.8290013618171538
  (3875, 3566)  0.5592465843483576
```

3 Building and Evaluating the model

4 1. LogisticRegression

[38]:
```
# Step 1: Vectorize the text data using TF-IDF
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vectorizer = TfidfVectorizer(stop_words='english')
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
X_test_tfidf = tfidf_vectorizer.transform(X_test)

# Step 2: Train and evaluate the Logistic Regression model
from sklearn.linear_model import LogisticRegression

logreg_classifier = LogisticRegression()
logreg_classifier.fit(X_train_tfidf, Y_train)

# Step 3: Make predictions and evaluate the model
Y_pred_logreg = logreg_classifier.predict(X_test_tfidf)

# Now you can evaluate the performance of the classifier using metrics like␣
 ↪accuracy, precision, recall, etc.
from sklearn.metrics import accuracy_score, classification_report

accuracy_logreg = accuracy_score(Y_test, Y_pred_logreg)
print("Logistic Regression Accuracy:", accuracy_logreg)

report_logreg = classification_report(Y_test, Y_pred_logreg)
print("Logistic Regression Classification Report:\n", report_logreg)
```

```
Logistic Regression Accuracy: 0.9443155452436195
Logistic Regression Classification Report:
               precision    recall  f1-score   support
```

```
              0       0.94      1.00      0.97      1128
              1       0.96      0.59      0.73       165

       accuracy                           0.94      1293
      macro avg       0.95      0.79      0.85      1293
   weighted avg       0.95      0.94      0.94      1293
```

5 2. Support Vector Machine (SVM)

[39]:
```
# Step 1: Vectorize the text data using TF-IDF
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vectorizer = TfidfVectorizer(stop_words='english')
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
X_test_tfidf = tfidf_vectorizer.transform(X_test)

# Step 2: Train and evaluate the SVM model
from sklearn.svm import SVC

svm_classifier = SVC(kernel='linear')
svm_classifier.fit(X_train_tfidf, Y_train)

# Step 3: Make predictions and evaluate the model
Y_pred_svm = svm_classifier.predict(X_test_tfidf)

# Now you can evaluate the performance of the classifier using metrics like␣
 ↪accuracy, precision, recall, etc.
from sklearn.metrics import accuracy_score, classification_report

accuracy_svm = accuracy_score(Y_test, Y_pred_svm)
print("SVM Accuracy:", accuracy_svm)

report_svm = classification_report(Y_test, Y_pred_svm)
print("SVM Classification Report:\n", report_svm)
```

```
SVM Accuracy: 0.9767981438515081
SVM Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.99      0.99      1128
           1       0.96      0.85      0.90       165

    accuracy                           0.98      1293
   macro avg       0.97      0.92      0.95      1293
weighted avg       0.98      0.98      0.98      1293
```

6 3.Naive Bayes model

[40]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vectorizer = TfidfVectorizer(stop_words='english')
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
X_test_tfidf = tfidf_vectorizer.transform(X_test)

# Step 2: Train and evaluate Naive Bayes classifier
from sklearn.naive_bayes import MultinomialNB

naive_bayes_classifier = MultinomialNB()
naive_bayes_classifier.fit(X_train_tfidf, Y_train)

# Step 3: Make predictions and evaluate the Naive Bayes model
Y_pred_nb = naive_bayes_classifier.predict(X_test_tfidf)

# Now you can evaluate the performance of the Naive Bayes classifier using␣
 ↪metrics like accuracy, precision, recall, etc.
from sklearn.metrics import accuracy_score, classification_report

accuracy_nb = accuracy_score(Y_test, Y_pred_nb)
print("Naive Bayes Accuracy:", accuracy_nb)

report_nb = classification_report(Y_test, Y_pred_nb)
print("Naive Bayes Classification Report:\n", report_nb)
```

```
Naive Bayes Accuracy: 0.9574632637277649
Naive Bayes Classification Report:
               precision    recall  f1-score   support

           0       0.95      1.00      0.98      1128
           1       1.00      0.67      0.80       165

    accuracy                           0.96      1293
   macro avg       0.98      0.83      0.89      1293
weighted avg       0.96      0.96      0.95      1293
```

Based on the provided classification reports and accuracy scores, the Support Vector Machine (SVM) appears to be the best-fit model among the three algorithms (Logistic Regression, SVM, and Naive Bayes) for dataset.

SVM Accuracy: 0.9767981438515081

The SVM model shows consistent high performance across accuracy, F1-score, precision, and recall for both classes. This indicates that it is a robust model for your dataset and can effectively distinguish between the two classes (0 and 1) with high accuracy.

[41]:
```
import pickle
```

[42]:
```
pickle.dump(svm_classifier,open('model.pkl','wb'))
pickle.dump(tfidf,open('Vectorizer.pkl','wb'))
```

[ ]:

[ ]: