Stack Overflow: Tag Prediction

Business Problem

Description

Description

Stack Overflow is the largest, most trusted online community for developers to learn, share their programming knowledge, and build their careers.

Stack Overflow is something which every programmer use one way or another. Each month, over 50 million developers come to Stack Overflow to learn, share their knowledge, and build their careers. It features questions and answers on a wide range of topics in computer programming. The website serves as a platform for users to ask and answer questions, and, through membership and active participation, to vote questions and answers up or down and edit questions and answers in a fashion similar to a wiki or Digg. As of April 2014 Stack Overflow has over 4,000,000 registered users, and it exceeded 10,000,000 questions in late August 2015. Based on the type of tags assigned to questions, the top eight most discussed topics on the site are: Java, JavaScript, C#, PHP, Android, jQuery, Python and HTML.

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/

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

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.

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 programming. The number of questions from each site 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, sh
ould not contain tabs '\t' or ampersands '&')
```

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

The answer should come in the form of a table like
\n\n



           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


\n\n


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)
\n\n


The output is not coming,can anyone correct the code or tell me what\'s wrong?
\n'
```
Tags : 'c++ c'

Mapping the real-world problem to a Machine Learning Problem

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 thought as predicting properties of a data-point that are not mutually exclusive, such as topics that are relevant for a document. A question on Stackoverflow might be about any of C, Pointers, FileIO and/or memory-management at the same time or none of these.
__Credit__ : http://scikit-learn.org/stable/modules/multiclass.html

Performance metric

Micro-Averaged F1-Score (Mean F Score) : The F1 score can be interpreted as a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0. The relative contribution of precision and recall to the F1 score are equal. The formula for the F1 score is:

*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 metric when we have class imbalance.

'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

Required Libraries

In [1]:

```
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 #installing- https://pypi.org/project/scikit-multilearn/
#!pip install scikit-multilearn
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
import joblib
```

Data Reading and Exploration

using SQLite database

In [2]:

```
os.chdir("D:\\Projects\\Machine-Learning\\TagPredictor\\Train")
```

In [4]:

```
#Creating db file from csv
#Learn SQL: https://www.w3schools.com/sql/default.asp
if not os.path.isfile('D:\\Projects\\Machine-Learning\\TagPredictor\\Train\\train.db'): #checking if the path contains the datafile or not
    start = datetime.now() #using datatime function to calculate the time at which this code starts
    disk_engine = create_engine('sqlite:///train.db') #creating database engine to create database
    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, iterator=True, encoding='utf-8', ):
        df.index += index_start
        j+=1
        print('{} rows'.format(j*chunksize))
        df.to_sql('data', disk_engine, if_exists='append')#appending file to a database
        index_start = df.index[-1] + 1
    print("Time taken to run this cell :", datetime.now() - start)#printing the time elapsed
```

Counting no of rows

In [5]:

```
conn = sqlite3.connect("train.db") #connecting to the database
#counting the no of rows
rows = pd.read_sql_query("SELECT count(*) FROM data",conn)
print(f"No of rows in the database is:\n {rows['count(*)'].values[0]}") #returns a dataframe with coloumn name as "count(*)"
conn.close()
```

```
No of rows in the database is:
 6034196
```

Checking for duplicates

In [6]:

```
#creating a connection
conn = sqlite3.connect("train.db")
df_no_dup = pd.read_sql_query('SELECT Title, Body, Tags, COUNT(*) as cnt_dup FROM data GROUP BY Title, Body, Tags', conn)
#in the above query we are selecting the features title,body etc counting each row and then creating another column as cnt_dup which signifies the repeated/duplicate rows
conn.close()
```

In [7]:

```
df_no_dup.head()
```

Out[7]:

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

In [8]:

```
df_no_dup.shape
```

Out[8]:

```
(4206315, 4)
```

In [9]:

```
print("number of duplicate questions :", rows['count(*)'].values[0]- df_no_dup.shape[0], "(",(1-((d
f_no_dup.shape[0])/(rows['count(*)'].values[0])))*100,"% )")
```

```
number of duplicate questions : 1827881 ( 30.292038906260256 % )
```

In [10]:

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

Out[10]:

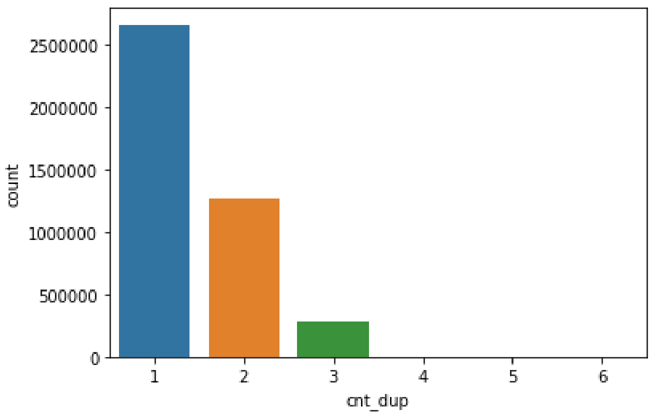
```
1    2656284
2    1272336
3     277575
4         90
5         25
6          5
Name: cnt_dup, dtype: int64
```

In [11]:

```
sns.countplot(x="cnt_dup", data=df_no_dup)
```

Out[11]:

```
&lt;matplotlib.axes._subplots.AxesSubplot at 0x14491dd3ac8&gt;
```



We can see that there are some questions that are repeated 4/5 and even 6times

In [12]:

```
print(f"Out of {rows['count(*)'].values[0]} rows:\n
{(df_no_dup.cnt_dup.value_counts().values[1]/rows['count(*)'].values[0])*100}% are repeated twice)
")
```

```
Out of 6034196 rows:
 21.085427122353998% are repeated twice)
```

In [13]:

```
print(f"Out of {rows['count(*)'].values[0]} rows:\n
{(df_no_dup.cnt_dup.value_counts().values[0]/rows['count(*)'].values[0])*100}% are present only on
ce)")
```

```
Out of 6034196 rows:
 44.02051242617907% are present only once)
```

Checking for the tags

In [14]:

```
df_no_dup.head()
```

Out[14]:

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

Checking for none values

How to check for null values in a dataframe : https://www.geeksforgeeks.org/working-with-missing-data-in-pandas/

In [15]:

```
bool_series = pd.isnull(df_no_dup["Tags"])
print(df_no_dup["Tags"][bool_series])
```

```
777547     None
962680     None
1126558    None
1256102    None
2430668    None
3329908    None
3551595    None
Name: Tags, dtype: object
```

In [16]:

```
df_no_dup["Tags"].fillna("None",inplace = True)
bool_series = pd.isnull(df_no_dup["Tags"])
print(df_no_dup["Tags"][bool_series])
```

```
Series([], Name: Tags, dtype: object)
```

In [17]:

```
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 : 0:00:11.140462
```

Out[17]:

	Title	Body	Tags	cnt_dup	tag_count
0	Implementing Boundary Value Analysis of S...	<pre><code>#include<iostream>\n#include&...	c++ c	1	2
1	Dynamic Datagrid Binding in Silverlight?	<p>I should do binding for datagrid dynamicall...	c# silverlight data-binding	1	3

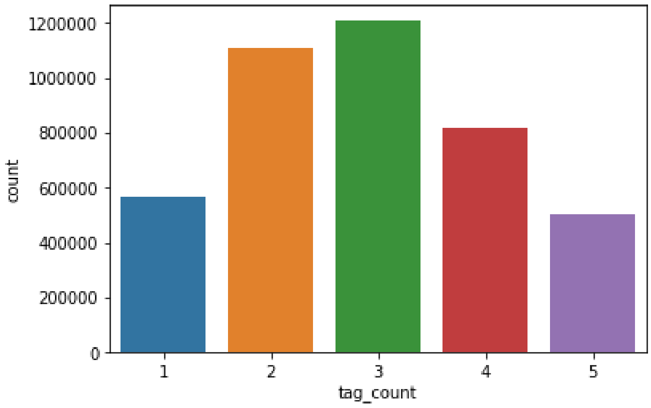
	Title	Body	Tags	cnt_dup	tag_count
2	Dynamic Datagrid Binding in Silverlight?	<p>I should do binding for datagrid dynamicall...	c# silverlight data-binding columns	1	4
3	java.lang.NoClassDefFoundError: javax/serv...	<p>I followed the guide in <a href="http://sta...	jsp jstl	1	2
4	java.sql.SQLException:[Microsoft][ODBC Dri...	I use the following code \n\n<pre><code>...	java jdbc	2	2

In [18]:

```
sns.countplot(x="tag_count", data=df_no_dup)
```

Out[18]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x14491ea6308>
```



In [19]:

```
df_no_dup["tag_count"].describe()
```

Out[19]:

```
count    4.206315e+06
mean     2.899439e+00
std      1.211917e+00
min      1.000000e+00
25%      2.000000e+00
50%      3.000000e+00
75%      4.000000e+00
max      5.000000e+00
Name: tag_count, dtype: float64
```

In [20]:

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

Out[20]:

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

Most of the questions have 2tags or 3 tags

In [21]:

```
#Creating a new database with no duplicates
if not os.path.isfile('D:\\Projects\\Machine-Learning\\TagPredictor\\Train\\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)
```

In [22]:

```
#creating the connection with database file.
if os.path.isfile('D:\\Projects\\Machine-Learning\\TagPredictor\\Train\\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.d
b file")
```

```
Time taken to run this cell : 0:01:45.457361
```

Analysis of Tags

Total number of unique tags

In [23]:

```
tag_data.head()
```

Out[23]:

	Tags
1	c# silverlight data-binding
2	c# silverlight data-binding columns
3	jsp jstl
4	java jdbc
5	facebook api facebook-php-sdk

In [24]:

```
#by default 'split()' will tokenize each tag using space.
#we are first splitting each tag using space and then tokenizing each tag

#intializating our count vectorizer
vectorizer = CountVectorizer(tokenizer = lambda x: x.split())

#fit_transform is the training on the tags data
tag_dtm = vectorizer.fit_transform(tag_data['Tags'])
```

In [25]:

```
print(f"No of datapoints: {tag_dtm.shape[0]}\nNo of unique datapoints: {tag_dtm.shape[1]}")
```

```
No of datapoints: 4206314
No of unique datapoints: 42049
```

In [26]:

```
#'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', '.class-fi
le', '.cs-file', '.doc', '.drv', '.ds-store']
```

Number of times a tag appeared

In [27]:

```
# 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 #axis=0 means column wise sum
result = dict(zip(tags, freqs))
```

In [28]:

```
#type(freqs)
freqs[0:10] #printing sample of frequencies
```

Out[28]:

```
array([ 18,  37,   1,  21, 138,  53,  14,  47,   1,   8], dtype=int64)
```

In [29]:

```
tags[0:10] #some of the tags
```

Out[29]:

```
['.a',
 '.app',
 '.asp.net-mvc',
 '.aspxauth',
 '.bash-profile',
 '.class-file',
 '.cs-file',
 '.doc',
 '.drv',
 '.ds-store']
```

In [30]:

```
#result is a dictionary which contains tags as the key and its frequency as a value
```

In [31]:

```
#Saving this dictionary to csv files.
if not os.path.isfile('D:\\Projects\\Machine-
Learning\\TagPredictor\\Train\\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()
```

Out[31]:

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

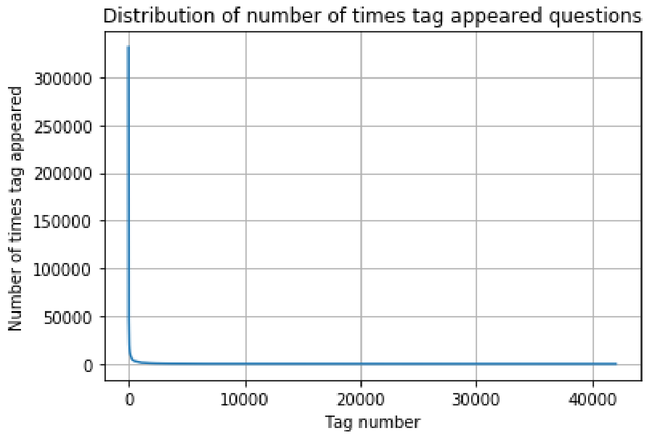
In [32]:

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

```
             Tags  Counts
4337           c#  331505
18069        java  299414
27250         php  284103
18157  javascript  265423
1234      android  235436
```

In [33]:

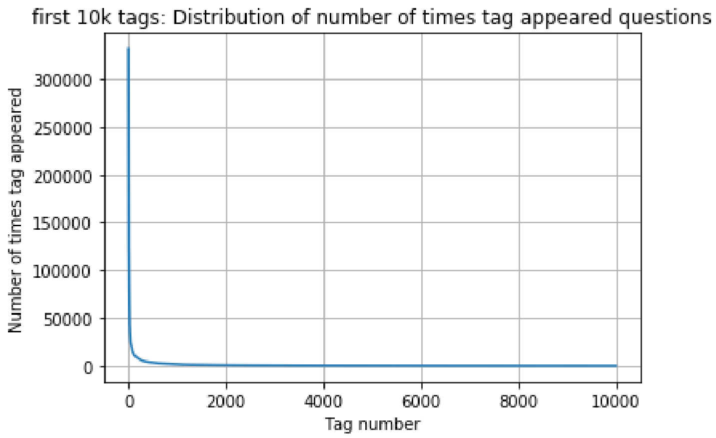
```
plt.plot(tag_counts)
plt.title("Distribution of number of times tag appeared questions")
plt.grid()
plt.xlabel("Tag number")
plt.ylabel("Number of times tag appeared")
plt.show()
```



We cannot observe much from this graph except for the fact that after some tags the count is falling sharply so no lets take a closer look

In [34]:

```
plt.plot(tag_counts[0:10000]) #taking only the first 10K tags and plotting
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])
```

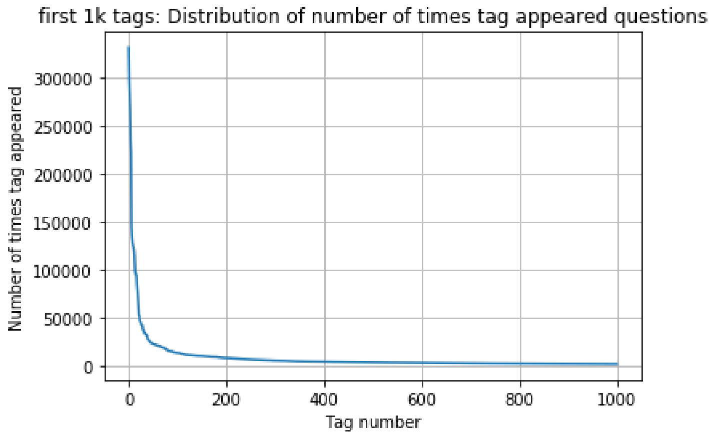



```
400 [331505  44829  22429  17728  13364  11162  10029   9148   8054   7151
```

```
   6466    5865    5370    4983    4526    4281    4144    3929    3750    3593
   3453    3299    3123    2986    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
     82      82      81      81      80      80      80      79      79      78
     78      78      78      77      77      76      76      76      75      75
     75      74      74      74      73      73      73      73      72      72]
```

In [35]:

```
plt.plot(tag_counts[0:1000]) #taking only 1K tags
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])
```



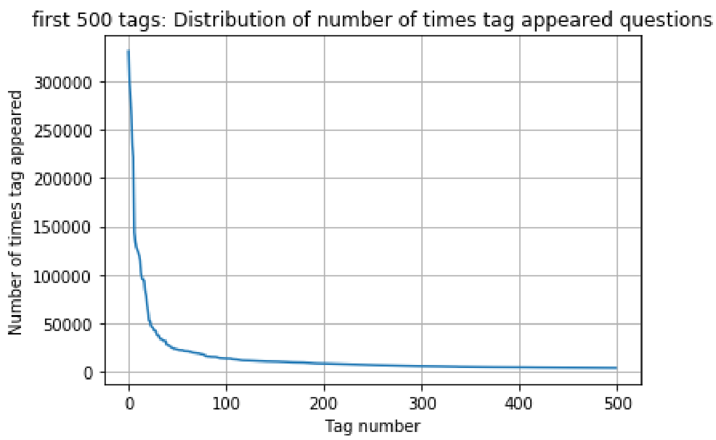

```
200 [331505 221533 122769  95160  62023  44829  37170  31897  26925  24537
  22429  21820  20957  19758  18905  17728  15533  15097  14884  13703
  13364  13157  12407  11658  11228  11162  10863  10600  10350  10224
  10029   9884   9719   9411   9252   9148   9040   8617   8361   8163
   8054   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   2986   2983   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]
```

We can see that after around 200tags the count decreasing sharply

In [36]:

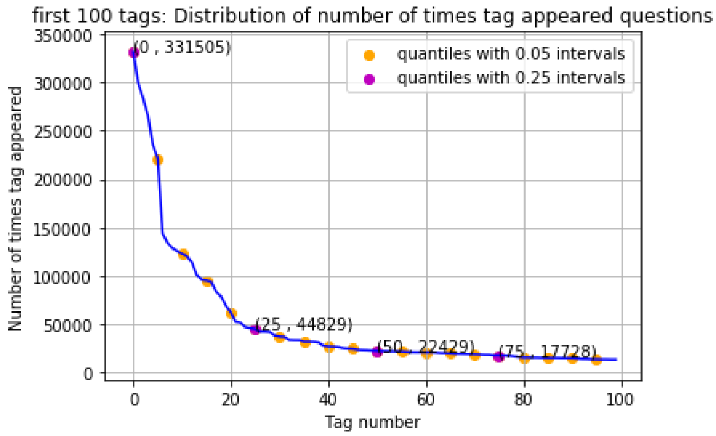
```
plt.plot(tag_counts[0:500])
plt.title('first 500 tags: Distribution of number of times tag appeared questions')
plt.grid()
plt.xlabel("Tag number")
plt.ylabel("Number of times tag appeared")
plt.show()
print(len(tag_counts[0:500:5]), tag_counts[0:500:5])
```




```
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
   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]
```

In [37]:

```
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 i
ntervals")
# 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 in
tervals")

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]
```

So on observing all the graph we can conclude that there are some tags that occur huge number of times whereas there also some tags which occur very small no. of times

In [38]:

```
# 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
```

There are **153 tags** which occured 1000 times

There are **14 tags** which occured 10000 times

In [39]:

```
tag_df_sorted.head(3)
```

Out[39]:

	Tags	Counts
4337	c#	331505
18069	java	299414
27250	php	284103

Top 3 frequently occuring tags are **C# , java , php**

Since some tags occur much more frequenctly than others, **Micro-averaged F1-score** is the appropriate metric for this probelm, because it takes care of the frequency of the value

Tags Per Question

In [40]:

```
#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 conve
```

```
rting this to [3, 4, 2, 2, 3]
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]
```

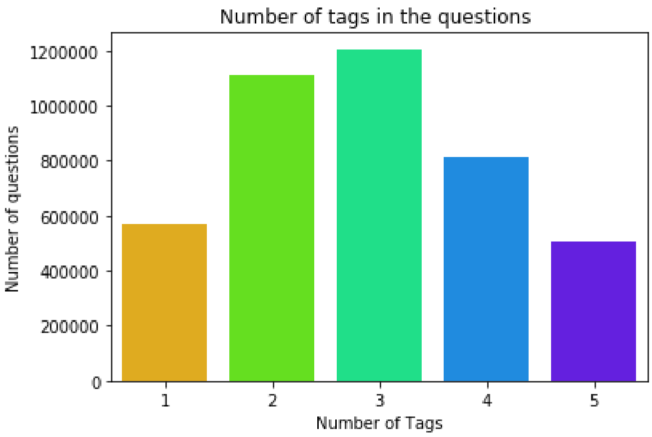
In [41]:

```
print( "Maximum number of tags per question: %d"%max(tag_quest_count))
print( "Minimum number of tags per question: %d"%min(tag_quest_count))
print( "Avg. number of tags per question: %f"% ((sum(tag_quest_count)*1.0)/len(tag_quest_count)))
```

```
Maximum number of tags per question: 5
Minimum number of tags per question: 1
Avg. number of tags per question: 2.899440
```

In [42]:

```
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()
```



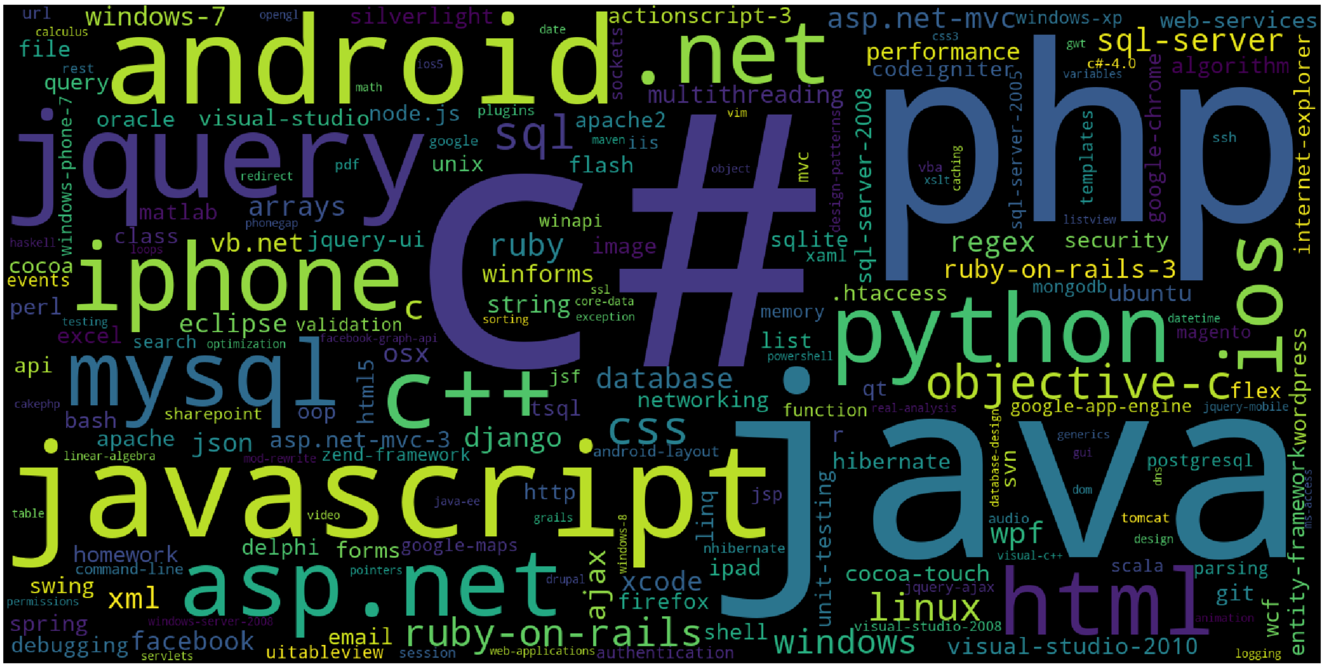
Observations:

1. Maximum number of tags per question: 5
2. Minimum number of tags per question: 1
3. Avg. number of tags per question: 2.899
4. Most of the questions are having 2 or 3 tags

In [43]:

```
# 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)
```



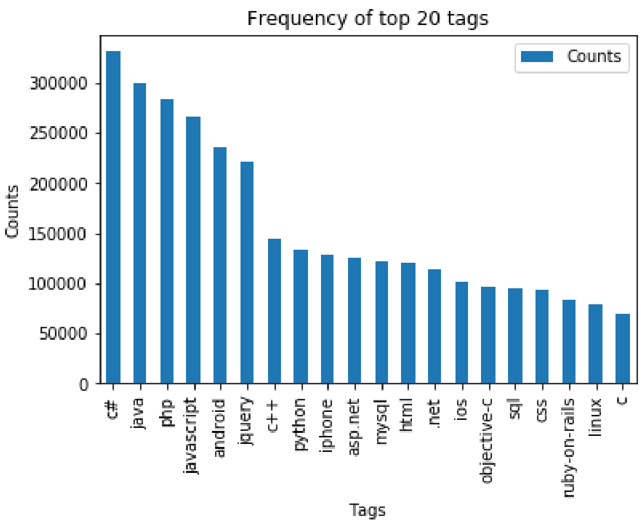

```
Time taken to run this cell : 0:00:17.094141
```

As we have seen earlier that the top 3 tags i.e. **c#, java and php** are the frequently occuring tags here also with the larger font represnting the frequency we can observed the same thing, apart from that ***android,javascript, iphone and jquery*** are also observed

The top 20 tags

In [44]:

```
i=np.arange(20)
tag_df_sorted.head(20).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()
```



Observations:

1. Majority of the most frequent tags are programming language.As it is obvious because stackoverflow is a platform mainly for coders

2. C# is the top most frequent programming language.
3. Android, IOS, Linux and windows are among the top most frequent operating systems.

Cleaning and preprocessing of Questions

Preprocessing

1. Sample 1M data points
2. Separate out code-snippets from Body
3. Remove Spcial characters from Question title and description (not in code)
4. Remove stop words (Except 'C')
5. Remove HTML Tags
6. Convert all the characters into small letters
7. Use SnowballStemmer to stem the words

In [45]:

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

In [46]:

```
#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()
```

In [47]:

```
sql_create_table = """CREATE TABLE IF NOT EXISTS QuestionsProcessed (question text NOT NULL, code
text, tags text, words_pre integer, words_post integer, is_code integer);"""
create_database_table("Titlemoreweight.db", sql_create_table)
```

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

In [48]:

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

read_db = 'D:\\Projects\\Machine-Learning\\TagPredictor\\Train\\train_no_dup.db'
write_db = 'D:\\Projects\\Machine-Learning\\TagPredictor\\Train\\Titlemoreweight.db'
train_datasize = 400000
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 500001;")
        # 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
```

In [49]:

```
#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=='c'))

    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) values (?,?,?,?,?,?)",tup)
    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
number of questions completed= 200000
number of questions completed= 300000
number of questions completed= 400000
number of questions completed= 500000
Avg. length of questions(Title+Body) before processing: 1239
Avg. length of questions(Title+Body) after processing: 424
Percent of questions containing code: 57
Time taken to run this cell : 0:31:06.709464
```

In [50]:

```
# 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()
```

In [51]:

```
if os.path.isfile(write_db):
    conn_r = create_connection("D:\\Projects\\Machine-
Learning\\TagPredictor\\Train\\Titlemoreweight.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 silverlight
bind datagrid dynam code wrote code debug code block seem bind correct grid come column form come
grid column although necessari bind nthank repli advance..',)
----------------------------------------------------------------------------------------------------
```

```
('java.lang.noclassdeffounderror javax servlet jsp tagext taglibraryvalid
java.lang.noclassdeffounderror javax servlet jsp tagext taglibraryvalid
java.lang.noclassdeffounderror javax servlet jsp tagext taglibraryvalid follow guid link instal js
tl got follow error tri launch jsp page java.lang.noclassdeffounderror javax servlet jsp tagext ta
glibraryvalid taglib declar instal jstl 1.1 tomcat webapp tri project work also tri version 1.2 js
tl still messag caus solv',)
----------------------------------------------------------------------------------------------------
('java.sql.sqlexcept microsoft odbc driver manag invalid descriptor index java.sql.sqlexcept
microsoft odbc driver manag invalid descriptor index java.sql.sqlexcept microsoft odbc driver
manag invalid descriptor index use follow code display caus solv',)
----------------------------------------------------------------------------------------------------
('better way updat feed fb php sdk better way updat feed fb php sdk better way updat feed fb php s
dk novic facebook api read mani tutori still confused.i find post feed api method like correct sec
ond way use curl someth like way better',)
----------------------------------------------------------------------------------------------------
('btnadd click event open two window record ad btnadd click event open two window record ad btnadd
click event open two window record ad open window search.aspx use code hav add button search.aspx
nwhen insert record btnadd click event open anoth window nafter insert record close window',)
----------------------------------------------------------------------------------------------------
('sql inject issu prevent correct form submiss php sql inject issu prevent correct form submiss ph
p sql inject issu prevent correct form submiss php check everyth think make sure input field safe
type sql inject good news safe bad news one tag mess form submiss place even touch life figur exac
t html use templat file forgiv okay entir php script get execut see data post none forum field pos
t problem use someth titl field none data get post current use print post see submit noth work fla
wless statement though also mention script work flawless local machin use host come across problem
state list input test mess',)
----------------------------------------------------------------------------------------------------
('countabl subaddit lebesgu measur countabl subaddit lebesgu measur countabl subaddit lebesgu meas
ur let lbrace rbrace sequenc set sigma -algebra mathcal want show left bigcup right leq sum left r
ight countabl addit measur defin set sigma algebra mathcal think use monoton properti somewher pro
of start appreci littl help nthank ad han answer make follow addit construct given han answer clea
r bigcup bigcup cap emptyset neq left bigcup right left bigcup right sum left right also construct
subset monoton left right leq left right final would sum leq sum result follow',)
----------------------------------------------------------------------------------------------------
('hql equival sql queri hql equival sql queri hql equival sql queri hql queri replac name class pr
operti name error occur hql error',)
----------------------------------------------------------------------------------------------------
('undefin symbol architectur i386 objc class skpsmtpmessag referenc error undefin symbol
architectur i386 objc class skpsmtpmessag referenc error undefin symbol architectur i386 objc
class skpsmtpmessag referenc error import framework send email applic background import framework
i.e skpsmtpmessag somebodi suggest get error collect2 ld return exit status import framework corre
ct sorc taken framework follow mfmailcomposeviewcontrol question lock field updat answer drag drop
folder project click copi nthat',)
----------------------------------------------------------------------------------------------------
```

In [52]:

```
#Taking 0.5 Million entries to a dataframe.
write_db = 'D:\\Projects\\Machine-Learning\\TagPredictor\\Train\\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""",
conn_r)
conn_r.commit()
conn_r.close()
```

In [53]:

```
preprocessed_data.head()
```

Out[53]:

	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

In [54]:

```
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 : 500000
number of dimensions : 2
```

Converting string Tags to multilable output variables

In [55]:

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

Selecting 500 Tags

In [56]:

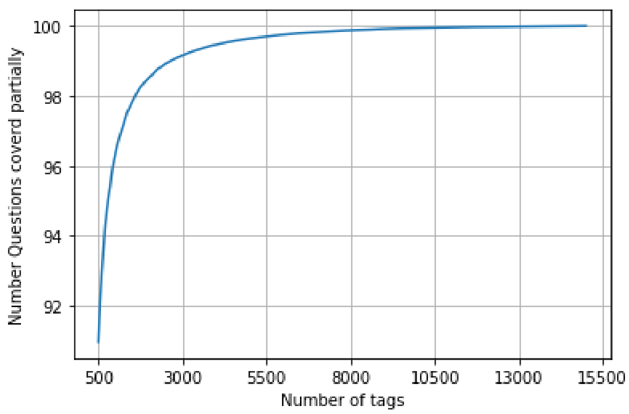
```
def tags_to_choose(n):
    t = multilabel_y.sum(axis=0).tolist()[0]
    sorted_tags_i = sorted(range(len(t)), key=lambda i: t[i], reverse=True)
    multilabel_yn=multilabel_y[:,sorted_tags_i[:n]]
    return multilabel_yn

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

In [57]:

```
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))
```

In [58]:

```
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
the tags)
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  99.157 % of questions
with  500 tags we are covering  90.956 % of questions
```

In [59]:

```
# 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_q
s)
```

```
number of questions that are not covered : 45221 out of  500000
```

In [60]:

```
x_train=preprocessed_data.head(train_datasize)
x_test=preprocessed_data.tail(preprocessed_data.shape[0] - 400000)

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

In [61]:

```
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 : (400000, 500)
Number of data points in test data : (100000, 500)
```

We could have used sklearn's library that will required to convert our sparse matrix into dense form so we tried this approach

Featurizing data

In [62]:

```
start = datetime.now()
vectorizer = TfidfVectorizer(min_df=0.00009, max_features=200000, smooth_idf=True, norm="l2", \
                             tokenizer = lambda x: x.split(), sublinear_tf=False, 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 : 0:02:49.671822
```

Smooth idf weights by adding one to document frequencies, as if an extra document was seen containing every term in the collection exactly once. Prevents zero divisions.|

takes care of if unseen values are present in test time

sublinear_tf Apply sublinear tf scaling, i.e. replace tf with 1 + log(tf).

In [63]:

```
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: (400000, 95585) Y : (400000, 500)
Dimensions of test data X: (100000, 95585) Y: (100000, 500)
```

Applying Logistic Regression(SGD Classifier) with OneVsRest Classifier

Hyperparameter tuning

In [40]:

```
from sklearn.model_selection import GridSearchCV
#start = datetime.now()
classifier = OneVsRestClassifier(SGDClassifier())

params = {
    "estimator__alpha" : [0.00001,0.0001,0.001,0.01],
    'estimator__loss':['log'],
    'estimator__penalty':['l2']
}

best = GridSearchCV(classifier , params, scoring = "f1_weighted")
best.fit(x_train_multilabel, y_train)
```

Out[40]:

```
GridSearchCV(estimator=OneVsRestClassifier(estimator=SGDClassifier()),
             param_grid={'estimator__alpha': [1e-05, 0.0001, 0.001, 0.01],
                         'estimator__loss': ['log'],
                         'estimator__penalty': ['l2']},
             scoring='f1_weighted')
```

In [41]:

```
best.best_params_
```

Out[41]:

```
{'estimator__alpha': 1e-05,
 'estimator__loss': 'log',
 'estimator__penalty': 'l2'}
```

In [42]:

```
start = datetime.now()
classifier = OneVsRestClassifier(SGDClassifier(loss='log', alpha=0.00001, penalty='l2'), 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 : 0.20904
Hamming loss  0.00290642
Micro-average quality numbers
Precision: 0.7777, Recall: 0.2295, F1-measure: 0.3544
Macro-average quality numbers
Precision: 0.5481, Recall: 0.1266, F1-measure: 0.1903
              precision    recall  f1-score   support

           0       0.96      0.60      0.74      5519
           1       0.70      0.24      0.35      8190
```

1	[illegible]	[illegible]	[illegible]	[illegible]
2	0.83	0.32	0.46	6529
3	0.83	0.37	0.52	3231
4	0.83	0.37	0.51	6430
5	0.83	0.30	0.44	2879
6	0.90	0.44	0.59	5086
7	0.89	0.50	0.64	4533
8	0.61	0.12	0.20	3000
9	0.84	0.50	0.62	2765
10	0.63	0.13	0.22	3051
11	0.75	0.27	0.39	3009
12	0.68	0.19	0.29	2630
13	0.77	0.17	0.28	1426
14	0.93	0.45	0.60	2548
15	0.74	0.10	0.18	2371
16	0.67	0.19	0.30	873
17	0.92	0.55	0.69	2151
18	0.68	0.17	0.27	2204
19	0.73	0.36	0.49	831
20	0.79	0.33	0.47	1860
21	0.28	0.06	0.10	2023
22	0.54	0.17	0.25	1513
23	0.93	0.38	0.54	1207
24	0.58	0.25	0.34	506
25	0.75	0.25	0.37	425
26	0.67	0.34	0.45	793
27	0.61	0.26	0.36	1291
28	0.74	0.26	0.39	1208
29	0.45	0.05	0.10	406
30	0.84	0.10	0.18	504
31	0.31	0.06	0.10	732
32	0.60	0.21	0.31	441
33	0.63	0.11	0.18	1645
34	0.76	0.18	0.29	1058
35	0.83	0.43	0.57	946
36	0.70	0.11	0.19	644
37	0.97	0.56	0.71	136
38	0.63	0.29	0.40	570
39	0.90	0.13	0.23	766
40	0.68	0.19	0.29	1132
41	0.48	0.16	0.24	174
42	0.74	0.29	0.42	210
43	0.86	0.36	0.51	433
44	0.67	0.41	0.51	626
45	0.79	0.21	0.33	852
46	0.79	0.28	0.42	534
47	0.36	0.10	0.16	350
48	0.76	0.41	0.53	496
49	0.81	0.49	0.61	785
50	0.16	0.03	0.04	475
51	0.44	0.04	0.07	305
52	0.44	0.02	0.03	251
53	0.70	0.26	0.38	914
54	0.49	0.12	0.19	728
55	0.00	0.00	0.00	258
56	0.48	0.12	0.19	821
57	0.56	0.04	0.08	541
58	0.84	0.15	0.25	748
59	0.96	0.51	0.66	724
60	0.28	0.03	0.05	660
61	0.89	0.14	0.24	235
62	0.94	0.61	0.74	718
63	0.88	0.48	0.62	468
64	0.53	0.25	0.34	191
65	0.36	0.07	0.12	429
66	0.40	0.03	0.06	415
67	0.75	0.34	0.46	274
68	0.85	0.32	0.47	510
69	0.71	0.30	0.42	466
70	0.26	0.03	0.05	305
71	0.53	0.11	0.18	247
72	0.81	0.36	0.50	401
73	0.98	0.60	0.75	86
74	0.76	0.26	0.39	120
75	0.91	0.57	0.70	129
76	0.00	0.00	0.00	473
77	0.48	0.22	0.30	143
78	0.83	0.26	0.39	347

78	[illegible]	0.20	0.39	347
79	0.73	0.10	0.17	479
80	0.59	0.23	0.33	279
81	0.96	0.05	0.09	461
82	0.00	0.00	0.00	298
83	0.82	0.30	0.44	396
84	0.57	0.20	0.29	184
85	0.69	0.13	0.22	573
86	0.00	0.00	0.00	325
87	0.57	0.13	0.21	273
88	0.47	0.16	0.24	135
89	0.30	0.06	0.09	232
90	0.64	0.24	0.35	409
91	0.59	0.10	0.17	420
92	0.81	0.37	0.50	408
93	0.72	0.34	0.46	241
94	0.29	0.02	0.04	211
95	0.43	0.05	0.08	277
96	0.43	0.01	0.03	410
97	0.95	0.11	0.19	501
98	0.82	0.43	0.57	136
99	0.58	0.18	0.27	239
100	0.62	0.03	0.06	324
101	0.97	0.41	0.58	277
102	0.95	0.52	0.67	613
103	0.59	0.08	0.15	157
104	0.23	0.03	0.05	295
105	0.94	0.23	0.37	334
106	1.00	0.01	0.01	335
107	0.77	0.24	0.37	389
108	0.59	0.09	0.16	251
109	0.63	0.27	0.38	317
110	1.00	0.01	0.02	187
111	0.00	0.00	0.00	140
112	0.55	0.11	0.18	154
113	0.57	0.06	0.11	332
114	0.46	0.15	0.22	323
115	0.48	0.06	0.11	344
116	0.76	0.28	0.41	370
117	0.62	0.13	0.21	313
118	0.78	0.14	0.23	874
119	0.51	0.10	0.16	293
120	0.00	0.00	0.00	200
121	0.83	0.27	0.41	463
122	0.50	0.07	0.12	119
123	0.00	0.00	0.00	256
124	0.91	0.54	0.68	195
125	0.48	0.07	0.13	138
126	0.86	0.31	0.46	376
127	0.43	0.02	0.05	122
128	0.27	0.02	0.03	252
129	0.47	0.06	0.10	144
130	0.25	0.01	0.03	150
131	0.33	0.00	0.01	210
132	0.59	0.09	0.15	361
133	0.96	0.29	0.45	453
134	0.89	0.60	0.72	124
135	0.00	0.00	0.00	91
136	0.75	0.16	0.27	128
137	0.66	0.27	0.38	218
138	1.00	0.02	0.05	243
139	0.44	0.10	0.16	149
140	0.71	0.16	0.26	318
141	0.33	0.06	0.11	159
142	0.66	0.27	0.39	274
143	0.90	0.45	0.60	362
144	0.70	0.06	0.11	118
145	0.69	0.25	0.37	164
146	0.61	0.14	0.23	461
147	0.67	0.22	0.33	159
148	0.39	0.08	0.13	166
149	0.99	0.19	0.32	346
150	1.00	0.02	0.03	350
151	1.00	0.38	0.55	55
152	0.87	0.26	0.40	387
153	0.40	0.03	0.05	150
154	0.71	0.02	0.03	281
155	0.33	0.01	0.02	202

155	0.55	0.01	0.02	202
156	0.80	0.50	0.62	130
157	0.45	0.02	0.04	245
158	0.91	0.42	0.58	177
159	0.53	0.13	0.21	130
160	0.50	0.05	0.10	336
161	0.95	0.40	0.56	220
162	0.40	0.02	0.03	229
163	0.90	0.23	0.37	316
164	0.79	0.17	0.28	283
165	0.65	0.13	0.22	197
166	0.54	0.15	0.23	101
167	0.40	0.06	0.11	231
168	0.59	0.12	0.19	370
169	0.42	0.12	0.18	258
170	0.38	0.05	0.09	101
171	0.41	0.17	0.24	89
172	0.57	0.20	0.30	193
173	0.44	0.10	0.17	309
174	0.60	0.03	0.07	172
175	0.96	0.54	0.69	95
176	0.97	0.32	0.48	346
177	0.94	0.16	0.27	322
178	0.63	0.26	0.37	232
179	0.50	0.02	0.05	125
180	0.67	0.11	0.19	145
181	0.40	0.03	0.05	77
182	0.00	0.00	0.00	182
183	0.61	0.16	0.25	257
184	0.00	0.00	0.00	216
185	0.38	0.03	0.06	242
186	0.36	0.05	0.09	165
187	0.81	0.32	0.46	263
188	0.44	0.02	0.04	174
189	0.88	0.10	0.18	136
190	0.95	0.20	0.33	202
191	0.44	0.03	0.06	134
192	0.77	0.17	0.28	230
193	0.47	0.10	0.17	90
194	0.68	0.30	0.42	185
195	0.00	0.00	0.00	156
196	0.00	0.00	0.00	160
197	0.00	0.00	0.00	266
198	0.50	0.01	0.01	284
199	1.00	0.01	0.01	145
200	0.95	0.41	0.57	212
201	0.71	0.09	0.15	317
202	0.87	0.27	0.41	427
203	0.38	0.02	0.04	232
204	0.53	0.09	0.16	217
205	0.51	0.12	0.19	527
206	0.00	0.00	0.00	124
207	0.50	0.03	0.06	103
208	0.94	0.25	0.40	287
209	0.37	0.04	0.07	193
210	0.86	0.08	0.15	220
211	1.00	0.01	0.03	140
212	0.17	0.01	0.01	161
213	0.55	0.08	0.14	72
214	0.63	0.20	0.31	396
215	0.91	0.07	0.14	134
216	0.67	0.01	0.01	400
217	0.50	0.13	0.21	75
218	0.97	0.39	0.56	219
219	0.91	0.15	0.25	210
220	1.00	0.18	0.30	298
221	0.96	0.26	0.41	266
222	0.74	0.14	0.23	290
223	0.00	0.00	0.00	128
224	0.82	0.26	0.40	159
225	0.59	0.06	0.11	164
226	0.64	0.21	0.31	144
227	0.57	0.17	0.27	276
228	0.00	0.00	0.00	235
229	1.00	0.00	0.01	216
230	0.44	0.04	0.07	228
231	0.67	0.28	0.40	64
232	0.25	0.01	0.02	103

232	0.25	0.01	0.02	103
233	0.68	0.08	0.14	216
234	1.00	0.02	0.03	116
235	0.50	0.16	0.24	77
236	1.00	0.46	0.63	67
237	1.00	0.02	0.04	218
238	0.33	0.01	0.03	139
239	0.00	0.00	0.00	94
240	0.58	0.14	0.23	77
241	0.50	0.01	0.02	167
242	0.88	0.16	0.27	86
243	0.67	0.03	0.07	58
244	0.62	0.06	0.10	269
245	0.09	0.01	0.02	112
246	0.98	0.36	0.53	255
247	0.44	0.14	0.21	58
248	0.00	0.00	0.00	81
249	0.00	0.00	0.00	131
250	0.23	0.03	0.06	93
251	0.72	0.14	0.23	154
252	1.00	0.01	0.02	129
253	0.70	0.17	0.27	83
254	0.00	0.00	0.00	191
255	0.00	0.00	0.00	219
256	0.67	0.02	0.03	130
257	0.58	0.16	0.25	93
258	0.68	0.18	0.29	217
259	0.50	0.04	0.08	141
260	1.00	0.06	0.11	143
261	0.56	0.02	0.04	219
262	0.58	0.10	0.17	107
263	0.31	0.06	0.10	236
264	0.34	0.09	0.15	119
265	0.57	0.06	0.10	72
266	0.00	0.00	0.00	70
267	0.29	0.02	0.04	107
268	0.75	0.20	0.31	169
269	0.33	0.04	0.07	129
270	0.75	0.31	0.44	159
271	0.79	0.08	0.14	190
272	0.68	0.08	0.14	248
273	0.92	0.28	0.43	264
274	0.88	0.36	0.51	105
275	1.00	0.01	0.02	104
276	0.00	0.00	0.00	115
277	0.85	0.34	0.48	170
278	0.79	0.08	0.14	145
279	0.91	0.19	0.31	230
280	0.63	0.21	0.32	80
281	0.67	0.31	0.43	217
282	0.70	0.19	0.30	175
283	0.40	0.01	0.01	269
284	0.73	0.11	0.19	74
285	0.89	0.23	0.36	206
286	0.95	0.24	0.39	227
287	1.00	0.08	0.14	130
288	0.50	0.02	0.03	129
289	0.00	0.00	0.00	80
290	0.33	0.01	0.02	99
291	0.79	0.05	0.10	208
292	0.00	0.00	0.00	67
293	0.87	0.24	0.37	109
294	0.57	0.11	0.19	140
295	0.17	0.02	0.03	241
296	0.33	0.07	0.11	72
297	0.25	0.02	0.03	107
298	0.91	0.16	0.28	61
299	1.00	0.21	0.34	77
300	1.00	0.01	0.02	111
301	0.00	0.00	0.00	126
302	0.00	0.00	0.00	73
303	0.73	0.18	0.29	176
304	1.00	0.22	0.36	230
305	0.98	0.31	0.47	156
306	0.63	0.21	0.32	146
307	0.00	0.00	0.00	98
308	0.00	0.00	0.00	78
309	1.00	0.01	0.02	94

309	1.00	0.01	0.02	94
310	0.78	0.09	0.16	162
311	0.78	0.16	0.26	116
312	0.47	0.12	0.19	57
313	1.00	0.02	0.03	65
314	0.49	0.15	0.23	138
315	0.59	0.09	0.15	195
316	0.57	0.19	0.28	69
317	0.33	0.01	0.01	134
318	0.51	0.14	0.21	148
319	0.81	0.19	0.30	161
320	0.25	0.03	0.05	104
321	0.89	0.27	0.41	156
322	0.59	0.14	0.23	134
323	0.57	0.20	0.30	232
324	0.45	0.05	0.10	92
325	0.55	0.03	0.06	197
326	0.00	0.00	0.00	126
327	0.00	0.00	0.00	115
328	1.00	0.20	0.34	198
329	0.71	0.10	0.17	125
330	1.00	0.02	0.05	81
331	0.40	0.02	0.04	94
332	0.00	0.00	0.00	56
333	0.29	0.01	0.01	260
334	0.00	0.00	0.00	60
335	0.57	0.04	0.07	110
336	0.78	0.25	0.38	71
337	0.17	0.02	0.03	66
338	0.59	0.19	0.29	150
339	0.00	0.00	0.00	54
340	0.90	0.18	0.31	195
341	0.00	0.00	0.00	79
342	0.75	0.08	0.14	38
343	0.56	0.12	0.19	43
344	0.57	0.06	0.11	68
345	0.59	0.18	0.27	73
346	0.00	0.00	0.00	116
347	0.90	0.08	0.15	111
348	0.38	0.05	0.08	63
349	0.90	0.27	0.41	104
350	0.62	0.18	0.28	44
351	0.50	0.03	0.05	40
352	1.00	0.10	0.19	136
353	0.45	0.09	0.15	54
354	0.00	0.00	0.00	134
355	0.56	0.08	0.14	120
356	0.42	0.04	0.06	228
357	0.83	0.06	0.10	269
358	0.85	0.14	0.24	80
359	0.77	0.16	0.27	140
360	0.33	0.02	0.03	125
361	0.95	0.21	0.34	169
362	0.00	0.00	0.00	56
363	0.96	0.32	0.48	154
364	0.00	0.00	0.00	58
365	0.00	0.00	0.00	71
366	1.00	0.24	0.39	54
367	1.00	0.01	0.02	116
368	0.00	0.00	0.00	54
369	0.00	0.00	0.00	71
370	0.00	0.00	0.00	61
371	0.00	0.00	0.00	71
372	0.69	0.17	0.28	52
373	0.38	0.02	0.04	150
374	0.50	0.02	0.04	93
375	0.00	0.00	0.00	67
376	0.00	0.00	0.00	76
377	1.00	0.02	0.04	106
378	0.00	0.00	0.00	86
379	0.00	0.00	0.00	14
380	1.00	0.03	0.06	122
381	0.00	0.00	0.00	104
382	0.00	0.00	0.00	66
383	0.85	0.10	0.18	110
384	0.00	0.00	0.00	155
385	0.50	0.02	0.04	50

386	0.20	0.03	0.05	64
387	0.00	0.00	0.00	93
388	0.50	0.01	0.02	102
389	0.00	0.00	0.00	108
390	0.98	0.26	0.41	178
391	0.70	0.06	0.11	115
392	1.00	0.14	0.25	42
393	0.00	0.00	0.00	134
394	0.00	0.00	0.00	112
395	0.69	0.05	0.10	176
396	0.00	0.00	0.00	125
397	0.84	0.07	0.13	224
398	1.00	0.19	0.32	63
399	0.00	0.00	0.00	59
400	0.50	0.13	0.20	63
401	1.00	0.01	0.02	98
402	1.00	0.01	0.01	162
403	0.80	0.05	0.09	83
404	0.91	0.53	0.67	19
405	0.75	0.03	0.06	92
406	1.00	0.05	0.09	41
407	0.71	0.12	0.20	43
408	0.83	0.03	0.06	160
409	0.00	0.00	0.00	50
410	0.00	0.00	0.00	19
411	0.00	0.00	0.00	175
412	0.00	0.00	0.00	72
413	0.00	0.00	0.00	95
414	0.33	0.01	0.02	97
415	0.00	0.00	0.00	48
416	0.56	0.12	0.20	83
417	0.00	0.00	0.00	40
418	0.00	0.00	0.00	91
419	0.62	0.09	0.16	90
420	0.43	0.08	0.14	37
421	0.00	0.00	0.00	66
422	0.56	0.19	0.29	73
423	0.43	0.05	0.10	56
424	0.94	0.45	0.61	33
425	0.00	0.00	0.00	76
426	0.50	0.02	0.05	81
427	1.00	0.19	0.31	150
428	0.93	0.48	0.64	29
429	0.00	0.00	0.00	389
430	1.00	0.04	0.07	167
431	0.00	0.00	0.00	123
432	0.45	0.13	0.20	39
433	0.38	0.06	0.11	82
434	1.00	0.30	0.47	66
435	0.63	0.13	0.21	93
436	0.68	0.15	0.25	87
437	0.50	0.01	0.02	86
438	0.86	0.18	0.30	104
439	0.00	0.00	0.00	100
440	0.00	0.00	0.00	141
441	0.53	0.09	0.16	110
442	0.18	0.02	0.03	123
443	0.00	0.00	0.00	71
444	0.00	0.00	0.00	109
445	1.00	0.08	0.15	48
446	0.35	0.08	0.13	76
447	0.00	0.00	0.00	38
448	0.69	0.22	0.34	81
449	0.43	0.02	0.04	132
450	0.41	0.09	0.14	81
451	1.00	0.04	0.08	76
452	0.00	0.00	0.00	44
453	0.00	0.00	0.00	44
454	0.89	0.23	0.36	70
455	0.00	0.00	0.00	155
456	0.75	0.07	0.13	43
457	0.50	0.03	0.05	72
458	0.00	0.00	0.00	62
459	1.00	0.04	0.08	69
460	0.00	0.00	0.00	119
461	1.00	0.03	0.05	79
462	0.57	0.09	0.15	47

```
         463       0.00      0.00      0.00       104
         464       0.75      0.14      0.24       106
         465       0.00      0.00      0.00        64
         466       0.71      0.07      0.13       173
         467       0.88      0.07      0.12       107
         468       0.00      0.00      0.00       126
         469       0.00      0.00      0.00       114
         470       0.98      0.34      0.50       140
         471       0.00      0.00      0.00        79
         472       0.32      0.04      0.07       143
         473       0.67      0.01      0.02       158
         474       0.00      0.00      0.00       138
         475       0.00      0.00      0.00        59
         476       0.25      0.01      0.02        88
         477       0.91      0.17      0.29       176
         478       1.00      0.50      0.67        24
         479       0.00      0.00      0.00        92
         480       0.94      0.16      0.27       100
         481       0.75      0.03      0.06       103
         482       0.20      0.01      0.03        74
         483       0.94      0.16      0.28       105
         484       0.00      0.00      0.00        83
         485       0.00      0.00      0.00        82
         486       1.00      0.04      0.08        71
         487       0.20      0.02      0.03       120
         488       0.00      0.00      0.00       105
         489       0.78      0.08      0.15        87
         490       1.00      0.28      0.44        32
         491       0.00      0.00      0.00        69
         492       0.00      0.00      0.00        49
         493       0.00      0.00      0.00       117
         494       0.00      0.00      0.00        61
         495       0.00      0.00      0.00       344
         496       0.50      0.06      0.10        52
         497       0.00      0.00      0.00       137
         498       1.00      0.01      0.02        98
         499       1.00      0.01      0.02        79

   micro avg       0.78      0.23      0.35    173812
   macro avg       0.55      0.13      0.19    173812
weighted avg       0.68      0.23      0.33    173812
 samples avg       0.32      0.22      0.24    173812

Time taken to run this cell : 0:03:10.354285
```

In [45]:

```
joblib.dump(classifier, 'lr_with_more_title_weight.pkl')
```

Out[45]:

```
['lr_with_more_title_weight.pkl']
```

Logistic Regression

Hyperparameter Tuning

In [82]:

```
from sklearn.model_selection import GridSearchCV
#start = datetime.now()
classifier2 = OneVsRestClassifier(LogisticRegression(penalty='l1'))

params = {
    "estimator__C" : [0.0001,0.001,0.01,1],
    'estimator__penalty':['l2']
}

best2 = GridSearchCV(classifier2 , params, scoring = "f1_weighted")
best2.fit(x_train_multilabel, y_train)
```

Out[82]:

Out[82]:

```
GridSearchCV(estimator=OneVsRestClassifier(estimator=LogisticRegression(penalty='l1')),
             param_grid={'estimator__C': [0.0001, 0.001, 0.01, 1],
                         'estimator__penalty': ['l2']},
             scoring='f1_weighted')
```

In [83]:

```
best2.best_params_
```

Out[83]:

```
{'estimator__C': 1, 'estimator__penalty': 'l2'}
```

In [84]:

```
start = datetime.now()
classifier_2 = OneVsRestClassifier(LogisticRegression(C=1,penalty='l2'), n_jobs=-1)
classifier_2.fit(x_train_multilabel, y_train)
predictions_2 = classifier_2.predict(x_test_multilabel)
print("Accuracy :",metrics.accuracy_score(y_test, predictions_2))
print("Hamming loss ",metrics.hamming_loss(y_test,predictions_2))


precision = precision_score(y_test, predictions_2, average='micro')
recall = recall_score(y_test, predictions_2, average='micro')
f1 = f1_score(y_test, predictions_2, 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_2, average='macro')
recall = recall_score(y_test, predictions_2, average='macro')
f1 = f1_score(y_test, predictions_2, 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_2))
print("Time taken to run this cell :", datetime.now() - start)
```

```
Accuracy : 0.23742
Hamming loss  0.00275658
Micro-average quality numbers
Precision: 0.7422, Recall: 0.3172, F1-measure: 0.4444
Macro-average quality numbers
Precision: 0.5772, Recall: 0.2319, F1-measure: 0.3143
              precision    recall  f1-score   support

           0       0.95      0.68      0.79      5519
           1       0.70      0.31      0.43      8190
           2       0.81      0.38      0.52      6529
           3       0.83      0.45      0.59      3231
           4       0.81      0.41      0.55      6430
           5       0.82      0.35      0.49      2879
           6       0.88      0.50      0.64      5086
           7       0.88      0.55      0.68      4533
           8       0.60      0.13      0.21      3000
           9       0.83      0.55      0.66      2765
          10       0.61      0.19      0.29      3051
          11       0.71      0.35      0.47      3009
          12       0.64      0.25      0.36      2630
          13       0.77      0.26      0.39      1426
          14       0.91      0.51      0.66      2548
          15       0.66      0.18      0.28      2371
          16       0.66      0.23      0.34       873
          17       0.90      0.60      0.72      2151
          18       0.63      0.22      0.33      2204
          19       0.72      0.39      0.51       831
          20       0.78      0.41      0.54      1860
          21       0.30      0.09      0.13      2023
          22       0.54      0.23      0.32      1513
          23       0.92      0.49      0.64      1207
          24       0.57      0.28      0.38       506
```

25	0.69	0.30	0.42	425
26	0.66	0.39	0.49	793
27	0.61	0.34	0.43	1291
28	0.76	0.34	0.47	1208
29	0.47	0.09	0.15	406
30	0.79	0.15	0.26	504
31	0.31	0.08	0.13	732
32	0.61	0.26	0.36	441
33	0.61	0.20	0.30	1645
34	0.73	0.22	0.34	1058
35	0.83	0.53	0.65	946
36	0.69	0.19	0.30	644
37	0.98	0.62	0.76	136
38	0.64	0.34	0.45	570
39	0.88	0.25	0.39	766
40	0.62	0.29	0.39	1132
41	0.46	0.17	0.25	174
42	0.78	0.44	0.56	210
43	0.81	0.40	0.54	433
44	0.67	0.48	0.56	626
45	0.74	0.30	0.42	852
46	0.78	0.39	0.52	534
47	0.38	0.13	0.19	350
48	0.74	0.49	0.59	496
49	0.79	0.60	0.68	785
50	0.22	0.05	0.09	475
51	0.39	0.10	0.16	305
52	0.42	0.02	0.04	251
53	0.68	0.37	0.48	914
54	0.49	0.15	0.23	728
55	0.33	0.01	0.02	258
56	0.46	0.20	0.28	821
57	0.51	0.08	0.14	541
58	0.81	0.25	0.38	748
59	0.95	0.59	0.73	724
60	0.34	0.06	0.11	660
61	0.86	0.18	0.30	235
62	0.92	0.69	0.79	718
63	0.85	0.62	0.71	468
64	0.54	0.31	0.39	191
65	0.35	0.12	0.17	429
66	0.28	0.05	0.09	415
67	0.76	0.44	0.56	274
68	0.83	0.46	0.59	510
69	0.71	0.41	0.52	466
70	0.27	0.06	0.10	305
71	0.53	0.15	0.24	247
72	0.81	0.47	0.59	401
73	0.98	0.71	0.82	86
74	0.72	0.34	0.46	120
75	0.90	0.64	0.75	129
76	0.50	0.01	0.01	473
77	0.44	0.25	0.32	143
78	0.81	0.41	0.55	347
79	0.73	0.21	0.32	479
80	0.60	0.33	0.43	279
81	0.85	0.15	0.25	461
82	0.50	0.04	0.07	298
83	0.80	0.42	0.55	396
84	0.53	0.26	0.35	184
85	0.63	0.20	0.31	573
86	0.52	0.04	0.07	325
87	0.50	0.21	0.30	273
88	0.45	0.22	0.30	135
89	0.36	0.09	0.15	232
90	0.58	0.33	0.42	409
91	0.64	0.22	0.33	420
92	0.79	0.50	0.62	408
93	0.69	0.43	0.53	241
94	0.36	0.05	0.08	211
95	0.35	0.08	0.13	277
96	0.31	0.04	0.07	410
97	0.90	0.26	0.40	501
98	0.79	0.55	0.65	136
99	0.55	0.29	0.38	239
100	0.65	0.10	0.18	324
101	0.94	0.54	0.69	277

102	0.93	0.66	0.77	613
103	0.44	0.13	0.20	157
104	0.25	0.06	0.09	295
105	0.82	0.34	0.48	334
106	0.92	0.10	0.19	335
107	0.75	0.44	0.56	389
108	0.60	0.20	0.30	251
109	0.57	0.37	0.45	317
110	0.73	0.06	0.11	187
111	0.54	0.05	0.09	140
112	0.65	0.29	0.40	154
113	0.64	0.15	0.25	332
114	0.48	0.26	0.34	323
115	0.50	0.19	0.28	344
116	0.75	0.46	0.57	370
117	0.62	0.20	0.30	313
118	0.80	0.57	0.66	874
119	0.47	0.17	0.25	293
120	0.00	0.00	0.00	200
121	0.77	0.42	0.55	463
122	0.48	0.10	0.17	119
123	1.00	0.01	0.02	256
124	0.91	0.66	0.76	195
125	0.38	0.09	0.15	138
126	0.83	0.48	0.61	376
127	0.33	0.03	0.06	122
128	0.20	0.02	0.04	252
129	0.49	0.15	0.23	144
130	0.36	0.05	0.09	150
131	0.30	0.01	0.03	210
132	0.67	0.22	0.33	361
133	0.95	0.48	0.64	453
134	0.88	0.71	0.79	124
135	0.14	0.01	0.02	91
136	0.73	0.26	0.38	128
137	0.59	0.33	0.42	218
138	0.74	0.11	0.19	243
139	0.46	0.17	0.25	149
140	0.75	0.36	0.48	318
141	0.33	0.09	0.14	159
142	0.64	0.33	0.44	274
143	0.88	0.65	0.75	362
144	0.62	0.17	0.27	118
145	0.70	0.35	0.46	164
146	0.56	0.25	0.34	461
147	0.68	0.36	0.48	159
148	0.34	0.11	0.17	166
149	0.99	0.41	0.58	346
150	0.72	0.06	0.11	350
151	0.97	0.55	0.70	55
152	0.82	0.46	0.59	387
153	0.50	0.09	0.15	150
154	0.63	0.09	0.15	281
155	0.32	0.04	0.08	202
156	0.82	0.62	0.70	130
157	0.35	0.04	0.08	245
158	0.94	0.58	0.72	177
159	0.52	0.25	0.34	130
160	0.51	0.12	0.20	336
161	0.90	0.54	0.68	220
162	0.28	0.04	0.08	229
163	0.92	0.37	0.53	316
164	0.76	0.33	0.46	283
165	0.64	0.26	0.37	197
166	0.62	0.32	0.42	101
167	0.46	0.14	0.22	231
168	0.56	0.20	0.29	370
169	0.43	0.17	0.25	258
170	0.37	0.07	0.12	101
171	0.41	0.24	0.30	89
172	0.58	0.33	0.42	193
173	0.44	0.19	0.26	309
174	0.55	0.09	0.16	172
175	0.95	0.66	0.78	95
176	0.96	0.51	0.67	346
177	0.95	0.40	0.56	322
178	0.66	0.44	0.52	232

179	0.31	0.04	0.07	125
180	0.63	0.23	0.34	145
181	0.36	0.05	0.09	77
182	0.21	0.03	0.06	182
183	0.61	0.28	0.38	257
184	0.27	0.03	0.05	216
185	0.34	0.05	0.09	242
186	0.40	0.10	0.16	165
187	0.77	0.52	0.62	263
188	0.44	0.10	0.16	174
189	0.78	0.32	0.45	136
190	0.94	0.43	0.59	202
191	0.41	0.10	0.17	134
192	0.72	0.33	0.46	230
193	0.43	0.17	0.24	90
194	0.59	0.40	0.48	185
195	0.39	0.04	0.08	156
196	0.50	0.04	0.07	160
197	0.48	0.05	0.08	266
198	0.35	0.04	0.07	284
199	0.44	0.03	0.05	145
200	0.94	0.62	0.75	212
201	0.71	0.19	0.30	317
202	0.81	0.49	0.61	427
203	0.36	0.07	0.12	232
204	0.60	0.24	0.34	217
205	0.51	0.34	0.41	527
206	0.11	0.01	0.02	124
207	0.42	0.08	0.13	103
208	0.92	0.44	0.59	287
209	0.37	0.08	0.13	193
210	0.79	0.28	0.41	220
211	0.83	0.11	0.19	140
212	0.12	0.01	0.02	161
213	0.58	0.25	0.35	72
214	0.65	0.37	0.47	396
215	0.94	0.22	0.35	134
216	0.55	0.07	0.12	400
217	0.47	0.19	0.27	75
218	0.97	0.65	0.78	219
219	0.83	0.31	0.45	210
220	0.96	0.52	0.67	298
221	0.97	0.53	0.68	266
222	0.77	0.34	0.47	290
223	0.17	0.01	0.01	128
224	0.79	0.40	0.53	159
225	0.59	0.20	0.30	164
226	0.60	0.31	0.41	144
227	0.58	0.29	0.39	276
228	0.16	0.01	0.02	235
229	0.33	0.00	0.01	216
230	0.42	0.12	0.19	228
231	0.69	0.42	0.52	64
232	0.64	0.07	0.12	103
233	0.74	0.24	0.36	216
234	0.86	0.05	0.10	116
235	0.52	0.29	0.37	77
236	0.97	0.58	0.73	67
237	0.50	0.05	0.08	218
238	0.38	0.08	0.13	139
239	0.00	0.00	0.00	94
240	0.56	0.23	0.33	77
241	0.57	0.08	0.14	167
242	0.86	0.28	0.42	86
243	0.25	0.03	0.06	58
244	0.59	0.19	0.29	269
245	0.24	0.04	0.08	112
246	0.96	0.67	0.79	255
247	0.44	0.21	0.28	58
248	0.00	0.00	0.00	81
249	0.11	0.01	0.01	131
250	0.42	0.14	0.21	93
251	0.67	0.23	0.34	154
252	0.75	0.02	0.05	129
253	0.69	0.33	0.44	83
254	0.34	0.05	0.09	191
255	0.18	0.03	0.05	219

256	0.43	0.05	0.08	130
257	0.53	0.23	0.32	93
258	0.68	0.40	0.50	217
259	0.38	0.09	0.15	141
260	0.95	0.14	0.24	143
261	0.62	0.11	0.18	219
262	0.60	0.23	0.34	107
263	0.44	0.20	0.27	236
264	0.33	0.16	0.21	119
265	0.58	0.15	0.24	72
266	0.00	0.00	0.00	70
267	0.32	0.08	0.13	107
268	0.72	0.37	0.49	169
269	0.31	0.09	0.13	129
270	0.77	0.48	0.59	159
271	0.87	0.25	0.39	190
272	0.64	0.19	0.29	248
273	0.91	0.59	0.71	264
274	0.90	0.58	0.71	105
275	0.60	0.06	0.11	104
276	0.08	0.01	0.02	115
277	0.88	0.55	0.67	170
278	0.70	0.21	0.32	145
279	0.92	0.48	0.63	230
280	0.60	0.38	0.46	80
281	0.66	0.48	0.55	217
282	0.71	0.37	0.48	175
283	0.32	0.06	0.09	269
284	0.67	0.22	0.33	74
285	0.87	0.42	0.56	206
286	0.91	0.54	0.68	227
287	0.88	0.28	0.42	130
288	0.41	0.05	0.10	129
289	0.33	0.01	0.02	80
290	0.24	0.07	0.11	99
291	0.82	0.25	0.39	208
292	0.43	0.04	0.08	67
293	0.86	0.44	0.58	109
294	0.45	0.20	0.28	140
295	0.24	0.10	0.14	241
296	0.33	0.10	0.15	72
297	0.38	0.06	0.10	107
298	0.89	0.41	0.56	61
299	0.97	0.40	0.57	77
300	0.24	0.05	0.09	111
301	0.00	0.00	0.00	126
302	0.00	0.00	0.00	73
303	0.62	0.32	0.42	176
304	0.97	0.62	0.75	230
305	0.98	0.51	0.67	156
306	0.56	0.34	0.42	146
307	0.30	0.06	0.10	98
308	0.00	0.00	0.00	78
309	0.88	0.07	0.14	94
310	0.79	0.28	0.41	162
311	0.82	0.39	0.53	116
312	0.54	0.23	0.32	57
313	0.75	0.05	0.09	65
314	0.51	0.30	0.38	138
315	0.56	0.17	0.27	195
316	0.47	0.23	0.31	69
317	0.45	0.10	0.16	134
318	0.52	0.32	0.39	148
319	0.85	0.37	0.52	161
320	0.26	0.12	0.17	104
321	0.85	0.51	0.64	156
322	0.60	0.32	0.42	134
323	0.60	0.37	0.45	232
324	0.43	0.13	0.20	92
325	0.46	0.19	0.27	197
326	0.14	0.02	0.03	126
327	0.10	0.01	0.02	115
328	0.99	0.49	0.66	198
329	0.68	0.26	0.37	125
330	0.82	0.11	0.20	81
331	0.50	0.10	0.16	94
332	0.60	0.05	0.10	56

333	0.19	0.02	0.04	260
334	0.38	0.05	0.09	60
335	0.53	0.07	0.13	110
336	0.67	0.41	0.51	71
337	0.17	0.03	0.05	66
338	0.52	0.33	0.40	150
339	0.00	0.00	0.00	54
340	0.85	0.46	0.59	195
341	0.83	0.25	0.39	79
342	0.50	0.18	0.27	38
343	0.65	0.35	0.45	43
344	0.57	0.19	0.29	68
345	0.70	0.32	0.43	73
346	0.08	0.01	0.02	116
347	0.89	0.30	0.45	111
348	0.38	0.10	0.15	63
349	0.86	0.53	0.65	104
350	0.61	0.39	0.47	44
351	0.69	0.23	0.34	40
352	0.98	0.32	0.49	136
353	0.38	0.15	0.21	54
354	0.56	0.04	0.07	134
355	0.68	0.30	0.42	120
356	0.51	0.18	0.27	228
357	0.72	0.27	0.39	269
358	0.75	0.30	0.43	80
359	0.83	0.36	0.50	140
360	0.38	0.12	0.18	125
361	0.93	0.51	0.66	169
362	0.11	0.04	0.05	56
363	0.94	0.54	0.69	154
364	0.71	0.09	0.15	58
365	0.20	0.07	0.10	71
366	1.00	0.50	0.67	54
367	0.37	0.06	0.10	116
368	0.00	0.00	0.00	54
369	0.00	0.00	0.00	71
370	0.00	0.00	0.00	61
371	0.50	0.04	0.08	71
372	0.68	0.33	0.44	52
373	0.77	0.29	0.43	150
374	0.43	0.10	0.16	93
375	0.25	0.03	0.05	67
376	0.00	0.00	0.00	76
377	0.74	0.19	0.30	106
378	0.17	0.01	0.02	86
379	0.50	0.07	0.12	14
380	1.00	0.30	0.47	122
381	0.36	0.04	0.07	104
382	0.35	0.09	0.14	66
383	0.56	0.25	0.35	110
384	0.33	0.01	0.01	155
385	0.71	0.10	0.18	50
386	0.17	0.05	0.07	64
387	0.62	0.05	0.10	93
388	0.64	0.16	0.25	102
389	0.14	0.01	0.02	108
390	0.96	0.54	0.69	178
391	0.58	0.12	0.20	115
392	0.88	0.33	0.48	42
393	0.00	0.00	0.00	134
394	0.43	0.03	0.05	112
395	0.58	0.19	0.29	176
396	0.67	0.06	0.12	125
397	0.82	0.24	0.37	224
398	0.93	0.44	0.60	63
399	1.00	0.02	0.03	59
400	0.52	0.25	0.34	63
401	0.57	0.16	0.25	98
402	0.61	0.14	0.23	162
403	0.46	0.13	0.21	83
404	0.83	0.79	0.81	19
405	0.44	0.08	0.13	92
406	0.88	0.17	0.29	41
407	0.65	0.30	0.41	43
408	0.82	0.29	0.43	160
409	0.21	0.10	0.14	50

409	0.21	0.10	0.14	50
410	0.00	0.00	0.00	19
411	0.32	0.07	0.11	175
412	0.25	0.03	0.05	72
413	0.75	0.06	0.12	95
414	0.14	0.02	0.04	97
415	0.36	0.08	0.14	48
416	0.48	0.27	0.34	83
417	0.25	0.03	0.05	40
418	0.55	0.13	0.21	91
419	0.50	0.19	0.27	90
420	0.39	0.24	0.30	37
421	0.09	0.02	0.03	66
422	0.65	0.38	0.48	73
423	0.52	0.20	0.29	56
424	0.93	0.76	0.83	33
425	0.00	0.00	0.00	76
426	0.30	0.04	0.07	81
427	0.99	0.47	0.64	150
428	0.95	0.66	0.78	29
429	0.99	0.34	0.51	389
430	0.66	0.26	0.38	167
431	0.61	0.09	0.16	123
432	0.55	0.31	0.39	39
433	0.32	0.15	0.20	82
434	1.00	0.52	0.68	66
435	0.62	0.34	0.44	93
436	0.64	0.29	0.40	87
437	0.12	0.01	0.02	86
438	0.72	0.40	0.52	104
439	0.71	0.10	0.18	100
440	0.00	0.00	0.00	141
441	0.43	0.25	0.31	110
442	0.43	0.12	0.19	123
443	0.45	0.07	0.12	71
444	0.60	0.06	0.10	109
445	0.42	0.17	0.24	48
446	0.36	0.16	0.22	76
447	0.46	0.16	0.24	38
448	0.69	0.47	0.56	81
449	0.66	0.22	0.33	132
450	0.46	0.22	0.30	81
451	0.94	0.21	0.34	76
452	0.00	0.00	0.00	44
453	0.00	0.00	0.00	44
454	0.88	0.43	0.58	70
455	0.59	0.08	0.15	155
456	0.58	0.16	0.25	43
457	0.55	0.15	0.24	72
458	0.50	0.08	0.14	62
459	0.75	0.17	0.28	69
460	0.00	0.00	0.00	119
461	0.79	0.14	0.24	79
462	0.54	0.15	0.23	47
463	0.33	0.04	0.07	104
464	0.72	0.29	0.42	106
465	0.92	0.17	0.29	64
466	0.58	0.24	0.34	173
467	0.86	0.28	0.42	107
468	0.75	0.12	0.21	126
469	0.00	0.00	0.00	114
470	0.95	0.72	0.82	140
471	0.88	0.19	0.31	79
472	0.39	0.24	0.30	143
473	0.75	0.25	0.37	158
474	0.50	0.04	0.07	138
475	0.00	0.00	0.00	59
476	0.53	0.18	0.27	88
477	0.88	0.48	0.62	176
478	1.00	0.71	0.83	24
479	0.14	0.01	0.02	92
480	0.83	0.38	0.52	100
481	0.54	0.20	0.30	103
482	0.50	0.19	0.27	74
483	0.80	0.43	0.56	105
484	0.00	0.00	0.00	83
485	0.00	0.00	0.00	82
486	0.50	0.13	0.20	71

```
        486       0.50      0.13      0.20        71
        487       0.41      0.12      0.19       120
        488       0.33      0.01      0.02       105
        489       0.79      0.25      0.38        87
        490       1.00      0.72      0.84        32
        491       0.00      0.00      0.00        69
        492       0.00      0.00      0.00        49
        493       0.00      0.00      0.00       117
        494       0.43      0.10      0.16        61
        495       0.99      0.39      0.56       344
        496       0.29      0.10      0.14        52
        497       0.59      0.15      0.23       137
        498       0.27      0.03      0.06        98
        499       0.92      0.14      0.24        79

  micro avg       0.74      0.32      0.44    173812
  macro avg       0.58      0.23      0.31    173812
weighted avg       0.68      0.32      0.42    173812
 samples avg       0.41      0.30      0.33    173812

Time taken to run this cell : 0:25:58.353092
```

Linear SVM(SGD Classifier) with OneVsResClassifier

In [85]:

```
start = datetime.now()
classifier = OneVsRestClassifier(SGDClassifier(loss='hinge', alpha=0.0001, penalty='l2'))
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 : 0.19264
Hamming loss  0.00300582
Micro-average quality numbers
Precision: 0.8629, Recall: 0.1609, F1-measure: 0.2712
Macro-average quality numbers
Precision: 0.2257, Recall: 0.0717, F1-measure: 0.0956
              precision    recall  f1-score   support

           0       0.96      0.51      0.67      5519
           1       0.65      0.17      0.26      8190
           2       0.86      0.27      0.41      6529
           3       0.81      0.31      0.45      3231
           4       0.88      0.30      0.45      6430
           5       0.84      0.26      0.39      2879
           6       0.92      0.43      0.58      5086
           7       0.91      0.47      0.62      4533
           8       0.62      0.12      0.20      3000
           9       0.83      0.41      0.55      2765
          10       0.00      0.00      0.00      3051
          11       0.85      0.24      0.38      3009
          12       0.75      0.12      0.20      2630
```

13	1.00	0.00	0.00	1426
14	0.95	0.45	0.61	2548
15	0.94	0.04	0.08	2371
16	0.66	0.25	0.36	873
17	0.91	0.54	0.68	2151
18	1.00	0.00	0.01	2204
19	0.70	0.43	0.54	831
20	0.82	0.34	0.48	1860
21	0.00	0.00	0.00	2023
22	0.00	0.00	0.00	1513
23	0.95	0.37	0.54	1207
24	0.00	0.00	0.00	506
25	0.82	0.24	0.38	425
26	0.79	0.04	0.08	793
27	0.00	0.00	0.00	1291
28	0.87	0.25	0.39	1208
29	0.00	0.00	0.00	406
30	0.83	0.10	0.18	504
31	0.00	0.00	0.00	732
32	0.79	0.03	0.07	441
33	0.00	0.00	0.00	1645
34	0.76	0.02	0.04	1058
35	0.90	0.25	0.39	946
36	0.78	0.03	0.06	644
37	0.98	0.67	0.79	136
38	0.53	0.03	0.06	570
39	0.93	0.09	0.16	766
40	0.00	0.00	0.00	1132
41	0.00	0.00	0.00	174
42	0.76	0.36	0.49	210
43	0.84	0.38	0.52	433
44	1.00	0.00	0.00	626
45	0.00	0.00	0.00	852
46	0.00	0.00	0.00	534
47	0.00	0.00	0.00	350
48	0.80	0.39	0.52	496
49	0.81	0.56	0.66	785
50	0.00	0.00	0.00	475
51	0.00	0.00	0.00	305
52	0.00	0.00	0.00	251
53	0.00	0.00	0.00	914
54	0.00	0.00	0.00	728
55	0.00	0.00	0.00	258
56	0.00	0.00	0.00	821
57	0.00	0.00	0.00	541
58	1.00	0.02	0.04	748
59	0.94	0.59	0.72	724
60	0.00	0.00	0.00	660
61	0.91	0.17	0.29	235
62	0.93	0.64	0.76	718
63	0.88	0.52	0.65	468
64	0.00	0.00	0.00	191
65	0.00	0.00	0.00	429
66	0.00	0.00	0.00	415
67	0.78	0.41	0.53	274
68	0.84	0.43	0.57	510
69	0.00	0.00	0.00	466
70	0.00	0.00	0.00	305
71	0.00	0.00	0.00	247
72	0.84	0.33	0.47	401
73	0.98	0.72	0.83	86
74	0.87	0.23	0.36	120
75	0.89	0.72	0.79	129
76	0.00	0.00	0.00	473
77	0.00	0.00	0.00	143
78	0.80	0.31	0.44	347
79	0.00	0.00	0.00	479
80	0.75	0.01	0.02	279
81	0.00	0.00	0.00	461
82	0.00	0.00	0.00	298
83	0.00	0.00	0.00	396
84	0.00	0.00	0.00	184
85	0.00	0.00	0.00	573
86	0.00	0.00	0.00	325
87	0.00	0.00	0.00	273
88	0.00	0.00	0.00	135
89	0.00	0.00	0.00	232

90	0.00	0.00	0.00	409
91	0.00	0.00	0.00	420
92	0.80	0.24	0.36	408
93	0.76	0.14	0.24	241
94	0.00	0.00	0.00	211
95	0.00	0.00	0.00	277
96	0.00	0.00	0.00	410
97	0.91	0.02	0.04	501
98	0.74	0.63	0.68	136
99	0.00	0.00	0.00	239
100	0.00	0.00	0.00	324
101	0.96	0.44	0.60	277
102	0.95	0.59	0.73	613
103	0.00	0.00	0.00	157
104	0.00	0.00	0.00	295
105	0.87	0.16	0.28	334
106	0.00	0.00	0.00	335
107	0.00	0.00	0.00	389
108	0.00	0.00	0.00	251
109	0.00	0.00	0.00	317
110	0.00	0.00	0.00	187
111	0.00	0.00	0.00	140
112	0.00	0.00	0.00	154
113	0.00	0.00	0.00	332
114	0.00	0.00	0.00	323
115	0.00	0.00	0.00	344
116	0.00	0.00	0.00	370
117	0.00	0.00	0.00	313
118	0.00	0.00	0.00	874
119	0.00	0.00	0.00	293
120	0.00	0.00	0.00	200
121	0.94	0.03	0.06	463
122	0.00	0.00	0.00	119
123	0.00	0.00	0.00	256
124	0.89	0.72	0.80	195
125	0.00	0.00	0.00	138
126	1.00	0.01	0.02	376
127	0.00	0.00	0.00	122
128	0.00	0.00	0.00	252
129	0.00	0.00	0.00	144
130	0.00	0.00	0.00	150
131	0.00	0.00	0.00	210
132	0.00	0.00	0.00	361
133	0.97	0.32	0.48	453
134	0.88	0.73	0.80	124
135	0.00	0.00	0.00	91
136	0.00	0.00	0.00	128
137	0.74	0.06	0.12	218
138	0.00	0.00	0.00	243
139	0.00	0.00	0.00	149
140	0.00	0.00	0.00	318
141	0.00	0.00	0.00	159
142	0.00	0.00	0.00	274
143	0.91	0.34	0.49	362
144	0.00	0.00	0.00	118
145	0.00	0.00	0.00	164
146	0.00	0.00	0.00	461
147	0.57	0.05	0.09	159
148	0.00	0.00	0.00	166
149	0.99	0.31	0.47	346
150	0.00	0.00	0.00	350
151	0.93	0.51	0.66	55
152	0.85	0.03	0.06	387
153	0.00	0.00	0.00	150
154	0.00	0.00	0.00	281
155	0.00	0.00	0.00	202
156	0.82	0.62	0.70	130
157	0.00	0.00	0.00	245
158	0.89	0.56	0.69	177
159	1.00	0.01	0.02	130
160	0.00	0.00	0.00	336
161	0.95	0.45	0.61	220
162	0.00	0.00	0.00	229
163	0.91	0.29	0.44	316
164	0.00	0.00	0.00	283
165	0.00	0.00	0.00	197
166	0.00	0.00	0.00	101

167	0.00	0.00	0.00	231
168	0.00	0.00	0.00	370
169	0.00	0.00	0.00	258
170	0.00	0.00	0.00	101
171	0.00	0.00	0.00	89
172	0.00	0.00	0.00	193
173	0.00	0.00	0.00	309
174	0.00	0.00	0.00	172
175	0.94	0.77	0.84	95
176	0.96	0.27	0.43	346
177	0.98	0.15	0.25	322
178	1.00	0.00	0.01	232
179	0.00	0.00	0.00	125
180	0.42	0.03	0.06	145
181	0.00	0.00	0.00	77
182	0.00	0.00	0.00	182
183	0.00	0.00	0.00	257
184	0.00	0.00	0.00	216
185	0.00	0.00	0.00	242
186	0.00	0.00	0.00	165
187	0.83	0.19	0.31	263
188	0.00	0.00	0.00	174
189	0.00	0.00	0.00	136
190	0.96	0.22	0.35	202
191	0.00	0.00	0.00	134
192	1.00	0.01	0.03	230
193	0.00	0.00	0.00	90
194	0.00	0.00	0.00	185
195	0.00	0.00	0.00	156
196	0.00	0.00	0.00	160
197	0.00	0.00	0.00	266
198	0.00	0.00	0.00	284
199	0.00	0.00	0.00	145
200	0.94	0.56	0.70	212
201	0.00	0.00	0.00	317
202	0.00	0.00	0.00	427
203	0.00	0.00	0.00	232
204	0.00	0.00	0.00	217
205	0.00	0.00	0.00	527
206	0.00	0.00	0.00	124
207	0.00	0.00	0.00	103
208	1.00	0.01	0.03	287
209	0.00	0.00	0.00	193
210	0.00	0.00	0.00	220
211	0.00	0.00	0.00	140
212	0.00	0.00	0.00	161
213	0.00	0.00	0.00	72
214	0.00	0.00	0.00	396
215	0.00	0.00	0.00	134
216	0.00	0.00	0.00	400
217	0.00	0.00	0.00	75
218	0.98	0.63	0.77	219
219	0.00	0.00	0.00	210
220	1.00	0.04	0.07	298
221	0.97	0.41	0.58	266
222	0.00	0.00	0.00	290
223	0.00	0.00	0.00	128
224	1.00	0.01	0.02	159
225	0.00	0.00	0.00	164
226	0.00	0.00	0.00	144
227	0.00	0.00	0.00	276
228	0.00	0.00	0.00	235
229	0.00	0.00	0.00	216
230	0.00	0.00	0.00	228
231	0.68	0.33	0.44	64
232	0.00	0.00	0.00	103
233	1.00	0.00	0.01	216
234	0.00	0.00	0.00	116
235	1.00	0.01	0.03	77
236	0.98	0.60	0.74	67
237	0.00	0.00	0.00	218
238	0.00	0.00	0.00	139
239	0.00	0.00	0.00	94
240	0.00	0.00	0.00	77
241	0.00	0.00	0.00	167
242	0.82	0.16	0.27	86
243	0.00	0.00	0.00	58

244	0.00	0.00	0.00	269
245	0.00	0.00	0.00	112
246	0.96	0.53	0.68	255
247	0.00	0.00	0.00	58
248	0.00	0.00	0.00	81
249	0.00	0.00	0.00	131
250	0.00	0.00	0.00	93
251	0.00	0.00	0.00	154
252	0.00	0.00	0.00	129
253	0.50	0.01	0.02	83
254	0.00	0.00	0.00	191
255	0.00	0.00	0.00	219
256	0.00	0.00	0.00	130
257	0.00	0.00	0.00	93
258	1.00	0.02	0.04	217
259	0.00	0.00	0.00	141
260	1.00	0.03	0.05	143
261	0.00	0.00	0.00	219
262	0.00	0.00	0.00	107
263	0.00	0.00	0.00	236
264	0.00	0.00	0.00	119
265	0.00	0.00	0.00	72
266	0.00	0.00	0.00	70
267	0.00	0.00	0.00	107
268	0.00	0.00	0.00	169
269	0.00	0.00	0.00	129
270	0.00	0.00	0.00	159
271	0.00	0.00	0.00	190
272	0.00	0.00	0.00	248
273	0.91	0.24	0.38	264
274	0.90	0.51	0.65	105
275	0.00	0.00	0.00	104
276	0.00	0.00	0.00	115
277	0.87	0.32	0.47	170
278	0.00	0.00	0.00	145
279	0.91	0.18	0.30	230
280	0.00	0.00	0.00	80
281	1.00	0.06	0.11	217
282	1.00	0.03	0.06	175
283	0.00	0.00	0.00	269
284	0.00	0.00	0.00	74
285	0.00	0.00	0.00	206
286	0.93	0.18	0.30	227
287	0.00	0.00	0.00	130
288	0.00	0.00	0.00	129
289	0.00	0.00	0.00	80
290	0.00	0.00	0.00	99
291	0.00	0.00	0.00	208
292	0.00	0.00	0.00	67
293	0.00	0.00	0.00	109
294	0.00	0.00	0.00	140
295	0.00	0.00	0.00	241
296	0.00	0.00	0.00	72
297	0.00	0.00	0.00	107
298	0.00	0.00	0.00	61
299	0.00	0.00	0.00	77
300	0.00	0.00	0.00	111
301	0.00	0.00	0.00	126
302	0.00	0.00	0.00	73
303	0.00	0.00	0.00	176
304	0.98	0.45	0.62	230
305	0.97	0.40	0.56	156
306	0.00	0.00	0.00	146
307	0.00	0.00	0.00	98
308	0.00	0.00	0.00	78
309	0.00	0.00	0.00	94
310	0.00	0.00	0.00	162
311	0.73	0.07	0.13	116
312	0.00	0.00	0.00	57
313	0.00	0.00	0.00	65
314	0.00	0.00	0.00	138
315	0.00	0.00	0.00	195
316	0.00	0.00	0.00	69
317	0.00	0.00	0.00	134
318	0.00	0.00	0.00	148
319	1.00	0.01	0.01	161
320	0.00	0.00	0.00	104

321	1.00	0.01	0.01	156
322	0.00	0.00	0.00	134
323	0.00	0.00	0.00	232
324	0.00	0.00	0.00	92
325	0.00	0.00	0.00	197
326	0.00	0.00	0.00	126
327	0.00	0.00	0.00	115
328	1.00	0.31	0.48	198
329	0.00	0.00	0.00	125
330	0.00	0.00	0.00	81
331	0.00	0.00	0.00	94
332	0.00	0.00	0.00	56
333	0.00	0.00	0.00	260
334	0.00	0.00	0.00	60
335	0.00	0.00	0.00	110
336	0.00	0.00	0.00	71
337	0.00	0.00	0.00	66
338	0.00	0.00	0.00	150
339	0.00	0.00	0.00	54
340	0.00	0.00	0.00	195
341	0.00	0.00	0.00	79
342	0.00	0.00	0.00	38
343	0.00	0.00	0.00	43
344	0.00	0.00	0.00	68
345	0.00	0.00	0.00	73
346	0.00	0.00	0.00	116
347	0.00	0.00	0.00	111
348	0.00	0.00	0.00	63
349	0.92	0.12	0.21	104
350	0.00	0.00	0.00	44
351	0.00	0.00	0.00	40
352	1.00	0.04	0.08	136
353	0.00	0.00	0.00	54
354	0.00	0.00	0.00	134
355	0.00	0.00	0.00	120
356	0.00	0.00	0.00	228
357	0.00	0.00	0.00	269
358	0.00	0.00	0.00	80
359	0.00	0.00	0.00	140
360	0.00	0.00	0.00	125
361	1.00	0.04	0.07	169
362	0.00	0.00	0.00	56
363	0.96	0.31	0.46	154
364	0.00	0.00	0.00	58
365	0.00	0.00	0.00	71
366	1.00	0.57	0.73	54
367	0.00	0.00	0.00	116
368	0.00	0.00	0.00	54
369	0.00	0.00	0.00	71
370	0.00	0.00	0.00	61
371	0.00	0.00	0.00	71
372	0.00	0.00	0.00	52
373	0.00	0.00	0.00	150
374	0.00	0.00	0.00	93
375	0.00	0.00	0.00	67
376	0.00	0.00	0.00	76
377	0.00	0.00	0.00	106
378	0.00	0.00	0.00	86
379	0.00	0.00	0.00	14
380	1.00	0.01	0.02	122
381	0.00	0.00	0.00	104
382	0.00	0.00	0.00	66
383	0.00	0.00	0.00	110
384	0.00	0.00	0.00	155
385	0.00	0.00	0.00	50
386	0.00	0.00	0.00	64
387	0.00	0.00	0.00	93
388	0.00	0.00	0.00	102
389	0.00	0.00	0.00	108
390	0.96	0.31	0.47	178
391	0.00	0.00	0.00	115
392	1.00	0.02	0.05	42
393	0.00	0.00	0.00	134
394	0.00	0.00	0.00	112
395	0.00	0.00	0.00	176
396	0.00	0.00	0.00	125
397	0.00	0.00	0.00	224

398	1.00	0.05	0.09	63
399	0.00	0.00	0.00	59
400	0.00	0.00	0.00	63
401	0.00	0.00	0.00	98
402	0.00	0.00	0.00	162
403	0.00	0.00	0.00	83
404	0.73	0.84	0.78	19
405	0.00	0.00	0.00	92
406	0.00	0.00	0.00	41
407	0.00	0.00	0.00	43
408	0.00	0.00	0.00	160
409	0.00	0.00	0.00	50
410	0.00	0.00	0.00	19
411	0.00	0.00	0.00	175
412	0.00	0.00	0.00	72
413	0.00	0.00	0.00	95
414	0.00	0.00	0.00	97
415	0.00	0.00	0.00	48
416	0.00	0.00	0.00	83
417	0.00	0.00	0.00	40
418	0.00	0.00	0.00	91
419	0.00	0.00	0.00	90
420	0.00	0.00	0.00	37
421	0.00	0.00	0.00	66
422	0.00	0.00	0.00	73
423	0.00	0.00	0.00	56
424	0.93	0.82	0.87	33
425	0.00	0.00	0.00	76
426	0.00	0.00	0.00	81
427	1.00	0.18	0.31	150
428	0.95	0.66	0.78	29
429	0.00	0.00	0.00	389
430	0.00	0.00	0.00	167
431	0.00	0.00	0.00	123
432	0.00	0.00	0.00	39
433	0.00	0.00	0.00	82
434	1.00	0.50	0.67	66
435	0.00	0.00	0.00	93
436	0.00	0.00	0.00	87
437	0.00	0.00	0.00	86
438	0.00	0.00	0.00	104
439	0.00	0.00	0.00	100
440	0.00	0.00	0.00	141
441	0.00	0.00	0.00	110
442	0.00	0.00	0.00	123
443	0.00	0.00	0.00	71
444	0.00	0.00	0.00	109
445	0.00	0.00	0.00	48
446	0.00	0.00	0.00	76
447	0.00	0.00	0.00	38
448	0.00	0.00	0.00	81
449	0.00	0.00	0.00	132
450	0.00	0.00	0.00	81
451	0.00	0.00	0.00	76
452	0.00	0.00	0.00	44
453	0.00	0.00	0.00	44
454	0.75	0.09	0.15	70
455	0.00	0.00	0.00	155
456	0.00	0.00	0.00	43
457	0.00	0.00	0.00	72
458	0.00	0.00	0.00	62
459	0.00	0.00	0.00	69
460	0.00	0.00	0.00	119
461	0.00	0.00	0.00	79
462	0.00	0.00	0.00	47
463	0.00	0.00	0.00	104
464	0.00	0.00	0.00	106
465	0.00	0.00	0.00	64
466	0.00	0.00	0.00	173
467	0.00	0.00	0.00	107
468	0.00	0.00	0.00	126
469	0.00	0.00	0.00	114
470	1.00	0.28	0.44	140
471	0.00	0.00	0.00	79
472	0.00	0.00	0.00	143
473	0.00	0.00	0.00	158
474	0.00	0.00	0.00	138

```
         475       0.00      0.00      0.00        59
         476       0.00      0.00      0.00        88
         477       0.00      0.00      0.00       176
         478       0.94      0.71      0.81        24
         479       0.00      0.00      0.00        92
         480       0.00      0.00      0.00       100
         481       0.00      0.00      0.00       103
         482       0.00      0.00      0.00        74
         483       0.00      0.00      0.00       105
         484       0.00      0.00      0.00        83
         485       0.00      0.00      0.00        82
         486       0.00      0.00      0.00        71
         487       0.00      0.00      0.00       120
         488       0.00      0.00      0.00       105
         489       0.00      0.00      0.00        87
         490       1.00      0.72      0.84        32
         491       0.00      0.00      0.00        69
         492       0.00      0.00      0.00        49
         493       0.00      0.00      0.00       117
         494       0.00      0.00      0.00        61
         495       0.00      0.00      0.00       344
         496       0.00      0.00      0.00        52
         497       0.00      0.00      0.00       137
         498       0.00      0.00      0.00        98
         499       0.00      0.00      0.00        79

   micro avg       0.86      0.16      0.27    173812
   macro avg       0.23      0.07      0.10    173812
weighted avg       0.49      0.16      0.23    173812
 samples avg       0.25      0.16      0.18    173812

Time taken to run this cell : 0:07:31.119153
```

In [2]:

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["S.No.","Model", "Hyperparameter", "Regularization", "F1 micro", "F1 macro"]

x.add_row(["1.","Logistic Regression", 1, "L2", 0.444, 0.314])
x.add_row(["2.","SGD Classifier with log loss", 0.00001, "L2",0.354, 0.1903])
x.add_row(["3.","SGD Classifier with hinge loss", 0.0001, "L2", 0.271, 0.095])

print(x)
```

```
+--------+--------------------------------+----------------+----------------+----------+----------+
| S.No.  |             Model              | Hyperparameter | Regularization | F1 micro | F1 macro |
+--------+--------------------------------+----------------+----------------+----------+----------+
|   1.   |      Logistic Regression       |       1        |       L2       |  0.444   |  0.314   |
|   2.   |  SGD Classifier with log loss  |     1e-05      |       L2       |  0.354   |  0.1903  |
|   3.   | SGD Classifier with hinge loss |     0.0001     |       L2       |  0.271   |  0.095   |
+--------+--------------------------------+----------------+----------------+----------+----------+
```

Conclusion

1. Reading the data from the csv file and then using the SQLite we are storing into a database.
2. Checking for the dupliacate rows and found the around **30%** of the data is being repeated and so we removed them.
3. Analyze the tags and found that most of the questions have **3Tags or 2Tags**.
4. Plotted the distribution/frequency of the tags and found the around **153tags occured 1000times and 14Tags occured 10,000times**.
5. Top 3 tags are **C#, Java and php**.
6. Preprocessing of the text data(Questions).
7. Converting tags into desired target i.e Multilabel using countvectorizer.
8. Selecting the top 500tags as it contains 90% of the data and also to reduce the computation.
9. Featuring the text data using **Tfidf** vectorizer with ngram range (1,4)
10. Used Logistic Regression and SGD classifier with loss equal to log and hinge separately for building the model

In [ ]:

```

```