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
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
from tqdm import tqdm

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import train_test_split
```

Stack Overflow: Tag Prediction

1. Business Problem

1.1 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/

1.2 Source / useful links

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

1.3 Real World / Business Objectives and Constraints

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

2. Machine Learning problem

2.1 Data

2.1.1 Data Overview

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

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

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

Size of Train.csv - 6.75GB

Size of Test.csv - 2GB

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

The questions are randomized and contains a mix of verbose text sites as well as sites related to math and 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, should
not contain tabs '\t' or ampersands '&')
```

2.1.2 Example Data point

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

```
#include<
iostream>\n
#include<
stdlib.h>\n\n
using namespace std;\n\n
int main()\n
{\n
         int n,a[n],x,c,u[n],m[n],e[n][4];\n
         cout<<"Enter the number of variables";\n         cin>>n;\n\n
         cout<<"Enter the Lower, and Upper Limits of the variables";\n
         for(int y=1; y<n+1; y++)\n
         {\n
            cin>>m[y];\n
            cin>>u[y];\n
         }\n
         for(x=1; x<n+1; x++)\n
         {\n
            a[x] = (m[x] + u[x])/2;\n
         }\n
         c=(n*4)-4;\n
         for(int a1=1; a1<n+1; a1++)\n
         {\n\n
            e[a1][0] = m[a1];\n
            e[a1][1] = m[a1]+1;\n
            e[a1][2] = u[a1]-1;\n
            e[a1][3] = u[a1];\n
         }\n
         for(int i=1; i<n+1; i++)\n
         {\n
            for(int l=1; l<=i; l++)\n
            {\n
                if(l!=1)\n
                {\n
                    cout<<a[l]<<"\\t";\n
                }\n
            }\n
            for(int j=0; j<4; j++)\n
            {\n
                cout<<e[i][j];\n
                for(int k=0; k<n-(i+1); k++)\n
                {\n
                    cout<<a[k]<<"\\t";\n
                }\n
                cout<<"\\n";\n
            }\n
         }    \n\n
         system("PAUSE");\n
         return 0;    \n
}\n
```

\n\n

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
```

```
            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'

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

2.2.1 Type of Machine Learning Problem

It is a multi-label classification problem
Multi-label Classification: Multilabel classification assigns to each sample a set of target labels. This can be 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

2.2.2 Performance metric

Micro-Averaged F1-Score (Mean F Score) : The F1 score can be interpreted as a weighted average of the precision 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

3. Exploratory Data Analysis

3.1 Data Loading and Cleaning

3.1 Data Loading and Cleaning

3.1.1 Using Pandas with SQLite to Load the data

In [3]:

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

```
180000 rows
360000 rows
540000 rows
720000 rows
900000 rows
1080000 rows
1260000 rows
1440000 rows
1620000 rows
1800000 rows
1980000 rows
2160000 rows
2340000 rows
2520000 rows
2700000 rows
2880000 rows
3060000 rows
3240000 rows
3420000 rows
3600000 rows
3780000 rows
3960000 rows
4140000 rows
4320000 rows
4500000 rows
4680000 rows
4860000 rows
5040000 rows
5220000 rows
5400000 rows
5580000 rows
5760000 rows
5940000 rows
6120000 rows
Time taken to run this cell : 0:05:01.862037
```

3.1.2 Counting the number of rows

In [5]:

```
if os.path.isfile('train.db'):
    start = datetime.now()
    con = sqlite3.connect('train.db')
    num_rows = pd.read_sql_query("""SELECT count(*) FROM data""", con)
    #Always remember to close the database
    """When we used read_sql_query to run "SELECT count(*) FROM data" then we got a
    dataframe with column name count(*) and only one row with the total count of rows"""
    print("Number of rows in the database :","\n",num_rows['count(*)'].values[0])
    con.close()
```

```
    print("Time taken to count the number of rows :", datetime.now() - start)
else:
    print("Please download the train.db file from drive or run the above cell to genarate train.db file
")
```

```
Number of rows in the database :
 6034196
Time taken to count the number of rows : 0:01:14.279550
```

3.1.3 Checking for duplicates

In [2]:

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

```
operation started at  2019-02-22 21:59:30.226573
Time taken to run this cell : 0:30:31.578145
```

In [3]:

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

Out[3]:

	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 [6]:

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

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

In [7]:

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

Out[7]:

```
1    2656284
2    1272336
3     277575
4         90
5         25
```

```
6            5
Name: cnt_dup, dtype: int64
```

In [8]:

```
df_no_dup = df_no_dup.dropna()
#df_no_dup.iloc[777541:777549]
```

In [9]:

```
df_no_dup.to_csv('df_no_dup.csv',sep=',')
```

In [2]:

```
df_no_dup = pd.read_csv("df_no_dup.csv")
```

In [3]:

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

```
Time taken to run this cell : 0:00:03.032125
```

Out[3]:

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

In [4]:

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

Out[4]:

```
3    1206157
2    1111706
4     814996
1     568291
5     505158
Name: tag_count, dtype: int64
```

In [5]:

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

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

```
Time taken to run this cell : 0:02:15.046240
```

In [4]:

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

```
Time taken to run this cell : 0:00:03.996150
```

Out[4]:

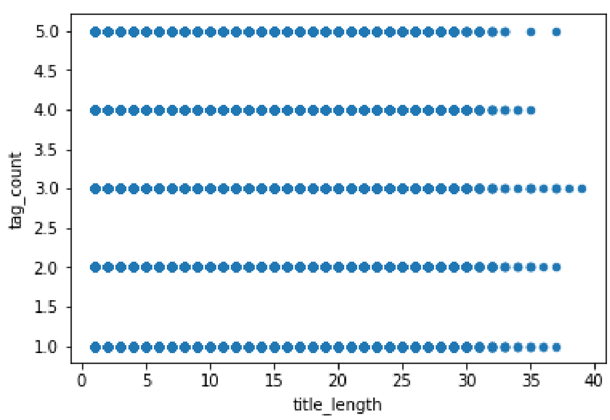
	Unnamed: 0	Title	Body	Tags	cnt_dup	tag_count	title_length
0	0	Implementing Boundary Value Analysis of S...	<pre><code>#include<iostream>\r\n#includ...	c++ c	1	2	16
1	1	Dynamic Datagrid Binding in Silverlight?	<p>I should do binding for datagrid dynamicall...	c# silverlight data-binding	1	3	9
2	2	Dynamic Datagrid Binding in Silverlight?	<p>I should do binding for datagrid dynamicall...	c# silverlight data-binding columns	1	4	9
3	3	java.lang.NoClassDefFoundError: javax/serv...	<p>I followed the guide in <a href="http://sta...	jsp jstl	1	2	6
4	4	java.sql.SQLException:[Microsoft][ODBC Dri...	I use the following code \r\n\r\n<pre><c...	java jdbc	2	2	10

In [7]:

```
df_no_dup.plot(kind='scatter',x='title_length',y='tag_count')
```

Out[7]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d7805216a0>
```

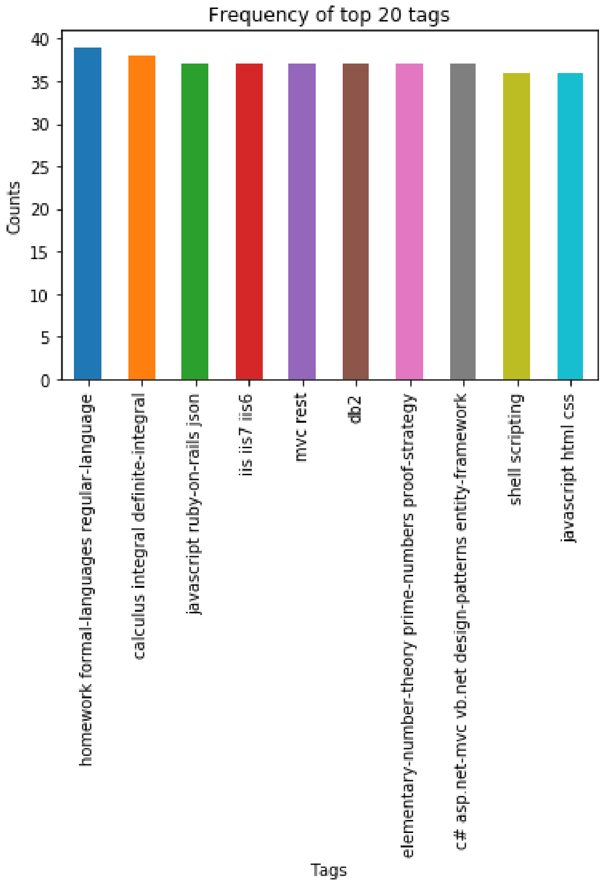


Number of tags doen't seems to match with length of the title

In [12]:

```
title_sorted = df_no_dup.sort_values(['title_length'], ascending=False)

i=np.arange(10)
title_sorted['title_length'].head(10).plot(kind='bar')
plt.title('Frequency of top 20 tags')
plt.xticks(i, title_sorted['Tags'])
plt.xlabel('Tags')
plt.ylabel('Counts')
plt.show()
```



Title length does not seems to match with tags. We can't say if the title length is this high then it is C# or its Java

In [17]:

```
from collections import Counter

splt_title = str(df_no_dup['Title']).split()
Counter = Counter(splt_title)
most_occur = Counter.most_common(20)

print(most_occur)
```

```
[('in', 18), ('to', 9), ('...', 8), ('is', 5), ('of', 4), ('not', 4), ('�', 4), ('on', 3), ('with', 3)
, ('the', 3), ('for', 3), ('a', 3), ('function', 3), ('symbol', 3), ('as', 3), ('1', 2), ('Dynamic', 2)
, ('Datagrid', 2), ('Binding', 2), ('Silverlight?', 2)]
```

We can't get much information from the most common words of title

3.2 Analysis of Tags

3.2.1 Total number of unique tags

In [6]:

```
# Importing & Initializing the "CountVectorizer" object, which
```

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

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

In [7]:

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

```
Number of data points : 4206307
Number of unique tags : 42048
```

In [8]:

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

3.2.3 Number of times a tag appeared

In [9]:

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

In [10]:

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

Out[10]:

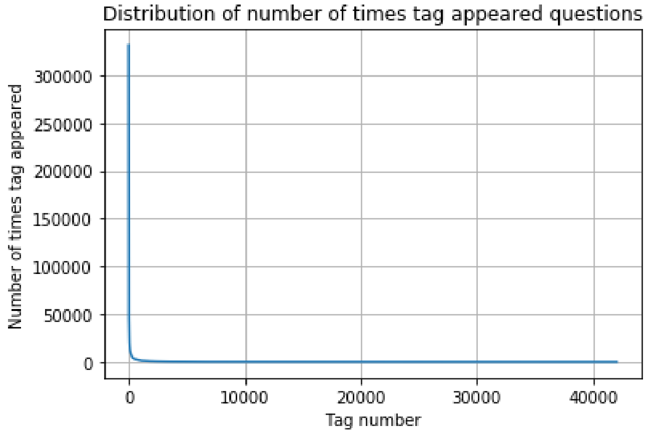
	Tags	Counts
0	.a	18
1	.app	37
2	.asp.net-mvc	1
3	.aspxauth	21
4	.bash-profile	138

In [11]:

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

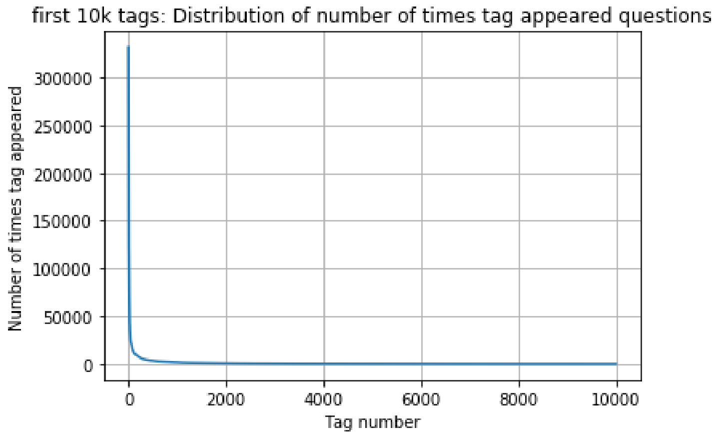
In [12]:

```
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()
```



In [13]:

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

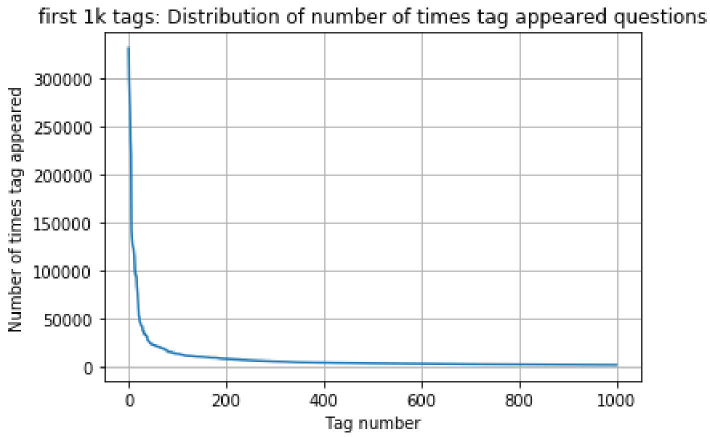



```
400 [331505  44829  22429  17728  13364  11162  10029   9148   8054   7151
   6466   5865   5370   4983   4526   4281   4144   3929   3750   3593
   3453   3299   3123   2986   2891   2738   2647   2527   2431   2331
   2259   2186   2097   2020   1959   1900   1828   1770   1723   1673
   1631   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 [14]:

```
plt.plot(tag_counts[0:1000])
plt.title('first 1k tags: Distribution of number of times tag appeared questions')
plt.grid()
plt.xlabel("Tag number")
plt.ylabel("Number of times tag appeared")
plt.show()
print(len(tag_counts[0:1000:5]), tag_counts[0:1000:5])
```

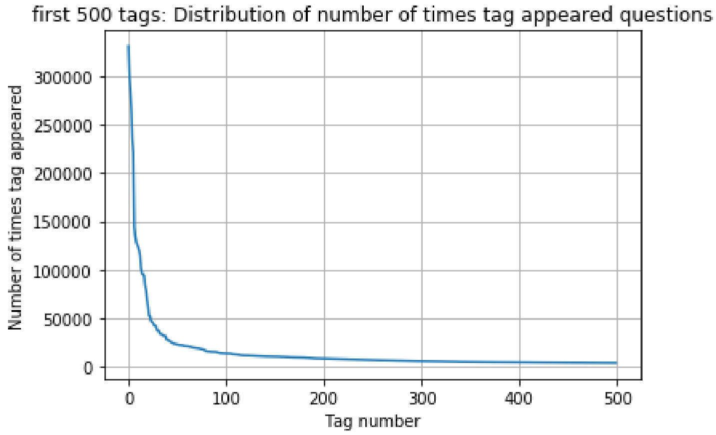



```
200 [331505 221533 122769  95160  62023  44829  37170  31897  26925  24537
  22429  21820  20957  19758  18905  17728  15533  15097  14884  13703
  13364  13157  12407  11658  11228  11162  10863  10600  10350  10224
  10029   9884   9719   9411   9252   9148   9040   8617   8361   8163
   8054   7867   7702   7564   7274   7151   7052   6847   6656   6553
   6466   6291   6183   6093   5971   5865   5760   5577   5490   5411
   5370   5283   5207   5107   5066   4983   4891   4785   4658   4549
   4526   4487   4429   4335   4310   4281   4239   4228   4195   4159
   4144   4088   4050   4002   3957   3929   3874   3849   3818   3797
   3750   3703   3685   3658   3615   3593   3564   3521   3505   3483
   3453   3427   3396   3363   3326   3299   3272   3232   3196   3168
   3123   3094   3073   3050   3012   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]
```

In [15]:

```
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])
```



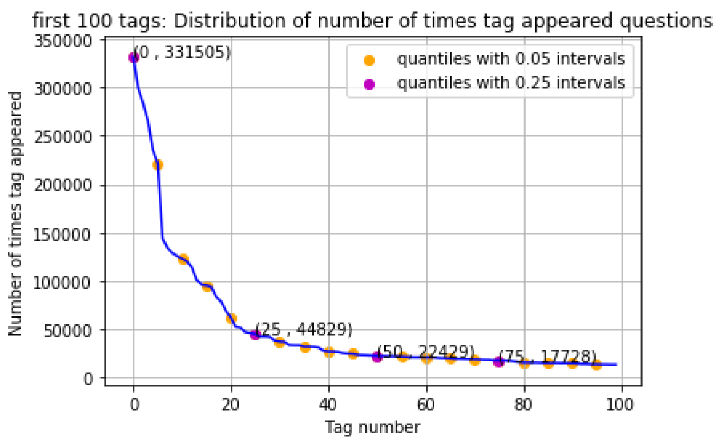

```
100 [331505 221533 122769  95160  62023  44829  37170  31897  26925  24537
  22429  21820  20957  19758  18905  17728  15533  15097  14884  13703
  13364  13157  12407  11658  11228  11162  10863  10600  10350  10224
  10029   9884   9719   9411   9252   9148   9040   8617   8361   8163
   8054   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 [16]:

```
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 inter
vals")
# 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 interv
als")

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]
```

In [17]:

```
# 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
```

Observations:

1. There are total 153 tags which are used more than 10000 times.
2. 14 tags are used more than 100000 times.
3. Most frequent tag (i.e. c#) is used 331505 times.
4. Since some tags occur much more frequenctly than others, Micro-averaged F1-score is the appropriate metric for this probelm.

3.2.4 Tags Per Question

In [18]:

```
#Storing the count of tag in each question in list 'tag_count'
tag_quest_count = tag_dtm.sum(axis=1).tolist()
#Converting each value in the 'tag_quest_count' to integer.
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 4206307 datapoints.
[3, 4, 2, 2, 3]
```

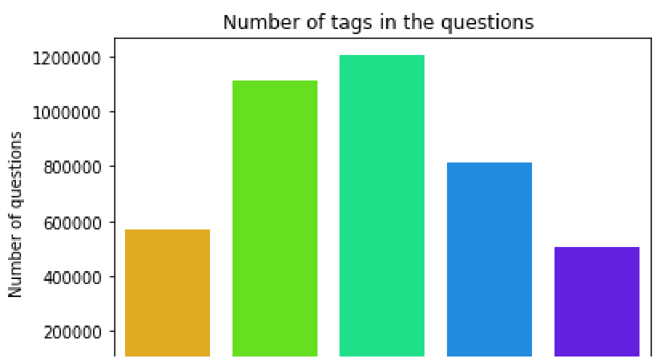
In [19]:

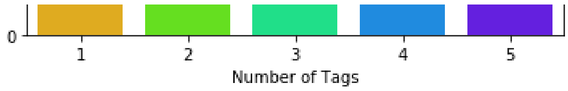
```
print( "Maximum number of tags per question: %d"%max(tag_quest_count))
print( "Minimum number of tags per question: %d"%min(tag_quest_count))
print( "Avg. number of tags per question: %f"% ((sum(tag_quest_count)*1.0)/len(tag_quest_count)))
```

```
Maximum number of tags per question: 5
Minimum number of tags per question: 1
Avg. number of tags per question: 2.899443
```

In [20]:

```
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

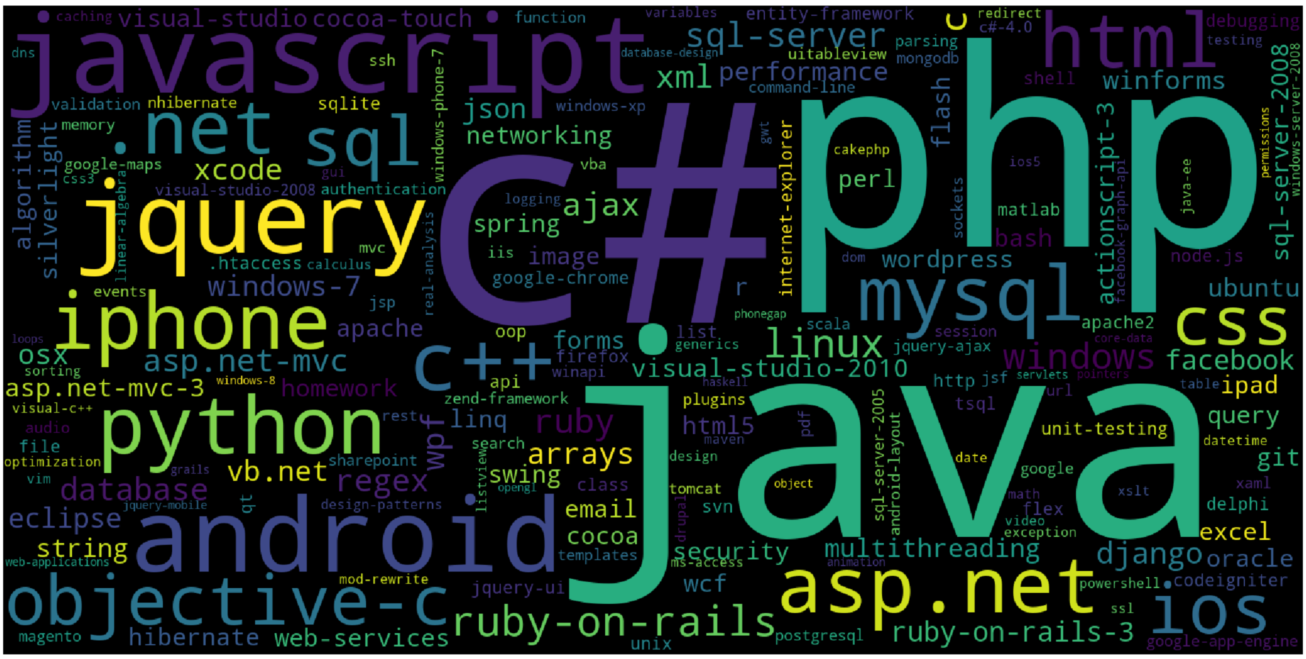
3.2.5 Most Frequent Tags

In [21]:

```
# 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)
```



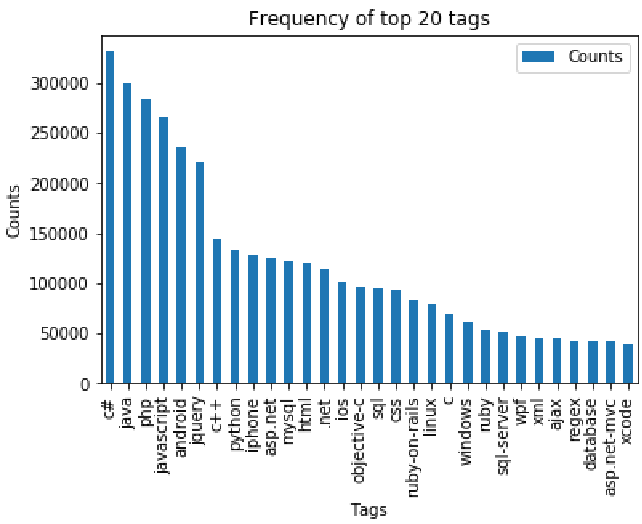

```
Time taken to run this cell : 0:00:04.744944
```

Observations:
A look at the word cloud shows that "c#", "java", "php", "asp.net", "javascript", "c++" are some of the most frequent tags.

3.2.6 The top 20 tags

In [22]:

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



Observations:

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

Below part is going to show 5 title, body for the most used tags

In [ ]:

```
counter = 0
for i in tag_df_sorted["Tags"][:20]:
    print("Tag",i)
    print("="*500)
    for index,row in df_no_dup.iterrows():
        if counter < 5:
            if i in row["Tags"]:
                print(row["Title"])
                print("\n\n")
                print(row["Body"])
                print("+"*100)
                counter+=1
            else:
                continue
        else:
            counter = 0
            break
```

1. only title or writing contents doesn't going to help in determination of tags, some times we need to consider the code part also. 2. we are planning to pay much attention on the words present in tags. 3. If a word from the tag is present in body or title we will give much weightage to it.

3.3 Cleaning and preprocessing of Questions

3.3.1 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 [40]:

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

while reexcuting execute the below

In [41]:

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

    return None

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

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

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

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

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

4. Machine Learning Models

4.1 Converting tags for multilabel problems

4.5 Modeling with less data points (0.5M data points) and more weight to title and 500 tags only.

In [42]:

```
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 [59]:

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

read_db = 'train_no_dup.db'
write_db = 'Titlemoreweight.db'
train_datasize = 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 [58]:

```
conn_r.close()
conn_w.close()
```

4.5.1 Preprocessing of questions

1. Separate Code from Body
2. Remove Spcial characters from Question title and description (not in code)
3. **Give more weightage to title : Add title three times to the question**
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 [60]:

```
#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)+" "+question+" "+code[:50]

    for tag in tag_df_sorted["Tags"][:20]:
        if tag in question:
            tag +=" "
            question +=" "
            question += (tag*5)
            break

#     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) val
ues (?,?,?,?,?,?)",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: 385
Percent of questions containing code: 57
```

```
Time taken to run this cell : 0:28:54.751956
```

In [61]:

```
# 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()
```

Sample quesitons after preprocessing of data

In [62]:

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

```
Questions after preprocessed
====================================================================================================
('dynam datagrid bind silverlight bind datagrid dynam code wrote code debug code block seem bind correc
t grid come column form come grid column although necessari bind nthank repli advance.. myclass myinsta
nc new myclass ndatagridobj c c c c c',)
----------------------------------------------------------------------------------------------------
('java.lang.noclassdeffounderror javax servlet jsp tagext taglibraryvalid follow guid link instal jstl
got follow error tri launch jsp page java.lang.noclassdeffounderror javax servlet jsp tagext taglibrary
valid taglib declar instal jstl 1.1 tomcat webapp tri project work also tri version 1.2 jstl still mess
ag caus solv lt taglib prefix c uri http java.sun.com java java java java java',)
----------------------------------------------------------------------------------------------------
('java.sql.sqlexcept microsoft odbc driver manag invalid descriptor index use follow code display caus
solv tri class.fornam sun.jdbc.odbc.jdbcodbc java java java java java',)
----------------------------------------------------------------------------------------------------
('better way updat feed fb php sdk novic facebook api read mani tutori still confused.i find post feed
api method like correct second way use curl someth like way better data array messag gt hello world c c
c c c',)
----------------------------------------------------------------------------------------------------
('btnadd click event open two window record ad open window search.aspx use code hav add button search.a
spx nwhen insert record btnadd click event open anoth window nafter insert record close window call sea
rch.aspx button click c c c c c',)
----------------------------------------------------------------------------------------------------
('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 exact html use
templat file forgiv okay entir php script get execut see data post none forum field post problem use so
meth titl field none data get post current use print post see submit noth work flawless statement thoug
h also mention script work flawless local machin use host come across problem state list input test mes
s variabl lt div width 100 gt cont php php php php php',)
----------------------------------------------------------------------------------------------------
('countabl subaddit lebesgu measur let lbrace rbrace sequenc set sigma -algebra mathcal want show left
bigcup right leq sum left right countabl addit measur defin set sigma algebra mathcal think use monoton
properti somewher proof start appreci littl help nthank ad han answer make follow addit construct given
han answer clear 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 c c c c c',)
----------------------------------------------------------------------------------------------------
('hql equival sql queri hql queri replac name class properti name error occur hql error select part.pai
d part.panam part.papartnumb c c c c c',)
----------------------------------------------------------------------------------------------------
('undefin symbol architectur i386 objc class skpsmtpmessag referenc error import framework send email a
pplic background import framework i.e skpsmtpmessag somebodi suggest get error collect2 ld return exit
status import framework correct sorc taken framework follow mfmailcomposeviewcontrol question lock fiel
d updat answer drag drop folder project click copi nthat undefin symbol architectur i386 ob c c c c c',
)
----------------------------------------------------------------------------------------------------
```

Saving Preprocessed data to a Database

In [63]:

```
#Taking 0.5 Million entries to a dataframe.
write_db = 'Titlemoreweight.db'
if os.path.isfile(write_db):
    conn_r = create_connection(write_db)
    if conn_r is not None:
        preprocessed_data = pd.read_sql_query("""SELECT question, Tags FROM QuestionsProcessed""", conn
_r)
conn_r.commit()
conn_r.close()
```

In [64]:

```
preprocessed_data.head()
```

Out[64]:

	question	tags
0	dynam datagrid bind silverlight bind datagrid ...	c# silverlight data-binding
1	dynam datagrid bind silverlight bind 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 novic faceboo...	facebook api facebook-php-sdk

In [65]:

```
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 [66]:

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

Selecting 500 Tags

In [67]:

```
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))
```

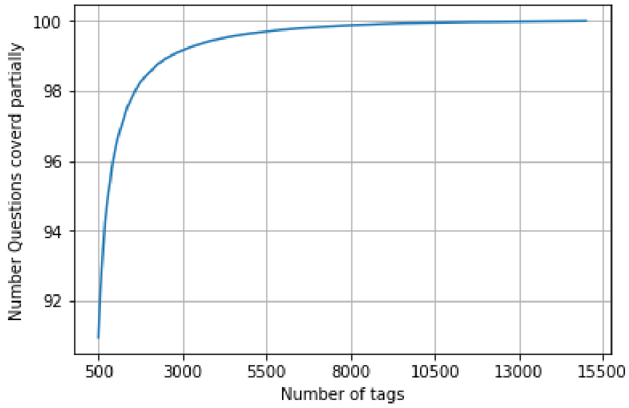
In [68]:

```
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 [69]:

```
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 [70]:

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

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

In [71]:

```
train_datasize = 400000
```

In [72]:

```
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 [73]:

```
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)
```

4.5.2 Featurizing data with BOW vectorizer

In [74]:

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

```
Time taken to run this cell : 0:35:11.094482
```

In [75]:

```
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, 104119) Y : (400000, 500)
Dimensions of test data X: (100000, 104119) Y: (100000, 500)
```

4.5.3 Applying Logistic Regression with OneVsRest Classifier

In [76]:

```
import operator
start = datetime.now()
print('start time',start)
n=100000
all_accuracy = {}
X_train, X_test_, y_train_, y_test_ = train_test_split(x_train_multilabel[0:n,:],y_train[0:n,:], train_
size=.8)

alpha_list = [0.00001,0.00005,0.0001,0.0005,0.001,0.05,0.01,0.1,0.5,1]
for alpha in tqdm(alpha_list):
    print("Value of Alpha",alpha)
    clf = OneVsRestClassifier(SGDClassifier(loss='log', alpha=alpha, penalty='l1'))
    clf.fit(X_train,y_train_)
    pred = clf.predict (X_test_)
    all_accuracy[alpha] = metrics.accuracy_score(y_test_, pred)

best_alpha = max(all_accuracy.items(), key=operator.itemgetter(1))[0]
print('The Best alpha',best_alpha)
```

```
start time 2019-02-23 12:54:13.310955
```

```
  0%|                                                                    | 0/10
[00:00<?, ?it/s]
```

```
Value of Alpha 1e-05
```

```
 10%|                                                     | 1/10 [06:50<1:01
:34, 410.49s/it]
```

```
Value of Alpha 5e-05
```

```
 20%|                                                     | 2/10 [12:33<52
:02, 390.33s/it]
```

```
Value of Alpha 0.0001
```

```
 30%|                                                     | 3/10 [17:35<42
:26, 363.74s/it]
```

```
Value of Alpha 0.0005
```

```
 40%|                                                     | 4/10 [21:50<33
```

```
 40%|████      | 4/10 [21:59<33
:22, 333.71s/it]
```

Value of Alpha 0.001

```
 50%|█████     | 5/10 [26:15<25
:51, 310.39s/it]
```

Value of Alpha 0.05

```
 60%|██████    | 6/10 [30:25<19
:29, 292.30s/it]
```

Value of Alpha 0.01

```
 70%|███████   | 7/10 [34:42<14
:05, 281.68s/it]
```

Value of Alpha 0.1

```
 80%|████████  | 8/10 [38:51<09
:04, 272.01s/it]
```

Value of Alpha 0.5

```
 90%|█████████ | 9/10 [42:56<04
:23, 263.91s/it]
```

Value of Alpha 1

```
100%|██████████| 10/10 [47:04<00
:00, 259.17s/it]
```

The Best alpha 0.001

In [77]:

```
all_accuracy
```

Out[77]:

```
{1e-05: 0.1319,
 5e-05: 0.128,
 0.0001: 0.13145,
 0.0005: 0.14945,
 0.001: 0.1618,
 0.05: 0.0948,
 0.01: 0.1234,
 0.1: 0.0912,
 0.5: 0.08825,
 1: 0.08825}
```

In [78]:

```
start1 = datetime.now()
print('start time',start1)
```

start time 2019-02-23 13:41:21.817631

In [79]:

```
#best_alpha = 0.001
```

In [80]:

```
from prettytable import PrettyTable

out_table = PrettyTable()
out_table.field_names = ["Model", "Vectrozier","Alpha","Accuracy", "Hamming loss","Micro-Precision","Mi
cro-Recall","Micro-F1-measure","Macro-Precision","Macro-Recall","Macro-F1-measure"]
```

In [82]:

```
start = datetime.now()

classifier = OneVsRestClassifier(SGDClassifier(loss='log', alpha=best_alpha, penalty='l1'))
classifier.fit(x_train_multilabel, y_train)
predictions = classifier.predict (x_test_multilabel)

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


precision_micro = precision_score(y_test, predictions, average='micro')
recall_micro = recall_score(y_test, predictions, average='micro')
f1_micro = f1_score(y_test, predictions, average='micro')

print("Micro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision_micro, recall_micro, f1_
micro))

precision_macro = precision_score(y_test, predictions, average='macro')
recall_macro = recall_score(y_test, predictions, average='macro')
f1_macro = f1_score(y_test, predictions, average='macro')

print("Macro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision_macro, recall_macro, f1_
macro))

out_table.add_row(["Logistic Regression","BOW",best_alpha,acc,ham_loss ,precision_micro,recall_micro,f1
_micro,precision_macro,recall_macro,f1_macro])


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

```
Accuracy : 0.18409
Hamming loss  0.00320644
Micro-average quality numbers
Precision: 0.5855, Recall: 0.2658, F1-measure: 0.3656
```

```
C:\Users\bolua\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\metrics\classification.py:11
35: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted s
amples.
  'precision', 'predicted', average, warn_for)
C:\Users\bolua\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\metrics\classification.py:11
35: UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in labels with no predicted sam
ples.
  'precision', 'predicted', average, warn_for)
```

```
Macro-average quality numbers
Precision: 0.3798, Recall: 0.1559, F1-measure: 0.1988
```

```
C:\Users\bolua\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\metrics\classification.py:11
35: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with n
o predicted samples.
  'precision', 'predicted', average, warn_for)
```

```
             precision    recall  f1-score   support

          0       0.86      0.67      0.75      5519
          1       0.32      0.32      0.32      8190
          2       0.75      0.30      0.43      6529
```

3	0.70	0.48	0.57	3231
4	0.76	0.34	0.47	6430
5	0.65	0.40	0.50	2879
6	0.78	0.54	0.64	5086
7	0.82	0.62	0.70	4533
8	0.54	0.13	0.21	3000
9	0.73	0.56	0.64	2765
10	0.50	0.09	0.16	3051
11	0.71	0.28	0.40	3009
12	0.62	0.18	0.28	2630
13	0.45	0.18	0.25	1426
14	0.84	0.49	0.62	2548
15	0.51	0.13	0.21	2371
16	0.54	0.30	0.38	873
17	0.78	0.64	0.71	2151
18	0.58	0.09	0.16	2204
19	0.61	0.35	0.45	831
20	0.63	0.52	0.57	1860
21	0.20	0.09	0.13	2023
22	0.32	0.16	0.21	1513
23	0.82	0.57	0.67	1207
24	0.48	0.20	0.28	506
25	0.72	0.30	0.42	425
26	0.57	0.29	0.39	793
27	0.54	0.21	0.30	1291
28	0.58	0.35	0.44	1208
29	0.22	0.15	0.18	406
30	0.65	0.21	0.31	504
31	0.24	0.05	0.09	732
32	0.52	0.17	0.26	441
33	0.38	0.06	0.10	1645
34	0.64	0.15	0.24	1058
35	0.68	0.64	0.66	946
36	0.66	0.12	0.20	644
37	0.68	0.73	0.70	136
38	0.45	0.31	0.37	570
39	0.81	0.19	0.31	766
40	0.44	0.09	0.15	1132
41	0.35	0.16	0.22	174
42	0.55	0.54	0.54	210
43	0.53	0.56	0.54	433
44	0.60	0.41	0.49	626
45	0.45	0.22	0.29	852
46	0.66	0.21	0.31	534
47	0.19	0.12	0.15	350
48	0.66	0.37	0.47	496
49	0.77	0.49	0.60	785
50	0.07	0.08	0.08	475
51	0.39	0.07	0.12	305
52	0.24	0.03	0.06	251
53	0.57	0.28	0.37	914
54	0.29	0.09	0.14	728
55	0.00	0.00	0.00	258
56	0.37	0.08	0.13	821
57	0.29	0.04	0.08	541
58	0.68	0.34	0.46	748
59	0.85	0.63	0.73	724
60	0.28	0.05	0.09	660
61	0.84	0.15	0.26	235
62	0.89	0.58	0.70	718
63	0.74	0.56	0.64	468
64	0.48	0.41	0.44	191
65	0.22	0.10	0.13	429
66	0.23	0.03	0.06	415
67	0.66	0.39	0.49	274
68	0.72	0.65	0.69	510
69	0.57	0.25	0.35	466
70	0.17	0.03	0.05	305
71	0.42	0.10	0.16	247
72	0.61	0.56	0.59	401
73	0.82	0.70	0.75	86
74	0.40	0.22	0.28	120
75	0.80	0.79	0.79	129
76	0.00	0.00	0.00	473
77	0.36	0.22	0.28	143
78	0.75	0.34	0.47	347
79	0.65	0.15	0.24	479

80	0.33	0.46	0.39	279
81	0.80	0.07	0.13	461
82	0.05	0.01	0.02	298
83	0.74	0.28	0.40	396
84	0.33	0.13	0.19	184
85	0.23	0.08	0.12	573
86	0.07	0.01	0.01	325
87	0.48	0.14	0.21	273
88	0.48	0.17	0.25	135
89	0.31	0.07	0.11	232
90	0.54	0.18	0.27	409
91	0.38	0.40	0.39	420
92	0.71	0.34	0.46	408
93	0.56	0.27	0.36	241
94	0.20	0.02	0.04	211
95	0.20	0.05	0.08	277
96	0.21	0.02	0.03	410
97	0.89	0.11	0.20	501
98	0.52	0.68	0.59	136
99	0.58	0.16	0.25	239
100	0.03	0.02	0.03	324
101	0.85	0.50	0.63	277
102	0.91	0.47	0.62	613
103	0.31	0.07	0.11	157
104	0.15	0.01	0.02	295
105	0.74	0.25	0.38	334
106	1.00	0.01	0.01	335
107	0.60	0.35	0.44	389
108	0.29	0.04	0.07	251
109	0.41	0.52	0.46	317
110	0.03	0.02	0.02	187
111	0.49	0.15	0.23	140
112	0.09	0.09	0.09	154
113	0.53	0.08	0.14	332
114	0.31	0.07	0.12	323
115	0.31	0.02	0.04	344
116	0.53	0.56	0.55	370
117	0.43	0.12	0.19	313
118	0.78	0.11	0.19	874
119	0.42	0.10	0.16	293
120	0.06	0.01	0.02	200
121	0.63	0.60	0.62	463
122	0.15	0.03	0.05	119
123	0.00	0.00	0.00	256
124	0.86	0.72	0.78	195
125	0.30	0.09	0.14	138
126	0.58	0.28	0.38	376
127	0.03	0.01	0.01	122
128	0.19	0.04	0.07	252
129	0.00	0.00	0.00	144
130	0.11	0.01	0.02	150
131	0.26	0.05	0.08	210
132	0.29	0.01	0.03	361
133	0.72	0.65	0.68	453
134	0.65	0.88	0.75	124
135	0.00	0.00	0.00	91
136	0.33	0.14	0.20	128
137	0.43	0.28	0.34	218
138	0.00	0.00	0.00	243
139	0.36	0.13	0.20	149
140	0.55	0.08	0.15	318
141	0.15	0.06	0.09	159
142	0.58	0.20	0.29	274
143	0.86	0.42	0.56	362
144	0.29	0.36	0.32	118
145	0.59	0.20	0.30	164
146	0.43	0.45	0.44	461
147	0.70	0.33	0.45	159
148	0.38	0.07	0.11	166
149	0.90	0.65	0.75	346
150	0.12	0.01	0.02	350
151	0.81	0.38	0.52	55
152	0.71	0.21	0.33	387
153	0.00	0.00	0.00	150
154	0.00	0.00	0.00	281
155	0.23	0.09	0.13	202
156	0.64	0.58	0.61	130

157	0.12	0.01	0.02	245
158	0.77	0.38	0.51	177
159	0.32	0.18	0.23	130
160	0.37	0.06	0.10	336
161	0.90	0.42	0.57	220
162	0.10	0.02	0.03	229
163	0.82	0.48	0.60	316
164	0.60	0.13	0.21	283
165	0.34	0.36	0.35	197
166	0.04	0.01	0.02	101
167	0.30	0.09	0.13	231
168	0.26	0.07	0.11	370
169	0.42	0.12	0.18	258
170	0.31	0.05	0.09	101
171	0.29	0.15	0.19	89
172	0.46	0.24	0.32	193
173	0.37	0.11	0.17	309
174	0.37	0.15	0.21	172
175	0.44	0.73	0.55	95
176	0.66	0.68	0.67	346
177	1.00	0.11	0.20	322
178	0.61	0.28	0.39	232
179	0.25	0.02	0.03	125
180	0.34	0.22	0.27	145
181	0.35	0.08	0.13	77
182	0.06	0.01	0.02	182
183	0.61	0.18	0.27	257
184	0.05	0.00	0.01	216
185	0.24	0.05	0.09	242
186	0.37	0.10	0.15	165
187	0.70	0.45	0.55	263
188	0.20	0.04	0.07	174
189	0.62	0.04	0.07	136
190	0.20	0.17	0.18	202
191	0.18	0.03	0.05	134
192	0.76	0.13	0.23	230
193	0.42	0.09	0.15	90
194	0.48	0.50	0.49	185
195	0.14	0.02	0.03	156
196	0.09	0.01	0.01	160
197	0.00	0.00	0.00	266
198	0.00	0.00	0.00	284
199	0.40	0.01	0.03	145
200	0.83	0.79	0.81	212
201	0.11	0.01	0.01	317
202	0.66	0.41	0.50	427
203	0.11	0.03	0.05	232
204	0.32	0.05	0.09	217
205	0.53	0.04	0.07	527
206	0.00	0.00	0.00	124
207	0.29	0.02	0.04	103
208	0.54	0.32	0.40	287
209	0.26	0.05	0.09	193
210	0.38	0.07	0.12	220
211	0.00	0.00	0.00	140
212	0.05	0.01	0.02	161
213	0.33	0.07	0.11	72
214	0.59	0.17	0.26	396
215	0.93	0.10	0.19	134
216	0.03	0.00	0.00	400
217	0.50	0.24	0.32	75
218	0.72	0.65	0.68	219
219	0.73	0.16	0.26	210
220	0.89	0.24	0.38	298
221	0.97	0.22	0.36	266
222	0.64	0.19	0.29	290
223	0.00	0.00	0.00	128
224	0.71	0.21	0.33	159
225	0.19	0.02	0.03	164
226	0.37	0.26	0.30	144
227	0.43	0.21	0.28	276
228	0.08	0.00	0.01	235
229	0.00	0.00	0.00	216
230	0.36	0.12	0.18	228
231	0.67	0.55	0.60	64
232	0.12	0.03	0.05	103
233	0.67	0.06	0.12	216

234	0.00	0.00	0.00	116
235	0.49	0.25	0.33	77
236	0.89	0.49	0.63	67
237	0.00	0.00	0.00	218
238	0.00	0.00	0.00	139
239	0.33	0.02	0.04	94
240	0.19	0.09	0.12	77
241	0.00	0.00	0.00	167
242	0.77	0.27	0.40	86
243	0.40	0.03	0.06	58
244	0.27	0.03	0.05	269
245	0.11	0.02	0.03	112
246	0.97	0.34	0.50	255
247	0.37	0.22	0.28	58
248	0.00	0.00	0.00	81
249	0.00	0.00	0.00	131
250	0.20	0.04	0.07	93
251	0.65	0.08	0.15	154
252	0.15	0.02	0.03	129
253	0.45	0.23	0.30	83
254	0.32	0.04	0.07	191
255	0.05	0.00	0.01	219
256	0.12	0.02	0.04	130
257	0.41	0.19	0.26	93
258	0.69	0.22	0.33	217
259	0.21	0.05	0.08	141
260	0.00	0.00	0.00	143
261	0.75	0.01	0.03	219
262	0.53	0.07	0.13	107
263	0.40	0.11	0.18	236
264	0.11	0.18	0.14	119
265	0.30	0.11	0.16	72
266	0.00	0.00	0.00	70
267	0.25	0.03	0.05	107
268	0.65	0.21	0.31	169
269	0.14	0.05	0.07	129
270	0.71	0.37	0.49	159
271	0.20	0.02	0.03	190
272	0.07	0.00	0.01	248
273	0.90	0.31	0.46	264
274	0.90	0.42	0.57	105
275	0.11	0.02	0.03	104
276	0.00	0.00	0.00	115
277	0.75	0.27	0.40	170
278	0.64	0.05	0.09	145
279	0.92	0.10	0.19	230
280	0.58	0.31	0.41	80
281	0.72	0.41	0.52	217
282	0.76	0.29	0.41	175
283	0.00	0.00	0.00	269
284	0.57	0.23	0.33	74
285	0.81	0.21	0.33	206
286	0.87	0.50	0.64	227
287	0.76	0.10	0.18	130
288	0.40	0.03	0.06	129
289	0.05	0.01	0.02	80
290	0.15	0.04	0.06	99
291	0.45	0.02	0.05	208
292	0.13	0.03	0.05	67
293	0.58	0.10	0.17	109
294	0.29	0.11	0.16	140
295	0.15	0.04	0.07	241
296	0.17	0.04	0.07	72
297	0.25	0.04	0.07	107
298	0.50	0.10	0.16	61
299	0.31	0.05	0.09	77
300	0.11	0.01	0.02	111
301	0.00	0.00	0.00	126
302	0.00	0.00	0.00	73
303	0.51	0.24	0.32	176
304	0.98	0.25	0.40	230
305	1.00	0.29	0.45	156
306	0.43	0.12	0.19	146
307	0.00	0.00	0.00	98
308	0.00	0.00	0.00	78
309	0.50	0.02	0.04	94
310	0.48	0.10	0.16	162

311	0.72	0.55	0.62	116
312	0.60	0.21	0.31	57
313	1.00	0.02	0.03	65
314	0.48	0.16	0.24	138
315	0.29	0.05	0.09	195
316	0.50	0.20	0.29	69
317	0.00	0.00	0.00	134
318	0.53	0.16	0.25	148
319	0.83	0.09	0.17	161
320	0.23	0.11	0.15	104
321	0.82	0.30	0.44	156
322	0.48	0.16	0.24	134
323	0.62	0.13	0.21	232
324	0.27	0.03	0.06	92
325	0.44	0.04	0.07	197
326	0.00	0.00	0.00	126
327	0.00	0.00	0.00	115
328	0.94	0.22	0.36	198
329	0.33	0.07	0.12	125
330	0.00	0.00	0.00	81
331	0.22	0.02	0.04	94
332	0.00	0.00	0.00	56
333	0.08	0.02	0.03	260
334	0.00	0.00	0.00	60
335	0.21	0.04	0.06	110
336	0.44	0.20	0.27	71
337	0.11	0.02	0.03	66
338	0.48	0.20	0.28	150
339	0.00	0.00	0.00	54
340	0.88	0.44	0.59	195
341	0.00	0.00	0.00	79
342	0.06	0.03	0.04	38
343	0.20	0.05	0.08	43
344	0.40	0.03	0.05	68
345	0.67	0.33	0.44	73
346	0.12	0.04	0.06	116
347	0.83	0.34	0.48	111
348	0.21	0.06	0.10	63
349	0.75	0.65	0.70	104
350	0.69	0.25	0.37	44
351	0.00	0.00	0.00	40
352	1.00	0.04	0.07	136
353	0.25	0.06	0.09	54
354	0.00	0.00	0.00	134
355	0.18	0.25	0.21	120
356	0.12	0.01	0.02	228
357	0.50	0.02	0.04	269
358	0.67	0.20	0.31	80
359	0.73	0.08	0.14	140
360	0.40	0.05	0.09	125
361	0.00	0.00	0.00	169
362	0.08	0.02	0.03	56
363	0.88	0.55	0.67	154
364	0.00	0.00	0.00	58
365	0.13	0.06	0.08	71
366	1.00	0.22	0.36	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	0.58	0.27	0.37	52
373	0.00	0.00	0.00	150
374	0.35	0.10	0.15	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	0.04	0.08	122
381	0.07	0.01	0.02	104
382	0.14	0.03	0.05	66
383	0.42	0.10	0.16	110
384	0.00	0.00	0.00	155
385	0.25	0.02	0.04	50
386	0.23	0.08	0.12	64
387	0.50	0.02	0.04	93

388	0.42	0.11	0.17	102
389	0.00	0.00	0.00	108
390	1.00	0.26	0.41	178
391	0.33	0.03	0.05	115
392	0.00	0.00	0.00	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	0.76	0.07	0.13	224
398	0.79	0.17	0.29	63
399	0.00	0.00	0.00	59
400	0.29	0.10	0.14	63
401	0.00	0.00	0.00	98
402	0.06	0.12	0.08	162
403	0.25	0.04	0.06	83
404	0.71	0.79	0.75	19
405	0.08	0.02	0.03	92
406	0.75	0.15	0.24	41
407	0.38	0.12	0.18	43
408	0.00	0.00	0.00	160
409	0.08	0.02	0.03	50
410	0.00	0.00	0.00	19
411	0.13	0.02	0.03	175
412	0.08	0.12	0.10	72
413	0.00	0.00	0.00	95
414	0.17	0.02	0.04	97
415	0.20	0.04	0.07	48
416	0.29	0.27	0.27	83
417	0.00	0.00	0.00	40
418	0.09	0.10	0.10	91
419	0.28	0.32	0.30	90
420	0.25	0.38	0.30	37
421	0.00	0.00	0.00	66
422	0.56	0.12	0.20	73
423	0.28	0.34	0.31	56
424	0.95	0.64	0.76	33
425	0.17	0.01	0.02	76
426	0.00	0.00	0.00	81
427	0.98	0.42	0.59	150
428	0.33	0.76	0.46	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	0.29	0.13	0.18	39
433	0.31	0.06	0.10	82
434	1.00	0.18	0.31	66
435	0.64	0.17	0.27	93
436	0.00	0.00	0.00	87
437	0.33	0.01	0.02	86
438	0.46	0.39	0.42	104
439	0.00	0.00	0.00	100
440	0.50	0.01	0.01	141
441	0.35	0.11	0.17	110
442	0.13	0.02	0.04	123
443	0.00	0.00	0.00	71
444	0.23	0.03	0.05	109
445	0.23	0.06	0.10	48
446	0.28	0.07	0.11	76
447	0.08	0.18	0.11	38
448	0.59	0.23	0.34	81
449	0.25	0.02	0.03	132
450	0.34	0.12	0.18	81
451	0.68	0.17	0.27	76
452	0.00	0.00	0.00	44
453	0.00	0.00	0.00	44
454	0.87	0.19	0.31	70
455	0.00	0.00	0.00	155
456	0.21	0.09	0.13	43
457	0.50	0.03	0.05	72
458	0.00	0.00	0.00	62
459	1.00	0.01	0.03	69
460	0.10	0.02	0.03	119
461	0.00	0.00	0.00	79
462	0.50	0.06	0.11	47
463	0.01	0.02	0.01	104
464	0.52	0.10	0.17	106

```
        465       0.00      0.00      0.00        64
        466       0.50      0.03      0.06       173
        467       1.00      0.02      0.04       107
        468       0.00      0.00      0.00       126
        469       0.00      0.00      0.00       114
        470       0.93      0.38      0.54       140
        471       0.00      0.00      0.00        79
        472       0.33      0.08      0.12       143
        473       0.14      0.01      0.01       158
        474       0.16      0.10      0.12       138
        475       0.00      0.00      0.00        59
        476       0.57      0.18      0.28        88
        477       0.86      0.14      0.24       176
        478       0.83      0.21      0.33        24
        479       0.00      0.00      0.00        92
        480       1.00      0.05      0.10       100
        481       0.06      0.01      0.02       103
        482       0.21      0.07      0.10        74
        483       0.78      0.61      0.68       105
        484       0.00      0.00      0.00        83
        485       0.00      0.00      0.00        82
        486       0.09      0.03      0.04        71
        487       0.40      0.08      0.14       120
        488       0.00      0.00      0.00       105
        489       0.80      0.09      0.16        87
        490       0.74      0.44      0.55        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       0.11      0.02      0.03        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

avg / total       0.53      0.27      0.33    173812

Time taken to run this cell : 0:23:24.775954
```

In [83]:

```
from sklearn.externals import joblib
joblib.dump(classifier, 'lr_with_more_title_weight.pkl')
```

Out[83]:

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

In [84]:

```
start = datetime.now()
classifier_2 = OneVsRestClassifier(SGDClassifier(loss='hinge', alpha=best_alpha, penalty='l1'), n_jobs=
-1)
classifier_2.fit(x_train_multilabel, y_train)
predictions_2 = classifier_2.predict(x_test_multilabel)
acc = metrics.accuracy_score(y_test, predictions_2)
ham_loss = metrics.hamming_loss(y_test,predictions_2)
print("Accuracy :",acc)
print("Hamming loss ",ham_loss)


precision_micro = precision_score(y_test, predictions_2, average='micro')
recall_micro = recall_score(y_test, predictions_2, average='micro')
f1_micro = f1_score(y_test, predictions_2, average='micro')

print("Micro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision_micro, recall_micro, f1_
micro))

precision_macro = precision_score(y_test, predictions_2, average='macro')
recall_macro = recall_score(y_test, predictions_2, average='macro')
f1_macro = f1_score(y_test, predictions_2, average='macro')
```

```python
print("Macro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision_macro, recall_macro, f1_
macro))

out_table.add_row(["Linear-SVM","BOW",best_alpha,acc,ham_loss ,precision_micro,recall_micro,f1_micro,pr
ecision_macro,recall_macro,f1_macro])

print (metrics.classification_report(y_test, predictions_2))
print("Time taken to run this cell :", datetime.now() - start)
```

```
Accuracy : 0.18745
Hamming loss  0.00317938
Micro-average quality numbers
Precision: 0.5877, Recall: 0.2863, F1-measure: 0.3850
```

```
C:\Users\bolua\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\metrics\classification.py:11
35: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted s
amples.
  'precision', 'predicted', average, warn_for)
C:\Users\bolua\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\metrics\classification.py:11
35: UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in labels with no predicted sam
ples.
  'precision', 'predicted', average, warn_for)
```

```
Macro-average quality numbers
Precision: 0.2802, Recall: 0.1854, F1-measure: 0.2040
```

```
C:\Users\bolua\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\metrics\classification.py:11
35: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with n
o predicted samples.
  'precision', 'predicted', average, warn_for)
```

```
             precision    recall  f1-score   support

          0       0.80      0.70      0.75      5519
          1       0.52      0.23      0.32      8190
          2       0.77      0.31      0.44      6529
          3       0.63      0.52      0.57      3231
          4       0.73      0.44      0.55      6430
          5       0.65      0.41      0.50      2879
          6       0.75      0.59      0.66      5086
          7       0.78      0.62      0.69      4533
          8       0.39      0.22      0.28      3000
          9       0.71      0.57      0.63      2765
         10       0.00      0.00      0.00      3051
         11       0.71      0.35      0.47      3009
         12       0.62      0.26      0.37      2630
         13       0.23      0.26      0.24      1426
         14       0.85      0.51      0.64      2548
         15       0.81      0.10      0.17      2371
         16       0.52      0.28      0.36       873
         17       0.72      0.64      0.68      2151
         18       0.31      0.25      0.28      2204
         19       0.47      0.53      0.50       831
         20       0.72      0.58      0.64      1860
         21       0.11      0.02      0.04      2023
         22       0.12      0.00      0.00      1513
         23       0.83      0.54      0.66      1207
         24       0.33      0.00      0.00       506
         25       0.61      0.44      0.51       425
         26       0.40      0.30      0.34       793
         27       0.58      0.13      0.22      1291
         28       0.37      0.45      0.41      1208
         29       0.21      0.22      0.22       406
         30       0.71      0.29      0.41       504
         31       0.12      0.17      0.14       732
         32       0.41      0.39      0.40       441
         33       0.00      0.00      0.00      1645
         34       0.51      0.33      0.40      1058
         35       0.76      0.45      0.56       946
         36       0.71      0.10      0.18       644
         37       0.86      0.83      0.85       136
         38       0.36      0.37      0.37       570
         39       0.59      0.43      0.50       766
```

39	0.59	0.45	0.50	766
40	0.00	0.00	0.00	1132
41	0.36	0.13	0.19	174
42	0.53	0.59	0.56	210
43	0.52	0.52	0.52	433
44	0.66	0.35	0.46	626
45	0.00	0.00	0.00	852
46	0.54	0.36	0.43	534
47	0.00	0.00	0.00	350
48	0.51	0.55	0.53	496
49	0.71	0.64	0.67	785
50	0.00	0.00	0.00	475
51	0.00	0.00	0.00	305
52	0.33	0.06	0.10	251
53	0.49	0.56	0.52	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	0.69	0.24	0.36	748
59	0.71	0.73	0.72	724
60	0.00	0.00	0.00	660
61	0.67	0.28	0.40	235
62	0.81	0.81	0.81	718
63	0.81	0.44	0.57	468
64	0.45	0.53	0.49	191
65	0.01	0.00	0.00	429
66	0.00	0.00	0.00	415
67	0.61	0.51	0.56	274
68	0.56	0.60	0.58	510
69	0.59	0.04	0.07	466
70	0.00	0.00	0.00	305
71	0.00	0.00	0.00	247
72	0.62	0.60	0.61	401
73	0.44	0.74	0.55	86
74	0.24	0.34	0.28	120
75	0.76	0.79	0.77	129
76	0.00	0.00	0.00	473
77	0.33	0.42	0.37	143
78	0.70	0.59	0.64	347
79	0.65	0.18	0.28	479
80	0.33	0.36	0.34	279
81	0.82	0.07	0.13	461
82	0.00	0.00	0.00	298
83	0.69	0.39	0.50	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	1.00	0.00	0.00	409
91	0.47	0.22	0.30	420
92	0.63	0.68	0.66	408
93	0.54	0.29	0.37	241
94	0.10	0.00	0.01	211
95	0.00	0.00	0.00	277
96	0.00	0.00	0.00	410
97	0.84	0.19	0.31	501
98	0.56	0.71	0.63	136
99	0.38	0.24	0.30	239
100	0.00	0.00	0.00	324
101	0.90	0.47	0.62	277
102	0.77	0.75	0.76	613
103	0.50	0.02	0.04	157
104	0.00	0.00	0.00	295
105	0.66	0.45	0.53	334
106	0.00	0.00	0.00	335
107	0.55	0.58	0.56	389
108	0.00	0.00	0.00	251
109	0.51	0.08	0.13	317
110	0.00	0.00	0.00	187
111	0.57	0.19	0.28	140
112	0.00	0.00	0.00	154
113	0.00	0.00	0.00	332
114	0.43	0.06	0.10	323
115	0.00	0.00	0.00	344
116	0.00	0.00	0.00	370

116	0.00	0.00	0.00	370
117	0.51	0.15	0.23	313
118	0.87	0.04	0.08	874
119	0.00	0.00	0.00	293
120	0.00	0.00	0.00	200
121	0.76	0.23	0.35	463
122	0.00	0.00	0.00	119
123	0.00	0.00	0.00	256
124	0.77	0.89	0.83	195
125	0.00	0.00	0.00	138
126	0.58	0.34	0.43	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	0.10	0.06	0.07	210
132	0.00	0.00	0.00	361
133	0.90	0.43	0.58	453
134	0.68	0.82	0.74	124
135	0.04	0.02	0.03	91
136	0.00	0.00	0.00	128
137	0.42	0.28	0.34	218
138	0.10	0.10	0.10	243
139	0.00	0.00	0.00	149
140	0.60	0.20	0.30	318
141	0.00	0.00	0.00	159
142	0.56	0.56	0.56	274
143	0.74	0.60	0.66	362
144	0.15	0.30	0.20	118
145	0.67	0.20	0.31	164
146	0.00	0.00	0.00	461
147	0.57	0.48	0.52	159
148	0.21	0.21	0.21	166
149	0.96	0.44	0.60	346
150	0.00	0.00	0.00	350
151	0.79	0.67	0.73	55
152	0.53	0.46	0.49	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	0.49	0.66	0.56	130
157	0.00	0.00	0.00	245
158	0.61	0.70	0.65	177
159	0.59	0.18	0.28	130
160	0.00	0.00	0.00	336
161	0.55	0.70	0.62	220
162	0.00	0.00	0.00	229
163	0.78	0.50	0.61	316
164	0.39	0.25	0.30	283
165	1.00	0.02	0.03	197
166	0.15	0.14	0.14	101
167	0.00	0.00	0.00	231
168	0.15	0.18	0.16	370
169	1.00	0.00	0.01	258
170	0.08	0.12	0.10	101
171	0.00	0.00	0.00	89
172	0.46	0.09	0.15	193
173	0.00	0.00	0.00	309
174	0.00	0.00	0.00	172
175	0.54	0.86	0.66	95
176	0.76	0.58	0.66	346
177	0.96	0.21	0.35	322
178	0.49	0.43	0.46	232
179	0.00	0.00	0.00	125
180	0.33	0.51	0.40	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	0.70	0.54	0.61	263
188	0.00	0.00	0.00	174
189	0.09	0.10	0.10	136
190	0.90	0.55	0.69	202
191	0.00	0.00	0.00	134
192	0.65	0.43	0.52	230
193	0.00	0.00	0.00	90

193	0.00	0.00	0.00	90
194	0.62	0.16	0.25	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	0.83	0.76	0.79	212
201	0.00	0.00	0.00	317
202	0.60	0.01	0.01	427
203	0.00	0.00	0.00	232
204	0.00	0.00	0.00	217
205	0.43	0.58	0.49	527
206	0.00	0.00	0.00	124
207	0.00	0.00	0.00	103
208	0.68	0.53	0.59	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	0.13	0.02	0.04	161
213	0.00	0.00	0.00	72
214	0.67	0.14	0.23	396
215	0.73	0.40	0.51	134
216	0.00	0.00	0.00	400
217	0.36	0.11	0.16	75
218	0.84	0.75	0.80	219
219	0.48	0.28	0.36	210
220	0.82	0.24	0.38	298
221	0.92	0.54	0.68	266
222	0.70	0.39	0.50	290
223	0.00	0.00	0.00	128
224	0.35	0.40	0.37	159
225	0.14	0.04	0.07	164
226	0.35	0.44	0.39	144
227	0.40	0.45	0.42	276
228	0.00	0.00	0.00	235
229	0.00	0.00	0.00	216
230	0.22	0.35	0.27	228
231	0.57	0.66	0.61	64
232	0.00	0.00	0.00	103
233	0.71	0.19	0.29	216
234	0.00	0.00	0.00	116
235	0.57	0.66	0.61	77
236	0.84	0.78	0.81	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	0.31	0.35	0.33	77
241	0.05	0.07	0.06	167
242	0.66	0.27	0.38	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	0.90	0.77	0.83	255
247	0.35	0.22	0.27	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	0.37	0.30	0.33	154
252	0.00	0.00	0.00	129
253	0.40	0.02	0.05	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	1.00	0.01	0.02	93
258	0.59	0.57	0.58	217
259	0.07	0.07	0.07	141
260	0.67	0.24	0.36	143
261	0.00	0.00	0.00	219
262	0.56	0.14	0.22	107
263	0.33	0.36	0.34	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	0.55	0.38	0.45	169
269	0.00	0.00	0.00	129
270	[illegible]	[illegible]	[illegible]	[illegible]

270	0.58	0.43	0.50	159
271	0.00	0.00	0.00	190
272	0.13	0.15	0.14	248
273	0.83	0.40	0.54	264
274	0.66	0.76	0.70	105
275	0.00	0.00	0.00	104
276	0.00	0.00	0.00	115
277	0.72	0.65	0.68	170
278	0.47	0.51	0.49	145
279	0.78	0.55	0.65	230
280	0.45	0.31	0.37	80
281	0.56	0.71	0.63	217
282	0.63	0.53	0.58	175
283	0.00	0.00	0.00	269
284	0.41	0.61	0.49	74
285	0.62	0.36	0.45	206
286	0.84	0.44	0.57	227
287	0.01	0.02	0.01	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	0.33	0.16	0.21	109
294	0.30	0.13	0.18	140
295	0.19	0.05	0.08	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	0.83	0.13	0.22	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	0.40	0.53	0.45	176
304	0.80	0.77	0.78	230
305	0.88	0.64	0.74	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	0.07	0.07	0.07	94
310	0.39	0.41	0.40	162
311	0.68	0.67	0.68	116
312	0.32	0.37	0.34	57
313	0.00	0.00	0.00	65
314	0.00	0.00	0.00	138
315	0.39	0.25	0.30	195
316	0.00	0.00	0.00	69
317	0.00	0.00	0.00	134
318	1.00	0.01	0.01	148
319	0.84	0.42	0.56	161
320	0.15	0.23	0.18	104
321	0.65	0.44	0.53	156
322	0.00	0.00	0.00	134
323	0.46	0.22	0.30	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	0.89	0.56	0.68	198
329	0.49	0.18	0.26	125
330	0.47	0.09	0.15	81
331	0.00	0.00	0.00	94
332	0.27	0.11	0.15	56
333	0.00	0.00	0.00	260
334	0.25	0.02	0.03	60
335	0.00	0.00	0.00	110
336	0.36	0.51	0.42	71
337	0.00	0.00	0.00	66
338	0.31	0.42	0.35	150
339	0.00	0.00	0.00	54
340	0.60	0.65	0.63	195
341	0.00	0.00	0.00	79
342	0.00	0.00	0.00	38
343	0.67	0.09	0.16	43
344	0.24	0.29	0.27	68
345	0.88	0.10	0.17	73
346	0.01	0.01	0.01	116

347	0.64	0.06	0.11	111
348	0.00	0.00	0.00	63
349	0.63	0.62	0.62	104
350	0.55	0.50	0.52	44
351	0.00	0.00	0.00	40
352	0.00	0.00	0.00	136
353	0.00	0.00	0.00	54
354	0.00	0.00	0.00	134
355	0.12	0.01	0.02	120
356	0.00	0.00	0.00	228
357	0.00	0.00	0.00	269
358	0.45	0.38	0.41	80
359	0.79	0.22	0.35	140
360	0.11	0.10	0.11	125
361	0.89	0.29	0.44	169
362	0.07	0.11	0.08	56
363	0.76	0.43	0.55	154
364	0.00	0.00	0.00	58
365	0.00	0.00	0.00	71
366	0.85	0.43	0.57	54
367	0.16	0.03	0.04	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	0.60	0.29	0.39	52
373	0.75	0.31	0.44	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	0.86	0.36	0.51	122
381	0.00	0.00	0.00	104
382	0.00	0.00	0.00	66
383	0.28	0.44	0.34	110
384	0.00	0.00	0.00	155
385	0.15	0.16	0.15	50
386	0.16	0.19	0.17	64
387	0.00	0.00	0.00	93
388	0.45	0.24	0.31	102
389	0.00	0.00	0.00	108
390	0.79	0.71	0.75	178
391	0.00	0.00	0.00	115
392	0.82	0.43	0.56	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	0.80	0.04	0.07	224
398	0.63	0.19	0.29	63
399	0.00	0.00	0.00	59
400	0.24	0.40	0.30	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	0.67	0.63	0.65	19
405	0.00	0.00	0.00	92
406	0.47	0.22	0.30	41
407	0.38	0.26	0.31	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	0.05	0.07	0.06	97
415	0.00	0.00	0.00	48
416	0.38	0.36	0.37	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	0.38	0.18	0.24	56

424	0.84	0.79	0.81	33
425	0.05	0.07	0.05	76
426	0.00	0.00	0.00	81
427	0.69	0.56	0.62	150
428	0.78	0.86	0.82	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	0.24	0.21	0.22	82
434	0.93	0.61	0.73	66
435	0.52	0.59	0.56	93
436	0.00	0.00	0.00	87
437	0.00	0.00	0.00	86
438	0.33	0.01	0.02	104
439	0.00	0.00	0.00	100
440	0.00	0.00	0.00	141
441	0.36	0.16	0.22	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	0.15	0.18	0.16	38
448	0.58	0.52	0.55	81
449	0.00	0.00	0.00	132
450	0.28	0.31	0.30	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	0.69	0.13	0.22	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	0.47	0.33	0.39	106
465	0.00	0.00	0.00	64
466	0.36	0.52	0.43	173
467	0.12	0.03	0.05	107
468	0.00	0.00	0.00	126
469	0.00	0.00	0.00	114
470	0.85	0.67	0.75	140
471	0.00	0.00	0.00	79
472	0.33	0.42	0.37	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	0.33	0.34	0.34	88
477	0.77	0.66	0.71	176
478	0.84	0.67	0.74	24
479	0.00	0.00	0.00	92
480	0.64	0.23	0.34	100
481	0.00	0.00	0.00	103
482	0.20	0.01	0.03	74
483	0.65	0.31	0.42	105
484	0.02	0.01	0.01	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	0.56	0.51	0.53	87
490	0.91	0.31	0.47	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	0.12	0.02	0.03	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

```
avg / total       0.45      0.29      0.33    173812

Time taken to run this cell : 0:13:11.320384
```

In [85]:

```python
print(out_table)
```

```
+---------------------+------------+-------+----------+--------------+--------------------+------------
---------+--------------------+--------------------+---------------------+---------------------+
|        Model        | Vectrozier | Alpha | Accuracy | Hamming loss |  Micro-Precision   |     Micro-R
ecall    |  Micro-F1-measure  |  Macro-Precision   |     Macro-Recall    |   Macro-F1-measure  |
+---------------------+------------+-------+----------+--------------+--------------------+------------
---------+--------------------+--------------------+---------------------+---------------------+
| Logistic Regression |    BOW     | 0.001 | 0.18409  |  0.00320644  | 0.5854792923404472 |  0.26579867
90325179 | 0.3656141183918962 | 0.3797871813543305 |  0.1559428573742044 | 0.19878062538812702 |
|      Linear-SVM     |    BOW     | 0.001 | 0.18745  |  0.00317938  | 0.5876530961745149 | 0.286263318
98833224 | 0.3849876780111498 | 0.2801658704241521 | 0.18538346621498564 | 0.20396162879047677 |
+---------------------+------------+-------+----------+--------------+--------------------+------------
---------+--------------------+--------------------+---------------------+---------------------+
```

Summerize work

We observed only with title and description it is hard to predict the Tags We used some of the code art also for our model we used first 50 character from our code for prediction We have not used three time weightage of title we considered top 20 most used tag and if any of the tag is present in our title or description or first 50 character of code we gave that tag more weightage by adding the tag to the sentance 5 times. Idea behind this is if Java, C, PHP etc, these kind of words are used in title or body part, there may be higher chance that our tags should be same. so we are giving more weightage in this case. At the end by considering only top 20 most used tags our result seems to be improved a little. More tags and more dimensions of BOW may take considerable amount of time resulting better result.

In [ ]:

```python
```