Stack Overflow: Tag Prediction

```
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import sqlite3
import csv
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from wordcloud import WordCloud
import re
import os
from sqlalchemy import create_engine # database connection
import datetime as dt
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem.snowball import SnowballStemmer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.multiclass import OneVsRestClassifier
from sklearn.linear_model import SGDClassifier
from sklearn import metrics
from sklearn.metrics import f1_score,precision_score,recall_score
from sklearn import svm
from sklearn.linear_model import LogisticRegression
from skmultilearn.adapt import mlknn
from skmultilearn.problem_transform import ClassifierChain
from skmultilearn.problem_transform import BinaryRelevance
from skmultilearn.problem_transform import LabelPowerset
from sklearn.naive_bayes import GaussianNB
from datetime import datetime
```

1. Business Problem

1.1 Description

Description

Stack Overflow is the largest, most trusted online community for developers to learn, share their programming know

Stack Overflow is something which every programmer use one way or another. Each month, over 50 million develop knowledge, and build their careers. It features questions and answers on a wide range of topics in computer prograr users to ask and answer questions, and, through membership and active participation, to vote questions and answe a fashion similar to a wiki or Digg. As of April 2014 Stack Overflow has over 4,000,000 registered users, and it excee Based on the type of tags assigned to questions, the top eight most discussed topics on the site are: Java, JavaScri

Problem Statemtent

Suggest the tags based on the content that was there in the question posted on Stackoverflow.

Source: https://www.kaggle.com/c/facebook-recruiting-iii-keyword-extraction/

1.2 Source / useful links

Data Source : https://www.kaggle.com/c/facebook-recruiting-iii-keyword-extraction/data
Youtube : https://youtu.be/nNDqbUhtIRg
Research paper : https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/tagging-1.pdf
Research paper : https://dl.acm.org/citation.cfm?id=2660970&dl=ACM&coll=DL

1.3 Real World / Business Objectives and Constraints

1. Predict as many tags as possible with high precision and recall.
2. Incorrect tags could impact customer experience on StackOverflow.
3. No strict latency constraints.

2. Machine Learning problem

2.1 Data

2.1.1 Data Overview

Refer: https://www.kaggle.com/c/facebook-recruiting-iii-keyword-extraction/data
All of the data is in 2 files: Train and Test.

```
Train.csv contains 4 columns: Id,Title,Body,Tags.

Test.csv contains the same columns but without the Tags, which you are to predict.

Size of Train.csv - 6.75GB

Size of Test.csv - 2GB

Number of rows in Train.csv = 6034195
```

The questions are randomized and contains a mix of verbose text sites as well as sites related to math and progran
may vary, and no filtering has been performed on the questions (such as closed questions).

Data Field Explaination

Dataset contains 6,034,195 rows. The columns in the table are:

```
Id - Unique identifier for each question

Title - The question's title

Body - The body of the question

Tags - The tags associated with the question in a space-seperated format (all lowercase, ampersands '&')
```

2.1.2 Example Data point

```
Title:  Implementing Boundary Value Analysis of Software Testing in a C++ program?
Body :

            #include<
            iostream>\n
            #include<
            stdlib.h>\n\n
            using namespace std;\n\n
            int main()\n
            {\n
                     int n,a[n],x,c,u[n],m[n],e[n][4];\n
                     cout<<"Enter the number of variables";\n         cin>>n;\n\n
                     cout<<"Enter the Lower, and Upper Limits of the variables";\n
                     for(int y=1; y<n+1; y++)\n
                     {\n
                        cin>>m[y];\n
                        cin>>u[y];\n
                     }\n
                     for(x=1; x<n+1; x++)\n
                     {\n
                        a[x] = (m[x] + u[x])/2;\n
                     }\n
                     c=(n*4)-4;\n
                     for(int a1=1; a1<n+1; a1++)\n
                     {\n\n
```

```
                e[a1][0] = m[a1];\n
                e[a1][1] = m[a1]+1;\n
                e[a1][2] = u[a1]-1;\n
                e[a1][3] = u[a1];\n
             }\n
             for(int i=1; i<n+1; i++)\n
             {\n
                for(int l=1; l<=i; l++)\n
                {\n
                    if(l!=1)\n
                    {\n
                        cout<<a[l]<<"\\t";\n
                    }\n
                }\n
                for(int j=0; j<4; j++)\n
                {\n
                    cout<<e[i][j];\n
                    for(int k=0; k<n-(i+1); k++)\n
                    {\n
                        cout<<a[k]<<"\\t";\n
                    }\n
                    cout<<"\\n";\n
                }\n
             }    \n\n
             system("PAUSE");\n
             return 0;    \n
    }\n


\n\n

        <p>The answer should come in the form of a table like</p>\n\n
        <pre><code>
        1            50              50\n
        2            50              50\n
        99           50              50\n
        100          50              50\n
        50           1               50\n
        50           2               50\n
        50           99              50\n
        50           100             50\n
        50           50              1\n
        50           50              2\n
        50           50              99\n
        50           50              100\n
        </code></pre>\n\n
        <p>if the no of inputs is 3 and their ranges are\n
```

```
        1,100\n
        1,100\n
        1,100\n
        (could be varied too)</p>\n\n
        <p>The output is not coming,can anyone correct the code or tell me what\'s wrong?<
```

```
Tags : 'c++ c'
```

2.2 Mapping the real-world problem to a Machine Learning Problem

2.2.1 Type of Machine Learning Problem

It is a multi-label classification problem
Multi-label Classification: Multilabel classification assigns to each sample a set of target labels. This can be though are not mutually exclusive, such as topics that are relevant for a document. A question on Stackoverflow might be a management at the same time or none of these.
__Credit__: http://scikit-learn.org/stable/modules/multiclass.html

2.2.2 Performance metric

Micro-Averaged F1-Score (Mean F Score) : The F1 score can be interpreted as a weighted average of the precision value at 1 and worst score at 0. The relative contribution of precision and recall to the F1 score are equal. The formu

*F1 = 2 * (precision * recall) / (precision + recall)*

In the multi-class and multi-label case, this is the weighted average of the F1 score of each class.

'Micro f1 score':
Calculate metrics globally by counting the total true positives, false negatives and false positives. This is a better me

'Macro f1 score':
Calculate metrics for each label, and find their unweighted mean. This does not take label imbalance into account.

https://www.kaggle.com/wiki/MeanFScore
http://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html

Hamming loss : The Hamming loss is the fraction of labels that are incorrectly predicted.
https://www.kaggle.com/wiki/HammingLoss

3. Exploratory Data Analysis

3.1 Data Loading and Cleaning

3.1.1 Using Pandas with SQLite to Load the data

```
#Creating db file from csv
#Learn SQL: https://www.w3schools.com/sql/default.asp
if not os.path.isfile('train.db'):
    start = datetime.now()
    disk_engine = create_engine('sqlite:///train.db')
    start = dt.datetime.now()
    chunksize = 180000
    j = 0
    index_start = 1
    for df in pd.read_csv('Train.csv', names=['Id', 'Title', 'Body', 'Tags'], chunksize=chunksize, i
        df.index += index_start
        j+=1
        print('{} rows'.format(j*chunksize))
        df.to_sql('data', disk_engine, if_exists='append')
        index_start = df.index[-1] + 1
    print("Time taken to run this cell :", datetime.now() - start)
```

3.1.2 Counting the number of rows

```
if os.path.isfile('train.db'):
    start = datetime.now()
    con = sqlite3.connect('train.db')
    num_rows = pd.read_sql_query("""SELECT count(*) FROM data""", con)
    #Always remember to close the database
    print("Number of rows in the database :","\n",num_rows['count(*)'].values[0])
    con.close()
    print("Time taken to count the number of rows :", datetime.now() - start)
else:
    print("Please download the train.db file from drive or run the above cell to genarate train.db f
```

```
Number of rows in the database :
 6034196
Time taken to count the number of rows : 0:01:15.750352
```

3.1.3 Checking for duplicates

```
#Learn SQl: https://www.w3schools.com/sql/default.asp
if os.path.isfile('train.db'):
    start = datetime.now()
    con = sqlite3.connect('train.db')
    df_no_dup = pd.read_sql_query('SELECT Title, Body, Tags, COUNT(*) as cnt_dup FROM data GROUP BY
    con.close()
    print("Time taken to run this cell :", datetime.now() - start)
else:
    print("Please download the train.db file from drive or run the first to genarate train.db file")
```

```
Time taken to run this cell : 0:04:33.560122
```

```
df_no_dup.head()
# we can observe that there are duplicates
```

	Title	Body
0	Implementing Boundary Value Analysis of S...	<pre><code>#include<iostream>\n#include&...
1	Dynamic Datagrid Binding in Silverlight?	<p>I should do binding for datagrid dynamicall...

```
print("number of duplicate questions :", num_rows['count(*)'].values[0]- df_no_dup.shape[0], "(",(1-
```

```
number of duplicate questions : 1827881 ( 30.2920389063 % )
```

```
# number of times each question appeared in our database
df_no_dup.cnt_dup.value_counts()
```

```
1    2656284
2    1272336
3     277575
4         90
5         25
6          5
Name: cnt_dup, dtype: int64
```

```
start = datetime.now()
df_no_dup["tag_count"] = df_no_dup["Tags"].apply(lambda text: len(text.split(" ")))
# adding a new feature number of tags per question
print("Time taken to run this cell :", datetime.now() - start)
df_no_dup.head()
```

```
Time taken to run this cell : 0:00:03.169523
```

	Title	Body	
0	Implementing Boundary Value Analysis of S...	<pre><code>#include<iostream>\n#include&...	
1	Dynamic Datagrid Binding in Silverlight?	<p>I should do binding for datagrid dynamicall...	
2	Dynamic Datagrid Binding in Silverlight?	<p>I should do binding for datagrid dynamicall...	c#
3	java.lang.NoClassDefFoundError: javax/serv...	<p>I followed the guide in <a href="http://sta...	
4	java.sql.SQLException:[Microsoft][ODBC Dri...	I use the following code \n\n<pre><code>...	

```
# distribution of number of tags per question
df_no_dup.tag_count.value_counts()
```

```
3    1206157
2    1111706
4     814996
1     568298
5     505158
Name: tag_count, dtype: int64
```

```
#Creating a new database with no duplicates
if not os.path.isfile('train_no_dup.db'):
    disk_dup = create_engine("sqlite:///train_no_dup.db")
    no_dup = pd.DataFrame(df_no_dup, columns=['Title', 'Body', 'Tags'])
    no_dup.to_sql('no_dup_train',disk_dup)
```

```
#This method seems more appropriate to work with this much data.
#creating the connection with database file.
if os.path.isfile('train_no_dup.db'):
    start = datetime.now()
    con = sqlite3.connect('train_no_dup.db')
    tag_data = pd.read_sql_query("""SELECT Tags FROM no_dup_train""", con)
    #Always remember to close the database
    con.close()

    # Let's now drop unwanted column.
    tag_data.drop(tag_data.index[0], inplace=True)
    #Printing first 5 columns from our data frame
    tag_data.head()
    print("Time taken to run this cell :", datetime.now() - start)
else:
    print("Please download the train.db file from drive or run the above cells to genarate train.db
```

```
Time taken to run this cell : 0:02:19.292131
```

3.2 Analysis of Tags

3.2.1 Total number of unique tags

```
# Importing & Initializing the "CountVectorizer" object, which
#is scikit-learn's bag of words tool.

#by default 'split()' will tokenize each tag using space.
vectorizer = CountVectorizer(tokenizer = lambda x: x.split())
# fit_transform() does two functions: First, it fits the model
# and learns the vocabulary; second, it transforms our training data
# into feature vectors. The input to fit_transform should be a list of strings.
tag_dtm = vectorizer.fit_transform(tag_data['Tags'])
```

```
print("Number of data points :", tag_dtm.shape[0])
print("Number of unique tags :", tag_dtm.shape[1])
```

```
Number of data points : 4206314
Number of unique tags : 42048
```

```
#'get_feature_name()' gives us the vocabulary.
tags = vectorizer.get_feature_names()
#Lets look at the tags we have.
print("Some of the tags we have :", tags[:10])
```

```
Some of the tags we have : ['.a', '.app', '.asp.net-mvc', '.aspxauth', '.bash-profile',
```

3.2.3 Number of times a tag appeared

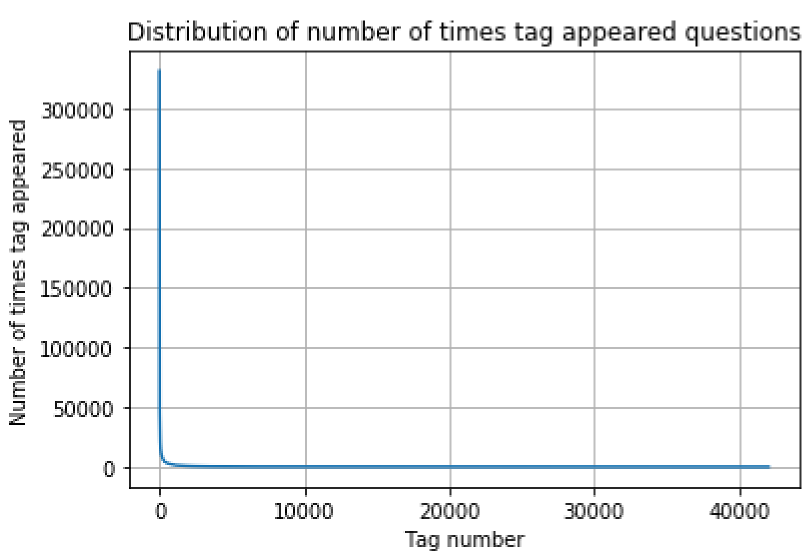
```
# https://stackoverflow.com/questions/15115765/how-to-access-sparse-matrix-elements
#Lets now store the document term matrix in a dictionary.
freqs = tag_dtm.sum(axis=0).A1
result = dict(zip(tags, freqs))
```

```
#Saving this dictionary to csv files.
if not os.path.isfile('tag_counts_dict_dtm.csv'):
    with open('tag_counts_dict_dtm.csv', 'w') as csv_file:
        writer = csv.writer(csv_file)
        for key, value in result.items():
            writer.writerow([key, value])
tag_df = pd.read_csv("tag_counts_dict_dtm.csv", names=['Tags', 'Counts'])
tag_df.head()
```

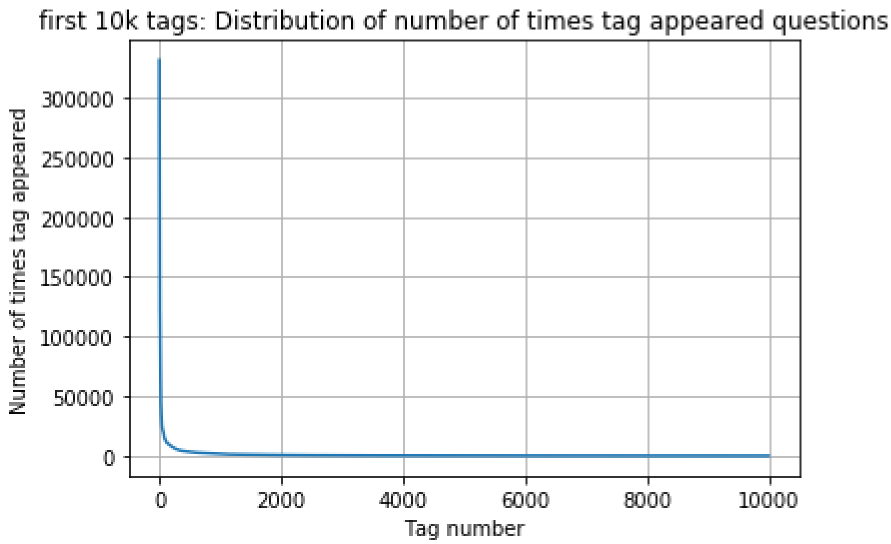
	Tags	Counts
0	.a	18
1	.app	37
2	.asp.net-mvc	1
3	.aspxauth	21
4	.bash-profile	138

```
tag_df_sorted = tag_df.sort_values(['Counts'], ascending=False)
tag_counts = tag_df_sorted['Counts'].values
```

```
plt.plot(tag_counts)
plt.title("Distribution of number of times tag appeared questions")
plt.grid()
plt.xlabel("Tag number")
plt.ylabel("Number of times tag appeared")
plt.show()
```



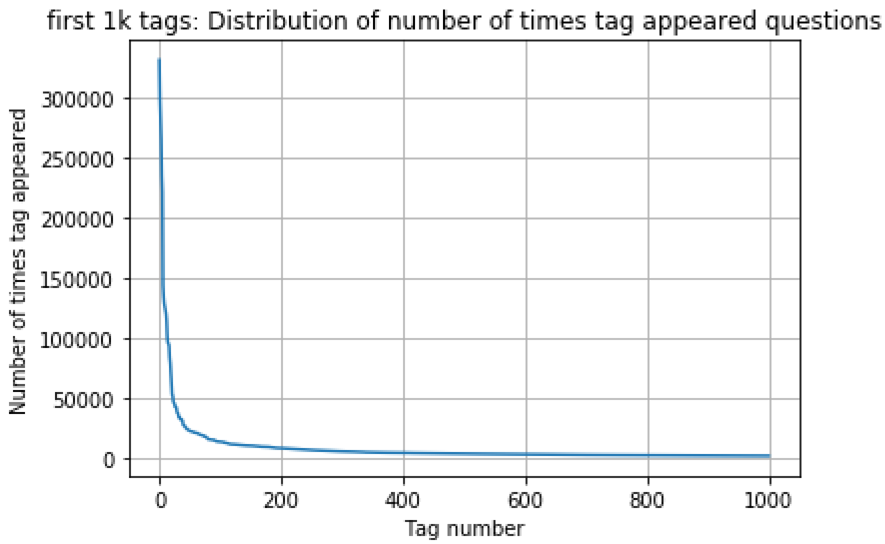
```
plt.plot(tag_counts[0:10000])
plt.title('first 10k tags: Distribution of number of times tag appeared questions')
plt.grid()
plt.xlabel("Tag number")
plt.ylabel("Number of times tag appeared")
plt.show()
print(len(tag_counts[0:10000:25]), tag_counts[0:10000:25])
```




```
400 [331505  44829  22429  17728  13364  11162  10029   9148   8054   7151
   6466   5865   5370   4983   4526   4281   4144   3929   3750   3593
   3453   3299   3123   2989   2891   2738   2647   2527   2431   2331
   2259   2186   2097   2020   1959   1900   1828   1770   1723   1673
   1631   1574   1532   1479   1448   1406   1365   1328   1300   1266
   1245   1222   1197   1181   1158   1139   1121   1101   1076   1056
   1038   1023   1006    983    966    952    938    926    911    891
    882    869    856    841    830    816    804    789    779    770
    752    743    733    725    712    702    688    678    671    658
    650    643    634    627    616    607    598    589    583    577
    568    559    552    545    540    533    526    518    512    506
    500    495    490    485    480    477    469    465    457    450
    447    442    437    432    426    422    418    413    408    403
    398    393    388    385    381    378    374    370    367    365
    361    357    354    350    347    344    342    339    336    332
    330    326    323    319    315    312    309    307    304    301
    299    296    293    291    289    286    284    281    278    276
    275    272    270    268    265    262    260    258    256    254
    252    250    249    247    245    243    241    239    238    236
    234    233    232    230    228    226    224    222    220    219
    217    215    214    212    210    209    207    205    204    203
    201    200    199    198    196    194    193    192    191    189
    188    186    185    183    182    181    180    179    178    177
    175    174    172    171    170    169    168    167    166    165
    164    162    161    160    159    158    157    156    156    155
    154    153    152    151    150    149    149    148    147    146
    145    144    143    142    142    141    140    139    138    137
    137    136    135    134    134    133    132    131    130    130
    129    128    128    127    126    126    125    124    124    123
    123    122    122    121    120    120    119    118    118    117
    117    116    116    115    115    114    113    113    112    111
    111    110    109    109    108    108    107    106    106    106
    105    105    104    104    103    103    102    102    101    101
    100    100     99     99     98     98     97     97     96     96
     95     95     94     94     93     93     93     92     92     91
     91     90     90     89     89     88     88     87     87     86
     86     86     85     85     84     84     83     83     83     82
```

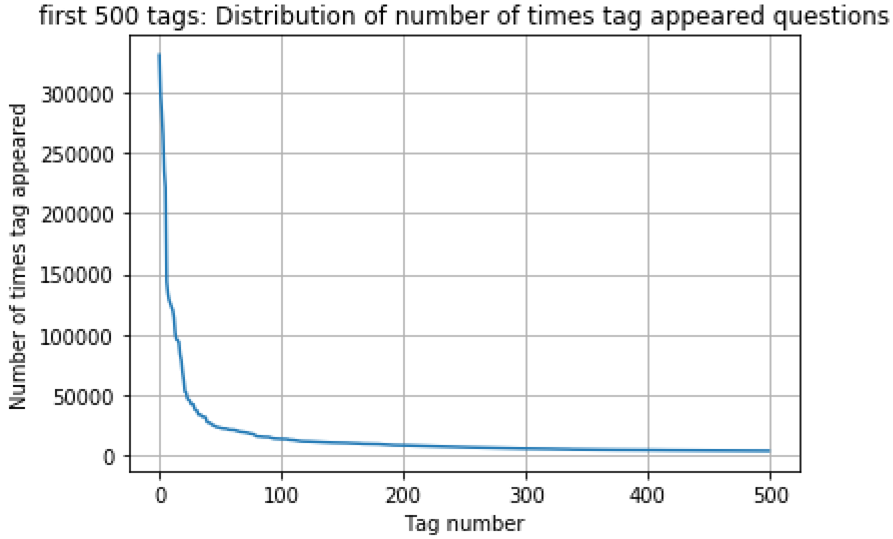
```
plt.plot(tag_counts[0:1000])
plt.title('first 1k tags: Distribution of number of times tag appeared questions')
plt.grid()
```

```
plt.xlabel("Tag number")
plt.ylabel("Number of times tag appeared")
plt.show()
print(len(tag_counts[0:1000:5]), tag_counts[0:1000:5])
```



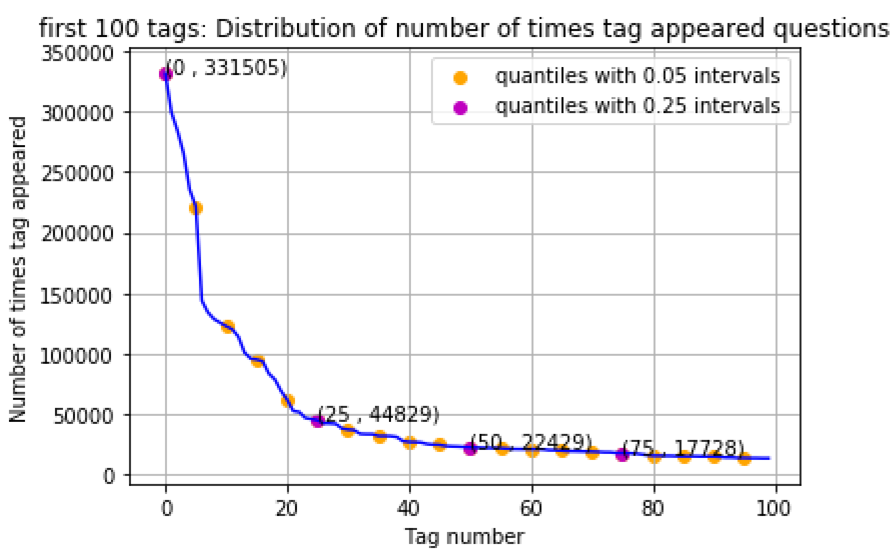

```
200 [331505 221533 122769  95160  62023  44829  37170  31897  26925  24537
  22429  21820  20957  19758  18905  17728  15533  15097  14884  13703
  13364  13157  12407  11658  11228  11162  10863  10600  10350  10224
  10029   9884   9719   9411   9252   9148   9040   8617   8361   8163
   8054   7867   7702   7564   7274   7151   7052   6847   6656   6553
   6466   6291   6183   6093   5971   5865   5760   5577   5490   5411
   5370   5283   5207   5107   5066   4983   4891   4785   4658   4549
   4526   4487   4429   4335   4310   4281   4239   4228   4195   4159
   4144   4088   4050   4002   3957   3929   3874   3849   3818   3797
   3750   3703   3685   3658   3615   3593   3564   3521   3505   3483
   3453   3427   3396   3363   3326   3299   3272   3232   3196   3168
   3123   3094   3073   3050   3012   2989   2984   2953   2934   2903
   2891   2844   2819   2784   2754   2738   2726   2708   2681   2669
   2647   2621   2604   2594   2556   2527   2510   2482   2460   2444
   2431   2409   2395   2380   2363   2331   2312   2297   2290   2281
   2259   2246   2222   2211   2198   2186   2162   2142   2132   2107
   2097   2078   2057   2045   2036   2020   2011   1994   1971   1965
   1959   1952   1940   1932   1912   1900   1879   1865   1855   1841
   1828   1821   1813   1801   1782   1770   1760   1747   1741   1734
   1723   1707   1697   1688   1683   1673   1665   1656   1646   1639]
```

```
plt.plot(tag_counts[0:500])
plt.title('first 500 tags: Distribution of number of times tag appeared questions')
plt.grid()
plt.xlabel("Tag number")
plt.ylabel("Number of times tag appeared")
plt.show()
print(len(tag_counts[0:500:5]), tag_counts[0:500:5])
```




```
100 [331505 221533 122769  95160  62023  44829  37170  31897  26925  24537
  22429  21820  20957  19758  18905  17728  15533  15097  14884  13703
  13364  13157  12407  11658  11228  11162  10863  10600  10350  10224
  10029   9884   9719   9411   9252   9148   9040   8617   8361   8163
   8054   7867   7702   7564   7274   7151   7052   6847   6656   6553
   6466   6291   6183   6093   5971   5865   5760   5577   5490   5411
```

```
plt.plot(tag_counts[0:100], c='b')
plt.scatter(x=list(range(0,100,5)), y=tag_counts[0:100:5], c='orange', label="quantiles with 0.05 in
# quantiles with 0.25 difference
plt.scatter(x=list(range(0,100,25)), y=tag_counts[0:100:25], c='m', label = "quantiles with 0.25 int

for x,y in zip(list(range(0,100,25)), tag_counts[0:100:25]):
    plt.annotate(s="({} , {})".format(x,y), xy=(x,y), xytext=(x-0.05, y+500))

plt.title('first 100 tags: Distribution of number of times tag appeared questions')
plt.grid()
plt.xlabel("Tag number")
plt.ylabel("Number of times tag appeared")
plt.legend()
plt.show()
print(len(tag_counts[0:100:5]), tag_counts[0:100:5])
```




```
20 [331505 221533 122769  95160  62023  44829  37170  31897  26925  24537
  22429  21820  20957  19758  18905  17728  15533  15097  14884  13703]
```

```
# Store tags greater than 10K in one list
lst_tags_gt_10k = tag_df[tag_df.Counts>10000].Tags
#Print the length of the list
print ('{} Tags are used more than 10000 times'.format(len(lst_tags_gt_10k)))
# Store tags greater than 100K in one list
lst_tags_gt_100k = tag_df[tag_df.Counts>100000].Tags
#Print the length of the list.
print ('{} Tags are used more than 100000 times'.format(len(lst_tags_gt_100k)))
```

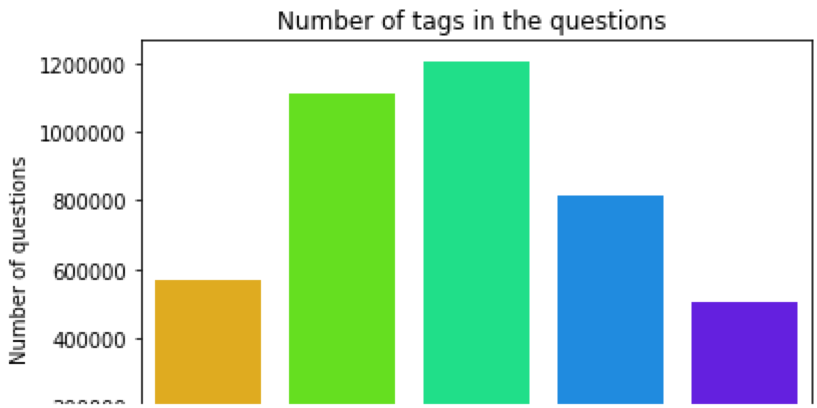
```
153 Tags are used more than 10000 times
14 Tags are used more than 100000 times
```

3.2.4 Tags Per Question

```
#Storing the count of tag in each question in list 'tag_count'
tag_quest_count = tag_dtm.sum(axis=1).tolist()
#Converting list of lists into single list, we will get [[3], [4], [2], [2], [3]] and we are convert
tag_quest_count=[int(j) for i in tag_quest_count for j in i]
print ('We have total {} datapoints.'.format(len(tag_quest_count)))

print(tag_quest_count[:5])
```

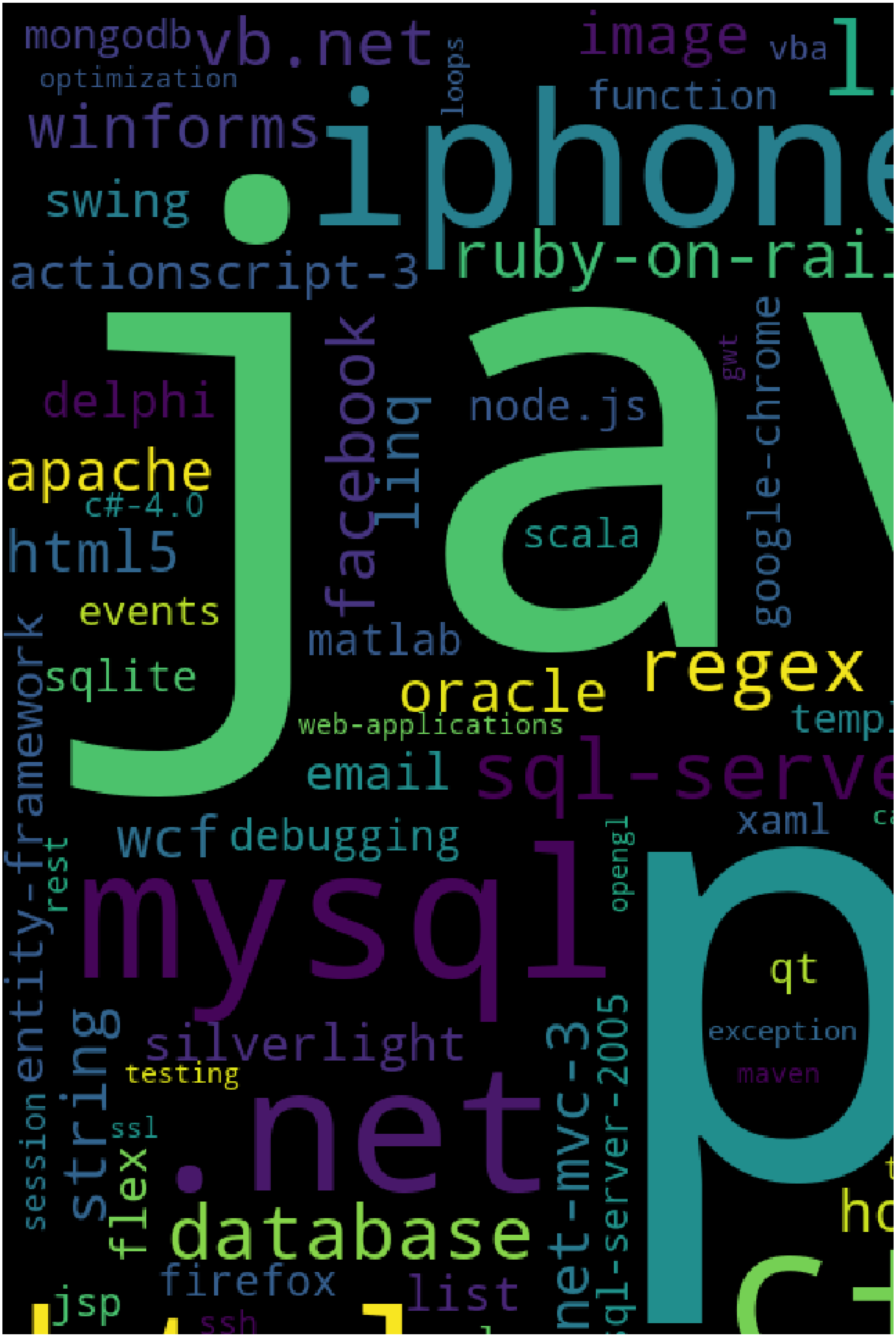
```
We have total 4206314 datapoints.
[3, 4, 2, 2, 3]
```

```
sns.countplot(tag_quest_count, palette='gist_rainbow')
plt.title("Number of tags in the questions ")
plt.xlabel("Number of Tags")
plt.ylabel("Number of questions")
plt.show()
```



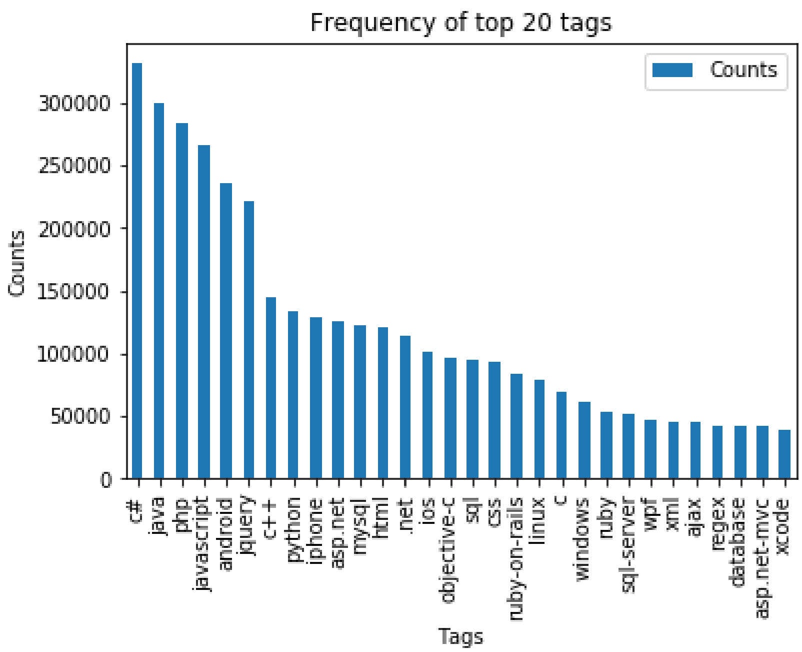
3.2.5 Most Frequent Tags

```
# Ploting word cloud
start = datetime.now()

# Lets first convert the 'result' dictionary to 'list of tuples'
tup = dict(result.items())
#Initializing WordCloud using frequencies of tags.
wordcloud = WordCloud(    background_color='black',
                          width=1600,
                          height=800,
                    ).generate_from_frequencies(tup)

fig = plt.figure(figsize=(30,20))
plt.imshow(wordcloud)
plt.axis('off')
plt.tight_layout(pad=0)
fig.savefig("tag.png")
plt.show()
print("Time taken to run this cell :", datetime.now() - start)
```


mongodb
vb.net
image
vba
optimization
loops
function
winforms
swing
actionscript-3
delphi
facebook
linq
node.js
gwt
google-chrome
apache
c#-4.0
html5
scala
events
entity-framework
matlab
sqlite
oracle
regex
web-applications
email
wcf
debugging
xaml
rest
opengl
mysql
qt
string
silverlight
asp.net-mvc-3
sql-server-2005
exception
testing
maven
session
ssl
flex
database
firefox
list
jsp
ssh

3.2.6 The top 20 tags

```
i=np.arange(30)
tag_df_sorted.head(30).plot(kind='bar')
plt.title('Frequency of top 20 tags')
plt.xticks(i, tag_df_sorted['Tags'])
plt.xlabel('Tags')
plt.ylabel('Counts')
plt.show()
```



4. Machine Learning Models

4.1 Modeling with less data points (0.1M data points) and more weight to tit

4.1.1 Preprocessing of questions

```
def striphtml(data):
    cleanr = re.compile('<.*?>')
    cleantext = re.sub(cleanr, ' ', str(data))
    return cleantext
stop_words = set(stopwords.words('english'))
stemmer = SnowballStemmer("english")
```

```
#http://www.sqlitetutorial.net/sqlite-python/create-tables/
def create_connection(db_file):
    """ create a database connection to the SQLite database
        specified by db_file
    :param db_file: database file
    :return: Connection object or None
    """
    try:
        conn = sqlite3.connect(db_file)
        return conn
    except Error as e:
        print(e)

    return None

def create_table(conn, create_table_sql):
    """ create a table from the create_table_sql statement
    :param conn: Connection object
    :param create_table_sql: a CREATE TABLE statement
    :return:
    """
    try:
        c = conn.cursor()
        c.execute(create_table_sql)
    except Error as e:
        print(e)

def checkTableExists(dbcon):
    cursr = dbcon.cursor()
    str = "select name from sqlite_master where type='table'"
    table_names = cursr.execute(str)
    print("Tables in the databse:")
    tables =table_names.fetchall()
    print(tables[0][0])
    return(len(tables))

def create_database_table(database, query):
    conn = create_connection(database)
    if conn is not None:
        create_table(conn, query)
        checkTableExists(conn)
    else:
        print("Error! cannot create the database connection.")
    conn.close()


sql_create_table = """CREATE TABLE IF NOT EXISTS QuestionsProcessed (question text NOT NULL, code te
create_database_table("Titlemoreweight.db", sql_create_table)
```

```
Tables in the databse:
QuestionsProcessed
```

```
# http://www.sqlitetutorial.net/sqlite-delete/
# https://stackoverflow.com/questions/2279706/select-random-row-from-a-sqlite-table

read_db = 'train_no_dup.db'
write_db = 'Titlemoreweight.db'
train_datasize = 80000
if os.path.isfile(read_db):
    conn_r = create_connection(read_db)
    if conn_r is not None:
        reader =conn_r.cursor()
        # for selecting first 0.5M rows
        reader.execute("SELECT Title, Body, Tags From no_dup_train LIMIT 100001;")
        # for selecting random points
        #reader.execute("SELECT Title, Body, Tags From no_dup_train ORDER BY RANDOM() LIMIT 500001;"
```

```
if os.path.isfile(write_db):
    conn_w = create_connection(write_db)
    if conn_w is not None:
        tables = checkTableExists(conn_w)
        writer =conn_w.cursor()
        if tables != 0:
            writer.execute("DELETE FROM QuestionsProcessed WHERE 1")
            print("Cleared All the rows")
```

```
Tables in the databse:
QuestionsProcessed
Cleared All the rows
```

4.1.2 Preprocessing of questions

```
#http://www.bernzilla.com/2008/05/13/selecting-a-random-row-from-an-sqlite-table/
start = datetime.now()
preprocessed_data_list=[]
reader.fetchone()
questions_with_code=0
len_pre=0
len_post=0
questions_proccesed = 0
for row in reader:

    is_code = 0

    title, question, tags = row[0], row[1], str(row[2])

    if '<code>' in question:
        questions_with_code+=1
        is_code = 1
    x = len(question)+len(title)
    len_pre+=x

    code = str(re.findall(r'<code>(.*?)</code>', question, flags=re.DOTALL))

    question=re.sub('<code>(.*?)</code>', '', question, flags=re.MULTILINE|re.DOTALL)
    question=striphtml(question.encode('utf-8'))

    title=title.encode('utf-8')

    # adding title three time to the data to increase its weight
    # add tags string to the training data

    question=str(title)+" "+str(title)+" "+str(title)+" "+question

#     if questions_proccesed<=train_datasize:
#         question=str(title)+" "+str(title)+" "+str(title)+" "+question+" "+str(tags)
#     else:
#         question=str(title)+" "+str(title)+" "+str(title)+" "+question

    question=re.sub(r'[^A-Za-z0-9#+.\-]+',' ',question)
    words=word_tokenize(str(question.lower()))

    #Removing all single letter and and stopwords from question exceptt for the letter 'c'
    question=' '.join(str(stemmer.stem(j)) for j in words if j not in stop_words and (len(j)!=1 or j

    len_post+=len(question)
    tup = (question,code,tags,x,len(question),is_code)
    questions_proccesed += 1
    writer.execute("insert into QuestionsProcessed(question,code,tags,words_pre,words_post,is_code)
    if (questions_proccesed%100000==0):
        print("number of questions completed=",questions_proccesed)
```

```
no_dup_avg_len_pre=(len_pre*1.0)/questions_proccesed
no_dup_avg_len_post=(len_post*1.0)/questions_proccesed

print( "Avg. length of questions(Title+Body) before processing: %d"%no_dup_avg_len_pre)
print( "Avg. length of questions(Title+Body) after processing: %d"%no_dup_avg_len_post)
print ("Percent of questions containing code: %d"%((questions_with_code*100.0)/questions_proccesed))

print("Time taken to run this cell :", datetime.now() - start)
```

```
number of questions completed= 100000
Avg. length of questions(Title+Body) before processing: 1232
Avg. length of questions(Title+Body) after processing: 441
Percent of questions containing code: 57
Time taken to run this cell : 0:07:47.230432
```

```
# never forget to close the conections or else we will end up with database locks
conn_r.commit()
conn_w.commit()
conn_r.close()
conn_w.close()
```

```
if os.path.isfile(write_db):
    conn_r = create_connection(write_db)
    if conn_r is not None:
        reader =conn_r.cursor()
        reader.execute("SELECT question From QuestionsProcessed LIMIT 10")
        print("Questions after preprocessed")
        print('='*100)
        reader.fetchone()
        for row in reader:
            print(row)
            print('-'*100)
conn_r.commit()
conn_r.close()
```

```
Questions after preprocessed
====================================================================================================
('dynam datagrid bind silverlight dynam datagrid bind silverlight dynam datagrid bind si
----------------------------------------------------------------------------------------------------
('java.lang.noclassdeffounderror javax servlet jsp tagext taglibraryvalid java.lang.nocl
----------------------------------------------------------------------------------------------------
('java.sql.sqlexcept microsoft odbc driver manag invalid descriptor index java.sql.sqlex
----------------------------------------------------------------------------------------------------
('better way updat feed fb php sdk better way updat feed fb php sdk better way updat fee
----------------------------------------------------------------------------------------------------
('btnadd click event open two window record ad btnadd click event open two window record
----------------------------------------------------------------------------------------------------
('sql inject issu prevent correct form submiss php sql inject issu prevent correct form 
----------------------------------------------------------------------------------------------------
('countabl subaddit lebesgu measur countabl subaddit lebesgu measur countabl subaddit le
----------------------------------------------------------------------------------------------------
('hql equival sql queri hql equival sql queri hql equival sql queri hql queri replac nam
----------------------------------------------------------------------------------------------------
('undefin symbol architectur i386 objc class skpsmtpmessag referenc error undefin symbol
----------------------------------------------------------------------------------------------------
```

```
#Taking 0.1 Million entries to a dataframe.
write_db = 'Titlemoreweight.db'
if os.path.isfile(write_db):
    conn_r = create_connection(write_db)
    if conn_r is not None:
        preprocessed_data = pd.read_sql_query("""SELECT question, Tags FROM QuestionsProcessed""", c
conn_r.commit()
conn_r.close()
```

```
preprocessed_data.head(5)
```

	question	tags
0	dynam datagrid bind silverlight dynam datagrid...	c# silverlight data-binding
1	dynam datagrid bind silverlight dynam datagrid...	c# silverlight data-binding columns
2	java.lang.noclassdeffounderror javax servlet j...	jsp jstl
3	java.sql.sqlexcept microsoft odbc driver manag...	java jdbc
4	better way updat feed fb php sdk better way up...	facebook api facebook-php-sdk

```
print("number of data points in sample :", preprocessed_data.shape[0])
print("number of dimensions :", preprocessed_data.shape[1])
```

```
number of data points in sample : 100000
number of dimensions : 2
```

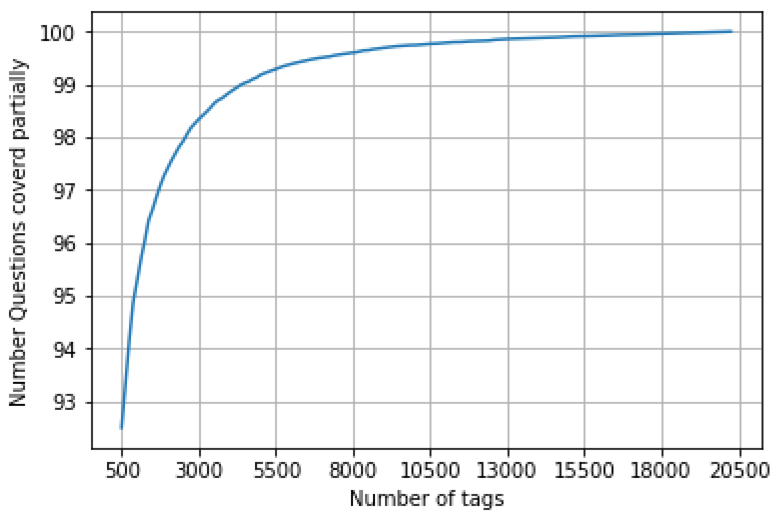
```
vectorizer = CountVectorizer(tokenizer = lambda x: x.split(), binary='true')
multilabel_y = vectorizer.fit_transform(preprocessed_data['tags'])
```

```
def tags_to_choose(n):
    t = multilabel_y.sum(axis=0).tolist()[0]
    sorted_tags_i = sorted(range(len(t)), key=lambda i: t[i], reverse=True)
    multilabel_yn=multilabel_y[:,sorted_tags_i[:n]]
    return multilabel_yn
```

```
def questions_explained_fn(n):
    multilabel_yn = tags_to_choose(n)
    x= multilabel_yn.sum(axis=1)
    return (np.count_nonzero(x==0))
```

```
questions_explained = []
total_tags=multilabel_y.shape[1]
total_qs=preprocessed_data.shape[0]
for i in range(500, total_tags, 100):
    questions_explained.append(np.round(((total_qs-questions_explained_fn(i))/total_qs)*100,3))
```

```
fig, ax = plt.subplots()
ax.plot(questions_explained)
xlabel = list(500+np.array(range(-50,450,50))*50)
ax.set_xticklabels(xlabel)
plt.xlabel("Number of tags")
plt.ylabel("Number Questions coverd partially")
plt.grid()
plt.show()
# you can choose any number of tags based on your computing power, minimun is 500(it covers 90% of t
print("with ",5500,"tags we are covering ",questions_explained[50],"% of questions")
print("with ",500,"tags we are covering ",questions_explained[0],"% of questions")
```




```
with  5500 tags we are covering  99.481 % of questions
with  500 tags we are covering  92.5 % of questions
```

```
# we will be taking 500 tags
multilabel_yx = tags_to_choose(500)
print("number of questions that are not covered :", questions_explained_fn(500),"out of ", total_qs)
```

```
number of questions that are not covered : 7500 out of  100000
```

```
x_train=preprocessed_data.head(train_datasize)
x_test=preprocessed_data.tail(preprocessed_data.shape[0] - 80000)

y_train = multilabel_yx[0:train_datasize,:]
y_test = multilabel_yx[train_datasize:preprocessed_data.shape[0],:]
```

```
print("Number of data points in train data :", y_train.shape)
print("Number of data points in test data :", y_test.shape)
```

```
Number of data points in train data : (80000, 500)
Number of data points in test data : (20000, 500)
```

4.1 3 Featurizing data with Countvectorizer

```
from sklearn.feature_extraction.text import CountVectorizer
start = datetime.now()
vectorizer =CountVectorizer (min_df=0.00009, max_features=200000, \
                             tokenizer = lambda x: x.split(), ngram_range=(1,4))
x_train_multilabel = vectorizer.fit_transform(x_train['question'])
x_test_multilabel = vectorizer.transform(x_test['question'])
print("Time taken to run this cell :", datetime.now() - start)
```

```
Time taken to run this cell : 0:02:03.200459
```

```
print("Dimensions of train data X:",x_train_multilabel.shape, "Y :",y_train.shape)
print("Dimensions of test data X:",x_test_multilabel.shape,"Y:",y_test.shape)
```

```
Dimensions of train data X: (80000, 101734) Y : (80000, 500)
Dimensions of test data X: (20000, 101734) Y: (20000, 500)
```

4.1.4 Applying Logistic Regression with OneVsRest Classifier

```
# HYper parameter tuning
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
start = datetime.now()

param_grid = dict(estimator__C=[0.001,0.01,1,100,1000])

gsv = GridSearchCV(OneVsRestClassifier(LogisticRegression()), param_grid=param_grid, verbose=5, n_jo
gsv.fit(x_train_multilabel, y_train)

print('The best value of hyper parameter is ', gsv.best_params_)
print('The best score is ', gsv.best_score_)
print("Time taken to run this cell :", datetime.now() - start)
```

```
Fitting 3 folds for each of 5 candidates, totalling 15 fits
[Parallel(n_jobs=-1)]: Done  12 out of  15 | elapsed: 411.4min remaining: 102.9min
[Parallel(n_jobs=-1)]: Done  15 out of  15 | elapsed: 478.7min finished
The best value of hyper parameter is  {'estimator__C': 1}
The best score is  0.1735875
Time taken to run this cell : 9:19:29.762581
```

```
c = gsv.best_params_['estimator__C']
print(gsv.best_estimator_)
```

```
OneVsRestClassifier(estimator=LogisticRegression(C=1, class_weight=None, dual=False, fit
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False),
          n_jobs=1)
```

```
classifier = OneVsRestClassifier(LogisticRegression(C=c))
classifier.fit(x_train_multilabel, y_train)
predictions = classifier.predict (x_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test,predictions))


precision = precision_score(y_test, predictions, average='micro')
recall = recall_score(y_test, predictions, average='micro')
f1 = f1_score(y_test, predictions, average='micro')

print("Micro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test, predictions, average='macro')
recall = recall_score(y_test, predictions, average='macro')
f1 = f1_score(y_test, predictions, average='macro')

print("Macro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print (metrics.classification_report(y_test, predictions))
```

```
Accuracy : 0.17965
Hamming loss  0.0032023
Micro-average quality numbers
Precision: 0.6400, Recall: 0.3324, F1-measure: 0.4376
Macro-average quality numbers
Precision: 0.4829, Recall: 0.2487, F1-measure: 0.3104
              precision    recall  f1-score   support

           0       0.74      0.40      0.52       820
           1       0.62      0.29      0.39      1931
           2       0.40      0.15      0.22       544
           3       0.57      0.20      0.29       222
           4       0.72      0.50      0.59      1311
           5       0.80      0.48      0.60      1014
           6       0.70      0.42      0.52      1374
           7       0.76      0.59      0.66       702
           8       0.89      0.62      0.73      1424
           9       0.74      0.57      0.65      1037
          10       0.75      0.56      0.64       797
          11       0.63      0.41      0.50       156
          12       0.64      0.39      0.48        36
          13       0.70      0.41      0.52       610
          14       0.44      0.25      0.32       405
          15       0.62      0.18      0.28       144
          16       0.51      0.25      0.34       425
          17       0.59      0.29      0.39       485
          18       0.80      0.64      0.71       269
          19       0.87      0.55      0.68       518
          20       0.54      0.27      0.36       529
          21       0.80      0.58      0.67       294
          22       0.81      0.40      0.53       520
          23       0.51      0.29      0.37       246
          24       0.62      0.37      0.46       312
          25       0.52      0.31      0.39       314
          26       0.63      0.26      0.37       190
          27       0.27      0.10      0.14       342
          28       0.42      0.20      0.27        96
          29       0.25      0.06      0.10        32
          30       0.66      0.38      0.48       747
          31       0.44      0.29      0.35        14
          32       0.63      0.60      0.62       166
          33       0.57      0.32      0.41       171
          34       0.61      0.29      0.39       256
          35       0.82      0.54      0.65       199
          36       0.15      0.07      0.09        60
          37       0.33      0.20      0.25       203
          38       0.73      0.46      0.56       201
          39       0.47      0.31      0.37       208
          40       0.17      0.08      0.11        13
          41       0.45      0.13      0.20       154
          42       0.39      0.30      0.34        69
          43       0.30      0.13      0.18       426
          44       0.43      0.19      0.27        77
          45       0.52      0.26      0.35       223
          46       0.57      0.32      0.41       144
          47       0.85      0.44      0.58       245
          48       0.63      0.19      0.29        91
```

```
          49       0.59      0.33      0.42       157
          50       0.91      0.63      0.74       132
          51       0.84      0.63      0.72        41
          52       0.59      0.40      0.48       124
          53       0.21      0.22      0.22        96
          54       0.20      0.09      0.13       128
          55       0.62      0.33      0.43        46
          56       0.69      0.61      0.65       151
          57       0.33      0.01      0.02        80
          58       0.48      0.17      0.25        65
          59       0.42      0.13      0.20       182
          60       0.93      0.65      0.76       148
          61       0.38      0.13      0.20       196
          62       0.32      0.16      0.21        58
          63       0.80      0.28      0.41        43
          64       0.62      0.25      0.36       197
          65       0.62      0.34      0.44        82
          66       0.66      0.46      0.54        50
          67       0.67      0.48      0.56       105
          68       0.23      0.06      0.10        98
          69       0.20      0.05      0.08       238
          70       0.40      0.06      0.10        35
          71       0.60      0.44      0.51        54
          72       0.33      0.08      0.13        25
          73       0.41      0.24      0.30        29
          74       0.25      0.07      0.11        29
          75       0.43      0.23      0.30        40
          76       0.81      0.50      0.62       105
          77       0.62      0.54      0.58        28
          78       0.21      0.07      0.10       202
          79       0.58      0.41      0.48        37
          80       0.83      0.33      0.48        15
          81       0.43      0.35      0.38        52
          82       0.41      0.26      0.32        50
          83       0.21      0.05      0.09        56
          84       0.67      0.48      0.56        54
          85       0.50      0.56      0.53        34
          86       0.29      0.17      0.21        30
          87       0.62      0.52      0.57        29
          88       0.82      0.75      0.78        24
          89       0.85      0.80      0.82       117
          90       0.21      0.09      0.13        66
          91       0.54      0.21      0.30        68
          92       0.79      0.28      0.42        67
          93       0.61      0.39      0.48        28
          94       0.44      0.24      0.31        17
          95       0.86      0.49      0.62        51
          96       0.65      0.38      0.48        53
          97       0.25      0.03      0.06        61
          98       0.18      0.03      0.04        79
          99       0.67      0.44      0.53        18
         100       1.00      0.09      0.17        11
         101       0.67      0.49      0.56       207
         102       0.00      0.00      0.00         6
         103       0.00      0.00      0.00        30
         104       0.33      0.07      0.12        54
         105       0.85      0.44      0.58        39
         106       0.31      0.13      0.18        70
```

107	0.67	0.14	0.24	14
108	0.57	0.12	0.20	66
109	0.52	0.28	0.36	50
110	0.73	0.18	0.29	87
111	0.44	0.39	0.42	51
112	0.75	0.01	0.02	291
113	0.95	0.76	0.84	49
114	0.43	0.11	0.17	110
115	0.14	0.04	0.06	28
116	0.00	0.00	0.00	5
117	0.30	0.11	0.16	56
118	0.77	0.40	0.53	125
119	0.90	0.41	0.56	44
120	0.69	0.26	0.38	42
121	0.43	0.18	0.26	55
122	0.72	0.43	0.54	68
123	0.12	0.05	0.07	82
124	0.00	0.00	0.00	0
125	0.83	0.71	0.77	7
126	0.20	0.11	0.14	18
127	0.60	0.10	0.17	31
128	0.86	0.46	0.60	13
129	0.69	0.48	0.56	50
130	0.21	0.05	0.09	91
131	0.74	0.57	0.65	35
132	0.42	0.19	0.26	26
133	0.15	0.06	0.09	32
134	0.61	0.40	0.48	35
135	0.91	0.54	0.68	37
136	0.00	0.00	0.00	55
137	0.33	0.41	0.37	41
138	0.24	0.27	0.25	15
139	0.27	0.10	0.15	99
140	0.94	0.53	0.68	86
141	0.68	0.28	0.40	53
142	0.67	0.06	0.10	36
143	0.55	0.48	0.52	66
144	0.64	0.39	0.49	64
145	0.18	0.08	0.11	25
146	0.20	0.07	0.11	125
147	0.28	0.33	0.30	15
148	0.72	0.44	0.55	48
149	0.42	0.26	0.32	65
150	0.33	0.09	0.14	11
151	0.33	0.27	0.30	15
152	0.30	0.15	0.20	52
153	0.44	0.39	0.41	18
154	0.43	0.19	0.26	16
155	0.80	0.20	0.32	20
156	0.51	0.20	0.29	121
157	0.51	0.31	0.38	107
158	0.50	0.07	0.12	15
159	0.72	0.47	0.57	105
160	0.58	0.32	0.41	69
161	0.62	0.29	0.39	56
162	0.00	0.00	0.00	47
163	0.20	0.03	0.06	121
164	0.52	0.32	0.39	41

164	0.32	0.32	0.39	41
165	1.00	0.00	0.01	229
166	0.78	0.14	0.24	98
167	0.55	0.18	0.27	33
168	0.50	0.16	0.24	44
169	0.64	0.47	0.54	45
170	0.83	0.29	0.43	51
171	0.00	0.00	0.00	18
172	0.56	0.38	0.45	48
173	0.62	0.42	0.50	12
174	0.36	0.15	0.21	62
175	0.79	0.50	0.61	44
176	0.95	0.70	0.81	30
177	0.57	0.43	0.49	30
178	0.00	0.00	0.00	0
179	1.00	1.00	1.00	1
180	0.62	0.33	0.43	40
181	0.31	0.09	0.14	44
182	0.00	0.00	0.00	2
183	0.60	0.36	0.45	75
184	0.50	0.50	0.50	4
185	0.67	0.22	0.33	64
186	0.40	0.33	0.36	12
187	0.97	0.55	0.70	55
188	0.82	0.58	0.68	64
189	0.43	0.16	0.23	96
190	0.00	0.00	0.00	22
191	0.90	0.24	0.38	76
192	0.68	0.42	0.52	45
193	0.80	0.29	0.42	14
194	0.66	0.38	0.48	50
195	1.00	0.25	0.40	20
196	0.83	0.57	0.68	35
197	0.65	0.23	0.34	94
198	0.00	0.00	0.00	14
199	0.12	0.04	0.06	25
200	0.62	0.09	0.16	54
201	0.50	0.05	0.08	22
202	0.31	0.09	0.14	43
203	1.00	0.02	0.05	43
204	0.97	0.52	0.67	62
205	0.00	0.00	0.00	3
206	0.31	0.09	0.14	43
207	0.50	0.14	0.22	7
208	0.25	0.12	0.17	8
209	0.20	0.02	0.04	42
210	0.36	0.40	0.38	10
211	0.37	0.17	0.24	40
212	0.80	0.35	0.48	23
213	0.00	0.00	0.00	6
214	0.72	0.45	0.55	47
215	0.50	0.08	0.14	62
216	0.69	0.32	0.44	77
217	0.22	0.09	0.13	22
218	0.33	0.33	0.33	3
219	0.00	0.00	0.00	28
220	0.71	0.06	0.11	81
221	0.27	0.10	0.14	31

222	0.50	0.03	0.06	34
223	1.00	0.33	0.50	60
224	0.33	0.20	0.25	10
225	0.86	0.60	0.71	10
226	0.73	0.68	0.71	92
227	0.78	0.54	0.64	13
228	0.50	0.15	0.24	13
229	0.86	0.74	0.80	43
230	0.36	0.11	0.17	35
231	0.00	0.00	0.00	4
232	0.38	0.15	0.21	20
233	0.41	0.16	0.23	145
234	0.82	0.49	0.61	55
235	0.00	0.00	0.00	2
236	0.38	0.08	0.13	37
237	0.70	0.36	0.47	90
238	0.33	0.03	0.06	58
239	0.50	0.25	0.33	20
240	0.97	0.46	0.62	61
241	0.86	0.57	0.69	42
242	0.59	0.67	0.62	30
243	0.79	0.50	0.61	66
244	0.57	0.19	0.29	42
245	0.09	0.03	0.05	31
246	0.75	0.50	0.60	6
247	0.22	0.11	0.15	18
248	0.80	0.47	0.59	51
249	0.70	0.41	0.52	17
250	0.50	0.36	0.42	22
251	0.74	0.33	0.45	52
252	0.67	0.14	0.23	29
253	0.08	0.04	0.05	28
254	0.50	0.10	0.17	10
255	0.20	0.20	0.20	5
256	0.25	0.33	0.29	3
257	0.67	0.34	0.45	41
258	0.33	0.10	0.15	30
259	0.50	0.33	0.40	3
260	0.00	0.00	0.00	38
261	0.00	0.00	0.00	1
262	0.89	0.42	0.57	19
263	0.00	0.00	0.00	14
264	0.25	0.11	0.15	37
265	0.11	0.11	0.11	9
266	0.26	0.24	0.25	45
267	0.57	0.52	0.54	33
268	0.77	0.62	0.69	16
269	0.56	0.40	0.47	35
270	0.38	0.27	0.32	11
271	0.00	0.00	0.00	30
272	0.25	0.25	0.25	8
273	0.09	0.05	0.06	21
274	0.51	0.15	0.23	123
275	0.47	0.24	0.32	67
276	0.84	0.80	0.82	20
277	0.00	0.00	0.00	14
278	0.60	0.16	0.25	19
279	0.83	0.42	0.56	12

280	0.00	0.00	0.00	15
281	0.91	0.59	0.71	17
282	1.00	0.63	0.78	41
283	0.71	0.33	0.45	15
284	0.60	0.24	0.35	74
285	0.57	0.11	0.18	38
286	0.25	0.12	0.17	16
287	0.33	0.07	0.11	30
288	0.94	0.54	0.68	28
289	0.00	0.00	0.00	21
290	0.85	0.54	0.66	41
291	0.29	0.17	0.21	12
292	0.57	0.17	0.26	24
293	0.44	0.35	0.39	20
294	0.18	0.09	0.12	23
295	0.20	0.03	0.06	29
296	0.25	0.07	0.11	28
297	0.31	0.10	0.15	42
298	0.17	0.02	0.03	53
299	0.25	0.03	0.05	36
300	0.38	0.12	0.19	41
301	0.59	0.43	0.50	37
302	0.85	0.42	0.56	26
303	0.29	0.18	0.22	11
304	0.31	0.13	0.18	31
305	0.45	0.29	0.36	17
306	0.67	0.22	0.33	9
307	0.40	0.33	0.36	6
308	0.00	0.00	0.00	34
309	0.68	0.30	0.42	43
310	0.00	0.00	0.00	30
311	0.29	0.12	0.17	50
312	0.00	0.00	0.00	24
313	0.50	0.05	0.09	42
314	0.43	0.14	0.21	22
315	0.33	0.02	0.03	58
316	0.00	0.00	0.00	10
317	0.34	0.19	0.25	57
318	0.60	0.30	0.40	10
319	0.00	0.00	0.00	11
320	0.50	0.18	0.27	11
321	0.67	0.50	0.57	8
322	0.89	0.36	0.52	22
323	0.94	0.57	0.71	28
324	0.69	0.50	0.58	50
325	0.75	0.17	0.27	18
326	0.11	0.03	0.05	33
327	0.15	0.12	0.13	17
328	0.75	0.10	0.18	29
329	0.50	0.29	0.36	7
330	0.56	0.50	0.53	10
331	0.23	0.12	0.16	25
332	1.00	1.00	1.00	2
333	0.57	0.36	0.44	11
334	0.00	0.00	0.00	24
335	1.00	0.20	0.33	5
336	0.67	0.06	0.11	33
337	0.60	0.20	0.30	30

337	0.60	0.20	0.30	30
338	0.96	0.52	0.68	42
339	0.25	0.04	0.07	26
340	0.48	0.31	0.37	36
341	1.00	0.46	0.63	13
342	0.60	0.55	0.57	11
343	0.80	0.40	0.53	10
344	0.33	0.10	0.15	21
345	0.00	0.00	0.00	0
346	0.00	0.00	0.00	6
347	0.33	0.08	0.13	12
348	0.25	0.08	0.12	13
349	0.57	0.17	0.26	24
350	0.69	0.33	0.45	27
351	0.41	0.16	0.23	43
352	0.00	0.00	0.00	30
353	0.50	0.23	0.31	22
354	0.40	0.06	0.11	31
355	0.54	0.70	0.61	10
356	0.33	0.05	0.09	20
357	0.71	0.60	0.65	20
358	0.47	0.29	0.36	28
359	0.50	0.33	0.40	21
360	0.33	0.08	0.13	25
361	0.55	0.49	0.52	35
362	0.87	0.56	0.68	36
363	0.57	0.24	0.33	17
364	1.00	0.31	0.47	13
365	0.50	0.05	0.09	21
366	0.00	0.00	0.00	18
367	0.33	0.02	0.04	97
368	0.67	0.48	0.56	29
369	1.00	0.50	0.67	12
370	0.50	0.15	0.24	13
371	0.22	0.11	0.15	18
372	0.00	0.00	0.00	6
373	0.50	0.33	0.40	6
374	0.67	0.20	0.31	30
375	0.20	0.19	0.19	27
376	0.50	0.04	0.07	28
377	0.00	0.00	0.00	2
378	0.29	0.50	0.36	4
379	0.00	0.00	0.00	19
380	0.29	0.40	0.33	5
381	1.00	0.39	0.56	18
382	0.33	0.14	0.19	22
383	0.00	0.00	0.00	16
384	0.83	0.38	0.53	13
385	0.20	0.06	0.09	18
386	0.90	0.82	0.86	11
387	0.47	0.40	0.43	88
388	0.00	0.00	0.00	13
389	0.00	0.00	0.00	6
390	0.00	0.00	0.00	6
391	1.00	0.43	0.60	51
392	0.00	0.00	0.00	13
393	0.56	0.24	0.34	37
394	0.00	0.00	0.00	6

395	0.33	0.11	0.17	9
396	0.00	0.00	0.00	13
397	1.00	0.50	0.67	6
398	0.52	0.38	0.44	29
399	0.95	0.64	0.76	33
400	0.50	0.03	0.06	31
401	0.50	0.06	0.11	50
402	0.90	0.50	0.64	18
403	0.50	0.14	0.22	7
404	0.65	0.50	0.57	26
405	0.84	0.64	0.73	56
406	0.75	0.75	0.75	4
407	0.20	0.12	0.15	17
408	0.50	0.09	0.15	11
409	0.50	0.06	0.10	18
410	0.50	0.20	0.29	10
411	0.50	0.13	0.21	45
412	0.71	0.25	0.37	20
413	0.67	0.08	0.14	25
414	0.50	0.15	0.23	20
415	0.00	0.00	0.00	6
416	0.29	0.08	0.12	26
417	0.67	0.20	0.31	10
418	0.00	0.00	0.00	18
419	0.60	0.50	0.55	6
420	0.62	0.47	0.53	17
421	0.00	0.00	0.00	1
422	0.00	0.00	0.00	6
423	0.00	0.00	0.00	12
424	1.00	0.25	0.40	4
425	1.00	0.36	0.53	11
426	0.33	0.09	0.14	11
427	0.75	0.75	0.75	8
428	0.71	0.19	0.30	26
429	0.55	0.40	0.46	40
430	0.00	0.00	0.00	2
431	0.00	0.00	0.00	35
432	0.50	0.07	0.12	15
433	0.00	0.00	0.00	18
434	0.00	0.00	0.00	0
435	0.00	0.00	0.00	0
436	0.33	0.07	0.12	28
437	0.36	0.12	0.18	33
438	0.82	0.45	0.58	20
439	0.50	0.06	0.10	36
440	1.00	0.11	0.20	18
441	0.50	0.44	0.47	18
442	0.64	0.56	0.60	16
443	0.00	0.00	0.00	22
444	0.00	0.00	0.00	6
445	0.88	0.33	0.48	21
446	0.81	0.37	0.51	46
447	0.29	0.06	0.10	69
448	0.00	0.00	0.00	7
449	0.00	0.00	0.00	3
450	0.22	0.04	0.07	52
451	0.00	0.00	0.00	16
452	1.00	0.76	0.87	17

```
        453       0.00      0.00      0.00        13
        454       0.67      0.18      0.29        11
        455       0.00      0.00      0.00        12
        456       0.40      0.33      0.36         6
        457       0.33      0.11      0.17        18
        458       0.20      0.07      0.10        15
        459       0.90      0.32      0.47        28
        460       0.00      0.00      0.00        18
        461       0.50      0.50      0.50        10
        462       0.50      0.17      0.25        24
        463       0.67      0.11      0.19        18
        464       0.93      0.36      0.52        39
        465       0.20      0.09      0.13        11
        466       0.18      0.06      0.09        35
        467       0.25      0.10      0.14        21
        468       0.00      0.00      0.00        37
        469       1.00      0.20      0.33         5
        470       0.33      0.12      0.18         8
        471       0.78      0.19      0.30        37
        472       0.00      0.00      0.00        47
        473       0.44      0.29      0.35        14
        474       1.00      0.52      0.69        23
        475       0.58      0.38      0.46        66
        476       0.00      0.00      0.00         3
        477       0.50      0.16      0.24        19
        478       0.00      0.00      0.00         1
        479       0.22      0.09      0.12        23
        480       0.50      0.05      0.09        60
        481       0.40      0.08      0.13        26
        482       0.67      0.50      0.57         4
        483       0.00      0.00      0.00         8
        484       0.89      0.35      0.50        23
        485       0.71      0.28      0.40        18
        486       0.60      0.25      0.35        12
        487       0.86      0.21      0.33        29
        488       1.00      1.00      1.00         1
        489       1.00      0.17      0.29         6
        490       0.33      0.14      0.20         7
        491       0.00      0.00      0.00         3
        492       0.30      0.30      0.30        10
        493       0.53      0.42      0.47        19
        494       0.33      0.14      0.20         7
        495       0.67      0.25      0.36         8
        496       0.22      0.11      0.15        18
        497       0.50      0.01      0.03        72
        498       0.00      0.00      0.00         8
        499       0.40      0.12      0.19        32

avg / total       0.60      0.33      0.41     37472

C:\Users\BALARAMI REDDY\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:11
  'precision', 'predicted', average, warn_for)
C:\Users\BALARAMI REDDY\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:11
  'recall', 'true', average, warn_for)
C:\Users\BALARAMI REDDY\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:11
  'precision', 'predicted', average, warn_for)
C:\Users\BALARAMI REDDY\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:11
```

```
  'recall', 'true', average, warn_for)
C:\Users\BALARAMI REDDY\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:11
  'precision', 'predicted', average, warn_for)
C:\Users\BALARAMI REDDY\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:11
  'recall', 'true', average, warn_for)
```

```
import pandas as pd
results=pd.DataFrame(columns=["Classification model", "Hyperparameter", "Regularization", "F1 micro"
new = ['Logistic Regression',1,'L1',0.436,0.3104]
results.loc[0] = new
```

4.1.5 Support Vector Classification(SGD Classifier with hinge loss)

```
start = datetime.now()
classifier = OneVsRestClassifier(SGDClassifier(loss='hinge', alpha=0.00001, penalty='l1'), n_jobs=-1
classifier.fit(x_train_multilabel, y_train)
predictions = classifier.predict (x_test_multilabel)


print("Accuracy :",metrics.accuracy_score(y_test, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test,predictions))


precision = precision_score(y_test, predictions, average='micro')
recall = recall_score(y_test, predictions, average='micro')
f1 = f1_score(y_test, predictions, average='micro')

print("Micro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test, predictions, average='macro')
recall = recall_score(y_test, predictions, average='macro')
f1 = f1_score(y_test, predictions, average='macro')

print("Macro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print (metrics.classification_report(y_test, predictions))
print("Time taken to run this cell :", datetime.now() - start)
```

```
Accuracy : 0.08925
Hamming loss  0.0069582
Micro-average quality numbers
Precision: 0.2529, Recall: 0.4385, F1-measure: 0.3208
Macro-average quality numbers
Precision: 0.1738, Recall: 0.3489, F1-measure: 0.2174
              precision    recall  f1-score   support

           0       0.48      0.49      0.48       820
           1       0.40      0.41      0.41      1931
           2       0.17      0.26      0.20       544
           3       0.22      0.29      0.25       222
           4       0.49      0.61      0.54      1311
           5       0.45      0.60      0.51      1014
           6       0.51      0.51      0.51      1374
           7       0.52      0.65      0.58       702
           8       0.75      0.71      0.73      1424
           9       0.68      0.67      0.67      1037
          10       0.53      0.65      0.58       797
          11       0.17      0.47      0.25       156
          12       0.13      0.50      0.20        36
          13       0.53      0.51      0.52       610
          14       0.27      0.36      0.31       405
          15       0.15      0.37      0.21       144
          16       0.32      0.32      0.32       425
          17       0.39      0.31      0.35       485
          18       0.52      0.71      0.60       269
          19       0.52      0.67      0.59       518
          20       0.31      0.62      0.41       529
          21       0.63      0.68      0.65       294
          22       0.54      0.53      0.54       520
          23       0.25      0.33      0.28       246
          24       0.26      0.47      0.34       312
          25       0.26      0.40      0.31       314
          26       0.21      0.38      0.27       190
          27       0.14      0.13      0.14       342
          28       0.14      0.40      0.20        96
          29       0.09      0.31      0.14        32
          30       0.33      0.47      0.39       747
          31       0.06      0.50      0.11        14
          32       0.31      0.62      0.41       166
          33       0.34      0.42      0.37       171
          34       0.32      0.36      0.34       256
          35       0.53      0.57      0.55       199
          36       0.07      0.13      0.09        60
          37       0.14      0.37      0.20       203
          38       0.47      0.49      0.48       201
          39       0.26      0.34      0.30       208
          40       0.01      0.15      0.02        13
          41       0.13      0.26      0.18       154
          42       0.27      0.43      0.34        69
          43       0.28      0.55      0.37       426
          44       0.15      0.39      0.22        77
          45       0.30      0.43      0.35       223
          46       0.18      0.38      0.24       144
          47       0.40      0.63      0.49       245
          48       0.13      0.19      0.15        91
```

49	0.21	0.42	0.28	157
50	0.44	0.74	0.55	132
51	0.43	0.83	0.57	41
52	0.27	0.47	0.34	124
53	0.11	0.22	0.14	96
54	0.05	0.17	0.08	128
55	0.15	0.48	0.23	46
56	0.49	0.63	0.55	151
57	0.06	0.15	0.09	80
58	0.09	0.28	0.14	65
59	0.25	0.23	0.24	182
60	0.55	0.78	0.65	148
61	0.21	0.23	0.22	196
62	0.07	0.19	0.10	58
63	0.10	0.35	0.15	43
64	0.32	0.40	0.36	197
65	0.27	0.37	0.31	82
66	0.40	0.70	0.51	50
67	0.35	0.56	0.43	105
68	0.09	0.10	0.09	98
69	0.18	0.16	0.17	238
70	0.04	0.20	0.07	35
71	0.25	0.46	0.32	54
72	0.13	0.36	0.19	25
73	0.23	0.38	0.29	29
74	0.03	0.14	0.04	29
75	0.08	0.28	0.12	40
76	0.48	0.56	0.52	105
77	0.33	0.57	0.42	28
78	0.11	0.23	0.15	202
79	0.26	0.57	0.35	37
80	0.12	0.40	0.19	15
81	0.14	0.46	0.21	52
82	0.11	0.30	0.16	50
83	0.03	0.14	0.05	56
84	0.23	0.50	0.32	54
85	0.29	0.71	0.41	34
86	0.09	0.30	0.14	30
87	0.31	0.62	0.41	29
88	0.23	0.79	0.36	24
89	0.46	0.85	0.60	117
90	0.06	0.20	0.09	66
91	0.19	0.31	0.23	68
92	0.16	0.36	0.22	67
93	0.07	0.43	0.12	28
94	0.11	0.29	0.16	17
95	0.21	0.57	0.31	51
96	0.26	0.49	0.34	53
97	0.07	0.13	0.09	61
98	0.04	0.10	0.05	79
99	0.22	0.50	0.31	18
100	0.02	0.09	0.03	11
101	0.34	0.57	0.42	207
102	0.02	0.17	0.04	6
103	0.02	0.07	0.03	30
104	0.12	0.13	0.12	54
105	0.41	0.51	0.45	39
106	0.11	0.29	0.16	70

107	0.02	0.14	0.03	14
108	0.14	0.29	0.19	66
109	0.12	0.38	0.18	50
110	0.14	0.23	0.18	87
111	0.17	0.37	0.24	51
112	0.69	0.22	0.33	291
113	0.57	0.76	0.65	49
114	0.22	0.16	0.19	110
115	0.04	0.11	0.06	28
116	0.00	0.00	0.00	5
117	0.07	0.12	0.09	56
118	0.41	0.51	0.45	125
119	0.50	0.64	0.56	44
120	0.41	0.45	0.43	42
121	0.10	0.16	0.13	55
122	0.31	0.59	0.41	68
123	0.06	0.20	0.09	82
124	0.00	0.00	0.00	0
125	0.16	0.57	0.25	7
126	0.03	0.17	0.04	18
127	0.17	0.35	0.23	31
128	0.17	0.31	0.22	13
129	0.30	0.54	0.39	50
130	0.07	0.15	0.09	91
131	0.47	0.69	0.56	35
132	0.04	0.19	0.07	26
133	0.08	0.12	0.10	32
134	0.22	0.26	0.24	35
135	0.44	0.59	0.51	37
136	0.01	0.02	0.01	55
137	0.10	0.51	0.17	41
138	0.15	0.33	0.20	15
139	0.08	0.21	0.11	99
140	0.29	0.65	0.40	86
141	0.27	0.42	0.33	53
142	0.12	0.31	0.17	36
143	0.36	0.56	0.44	66
144	0.41	0.42	0.42	64
145	0.04	0.12	0.06	25
146	0.10	0.15	0.12	125
147	0.11	0.47	0.17	15
148	0.39	0.54	0.46	48
149	0.13	0.32	0.19	65
150	0.03	0.27	0.05	11
151	0.11	0.40	0.17	15
152	0.08	0.27	0.13	52
153	0.25	0.50	0.33	18
154	0.15	0.25	0.19	16
155	0.19	0.35	0.25	20
156	0.20	0.34	0.25	121
157	0.26	0.38	0.31	107
158	0.01	0.13	0.02	15
159	0.37	0.66	0.47	105
160	0.23	0.35	0.28	69
161	0.17	0.39	0.24	56
162	0.04	0.09	0.05	47
163	0.09	0.16	0.11	121
164	0.14	0.44	0.22	41

164	0.14	0.44	0.22	41
165	0.25	0.03	0.05	229
166	0.24	0.27	0.25	98
167	0.15	0.33	0.21	33
168	0.16	0.32	0.21	44
169	0.26	0.47	0.33	45
170	0.57	0.47	0.52	51
171	0.01	0.06	0.02	18
172	0.31	0.60	0.41	48
173	0.09	0.42	0.14	12
174	0.12	0.29	0.17	62
175	0.47	0.77	0.58	44
176	0.42	0.73	0.53	30
177	0.20	0.47	0.28	30
178	0.00	0.00	0.00	0
179	0.06	1.00	0.11	1
180	0.07	0.42	0.12	40
181	0.05	0.11	0.07	44
182	0.03	0.50	0.06	2
183	0.24	0.51	0.33	75
184	0.02	0.25	0.04	4
185	0.22	0.20	0.21	64
186	0.08	0.58	0.14	12
187	0.53	0.67	0.59	55
188	0.31	0.66	0.42	64
189	0.14	0.25	0.18	96
190	0.05	0.27	0.08	22
191	0.30	0.39	0.34	76
192	0.24	0.49	0.32	45
193	0.04	0.07	0.05	14
194	0.23	0.52	0.32	50
195	0.09	0.45	0.15	20
196	0.52	0.63	0.57	35
197	0.26	0.37	0.31	94
198	0.06	0.36	0.10	14
199	0.02	0.04	0.03	25
200	0.13	0.20	0.16	54
201	0.05	0.23	0.09	22
202	0.07	0.21	0.10	43
203	0.03	0.07	0.04	43
204	0.45	0.79	0.57	62
205	0.00	0.00	0.00	3
206	0.06	0.23	0.09	43
207	0.03	0.29	0.05	7
208	0.03	0.12	0.04	8
209	0.15	0.19	0.17	42
210	0.16	0.50	0.24	10
211	0.11	0.30	0.16	40
212	0.13	0.48	0.20	23
213	0.00	0.00	0.00	6
214	0.24	0.49	0.32	47
215	0.12	0.11	0.11	62
216	0.27	0.35	0.30	77
217	0.03	0.09	0.05	22
218	0.07	0.33	0.12	3
219	0.04	0.07	0.05	28
220	0.14	0.22	0.17	81
221	0.11	0.32	0.16	31

222	0.04	0.21	0.07	34
223	0.37	0.47	0.41	60
224	0.06	0.50	0.11	10
225	0.26	0.50	0.34	10
226	0.65	0.77	0.70	92
227	0.22	0.46	0.30	13
228	0.04	0.23	0.07	13
229	0.53	0.74	0.62	43
230	0.14	0.17	0.16	35
231	0.00	0.00	0.00	4
232	0.12	0.20	0.15	20
233	0.13	0.30	0.18	145
234	0.49	0.49	0.49	55
235	0.00	0.00	0.00	2
236	0.15	0.22	0.17	37
237	0.51	0.51	0.51	90
238	0.14	0.14	0.14	58
239	0.10	0.35	0.15	20
240	0.41	0.57	0.48	61
241	0.45	0.79	0.57	42
242	0.19	0.73	0.31	30
243	0.39	0.53	0.45	66
244	0.22	0.31	0.25	42
245	0.06	0.13	0.08	31
246	0.14	0.33	0.20	6
247	0.03	0.17	0.06	18
248	0.42	0.65	0.51	51
249	0.10	0.41	0.16	17
250	0.22	0.50	0.30	22
251	0.36	0.52	0.42	52
252	0.02	0.03	0.03	29
253	0.05	0.11	0.06	28
254	0.04	0.10	0.05	10
255	0.01	0.20	0.03	5
256	0.09	0.67	0.15	3
257	0.24	0.34	0.28	41
258	0.09	0.23	0.13	30
259	0.08	0.67	0.14	3
260	0.02	0.03	0.02	38
261	0.00	0.00	0.00	1
262	0.17	0.32	0.22	19
263	0.02	0.07	0.03	14
264	0.03	0.16	0.05	37
265	0.04	0.33	0.07	9
266	0.06	0.16	0.09	45
267	0.33	0.61	0.43	33
268	0.27	0.75	0.39	16
269	0.21	0.57	0.31	35
270	0.07	0.36	0.11	11
271	0.01	0.03	0.01	30
272	0.06	0.12	0.08	8
273	0.13	0.29	0.18	21
274	0.07	0.21	0.11	123
275	0.10	0.48	0.16	67
276	0.33	0.90	0.49	20
277	0.00	0.00	0.00	14
278	0.05	0.11	0.07	19
279	0.21	0.50	0.30	12

280	0.00	0.00	0.00	15
281	0.42	0.65	0.51	17
282	0.56	0.78	0.65	41
283	0.19	0.40	0.26	15
284	0.27	0.35	0.30	74
285	0.15	0.34	0.21	38
286	0.04	0.19	0.07	16
287	0.03	0.13	0.05	30
288	0.35	0.64	0.45	28
289	0.04	0.05	0.04	21
290	0.43	0.63	0.51	41
291	0.03	0.42	0.06	12
292	0.08	0.25	0.13	24
293	0.13	0.60	0.21	20
294	0.06	0.26	0.09	23
295	0.05	0.21	0.08	29
296	0.05	0.18	0.08	28
297	0.10	0.17	0.12	42
298	0.05	0.13	0.08	53
299	0.09	0.11	0.10	36
300	0.17	0.29	0.21	41
301	0.18	0.59	0.27	37
302	0.32	0.54	0.40	26
303	0.11	0.36	0.17	11
304	0.07	0.23	0.11	31
305	0.08	0.29	0.13	17
306	0.05	0.22	0.08	9
307	0.05	0.33	0.08	6
308	0.00	0.00	0.00	34
309	0.35	0.44	0.39	43
310	0.02	0.13	0.03	30
311	0.16	0.34	0.22	50
312	0.02	0.08	0.03	24
313	0.07	0.12	0.09	42
314	0.13	0.36	0.19	22
315	0.07	0.05	0.06	58
316	0.06	0.30	0.11	10
317	0.15	0.37	0.21	57
318	0.13	0.50	0.20	10
319	0.01	0.18	0.03	11
320	0.07	0.27	0.11	11
321	0.24	0.62	0.34	8
322	0.19	0.45	0.27	22
323	0.17	0.64	0.27	28
324	0.33	0.42	0.37	50
325	0.05	0.22	0.08	18
326	0.07	0.18	0.10	33
327	0.03	0.18	0.05	17
328	0.04	0.24	0.07	29
329	0.22	0.57	0.32	7
330	0.14	0.40	0.21	10
331	0.06	0.32	0.11	25
332	0.18	1.00	0.31	2
333	0.14	0.45	0.22	11
334	0.00	0.00	0.00	24
335	0.03	0.20	0.05	5
336	0.00	0.00	0.00	33
337	0.14	0.37	0.21	30

337	0.14	0.37	0.21	30
338	0.34	0.64	0.45	42
339	0.07	0.12	0.09	26
340	0.30	0.50	0.37	36
341	0.29	0.69	0.41	13
342	0.14	0.36	0.20	11
343	0.27	0.70	0.39	10
344	0.14	0.29	0.18	21
345	0.00	0.00	0.00	0
346	0.01	0.17	0.01	6
347	0.01	0.17	0.01	12
348	0.04	0.15	0.07	13
349	0.12	0.29	0.17	24
350	0.28	0.56	0.38	27
351	0.14	0.26	0.18	43
352	0.00	0.00	0.00	30
353	0.19	0.36	0.25	22
354	0.07	0.39	0.12	31
355	0.06	0.60	0.11	10
356	0.08	0.20	0.11	20
357	0.38	0.75	0.50	20
358	0.15	0.39	0.21	28
359	0.22	0.43	0.29	21
360	0.08	0.24	0.12	25
361	0.22	0.40	0.28	35
362	0.55	0.67	0.60	36
363	0.05	0.35	0.09	17
364	0.11	0.38	0.17	13
365	0.05	0.10	0.07	21
366	0.21	0.39	0.27	18
367	0.23	0.11	0.15	97
368	0.21	0.45	0.29	29
369	0.24	0.75	0.36	12
370	0.14	0.38	0.21	13
371	0.09	0.17	0.12	18
372	0.02	0.17	0.04	6
373	0.08	0.33	0.12	6
374	0.22	0.40	0.28	30
375	0.14	0.22	0.17	27
376	0.07	0.18	0.10	28
377	0.00	0.00	0.00	2
378	0.12	0.50	0.19	4
379	0.03	0.05	0.04	19
380	0.17	0.60	0.26	5
381	0.20	0.56	0.30	18
382	0.21	0.50	0.30	22
383	0.02	0.06	0.03	16
384	0.23	0.54	0.33	13
385	0.08	0.22	0.11	18
386	0.34	0.91	0.50	11
387	0.29	0.42	0.35	88
388	0.02	0.08	0.03	13
389	0.00	0.00	0.00	6
390	0.00	0.00	0.00	6
391	0.38	0.69	0.49	51
392	0.03	0.08	0.04	13
393	0.34	0.51	0.41	37
394	0.00	0.00	0.00	6

```
         395       0.00      0.00      0.00         9
         396       0.02      0.08      0.03        13
         397       0.09      0.50      0.15         6
         398       0.10      0.55      0.17        29
         399       0.51      0.76      0.61        33
         400       0.04      0.10      0.05        31
         401       0.12      0.14      0.13        50
         402       0.30      0.72      0.42        18
         403       0.02      0.14      0.03         7
         404       0.16      0.54      0.25        26
         405       0.66      0.79      0.72        56
         406       0.08      0.50      0.14         4
         407       0.09      0.24      0.13        17
         408       0.22      0.36      0.28        11
         409       0.02      0.17      0.04        18
         410       0.11      0.50      0.18        10
         411       0.14      0.20      0.16        45
         412       0.29      0.35      0.32        20
         413       0.13      0.44      0.20        25
         414       0.07      0.35      0.12        20
         415       0.00      0.00      0.00         6
         416       0.05      0.12      0.07        26
         417       0.18      0.30      0.22        10
         418       0.04      0.11      0.05        18
         419       0.27      0.50      0.35         6
         420       0.20      0.53      0.29        17
         421       0.00      0.00      0.00         1
         422       0.00      0.00      0.00         6
         423       0.00      0.00      0.00        12
         424       0.03      0.25      0.06         4
         425       0.15      0.36      0.21        11
         426       0.00      0.00      0.00        11
         427       0.14      0.62      0.22         8
         428       0.20      0.31      0.24        26
         429       0.31      0.47      0.37        40
         430       0.00      0.00      0.00         2
         431       0.01      0.03      0.01        35
         432       0.10      0.20      0.13        15
         433       0.00      0.00      0.00        18
         434       0.00      0.00      0.00         0
         435       0.00      0.00      0.00         0
         436       0.05      0.07      0.06        28
         437       0.12      0.39      0.18        33
         438       0.41      0.45      0.43        20
         439       0.03      0.14      0.04        36
         440       0.08      0.17      0.11        18
         441       0.25      0.50      0.33        18
         442       0.27      0.56      0.37        16
         443       0.06      0.09      0.07        22
         444       0.00      0.00      0.00         6
         445       0.26      0.43      0.32        21
         446       0.43      0.48      0.45        46
         447       0.10      0.16      0.12        69
         448       0.00      0.00      0.00         7
         449       0.02      0.33      0.04         3
         450       0.09      0.17      0.12        52
         451       0.01      0.06      0.01        16
         452       0.44      0.88      0.59        17
```

```
        453       0.04      0.23      0.06        13
        454       0.19      0.45      0.26        11
        455       0.01      0.08      0.02        12
        456       0.14      0.83      0.23         6
        457       0.02      0.06      0.03        18
        458       0.01      0.07      0.02        15
        459       0.42      0.54      0.47        28
        460       0.05      0.06      0.05        18
        461       0.08      0.40      0.14        10
        462       0.14      0.17      0.15        24
        463       0.11      0.39      0.18        18
        464       0.57      0.54      0.55        39
        465       0.13      0.36      0.20        11
        466       0.05      0.11      0.07        35
        467       0.10      0.19      0.13        21
        468       0.19      0.22      0.20        37
        469       0.11      0.40      0.17         5
        470       0.05      0.25      0.08         8
        471       0.50      0.35      0.41        37
        472       0.06      0.17      0.09        47
        473       0.10      0.43      0.17        14
        474       0.25      0.65      0.36        23
        475       0.59      0.65      0.62        66
        476       0.00      0.00      0.00         3
        477       0.18      0.21      0.20        19
        478       0.04      1.00      0.07         1
        479       0.05      0.13      0.07        23
        480       0.04      0.15      0.06        60
        481       0.06      0.23      0.10        26
        482       0.08      0.75      0.14         4
        483       0.25      0.38      0.30         8
        484       0.19      0.30      0.23        23
        485       0.18      0.39      0.25        18
        486       0.16      0.58      0.25        12
        487       0.23      0.31      0.26        29
        488       0.02      1.00      0.04         1
        489       0.15      0.67      0.24         6
        490       0.05      0.43      0.09         7
        491       0.00      0.00      0.00         3
        492       0.06      0.30      0.09        10
        493       0.10      0.26      0.14        19
        494       0.03      0.14      0.05         7
        495       0.26      0.62      0.37         8
        496       0.14      0.39      0.21        18
        497       0.02      0.07      0.03        72
        498       0.04      0.25      0.07         8
        499       0.21      0.44      0.29        32

avg / total       0.33      0.44      0.37     37472

Time taken to run this cell : 0:04:33.099928
C:\Users\BALARAMI REDDY\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:11
  'recall', 'true', average, warn_for)
C:\Users\BALARAMI REDDY\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:11
  'recall', 'true', average, warn_for)
C:\Users\BALARAMI REDDY\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:11
  'recall', 'true', average, warn_for)
```

```
new = ['SVM SGDClassifier',0.00001,'L1',0.3208,0.2174]
results.loc[1] = new
```

Conclusions

```
results
```

	Classification model	Hyperparameter	Regularization	F1 micro	F1 macro
0	Logistic Regression	1	L1	0.4360	0.3104
1	SVM SGDClassifier	1e-05	L1	0.3208	0.2174

1. We performed Exploratory data analysis on the data set at first
2. We removed.the duplicates, and other data cleaning and preprocessing steps like stemming were performed.
3. The OneVsRestClassifier was trained for the data with logistic regression, some hyperparameter tuning, l1 reg
4. Again OneVsRestClassifier was trained for the data with Support Vector Classification(SGD Classifier with hin results F1 macro = 0.2174