In [0]:

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6
qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%
b&scope=email%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.
2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.photos.readonly%20https%3A%2F%2Fww
ogleapis.com%2Fauth%2Fpeopleapi.readonly&response_type=code

Enter your authorization code:
··········
Mounted at /content/gdrive
```

In [0]:

```
!pip install scikit-multilearn
```

```
Collecting scikit-multilearn
  Downloading
https://files.pythonhosted.org/packages/bb/1f/e6ff649c72a1cdf2c7a1d31eb21705110ce1c5d3e7e26b2cc300e
272/scikit_multilearn-0.2.0-py3-none-any.whl (89kB)
     |████████████████████████████████| 92kB 3.1MB/s
Installing collected packages: scikit-multilearn
Successfully installed scikit-multilearn-0.2.0
```

In [0]:

```
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
```

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, sh
ould 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
           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

3.1.1 Using Pandas with SQLite to Load the data

In [0]:

```
#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('gdrive/My Drive/Colab Notebooks/Train.csv',nrows=200000, names=['Id', 'T
itle', 'Body', 'Tags'], chunksize=chunksize, iterator=True, encoding='utf-8', ):
        df.index += index_start
        j+=1
        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)
```

```
Time taken to run this cell : 0:00:18.113470
```

3.1.2 Counting the number of rows

In [0]:

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

```
Number of rows in the database :
 200000
Time taken to count the number of rows : 0:00:00.009961
```

3.1.3 Checking for duplicates

In [0]:

```
#Learn SQl: https://www.w3schools.com/sql/default.asp
if os.path.isfile('train.db'):
    start = datetime.now()
    con = sqlite3.connect('train.db')
    df_no_dup = pd.read_sql_query('SELECT Title, Body, Tags, COUNT(*) as cnt_dup FROM data GROUP BY Title, 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")
```

```
Time taken to run this cell : 0:00:02.063599
```

In [0]:

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

Out[0]:

	Title	Body	Tags	cnt_dup
0	*** Exception: Prelude.read: no parse in Hask...	<p>This portion of code should read in two or ...	parsing haskell expression	1
1	Accessing @Local Session Bean from an exposed...	<p>What I am trying to do should be very strai...	ejb resteasy	1
2	Controlling the Lego WeDo Device	<p>Has anyone written a API for the Lego WeDo ...	c# api lego	1
3	Dialog not getting recreated on orientation c...	<p>called using: </p>\n\n<p>showDialog(DIALOG_L...	android dialog display orientation	1
4	Dynamically changing localization in VS2010 s...	<p>I would like to know how to change localiza...	c# visual-studio-2010 localization	1

In [0]:

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

```
number of duplicate questions : 2250 ( 1.1249999999999982 % )
```

In [0]:

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

Out[0]:

```
1    195507
2      2236
3         7
Name: cnt_dup, dtype: int64
```

In [0]:

```
df_no_dup['Tags']=df_no_dup['Tags'].apply(str)
```

In [0]:

```
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 : 0:00:00.151672
```

Out[0]:

	Title	Body	Tags	cnt_dup	tag_count
0	*** Exception: Prelude.read: no parse in Hask...	<p>This portion of code should read in two or ...	parsing haskell expression	1	3
1	Accessing @Local Session Bean from an exposed...	<p>What I am trying to do should be very strai...	ejb resteasy	1	2
2	Controlling the Lego WeDo Device	<p>Has anyone written a API for the Lego WeDo ...	c# api lego	1	3
3	Dialog not getting recreated on orientation c...	<p>called using: </p>\n\n<p>showDialog(DIALOG_L...	android dialog display orientation	1	4
4	Dynamically changing localization in VS2010 s...	<p>I would like to know how to change localiza...	c# visual-studio-2010 localization	1	3

In [0]:

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

Out[0]:

```
3    56635
2    52585
4    37959
1    27163
5    23408
Name: tag_count, dtype: int64
```

In [0]:

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

In [0]:

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

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

```
Time taken to run this cell : 0:00:00.309526
```

3.2 Analysis of Tags

3.2.1 Total number of unique tags

In [0]:

```
# 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 [0]:

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

```
Number of data points : 197749
Number of unique tags : 23686
```

In [0]:

```
#'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', '.aspxauth', '.bash-profile', '.class-file', '.doc', '.e
ach', '.emf', '.hgtags', '.htaccess']
```

3.2.3 Number of times a tag appeared

In [0]:

```
# 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 [0]:

```
#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[0]:

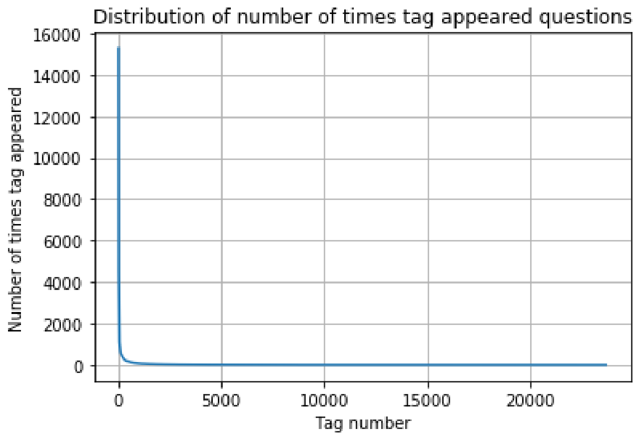
	Tags	Counts
0	.a	3
1	.app	1
2	.aspxauth	1
3	.bash-profile	3
4	.class-file	2

In [0]:

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

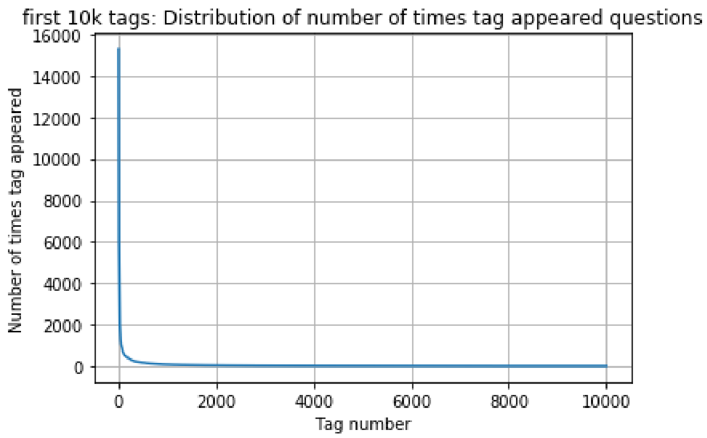
In [0]:

```
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 [0]:

```
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])
```

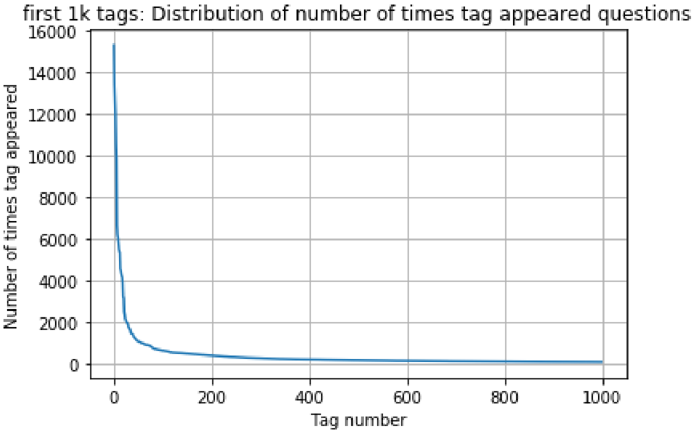



```
400 [15289  1995  1056   830   625   523   482   427   380   345   313   276
   255   233   217   204   196   189   179   171   164   157   149   140
   135   131   125   121   115   112   107   104    99    97    94    89
    86    84    82    79    77    75    73    71    69    67    66    64
    63    61    60    59    58    57    56    55    54    53    52    51
    49    48    48    47    46    45    45    44    43    43    42    41
    40    40    39    39    38    37    37    36    36    35    35    34
    34    34    33    32    32    31    31    31    30    30    30    29
    29    28    28    28    27    27    26    26    26    25    25    25
    24    24    24    24    23    23    23    22    22    22    22    21
    21    21    21    21    20    20    20    20    20    19    19    19
    19    19    19    18    18    18    18    18    18    17    17    17
    17    17    17    16    16    16    16    16    16    16    15    15
    15    15    15    15    15    15    14    14    14    14    14    14
```

```
     14      14      14      13      13      13      13      13      13      13      13      13
     12      12      12      12      12      12      12      12      12      12      11      11
     11      11      11      11      11      11      11      11      11      10      10      10
     10      10      10      10      10      10      10      10      10      10      10       9
      9       9       9       9       9       9       9       9       9       9       9       9
      9       9       9       8       8       8       8       8       8       8       8       8
      8       8       8       8       8       8       8       8       8       8       8       7
      7       7       7       7       7       7       7       7       7       7       7       7
      7       7       7       7       7       7       7       7       7       7       7       6
      6       6       6       6       6       6       6       6       6       6       6       6
      6       6       6       6       6       6       6       6       6       6       6       6
      6       6       6       6       6       6       6       5       5       5       5       5
      5       5       5       5       5       5       5       5       5       5       5       5
      5       5       5       5       5       5       5       5       5       5       5       5
      5       5       5       5       5       5       5       5       5       5       5       5
      5       4       4       4       4       4       4       4       4       4       4       4
      4       4       4       4       4       4       4       4       4       4       4       4
      4       4       4       4       4       4       4       4       4       4       4       4
      4       4       4       4       4       4       4       4       4       4       4       4
      4       4       4       4]
```

In [0]:

```
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])
```

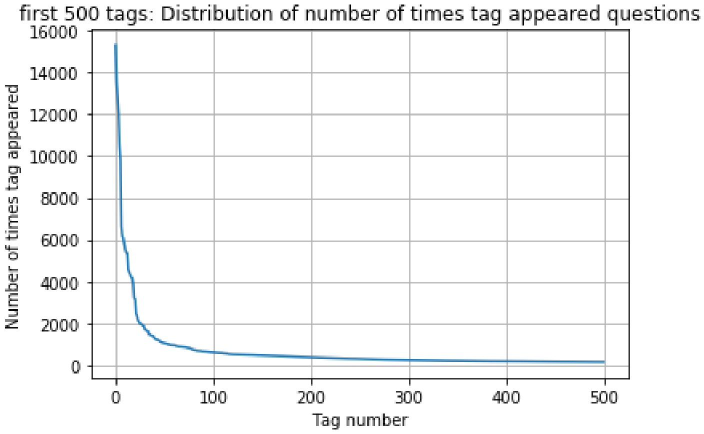



```
200 [15289  9803  5517  4334  3165  1995  1718  1450  1293  1153  1056  1003
    961   907   887   830   742   702   668   645   625   602   584   549
    530   523   512   508   500   490   482   467   455   444   438   427
    419   412   397   389   380   371   367   357   351   345   332   327
    323   318   313   304   296   290   283   276   272   267   264   260
    255   248   244   239   237   233   231   228   222   220   217   216
    210   209   206   204   202   200   199   197   196   194   193   193
    190   189   186   185   183   182   179   177   176   175   173   171
    169   167   166   164   164   162   162   160   158   157   154   153
    152   151   149   148   146   144   142   140   139   137   137   136
    135   134   133   133   132   131   130   129   127   126   125   124
    124   123   122   121   121   118   117   116   115   115   114   113
    112   112   111   110   109   109   107   107   106   106   105   104
    103   102   100   100    99    99    99    98    98    97    97    96
     95    95    94    93    92    92    91    89    88    87    87    86
     86    86    85    85    84    84    83    83    82    82    82    81
     81    80    80    79    79    78    77    77]
```

In [0]:

```
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])
```



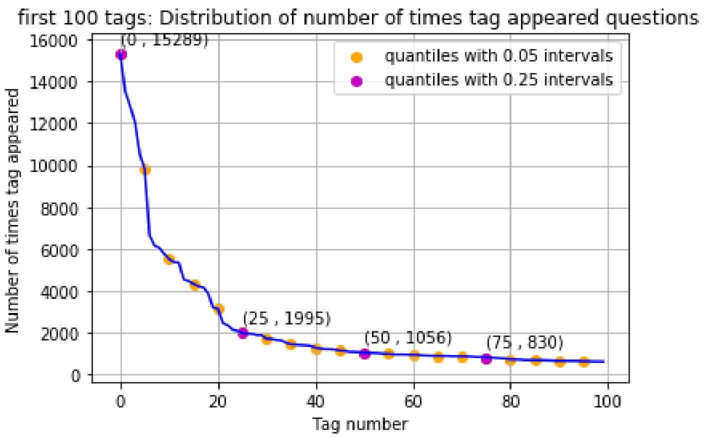

```
100 [15289  9803  5517  4334  3165  1995  1718  1450  1293  1153  1056  1003
   961   907   887   830   742   702   668   645   625   602   584   549
   530   523   512   508   500   490   482   467   455   444   438   427
   419   412   397   389   380   371   367   357   351   345   332   327
   323   318   313   304   296   290   283   276   272   267   264   260
   255   248   244   239   237   233   231   228   222   220   217   216
   210   209   206   204   202   200   199   197   196   194   193   193
   190   189   186   185   183   182   179   177   176   175   173   171
   169   167   166   164]
```

In [0]:

```
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 [15289  9803  5517  4334  3165  1995  1718  1450  1293  1153  1056  1003
   961   907   887   830   742   702   668   645]
```

In [0]:

```
# 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)))
```

```
5 Tags are used more than 10000 times
0 Tags are used more than 100000 times
```

Observations:

1. There are total 5 tags which are used more than 10000 times.
2. 0 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 [0]:

```
#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 197749 datapoints.
[2, 3, 4, 3, 4]
```

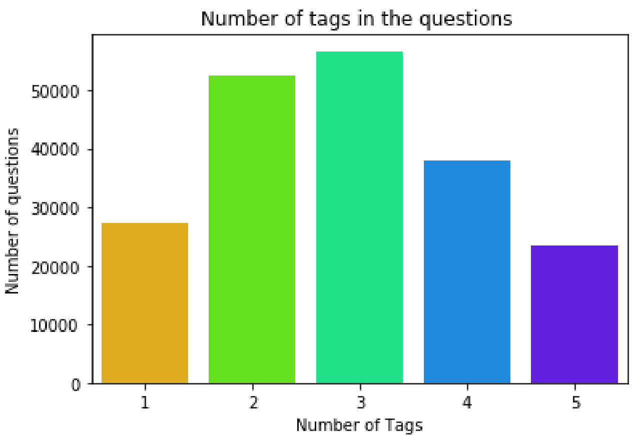
In [0]:

```
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: 2.888060
```

In [0]:

```
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

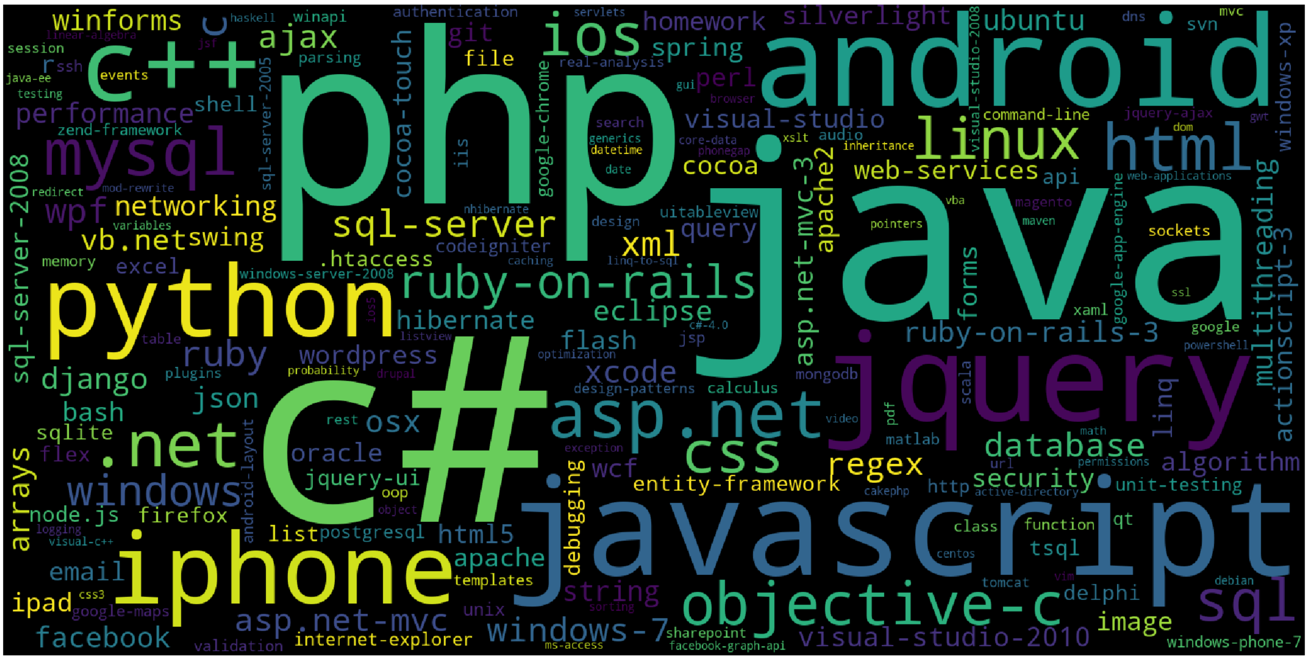
3.2.5 Most Frequent Tags

In [0]:

```
# 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 : 0:00:05.572168
```

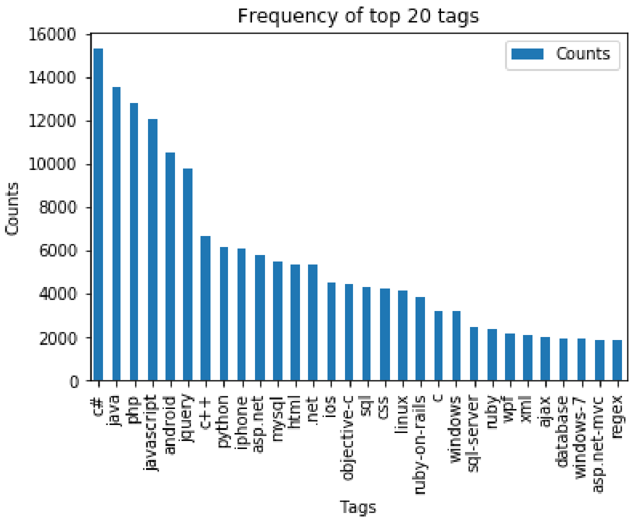
Observations:
A look at the word cloud shows that "c#", "java", "php", "asp.net", "javascript", "c++" are some of the most frequent tags.

3.2.6 The top 20 tags

In [0]:

```
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. Java, Android, IOS, Linux and windows are among the top most frequent operating systems.

3.3 Cleaning and preprocessing of Questions

3.3.1 Preprocessing

1. Sample 0.3M 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 [0]:

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

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

In [0]:

```
#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
```

In [0]:

```
# http://www.sqlitetutorial.net/sqlite-delete/
# https://stackoverflow.com/questions/2279706/select-random-row-from-a-sqlite-table
start = datetime.now()
read_db = 'train_no_dup.db'
write_db = 'Processed.db'
if os.path.isfile(read_db):
    conn_r = create_connection(read_db)
    if conn_r is not None:
        reader =conn_r.cursor()
        reader.execute("SELECT Title, Body, Tags From no_dup_train ORDER BY RANDOM() LIMIT
300000;")

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

```
Tables in the databse:
QuestionsProcessed
Cleared All the rows
Time taken to run this cell : 0:00:08.790824
```

we create a new data base to store the sampled and preprocessed questions

In [0]:

```
#http://www.bernzilla.com/2008/05/13/selecting-a-random-row-from-an-sqlite-table/
import nltk
nltk.download('punkt')

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], 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')

    question=str(title)+" "+str(question)
    question=re.sub(r'[^A-Za-z]+',' ',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)

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)
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
Avg. length of questions(Title+Body) before processing: 1151
Avg. length of questions(Title+Body) after processing: 328
Percent of questions containing code: 56
Time taken to run this cell : 0:05:10.829583
```

In [0]:

```
# dont forget to close the connections, or else you will end up with locks
conn_r.commit()
conn_w.commit()
conn_r.close()
conn_w.close()
```

In [0]:

```
if os.path.isfile(write_db):
```

```
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
====================================================================================================

('media player error decreas seek bar volum decreas progress increas increas volum real state problem media player audiomanag decreas volum sound seekbar decreas progress increas increas sound volum state pleas help sort problem nthank advanc',)
----------------------------------------------------------------------------------------------------
('mount amazon bucket directori freebsd find solut mount amazon bucket exist directori freebsd ni suspect need instal port configur find info',)
----------------------------------------------------------------------------------------------------
('unit test updatemodel method within mvc updat recent modifi view post control json object rather formcollect order get unit test control work set formvalu provid dictionari object stop updatemodel method throw nre result unit test howev simpli feel like right thing insight rework wo uld great appreci thank advanc',)
----------------------------------------------------------------------------------------------------
('problem macport pick wrong python previous instal python lion updat instal instal python packag use macport instal fail follow messag happen instal mercuri tri set default python activ one use m ake macport use version python want abl tell version place need use',)
----------------------------------------------------------------------------------------------------
('equival asp net login control java web framework world java web framework function come close as p net login control recommend way provid login authent new user registr etc java web world reusabl librari imagin everyon roll come asp net tri figur get thing done java thank',)
----------------------------------------------------------------------------------------------------
('stream data httprespons consol write consol applic need receiv larg amount data tri code like co de need wait entir respons write data consol recod stream data consol receiv thank',)
----------------------------------------------------------------------------------------------------
('make jira number field read describ titl look smart safe effici way set number field jira read s hort list approach guid plugin use attempt achiev instal deploy behaviour plugin result form permi ss error jira set basic edit field non writeabl investig revealv known issu fix anytim soon gone o ption jira exist field behaviour simpli offer option set field read hide option field need visibl potenti option would creat new screen scheme simpli exclud field edit screen associ new screen sch eme current project would littl disast mani project depend share henc make field read admin writeabl would much better solut instanc regard custom field ni creat post function workflow current project increas custom number field increment evertim issu task bug reopen essenc track nu mber reopen bring read requir develop abl chang valu field would throw statist alway help knowledg would great appreci thank time answer',)
----------------------------------------------------------------------------------------------------
('mac os termin select devic figur ni quit beginn termin problem brows file cd know brows differ d evic hard drive bootcamp partit exampl thank help edit need command line sinc refit shell',)
----------------------------------------------------------------------------------------------------
('disabl littl touch keyboard window edit control window version tablet support small keyboard ico n appear edit control get focus touch touch keyboard pop way disabl rather inconveni touch keyboard want disabl certain edit control code ie look window set giel',)
----------------------------------------------------------------------------------------------------
```

In [0]:

```
#Taking 1 Million entries to a dataframe.
write_db = 'Processed.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 [0]:

```
preprocessed_data.head()
```

Out[0]:

	question	tags
0	string also static string creation within meth...	java string
1	media player error decreas seek bar volum decr...	android mediaplayer android-audiomanager
2	mount amazon bucket directori freebsd find sol...	mount freebsd amazon-s3
3	unit test updatemodel method within mvc updat ...	asp.net-mvc unit-testing asp.net-mvc-3
4	problem macport pick wrong python previous ins...	python osx macports

In [0]:

```
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 : 197749
number of dimensions : 2
```

4. Machine Learning Models

4.1 Converting tags for multilabel problems

X	y1	y2	y3	y4
x1	0	1	1	0
x1	1	0	0	0
x1	0	1	0	0

In [0]:

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

We will sample the number of tags instead considering all of them (due to limitation of computing power)

In [0]:

```
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))
```

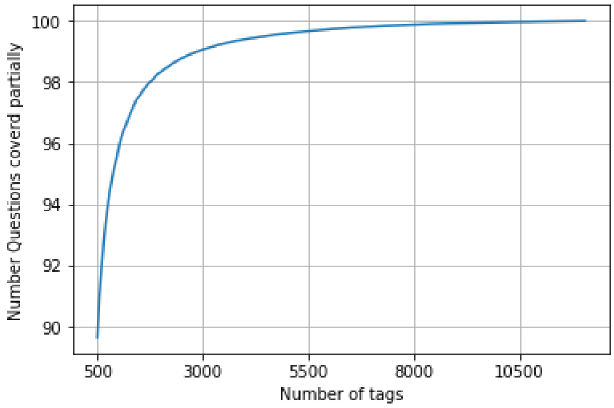
In [0]:

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

In [0]:

```
fig, ax = plt.subplots()
ax.plot(questions_explained)
```

```
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 50(it covers 90% of the tags)
print("with ",5500,"tags we are covering ",questions_explained[50],"% of questions")
```




```
with  5500 tags we are covering  99.062 % of questions
```

In [0]:

```
multilabel_yx = tags_to_choose(5500)
print("number of questions that are not covered :", questions_explained_fn(5500),"out of ", total_qs)
```

```
number of questions that are not covered : 1855 out of  197749
```

In [0]:

```
print("Number of tags in sample :", multilabel_y.shape[1])
print("number of tags taken :", multilabel_yx.shape[1],"(",(multilabel_yx.shape[1]/multilabel_y.shape[1])*100,"%)")
```

```
Number of tags in sample : 23686
number of tags taken : 5500 ( 23.220467786878324 %)
```

We consider top 15% tags which covers 99% of the questions

4.2 Split the data into test and train (80:20)

In [0]:

```
total_size=preprocessed_data.shape[0]
train_size=int(0.80*total_size)

x_train=preprocessed_data.head(train_size)
x_test=preprocessed_data.tail(total_size - train_size)

y_train = multilabel_yx[0:train_size,:]
y_test = multilabel_yx[train_size:total_size,:]
```

In [0]:

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

```
Number of data points in train data : (158199, 5500)
Number of data points in test data : (39550, 5500)
```

4.3 Featurizing data

In [0]:

```
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=(1,3)
x_tfidf_train_multilabel = vectorizer.fit_transform(x_train['question'])
x_tfidf_test_multilabel = vectorizer.transform(x_test['question'])
print("Time taken to run this cell :", datetime.now() - start)
```

```
Time taken to run this cell : 0:01:44.519934
```

In [0]:

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

```
Dimensions of train data X: (158199, 88660) Y : (158199, 5500)
Dimensions of test data X: (39550, 88660) Y: (39550, 5500)
```

In [0]:

```
# https://www.analyticsvidhya.com/blog/2017/08/introduction-to-multi-label-classification/
#https://stats.stackexchange.com/questions/117796/scikit-multi-label-classification
# classifier = LabelPowerset(GaussianNB())
"""
from skmultilearn.adapt import MLkNN
classifier = MLkNN(k=21)

# train
classifier.fit(x_train_multilabel, y_train)

# predict
predictions = classifier.predict(x_test_multilabel)
print(accuracy_score(y_test,predictions))
print(metrics.f1_score(y_test, predictions, average = 'macro'))
print(metrics.f1_score(y_test, predictions, average = 'micro'))
print(metrics.hamming_loss(y_test,predictions))

"""
# we are getting memory error because the multilearn package
# is trying to convert the data into dense matrix
# ---------------------------------------------------------------------------
#MemoryError                               Traceback (most recent call last)
#<ipython-input-170-f0e7c7f3e0be> in <module>()
#----> classifier.fit(x_train_multilabel, y_train)
```

Out[0]:

```
"\nfrom skmultilearn.adapt import MLkNN\nclassifier = MLkNN(k=21)\n\n#
train\nclassifier.fit(x_train_multilabel, y_train)\n\n# predict\npredictions =
classifier.predict(x_test_multilabel)\nprint(accuracy_score(y_test,predictions))\nprint(metrics.f1_
e(y_test, predictions, average = 'macro'))\nprint(metrics.f1_score(y_test, predictions, average =
'micro'))\nprint(metrics.hamming_loss(y_test,predictions))\n\n"
```

4.5 Modeling with less data points (80k data points) and more weight to title and 500 tags only.

In [0]:

```
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
```

questionsProcessed

In [0]:

```
# 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 = 60000
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 80000;")
        # for selecting random points
        #reader.execute("SELECT Title, Body, Tags From no_dup_train ORDER BY RANDOM() LIMIT 
80000;")

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

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

4.5.1 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 [0]:

```
#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)

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)
```

```
Avg. length of questions(Title+Body) before processing: 1130
Avg. length of questions(Title+Body) after processing: 411
Percent of questions containing code: 54
Time taken to run this cell : 0:02:57.503441
```

In [0]:

```
# 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 [0]:

```
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
====================================================================================================

('control lego wedo devic control lego wedo devic control lego wedo devic anyon written api lego w
edo devic c found python api hope someon done .net https github.com itdanih wedomor tri write use
libusb c know much usb python thank',)
----------------------------------------------------------------------------------------------------
('dialog get recreat orient chang dialog get recreat orient chang dialog get recreat orient chang
call use showdialog dialog long click menu id problem leak window orient chang activ get re-creat
```

```
never call oncreatedialog onpreparedialog know make differ activ within tab insight possibl caus m
assiv appreci',)
----------------------------------------------------------------------------------------------------
('dynam chang local vs2010 setup project dynam chang local vs2010 setup project dynam chang local v
s2010 setup project would like know chang local vs2008 setup project dynam scenario like oper
system langaug french user click setup.ex screen come french english langaug creat setup project k
eep local properti english show screen english chang local properti french show screen french want
dynam consid machin langaug display accordig',)
----------------------------------------------------------------------------------------------------
('encod sent data work encod sent data work encod sent data work got littl chatbox everyth work ex
cept send special latin char like xc3 xa4 xc3 xb6 xc3 xbc post send input soon enter press place b
odi header figur put header -- vbulletin templat server side insertshout logic tri use utf-8 chars
et luck also post help use kind ajax pull chatbox content work like charm special char direct inse
rt db phpmyadmin show correct problem insert char databas dump get var first entri point alreadi m
ess problem jqueri php new header',)
----------------------------------------------------------------------------------------------------
('file array array tree file array array tree file array array tree use creat program execut
command line captur output text file output huge pain figur isnt much way document clearcas plugin
anyhow would like skip file use output consol ... output appear like want basic load tree appear d
irectori list ... could sortabl easi tell latest version particular file directori one problem mul
tipl instanc directori file version particular file count may differ branch version ... troubl sli
ght experienc quit comprehend load array array neat go keep associ onlin exampl tree view find
hard code string dynam string anyon experi know trick cant decid visual studio line edit best use
split directori use ... later point get figur want re-send data clearcas via command prompt auto c
heckout associ file ... part seem easier point view ... post code close loop lan exampl treeview s
cratch head dotnetperl array tree d.morton msdn',)
----------------------------------------------------------------------------------------------------
('googl map locat base address googl map locat base address googl map locat base address im use go
ogl map applic work fine provid longitud latitud nbut display locat base address string',)
----------------------------------------------------------------------------------------------------
('insert custom field typo3 dam modul custom locat insert custom field typo3 dam modul custom
locat insert custom field typo3 dam modul custom locat introduc custom field dam modul work fine w
ant display custom field overview tab first tab edit document appear last tab ext tables.php line
add field dam modul',)
----------------------------------------------------------------------------------------------------
('preserv case use re.ignorecas .sub preserv case use re.ignorecas .sub preserv case use
re.ignorecas .sub return nmi name roger shrubber arrang design sell shrubberi like return origin c
ase name roger shrubber arrang design sell shrubberi sorri total noob help would great appreci',)
----------------------------------------------------------------------------------------------------
('itemcontainergenerator.containerfromitem return null itemcontainergenerator.containerfromitem re
turn null itemcontainergenerator.containerfromitem return null bit weird behavior seem work iter i
tem listbox.itemssourc properti seem get contain expect see listboxitem return get null idea bit c
ode use itemssourc current set dictionari contain number kvps',)
----------------------------------------------------------------------------------------------------
```

Saving Preprocessed data to a Database

In [0]:

```
#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 [0]:

```
preprocessed_data.head()
```

Out[0]:

	question	tags
0	access local session bean expos resteasi inter...	ejb resteasy
1	control lego wedo devic control lego wedo devi...	c# api lego
2	dialog get recreat orient chang dialog get rec...	android dialog display orientation
3	dynam chang local vs2010 setup project dynam c...	c# visual-studio-2010 localization

4	encod sent data work encod sent data work encod...	php jquery ajax encoding
	question	tags

In [0]:

```
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 : 79999
number of dimensions : 2
```

Converting string Tags to multilable output variables

In [0]:

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

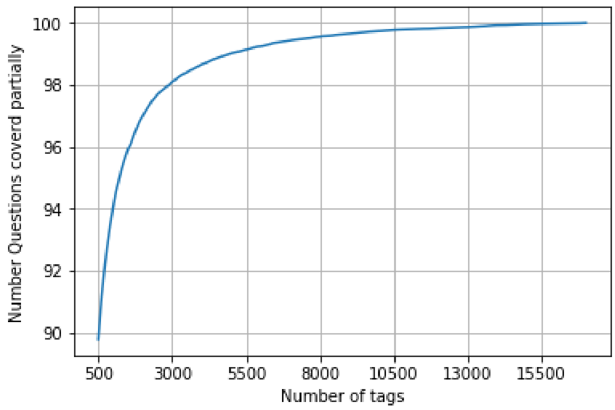
Selecting 500 Tags

In [0]:

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

In [0]:

```
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  99.136 % of questions
with  500 tags we are covering  89.772 % of questions
```

In [0]:

```
# 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 : 8182 out of  79999
```

In [0]:

```
x_train=preprocessed_data.head(train_datasize)
x_test=preprocessed_data.tail(preprocessed_data.shape[0] - 60000)

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

In [0]:

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

```
Number of data points in train data : (60000, 500)
Number of data points in test data : (19999, 500)
```

4.5.2 Featurizing data with BOW vectorizer

In [0]:

```
start = datetime.now()
vectorizer = CountVectorizer(min_df=0.00009, max_features=200000,ngram_range=(1,3))
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 : 0:00:44.179529
```

In [0]:

```
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: (60000, 99490) Y : (60000, 500)
Dimensions of test data X: (19999, 99490) Y: (19999, 500)
```

In [0]:

```
x_train_multilabel.shape
```

Out[0]:

```
(60000, 99490)
```

4.5.3 Applying SGD Logistic Regression with OneVsRest Classifier

In [69]:

```
import warnings
warnings.filterwarnings("ignore")
from sklearn.model_selection import GridSearchCV
start = datetime.now()
parameters={'estimator__alpha': [10**-5, 10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1]}
start = datetime.now()
clf = OneVsRestClassifier(SGDClassifier(loss='log', penalty='l1'))
gd_clf = GridSearchCV(estimator = clf, param_grid=parameters, cv=None, verbose=10, scoring='f1_micr
o',n_jobs=-1)
gd_clf.fit(x_train_multilabel, y_train)
best_alpha = gd_clf.best_estimator_.get_params()['estimator__alpha']
print('value of alpha after hyperparameter tuning : ',best_alpha)
print("Time taken to run this cell :", datetime.now() - start)
```

```
Fitting 3 folds for each of 7 candidates, totalling 21 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed: 19.3min
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed: 43.2min
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed: 79.5min
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed: 95.7min
[Parallel(n_jobs=-1)]: Done  21 out of  21 | elapsed: 114.8min remaining:    0.0s
[Parallel(n_jobs=-1)]: Done  21 out of  21 | elapsed: 114.8min finished
```

```
value of alpha after hyperparameter tuning :  0.001
Time taken to run this cell : 2:05:24.447594
```

In [71]:

```python
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)


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 : 0.1807590379518976
Hamming loss  0.0033051652582629133
Micro-average quality numbers
Precision: 0.5708, Recall: 0.3176, F1-measure: 0.4081
Macro-average quality numbers
Precision: 0.4132, Recall: 0.2529, F1-measure: 0.2931
              precision    recall  f1-score   support

           0       0.41      0.09      0.15      1691
           1       0.85      0.71      0.77      1192
           2       0.78      0.34      0.47      1435
           3       0.77      0.40      0.52      1275
           4       0.74      0.48      0.58      1105
           5       0.78      0.59      0.68       947
           6       0.57      0.16      0.25       621
           7       0.73      0.38      0.50       580
           8       0.42      0.11      0.18       606
           9       0.39      0.25      0.31       585
          10       0.68      0.37      0.48       716
          11       0.84      0.62      0.71       621
          12       0.68      0.48      0.56       417
          13       0.36      0.15      0.21       475
          14       0.53      0.21      0.30       416
          15       0.75      0.59      0.66       454
          16       0.54      0.28      0.37       522
          17       0.48      0.19      0.27       397
          18       0.34      0.04      0.07       313
          19       0.27      0.08      0.12       403
          20       0.79      0.58      0.67       354
          21       0.55      0.36      0.44       241
          22       0.80      0.81      0.80       158
          23       0.82      0.59      0.69       292
          24       0.44      0.30      0.36       172
```

25	0.79	0.34	0.48	260
26	0.58	0.68	0.63	141
27	0.27	0.13	0.17	185
28	0.38	0.20	0.26	218
29	0.79	0.61	0.69	178
30	0.42	0.44	0.43	214
31	0.59	0.44	0.50	170
32	0.60	0.30	0.40	219
33	0.41	0.43	0.42	176
34	0.62	0.42	0.50	163
35	0.62	0.24	0.35	127
36	0.55	0.20	0.29	161
37	0.59	0.42	0.49	158
38	0.85	0.46	0.59	72
39	0.57	0.26	0.35	97
40	0.77	0.27	0.40	90
41	0.00	0.00	0.00	76
42	0.76	0.49	0.59	152
43	0.26	0.10	0.14	81
44	0.45	0.18	0.26	157
45	0.62	0.60	0.61	154
46	0.26	0.22	0.24	121
47	0.54	0.45	0.49	126
48	0.61	0.79	0.69	115
49	0.50	0.43	0.46	68
50	0.27	0.49	0.35	90
51	0.20	0.08	0.12	123
52	0.22	0.07	0.10	133
53	0.75	0.77	0.76	60
54	0.43	0.35	0.38	100
55	0.22	0.09	0.13	89
56	0.42	0.18	0.25	115
57	0.72	0.58	0.64	71
58	0.47	0.38	0.42	66
59	0.60	0.41	0.48	98
60	0.26	0.28	0.27	101
61	0.51	0.28	0.36	92
62	0.17	0.14	0.15	114
63	0.93	0.64	0.76	107
64	0.58	0.46	0.51	90
65	0.50	0.17	0.25	78
66	0.40	0.39	0.39	100
67	0.80	0.53	0.64	91
68	0.91	0.50	0.64	96
69	0.86	0.72	0.78	124
70	0.21	0.12	0.15	86
71	0.43	0.19	0.27	119
72	0.52	0.52	0.52	88
73	0.48	0.25	0.33	93
74	0.70	0.65	0.68	78
75	0.16	0.16	0.16	58
76	0.58	0.55	0.56	62
77	0.00	0.00	0.00	99
78	0.61	0.58	0.59	92
79	0.03	0.01	0.01	100
80	0.91	0.55	0.69	56
81	0.67	0.02	0.04	92
82	0.51	0.35	0.42	88
83	0.51	0.25	0.34	96
84	0.82	0.71	0.76	75
85	0.95	0.78	0.86	50
86	0.32	0.13	0.19	75
87	0.81	0.63	0.71	67
88	0.79	0.77	0.78	39
89	0.39	0.17	0.23	54
90	0.29	0.03	0.05	73
91	0.60	0.16	0.25	93
92	0.70	0.47	0.56	68
93	0.47	0.36	0.41	58
94	0.35	0.19	0.25	74
95	0.89	0.56	0.68	45
96	0.07	0.03	0.04	78
97	0.28	0.05	0.09	91
98	0.36	0.27	0.31	60
99	0.58	0.24	0.34	80
100	0.67	0.58	0.62	45
101	0.40	0.06	0.11	33

102	0.77	0.71	0.74	28
103	0.55	0.42	0.48	57
104	0.53	0.08	0.15	95
105	0.45	0.33	0.38	39
106	0.35	0.33	0.34	51
107	0.82	0.43	0.56	87
108	0.09	0.04	0.05	54
109	0.53	0.34	0.41	53
110	0.61	0.51	0.56	53
111	0.54	0.54	0.54	59
112	0.00	0.00	0.00	78
113	0.58	0.17	0.27	40
114	0.32	0.08	0.13	72
115	0.59	0.54	0.56	63
116	0.75	0.61	0.68	70
117	0.48	0.18	0.26	56
118	0.33	0.15	0.20	41
119	0.00	0.00	0.00	42
120	0.36	0.11	0.17	74
121	0.28	0.22	0.25	49
122	0.88	0.42	0.57	52
123	0.82	0.36	0.50	75
124	0.11	0.02	0.03	51
125	0.08	0.04	0.05	56
126	0.81	0.31	0.45	55
127	0.44	0.15	0.23	52
128	0.17	0.11	0.13	72
129	0.35	0.11	0.16	57
130	0.23	0.35	0.28	20
131	0.80	0.84	0.82	61
132	0.38	0.22	0.28	49
133	0.32	0.12	0.18	66
134	0.58	0.10	0.17	71
135	0.55	0.48	0.51	44
136	0.90	0.70	0.79	37
137	0.32	0.22	0.26	50
138	0.71	0.09	0.16	54
139	0.21	0.11	0.14	47
140	0.28	0.17	0.21	58
141	0.92	0.23	0.37	47
142	0.00	0.00	0.00	44
143	0.31	0.41	0.35	27
144	0.92	0.38	0.53	32
145	0.00	0.00	0.00	17
146	0.31	0.40	0.35	35
147	0.80	0.58	0.67	64
148	0.11	0.04	0.06	48
149	0.08	0.10	0.09	30
150	0.18	0.16	0.17	32
151	0.40	0.05	0.09	39
152	0.00	0.00	0.00	51
153	0.24	0.27	0.25	52
154	0.80	0.18	0.30	44
155	0.91	0.48	0.62	42
156	0.63	0.34	0.44	64
157	0.23	0.23	0.23	22
158	0.23	0.25	0.24	36
159	0.94	0.30	0.45	57
160	0.40	0.07	0.12	56
161	0.34	0.39	0.36	44
162	0.53	0.41	0.46	39
163	0.63	0.63	0.63	52
164	0.25	0.12	0.16	43
165	0.27	0.11	0.15	37
166	0.27	0.18	0.21	51
167	0.70	0.32	0.44	66
168	0.25	0.08	0.12	38
169	0.33	0.25	0.29	56
170	0.00	0.00	0.00	53
171	0.14	0.07	0.09	42
172	0.57	0.53	0.55	38
173	0.00	0.00	0.00	51
174	0.82	0.74	0.78	57
175	0.22	0.32	0.26	34
176	0.60	0.46	0.52	39
177	0.16	0.13	0.14	39
178	0.14	0.06	0.08	36

179	0.00	0.00	0.00	44
180	0.94	0.62	0.74	47
181	0.11	0.12	0.11	26
182	0.00	0.00	0.00	43
183	0.31	0.50	0.38	40
184	0.23	0.18	0.20	50
185	1.00	0.03	0.05	40
186	0.41	0.29	0.34	45
187	0.76	0.33	0.46	40
188	0.00	0.00	0.00	26
189	0.38	0.22	0.28	50
190	0.00	0.00	0.00	66
191	0.22	0.06	0.09	34
192	0.75	0.50	0.60	24
193	0.25	0.15	0.19	26
194	0.36	0.55	0.44	31
195	0.90	0.29	0.43	63
196	0.83	0.72	0.77	40
197	0.71	0.47	0.56	51
198	0.33	0.23	0.27	40
199	0.00	0.00	0.00	48
200	0.18	0.16	0.17	38
201	0.23	0.13	0.17	45
202	0.17	0.04	0.06	26
203	0.54	0.61	0.58	31
204	0.97	0.62	0.76	53
205	0.62	0.14	0.23	35
206	0.72	0.52	0.60	25
207	0.00	0.00	0.00	39
208	0.59	0.28	0.38	36
209	0.25	0.04	0.07	46
210	0.24	0.10	0.14	42
211	0.80	0.73	0.76	64
212	0.38	0.16	0.23	37
213	0.61	0.44	0.51	43
214	0.35	0.35	0.35	17
215	0.97	0.64	0.77	53
216	0.89	0.53	0.66	59
217	0.40	0.16	0.23	38
218	0.37	0.48	0.42	46
219	0.95	0.61	0.74	33
220	0.32	0.21	0.25	48
221	0.00	0.00	0.00	14
222	0.30	0.14	0.19	21
223	0.00	0.00	0.00	33
224	0.00	0.00	0.00	50
225	0.93	0.54	0.68	26
226	0.61	0.71	0.66	49
227	0.46	0.18	0.26	34
228	0.90	0.86	0.88	21
229	0.29	0.16	0.21	37
230	0.07	0.04	0.05	26
231	0.45	0.18	0.26	28
232	0.00	0.00	0.00	42
233	1.00	0.61	0.76	51
234	0.15	0.05	0.08	40
235	0.33	0.32	0.32	19
236	0.54	0.23	0.32	31
237	0.68	0.81	0.74	32
238	0.75	0.77	0.76	31
239	0.00	0.00	0.00	32
240	0.39	0.54	0.45	26
241	0.30	0.18	0.22	34
242	0.21	0.28	0.24	18
243	0.27	0.09	0.13	35
244	0.33	0.05	0.08	22
245	0.38	0.12	0.19	24
246	0.55	0.30	0.39	40
247	0.33	0.35	0.34	17
248	0.64	0.43	0.52	37
249	0.20	0.08	0.12	24
250	0.08	0.06	0.06	18
251	0.00	0.00	0.00	33
252	0.85	0.64	0.73	44
253	0.25	0.08	0.12	37
254	0.00	0.00	0.00	36
255	0.74	0.38	0.50	37

256	0.00	0.00	0.00	24
257	0.00	0.00	0.00	42
258	0.00	0.00	0.00	20
259	0.76	0.48	0.59	27
260	0.60	0.58	0.59	26
261	0.25	0.10	0.14	30
262	0.96	0.83	0.89	30
263	0.24	0.18	0.21	38
264	0.43	0.09	0.15	32
265	0.44	0.15	0.23	46
266	0.10	0.21	0.13	19
267	0.21	0.17	0.19	18
268	0.00	0.00	0.00	31
269	0.69	0.48	0.56	23
270	0.19	0.15	0.17	39
271	0.00	0.00	0.00	27
272	1.00	0.54	0.70	28
273	0.57	0.33	0.42	36
274	0.25	0.03	0.06	31
275	0.00	0.00	0.00	30
276	0.67	1.00	0.80	16
277	0.23	0.24	0.23	21
278	0.00	0.00	0.00	36
279	0.25	0.04	0.06	28
280	0.20	0.06	0.10	31
281	0.05	0.05	0.05	22
282	0.25	0.19	0.21	27
283	0.57	0.21	0.31	19
284	0.33	0.13	0.19	31
285	0.60	0.14	0.23	21
286	0.00	0.00	0.00	20
287	0.88	0.61	0.72	23
288	0.00	0.00	0.00	13
289	0.25	0.02	0.04	46
290	0.45	0.13	0.20	39
291	0.00	0.00	0.00	31
292	1.00	0.17	0.29	30
293	1.00	0.43	0.60	14
294	0.63	0.55	0.59	31
295	0.13	0.11	0.12	38
296	0.12	0.08	0.10	25
297	0.00	0.00	0.00	28
298	0.28	0.29	0.29	17
299	0.00	0.00	0.00	15
300	0.33	0.29	0.31	7
301	0.93	0.61	0.74	23
302	0.67	0.18	0.29	11
303	0.92	0.58	0.71	19
304	0.00	0.00	0.00	29
305	0.08	0.04	0.05	25
306	0.27	0.16	0.20	25
307	0.76	0.57	0.65	23
308	0.57	0.22	0.32	18
309	0.00	0.00	0.00	17
310	0.33	0.15	0.21	27
311	0.75	0.38	0.50	24
312	0.35	0.23	0.28	30
313	0.12	0.26	0.16	31
314	0.42	0.39	0.41	28
315	0.65	0.57	0.61	30
316	0.26	0.18	0.21	28
317	0.00	0.00	0.00	21
318	0.00	0.00	0.00	8
319	0.21	0.33	0.26	12
320	0.00	0.00	0.00	35
321	0.71	0.36	0.48	28
322	0.38	0.29	0.32	21
323	0.50	0.05	0.09	20
324	0.46	0.48	0.47	23
325	0.00	0.00	0.00	25
326	0.83	0.67	0.74	15
327	0.65	0.48	0.55	27
328	0.81	0.45	0.58	29
329	0.95	0.50	0.66	38
330	0.12	0.15	0.14	27
331	0.23	0.19	0.21	26
332	1.00	0.03	0.06	32

333	0.00	0.00	0.00	17
334	0.14	0.06	0.09	32
335	0.30	0.47	0.36	30
336	0.59	0.62	0.61	16
337	0.00	0.00	0.00	19
338	0.00	0.00	0.00	24
339	0.00	0.00	0.00	33
340	0.30	0.18	0.22	17
341	0.55	0.64	0.59	36
342	0.80	0.50	0.62	16
343	0.00	0.00	0.00	15
344	0.19	0.12	0.15	24
345	0.42	0.24	0.30	21
346	0.07	0.05	0.06	20
347	0.38	0.46	0.41	13
348	0.00	0.00	0.00	19
349	0.60	0.60	0.60	20
350	0.18	0.08	0.11	26
351	0.22	0.09	0.13	22
352	0.20	0.05	0.08	19
353	0.00	0.00	0.00	30
354	0.08	0.04	0.05	25
355	0.92	0.52	0.67	23
356	0.08	0.04	0.06	24
357	0.55	0.38	0.45	29
358	0.00	0.00	0.00	25
359	0.82	0.69	0.75	39
360	0.80	0.17	0.29	23
361	0.11	0.19	0.14	21
362	0.00	0.00	0.00	17
363	0.11	0.33	0.17	9
364	0.62	0.19	0.29	26
365	0.86	0.78	0.82	23
366	0.55	0.20	0.29	30
367	0.59	0.62	0.61	16
368	0.50	0.25	0.33	4
369	0.00	0.00	0.00	34
370	1.00	0.42	0.59	19
371	0.54	0.30	0.39	23
372	0.00	0.00	0.00	34
373	0.30	0.11	0.16	27
374	0.21	0.21	0.21	19
375	0.64	0.30	0.41	23
376	0.67	0.67	0.67	18
377	0.00	0.00	0.00	25
378	0.45	0.38	0.41	24
379	0.00	0.00	0.00	21
380	0.00	0.00	0.00	17
381	0.00	0.00	0.00	21
382	0.25	0.09	0.13	22
383	0.00	0.00	0.00	34
384	0.64	0.35	0.45	20
385	0.67	0.20	0.31	10
386	0.08	0.06	0.07	17
387	0.00	0.00	0.00	20
388	0.37	0.45	0.41	22
389	0.00	0.00	0.00	18
390	0.38	0.38	0.38	24
391	0.67	0.12	0.20	17
392	0.29	0.09	0.14	22
393	0.00	0.00	0.00	4
394	0.38	0.46	0.41	13
395	0.00	0.00	0.00	22
396	0.25	0.12	0.16	17
397	0.20	0.05	0.08	20
398	0.80	0.50	0.62	8
399	0.00	0.00	0.00	15
400	0.00	0.00	0.00	19
401	0.00	0.00	0.00	12
402	0.00	0.00	0.00	19
403	0.50	0.25	0.33	20
404	0.00	0.00	0.00	17
405	0.12	0.07	0.09	15
406	0.50	0.22	0.30	23
407	1.00	0.25	0.40	12
408	0.02	0.05	0.02	21
409	0.00	0.00	0.00	15

410	0.23	0.50	0.32	12
411	0.64	0.35	0.45	20
412	0.47	0.36	0.41	25
413	0.50	0.11	0.18	18
414	0.00	0.00	0.00	32
415	1.00	0.07	0.14	27
416	0.17	0.07	0.10	15
417	1.00	0.11	0.20	9
418	0.33	0.30	0.32	10
419	1.00	0.12	0.21	25
420	0.62	0.57	0.59	14
421	0.13	0.29	0.18	14
422	0.00	0.00	0.00	20
423	0.42	0.26	0.32	19
424	0.00	0.00	0.00	10
425	0.00	0.00	0.00	32
426	1.00	0.03	0.05	36
427	0.00	0.00	0.00	22
428	0.17	0.04	0.06	27
429	0.00	0.00	0.00	17
430	0.60	0.33	0.43	18
431	0.50	0.29	0.36	21
432	0.50	0.15	0.24	13
433	0.11	0.09	0.10	11
434	0.00	0.00	0.00	3
435	0.07	0.04	0.05	26
436	0.29	0.37	0.33	19
437	0.00	0.00	0.00	20
438	0.29	0.31	0.30	16
439	0.33	0.04	0.07	27
440	0.09	0.04	0.05	27
441	0.00	0.00	0.00	23
442	0.00	0.00	0.00	14
443	0.00	0.00	0.00	19
444	0.17	0.05	0.07	21
445	0.19	0.24	0.21	17
446	0.00	0.00	0.00	7
447	0.53	0.40	0.46	20
448	0.83	0.45	0.59	11
449	1.00	0.12	0.21	17
450	0.83	0.43	0.57	23
451	0.00	0.00	0.00	22
452	0.70	0.41	0.52	17
453	0.40	0.15	0.22	13
454	1.00	0.08	0.15	12
455	0.07	0.06	0.06	18
456	1.00	0.11	0.19	19
457	0.50	0.04	0.08	23
458	1.00	0.35	0.52	17
459	0.00	0.00	0.00	8
460	0.40	0.18	0.25	11
461	0.92	0.61	0.73	18
462	0.71	0.38	0.50	13
463	0.00	0.00	0.00	10
464	0.05	0.08	0.06	24
465	0.40	0.24	0.30	17
466	0.46	0.23	0.31	26
467	0.23	0.50	0.32	14
468	0.22	0.57	0.32	7
469	0.58	0.55	0.56	20
470	0.33	0.15	0.21	20
471	0.86	0.43	0.57	14
472	0.43	0.13	0.20	23
473	0.17	0.08	0.11	24
474	0.92	0.44	0.59	25
475	0.50	0.19	0.28	21
476	0.57	0.20	0.30	20
477	0.40	0.14	0.21	14
478	0.00	0.00	0.00	7
479	1.00	0.22	0.36	23
480	0.50	0.19	0.28	21
481	0.86	0.18	0.30	33
482	0.09	0.05	0.06	22
483	0.00	0.00	0.00	16
484	0.00	0.00	0.00	15
485	0.33	0.18	0.23	17
486	0.00	0.00	0.00	10

```
         487       0.56      0.28      0.37        18
         488       1.00      0.15      0.27        13
         489       0.40      0.14      0.21        14
         490       0.38      0.50      0.43        10
         491       0.80      0.24      0.36        17
         492       0.22      0.07      0.11        29
         493       0.00      0.00      0.00        16
         494       0.29      0.44      0.35         9
         495       0.00      0.00      0.00        13
         496       1.00      0.42      0.59        26
         497       0.82      0.53      0.64        17
         498       0.00      0.00      0.00        18
         499       0.70      0.37      0.48        19

   micro avg       0.57      0.32      0.41     35876
   macro avg       0.41      0.25      0.29     35876
weighted avg       0.53      0.32      0.38     35876
 samples avg       0.39      0.31      0.32     35876

Time taken to run this cell : 0:09:39.988095
```

4.5.4 Applying Logistic Regression with OneVsRest Classifier

In [73]:

```
import warnings
warnings.filterwarnings("ignore")
from sklearn.model_selection import GridSearchCV
start = datetime.now()
parameters={'estimator__C': [10**-5, 10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1]}
lg_clf = OneVsRestClassifier(LogisticRegression(class_weight='balanced', penalty='l1'))
lg_gd_clf = GridSearchCV(estimator = lg_clf, param_grid=parameters, cv=3, verbose=10, scoring='f1_m
icro',n_jobs=15)
lg_gd_clf.fit(x_train_multilabel, y_train)
lg_best_alpha = lg_gd_clf.best_estimator_.get_params()['estimator__C']
print('value of alpha after hyperparameter tuning : ',lg_best_alpha)
print("Time taken to run this cell :", datetime.now() - start)
```

```
Fitting 3 folds for each of 7 candidates, totalling 21 fits
```

```
[Parallel(n_jobs=15)]: Using backend LokyBackend with 15 concurrent workers.
[Parallel(n_jobs=15)]: Done   4 out of  21 | elapsed: 44.5min remaining: 189.3min
[Parallel(n_jobs=15)]: Done   7 out of  21 | elapsed: 144.6min remaining: 289.1min
[Parallel(n_jobs=15)]: Done  10 out of  21 | elapsed: 238.4min remaining: 262.2min
[Parallel(n_jobs=15)]: Done  13 out of  21 | elapsed: 261.0min remaining: 160.6min
[Parallel(n_jobs=15)]: Done  16 out of  21 | elapsed: 272.4min remaining: 85.1min
[Parallel(n_jobs=15)]: Done  19 out of  21 | elapsed: 277.8min remaining: 29.2min
[Parallel(n_jobs=15)]: Done  21 out of  21 | elapsed: 280.6min finished
```

```
value of alpha after hyperparameter tuning :  10
Time taken to run this cell : 5:27:56.890444
```

In [74]:

```
start = datetime.now()
classifier_2 =
OneVsRestClassifier(LogisticRegression(penalty='l1',C=lg_best_alpha,class_weight='balanced'),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 : 0.14595729786489325
Hamming loss  0.004105705285264263
Micro-average quality numbers
Precision: 0.4322, Recall: 0.4602, F1-measure: 0.4458
Macro-average quality numbers
Precision: 0.3770, Recall: 0.4125, F1-measure: 0.3880
              precision    recall  f1-score   support

           0       0.34      0.38      0.36      1691
           1       0.84      0.79      0.81      1192
           2       0.55      0.52      0.53      1435
           3       0.52      0.54      0.53      1275
           4       0.59      0.58      0.58      1105
           5       0.68      0.65      0.66       947
           6       0.33      0.37      0.35       621
           7       0.48      0.48      0.48       580
           8       0.26      0.25      0.26       606
           9       0.32      0.36      0.34       585
          10       0.55      0.51      0.53       716
          11       0.73      0.67      0.70       621
          12       0.60      0.62      0.61       417
          13       0.32      0.33      0.32       475
          14       0.32      0.41      0.36       416
          15       0.61      0.61      0.61       454
          16       0.45      0.43      0.44       522
          17       0.42      0.45      0.44       397
          18       0.24      0.26      0.25       313
          19       0.26      0.24      0.25       403
          20       0.66      0.64      0.65       354
          21       0.39      0.40      0.39       241
          22       0.90      0.75      0.82       158
          23       0.72      0.69      0.71       292
          24       0.44      0.42      0.43       172
          25       0.58      0.49      0.53       260
          26       0.66      0.74      0.70       141
          27       0.21      0.23      0.22       185
          28       0.32      0.39      0.35       218
          29       0.67      0.71      0.69       178
          30       0.48      0.63      0.54       214
          31       0.52      0.51      0.51       170
          32       0.50      0.52      0.51       219
          33       0.38      0.51      0.44       176
          34       0.43      0.43      0.43       163
          35       0.31      0.38      0.34       127
          36       0.29      0.31      0.30       161
          37       0.44      0.51      0.47       158
          38       0.48      0.60      0.53        72
          39       0.33      0.38      0.36        97
          40       0.30      0.23      0.26        90
          41       0.13      0.16      0.15        76
          42       0.68      0.68      0.68       152
          43       0.18      0.21      0.19        81
          44       0.41      0.38      0.39       157
          45       0.64      0.62      0.63       154
          46       0.19      0.31      0.24       121
          47       0.47      0.48      0.47       126
          48       0.87      0.82      0.84       115
          49       0.47      0.51      0.49        68
          50       0.51      0.50      0.50        90
          51       0.38      0.32      0.34       123
          52       0.20      0.19      0.19       133
          53       0.69      0.73      0.71        60
          54       0.41      0.55      0.47       100
          55       0.20      0.24      0.22        89
          56       0.23      0.28      0.25       115
          57       0.58      0.65      0.61        71
          58       0.44      0.52      0.47        66
```

59	0.43	0.59	0.50	98
60	0.30	0.37	0.33	101
61	0.32	0.39	0.35	92
62	0.20	0.24	0.22	114
63	0.74	0.69	0.71	107
64	0.49	0.58	0.53	90
65	0.28	0.27	0.28	78
66	0.38	0.47	0.42	100
67	0.56	0.70	0.62	91
68	0.66	0.55	0.60	96
69	0.75	0.71	0.73	124
70	0.24	0.33	0.27	86
71	0.36	0.27	0.31	119
72	0.48	0.58	0.52	88
73	0.46	0.55	0.50	93
74	0.66	0.69	0.68	78
75	0.18	0.24	0.21	58
76	0.55	0.66	0.60	62
77	0.14	0.15	0.14	99
78	0.54	0.65	0.59	92
79	0.09	0.09	0.09	100
80	0.80	0.73	0.77	56
81	0.32	0.26	0.29	92
82	0.32	0.32	0.32	88
83	0.62	0.67	0.64	96
84	0.64	0.72	0.68	75
85	0.81	0.76	0.78	50
86	0.23	0.32	0.27	75
87	0.61	0.67	0.64	67
88	0.74	0.74	0.74	39
89	0.23	0.26	0.24	54
90	0.07	0.07	0.07	73
91	0.28	0.27	0.28	93
92	0.57	0.60	0.59	68
93	0.53	0.53	0.53	58
94	0.23	0.23	0.23	74
95	0.66	0.56	0.60	45
96	0.13	0.15	0.14	78
97	0.26	0.26	0.26	91
98	0.38	0.45	0.41	60
99	0.30	0.35	0.33	80
100	0.58	0.67	0.62	45
101	0.26	0.42	0.33	33
102	0.67	0.71	0.69	28
103	0.40	0.49	0.44	57
104	0.20	0.18	0.19	95
105	0.34	0.59	0.43	39
106	0.25	0.37	0.30	51
107	0.68	0.56	0.62	87
108	0.10	0.11	0.10	54
109	0.37	0.45	0.41	53
110	0.66	0.55	0.60	53
111	0.63	0.86	0.73	59
112	0.24	0.21	0.22	78
113	0.33	0.38	0.35	40
114	0.26	0.24	0.25	72
115	0.40	0.57	0.47	63
116	0.75	0.83	0.79	70
117	0.26	0.39	0.31	56
118	0.10	0.12	0.11	41
119	0.04	0.05	0.04	42
120	0.19	0.18	0.18	74
121	0.24	0.35	0.28	49
122	0.72	0.75	0.74	52
123	0.60	0.57	0.59	75
124	0.16	0.14	0.15	51
125	0.18	0.23	0.20	56
126	0.51	0.47	0.49	55
127	0.32	0.31	0.31	52
128	0.49	0.46	0.47	72
129	0.23	0.21	0.22	57
130	0.42	0.55	0.48	20
131	0.76	0.85	0.81	61
132	0.23	0.20	0.22	49
133	0.23	0.20	0.21	66
134	0.51	0.44	0.47	71
135	0.56	0.55	0.55	44

136	0.85	0.76	0.80	37
137	0.40	0.38	0.39	50
138	0.31	0.35	0.33	54
139	0.27	0.43	0.33	47
140	0.16	0.19	0.17	58
141	0.49	0.57	0.53	47
142	0.16	0.14	0.15	44
143	0.27	0.44	0.34	27
144	0.59	0.53	0.56	32
145	0.23	0.41	0.30	17
146	0.31	0.51	0.39	35
147	0.69	0.66	0.67	64
148	0.07	0.10	0.08	48
149	0.14	0.13	0.14	30
150	0.23	0.28	0.25	32
151	0.11	0.15	0.13	39
152	0.02	0.02	0.02	51
153	0.26	0.35	0.30	52
154	0.56	0.52	0.54	44
155	0.92	0.57	0.71	42
156	0.51	0.61	0.56	64
157	0.19	0.32	0.24	22
158	0.26	0.47	0.34	36
159	0.64	0.53	0.58	57
160	0.16	0.20	0.17	56
161	0.50	0.55	0.52	44
162	0.53	0.51	0.52	39
163	0.54	0.37	0.44	52
164	0.26	0.23	0.24	43
165	0.08	0.11	0.09	37
166	0.28	0.29	0.29	51
167	0.62	0.61	0.62	66
168	0.22	0.34	0.27	38
169	0.39	0.46	0.42	56
170	0.29	0.23	0.25	53
171	0.11	0.17	0.13	42
172	0.44	0.50	0.47	38
173	0.27	0.27	0.27	51
174	0.93	0.75	0.83	57
175	0.44	0.56	0.49	34
176	0.51	0.46	0.49	39
177	0.22	0.18	0.20	39
178	0.22	0.31	0.26	36
179	0.06	0.05	0.05	44
180	0.83	0.74	0.79	47
181	0.14	0.27	0.19	26
182	0.02	0.02	0.02	43
183	0.26	0.25	0.25	40
184	0.23	0.32	0.26	50
185	0.17	0.17	0.17	40
186	0.35	0.42	0.38	45
187	0.41	0.40	0.41	40
188	0.22	0.27	0.24	26
189	0.27	0.26	0.26	50
190	0.31	0.20	0.24	66
191	0.21	0.24	0.22	34
192	0.78	0.58	0.67	24
193	0.46	0.50	0.48	26
194	0.46	0.61	0.53	31
195	0.57	0.51	0.54	63
196	0.67	0.65	0.66	40
197	0.45	0.49	0.47	51
198	0.25	0.30	0.27	40
199	0.14	0.10	0.12	48
200	0.21	0.29	0.24	38
201	0.41	0.40	0.40	45
202	0.23	0.19	0.21	26
203	0.44	0.48	0.46	31
204	0.93	0.79	0.86	53
205	0.47	0.43	0.45	35
206	0.59	0.68	0.63	25
207	0.08	0.10	0.09	39
208	0.45	0.47	0.46	36
209	0.09	0.09	0.09	46
210	0.30	0.38	0.33	42
211	0.87	0.83	0.85	64
212	0.43	0.57	0.49	37

213	0.51	0.44	0.48	43
214	0.21	0.41	0.28	17
215	0.84	0.79	0.82	53
216	0.90	0.78	0.84	59
217	0.34	0.34	0.34	38
218	0.36	0.67	0.47	46
219	0.85	0.52	0.64	33
220	0.31	0.31	0.31	48
221	0.33	0.57	0.42	14
222	0.23	0.33	0.27	21
223	0.14	0.18	0.16	33
224	0.30	0.28	0.29	50
225	0.86	0.73	0.79	26
226	0.60	0.65	0.63	49
227	0.41	0.47	0.44	34
228	0.89	0.76	0.82	21
229	0.38	0.43	0.41	37
230	0.11	0.15	0.13	26
231	0.48	0.71	0.57	28
232	0.14	0.12	0.13	42
233	0.86	0.86	0.86	51
234	0.19	0.20	0.19	40
235	0.31	0.42	0.36	19
236	0.59	0.65	0.62	31
237	0.66	0.84	0.74	32
238	0.68	0.90	0.78	31
239	0.52	0.47	0.49	32
240	0.32	0.62	0.42	26
241	0.15	0.18	0.16	34
242	0.26	0.56	0.36	18
243	0.11	0.09	0.10	35
244	0.20	0.23	0.21	22
245	0.16	0.21	0.18	24
246	0.56	0.55	0.56	40
247	0.16	0.24	0.19	17
248	0.57	0.62	0.60	37
249	0.25	0.38	0.30	24
250	0.23	0.33	0.27	18
251	0.27	0.27	0.27	33
252	0.82	0.70	0.76	44
253	0.12	0.16	0.14	37
254	0.30	0.31	0.30	36
255	0.55	0.57	0.56	37
256	0.25	0.25	0.25	24
257	0.11	0.10	0.10	42
258	0.07	0.05	0.06	20
259	0.61	0.85	0.71	27
260	0.61	0.65	0.63	26
261	0.33	0.37	0.35	30
262	0.81	0.83	0.82	30
263	0.39	0.53	0.45	38
264	0.48	0.34	0.40	32
265	0.50	0.52	0.51	46
266	0.28	0.47	0.35	19
267	0.14	0.22	0.17	18
268	0.06	0.06	0.06	31
269	0.61	0.74	0.67	23
270	0.10	0.10	0.10	39
271	0.00	0.00	0.00	27
272	0.85	0.79	0.81	28
273	0.53	0.47	0.50	36
274	0.30	0.32	0.31	31
275	0.37	0.23	0.29	30
276	0.83	0.94	0.88	16
277	0.29	0.52	0.37	21
278	0.21	0.28	0.24	36
279	0.26	0.32	0.29	28
280	0.13	0.13	0.13	31
281	0.20	0.09	0.13	22
282	0.14	0.22	0.17	27
283	0.53	0.53	0.53	19
284	0.21	0.26	0.23	31
285	0.20	0.24	0.22	21
286	0.03	0.05	0.04	20
287	0.71	0.65	0.68	23
288	0.08	0.15	0.11	13
289	0.17	0.13	0.15	46

290	0.27	0.23	0.25	39
291	0.08	0.10	0.09	31
292	0.50	0.33	0.40	30
293	0.67	0.57	0.62	14
294	0.55	0.68	0.61	31
295	0.10	0.08	0.09	38
296	0.10	0.20	0.13	25
297	0.00	0.00	0.00	28
298	0.30	0.65	0.41	17
299	0.00	0.00	0.00	15
300	0.36	0.57	0.44	7
301	0.88	0.61	0.72	23
302	0.47	0.64	0.54	11
303	0.94	0.84	0.89	19
304	0.15	0.14	0.14	29
305	0.09	0.12	0.10	25
306	0.26	0.40	0.31	25
307	0.67	0.78	0.72	23
308	0.35	0.44	0.39	18
309	0.24	0.41	0.30	17
310	0.40	0.52	0.45	27
311	0.44	0.58	0.50	24
312	0.19	0.27	0.22	30
313	0.43	0.65	0.52	31
314	0.35	0.39	0.37	28
315	0.60	0.70	0.65	30
316	0.14	0.14	0.14	28
317	0.08	0.10	0.09	21
318	0.04	0.12	0.06	8
319	0.47	0.67	0.55	12
320	0.22	0.23	0.22	35
321	0.34	0.46	0.39	28
322	0.39	0.43	0.41	21
323	0.15	0.20	0.17	20
324	0.46	0.52	0.49	23
325	0.19	0.20	0.20	25
326	0.82	0.93	0.87	15
327	0.57	0.78	0.66	27
328	0.64	0.62	0.63	29
329	0.81	0.68	0.74	38
330	0.28	0.48	0.36	27
331	0.24	0.42	0.31	26
332	0.58	0.56	0.57	32
333	0.00	0.00	0.00	17
334	0.15	0.16	0.15	32
335	0.45	0.60	0.51	30
336	0.34	0.62	0.44	16
337	0.05	0.05	0.05	19
338	0.00	0.00	0.00	24
339	0.10	0.06	0.08	33
340	0.44	0.65	0.52	17
341	0.54	0.61	0.57	36
342	0.72	0.81	0.76	16
343	0.21	0.33	0.26	15
344	0.10	0.08	0.09	24
345	0.35	0.52	0.42	21
346	0.07	0.15	0.10	20
347	0.33	0.54	0.41	13
348	0.00	0.00	0.00	19
349	0.58	0.75	0.65	20
350	0.21	0.23	0.22	26
351	0.18	0.27	0.21	22
352	0.41	0.58	0.48	19
353	0.40	0.13	0.20	30
354	0.00	0.00	0.00	25
355	0.68	0.83	0.75	23
356	0.19	0.29	0.23	24
357	0.63	0.66	0.64	29
358	0.10	0.12	0.11	25
359	0.70	0.67	0.68	39
360	0.43	0.39	0.41	23
361	0.26	0.24	0.25	21
362	0.16	0.24	0.19	17
363	0.30	0.33	0.32	9
364	0.67	0.46	0.55	26
365	0.72	0.78	0.75	23
366	0.62	0.67	0.65	30

367	0.33	0.56	0.42	16
368	0.29	0.50	0.36	4
369	0.12	0.09	0.10	34
370	0.80	0.84	0.82	19
371	0.38	0.43	0.41	23
372	0.06	0.03	0.04	34
373	0.21	0.26	0.23	27
374	0.15	0.21	0.18	19
375	0.34	0.48	0.40	23
376	0.88	0.78	0.82	18
377	0.41	0.36	0.38	25
378	0.38	0.75	0.50	24
379	0.07	0.05	0.06	21
380	0.05	0.06	0.05	17
381	0.22	0.19	0.21	21
382	0.53	0.45	0.49	22
383	0.17	0.09	0.12	34
384	0.42	0.40	0.41	20
385	0.24	0.50	0.32	10
386	0.00	0.00	0.00	17
387	0.22	0.10	0.14	20
388	0.41	0.50	0.45	22
389	0.05	0.06	0.05	18
390	0.37	0.62	0.46	24
391	0.42	0.29	0.34	17
392	0.00	0.00	0.00	22
393	0.40	0.50	0.44	4
394	0.37	0.54	0.44	13
395	0.12	0.14	0.13	22
396	0.30	0.35	0.32	17
397	0.23	0.35	0.27	20
398	0.33	0.50	0.40	8
399	0.33	0.33	0.33	15
400	0.10	0.16	0.12	19
401	0.22	0.42	0.29	12
402	0.32	0.32	0.32	19
403	0.36	0.40	0.38	20
404	0.00	0.00	0.00	17
405	0.04	0.07	0.05	15
406	0.36	0.35	0.36	23
407	0.75	0.75	0.75	12
408	0.07	0.14	0.09	21
409	0.07	0.13	0.09	15
410	0.91	0.83	0.87	12
411	0.47	0.45	0.46	20
412	0.66	0.76	0.70	25
413	0.14	0.17	0.15	18
414	0.00	0.00	0.00	32
415	0.56	0.37	0.44	27
416	0.23	0.40	0.29	15
417	0.35	0.67	0.46	9
418	0.09	0.20	0.13	10
419	0.84	0.64	0.73	25
420	0.52	0.79	0.63	14
421	0.31	0.36	0.33	14
422	0.11	0.05	0.07	20
423	0.36	0.53	0.43	19
424	0.42	0.50	0.45	10
425	0.24	0.22	0.23	32
426	0.70	0.64	0.67	36
427	0.16	0.27	0.20	22
428	0.22	0.26	0.24	27
429	0.24	0.47	0.32	17
430	0.50	0.50	0.50	18
431	0.52	0.52	0.52	21
432	0.30	0.46	0.36	13
433	0.20	0.18	0.19	11
434	0.00	0.00	0.00	3
435	0.19	0.19	0.19	26
436	0.23	0.37	0.28	19
437	0.34	0.50	0.41	20
438	0.23	0.38	0.29	16
439	0.08	0.04	0.05	27
440	0.25	0.26	0.25	27
441	0.39	0.30	0.34	23
442	0.38	0.43	0.40	14
443	0.13	0.21	0.16	19

```
        444       0.20      0.33      0.25        21
        445       0.15      0.18      0.16        17
        446       0.05      0.14      0.07         7
        447       0.39      0.45      0.42        20
        448       0.67      0.91      0.77        11
        449       0.58      0.65      0.61        17
        450       0.79      0.65      0.71        23
        451       0.67      0.55      0.60        22
        452       0.70      0.82      0.76        17
        453       0.33      0.46      0.39        13
        454       0.64      0.58      0.61        12
        455       0.14      0.22      0.17        18
        456       0.23      0.37      0.28        19
        457       0.21      0.22      0.21        23
        458       0.79      0.65      0.71        17
        459       0.22      0.25      0.24         8
        460       0.21      0.27      0.24        11
        461       0.87      0.72      0.79        18
        462       0.58      0.54      0.56        13
        463       0.12      0.30      0.17        10
        464       0.50      0.25      0.33        24
        465       0.47      0.41      0.44        17
        466       0.43      0.50      0.46        26
        467       0.24      0.36      0.29        14
        468       0.12      0.43      0.19         7
        469       0.52      0.80      0.63        20
        470       0.38      0.50      0.43        20
        471       0.57      0.57      0.57        14
        472       0.30      0.43      0.36        23
        473       0.10      0.12      0.11        24
        474       0.62      0.40      0.49        25
        475       0.10      0.10      0.10        21
        476       0.47      0.40      0.43        20
        477       0.42      0.36      0.38        14
        478       0.06      0.14      0.08         7
        479       0.30      0.26      0.28        23
        480       0.45      0.62      0.52        21
        481       0.71      0.67      0.69        33
        482       0.38      0.36      0.37        22
        483       0.00      0.00      0.00        16
        484       0.12      0.13      0.12        15
        485       0.38      0.47      0.42        17
        486       0.00      0.00      0.00        10
        487       0.50      0.50      0.50        18
        488       0.47      0.54      0.50        13
        489       0.10      0.14      0.12        14
        490       0.27      0.60      0.37        10
        491       0.50      0.47      0.48        17
        492       0.40      0.28      0.33        29
        493       0.21      0.25      0.23        16
        494       0.88      0.78      0.82         9
        495       0.35      0.46      0.40        13
        496       0.88      0.88      0.88        26
        497       0.61      0.65      0.63        17
        498       0.26      0.28      0.27        18
        499       0.50      0.63      0.56        19

   micro avg       0.43      0.46      0.45     35876
   macro avg       0.38      0.41      0.39     35876
weighted avg       0.45      0.46      0.45     35876
 samples avg       0.41      0.44      0.39     35876

Time taken to run this cell : 0:34:05.534044
```

4.5.5 Applying SGD Linear SVM with OneVsRest Classifier

In [75]:

```
import warnings
warnings.filterwarnings("ignore")
from sklearn.model_selection import GridSearchCV
start = datetime.now()
parameters={'estimator__alpha': [10**-5, 10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1]}
svm_clf = OneVsRestClassifier(SGDClassifier(loss='hinge', penalty='l1'))
svm_gd_clf = GridSearchCV(estimator = svm_clf, param_grid=parameters, cv=3, verbose=10, scoring='f1
```

```
_micro',n_jobs=-1)
svm_gd_clf.fit(x_train_multilabel, y_train)
svm_best_alpha = svm_gd_clf.best_estimator_.get_params()['estimator__alpha']
print('value of alpha after hyperparameter tuning : ',best_alpha)
print("Time taken to run this cell :", datetime.now() - start)
```

```
Fitting 3 folds for each of 7 candidates, totalling 21 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed: 18.9min
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed: 38.7min
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed: 75.9min
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed: 95.9min
[Parallel(n_jobs=-1)]: Done  21 out of  21 | elapsed: 125.4min remaining:    0.0s
[Parallel(n_jobs=-1)]: Done  21 out of  21 | elapsed: 125.4min finished
```

```
value of alpha after hyperparameter tuning :  0.001
Time taken to run this cell : 2:14:08.598435
```

In [77]:

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


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


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

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

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

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

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

```
Accuracy : 0.17640882044102205
Hamming loss  0.003335066753337667
Micro-average quality numbers
Precision: 0.5632, Recall: 0.3136, F1-measure: 0.4029
Macro-average quality numbers
Precision: 0.3556, Recall: 0.2535, F1-measure: 0.2729
              precision    recall  f1-score   support

           0       0.39      0.05      0.09      1691
           1       0.86      0.68      0.76      1192
           2       0.82      0.33      0.47      1435
           3       0.66      0.44      0.53      1275
           4       0.70      0.55      0.62      1105
           5       0.83      0.58      0.68       947
           6       0.47      0.14      0.21       621
           7       0.78      0.27      0.40       580
           8       0.54      0.13      0.21       606
           9       0.39      0.05      0.09       585
          10       0.71      0.37      0.48       716
          11       0.81      0.64      0.71       621
          12       0.65      0.48      0.56       417
          13       0.50      0.11      0.18       475
          14       0.44      0.28      0.34       416
          15       0.78      0.62      0.69       454
```

16	0.51	0.28	0.36	322
17	0.60	0.18	0.28	397
18	0.23	0.04	0.07	313
19	0.12	0.01	0.03	403
20	0.77	0.64	0.70	354
21	0.52	0.32	0.39	241
22	0.69	0.79	0.74	158
23	0.78	0.66	0.72	292
24	0.55	0.28	0.37	172
25	0.79	0.41	0.54	260
26	0.62	0.70	0.66	141
27	0.00	0.00	0.00	185
28	0.08	0.01	0.02	218
29	0.69	0.70	0.69	178
30	0.36	0.45	0.40	214
31	0.56	0.45	0.50	170
32	0.44	0.33	0.38	219
33	0.38	0.44	0.41	176
34	0.64	0.37	0.47	163
35	0.65	0.29	0.40	127
36	0.48	0.34	0.40	161
37	0.43	0.52	0.47	158
38	0.71	0.50	0.59	72
39	0.72	0.22	0.33	97
40	0.60	0.29	0.39	90
41	0.00	0.00	0.00	76
42	0.81	0.61	0.69	152
43	0.15	0.19	0.17	81
44	0.58	0.20	0.30	157
45	0.63	0.62	0.62	154
46	0.00	0.00	0.00	121
47	0.63	0.44	0.52	126
48	0.80	0.80	0.80	115
49	0.61	0.51	0.56	68
50	0.51	0.48	0.49	90
51	0.70	0.13	0.22	123
52	0.21	0.13	0.16	133
53	0.59	0.67	0.62	60
54	0.50	0.41	0.45	100
55	0.00	0.00	0.00	89
56	0.00	0.00	0.00	115
57	0.64	0.59	0.61	71
58	0.34	0.39	0.36	66
59	0.40	0.52	0.45	98
60	0.38	0.24	0.29	101
61	0.49	0.25	0.33	92
62	0.22	0.09	0.12	114
63	0.88	0.72	0.79	107
64	0.47	0.49	0.48	90
65	0.00	0.00	0.00	78
66	0.37	0.43	0.40	100
67	0.67	0.64	0.65	91
68	0.92	0.49	0.64	96
69	0.82	0.69	0.75	124
70	0.00	0.00	0.00	86
71	0.59	0.24	0.35	119
72	0.45	0.50	0.48	88
73	0.88	0.08	0.14	93
74	0.46	0.55	0.50	78
75	0.00	0.00	0.00	58
76	0.51	0.69	0.59	62
77	0.00	0.00	0.00	99
78	0.70	0.58	0.63	92
79	0.18	0.02	0.04	100
80	0.75	0.68	0.71	56
81	1.00	0.05	0.10	92
82	0.52	0.38	0.44	88
83	0.54	0.21	0.30	96
84	0.72	0.61	0.66	75
85	0.82	0.66	0.73	50
86	0.00	0.00	0.00	75
87	0.76	0.63	0.69	67
88	0.79	0.77	0.78	39
89	0.00	0.00	0.00	54
90	0.29	0.08	0.13	73
91	0.53	0.11	0.18	93
92	0.60	0.59	0.59	68
93	0.60	0.43	0.50	58

93	0.60	0.43	0.50	58
94	0.00	0.00	0.00	74
95	0.62	0.56	0.59	45
96	0.02	0.01	0.02	78
97	0.00	0.00	0.00	91
98	0.44	0.32	0.37	60
99	0.35	0.31	0.33	80
100	0.57	0.60	0.59	45
101	0.00	0.00	0.00	33
102	0.79	0.79	0.79	28
103	0.38	0.60	0.46	57
104	0.36	0.08	0.14	95
105	0.18	0.28	0.22	39
106	0.26	0.24	0.25	51
107	0.78	0.40	0.53	87
108	0.00	0.00	0.00	54
109	0.45	0.47	0.46	53
110	0.62	0.47	0.54	53
111	0.57	0.56	0.56	59
112	0.00	0.00	0.00	78
113	0.52	0.28	0.36	40
114	0.00	0.00	0.00	72
115	0.57	0.41	0.48	63
116	0.55	0.70	0.62	70
117	0.28	0.27	0.28	56
118	0.11	0.20	0.14	41
119	0.00	0.00	0.00	42
120	0.00	0.00	0.00	74
121	0.20	0.33	0.25	49
122	0.86	0.58	0.69	52
123	0.76	0.49	0.60	75
124	0.00	0.00	0.00	51
125	0.15	0.12	0.13	56
126	0.86	0.33	0.47	55
127	0.36	0.08	0.13	52
128	0.00	0.00	0.00	72
129	0.18	0.12	0.15	57
130	0.17	0.25	0.20	20
131	0.78	0.77	0.78	61
132	0.25	0.02	0.04	49
133	0.00	0.00	0.00	66
134	0.90	0.13	0.22	71
135	0.69	0.41	0.51	44
136	0.77	0.62	0.69	37
137	0.33	0.02	0.04	50
138	0.53	0.30	0.38	54
139	0.55	0.13	0.21	47
140	0.28	0.17	0.21	58
141	0.59	0.28	0.38	47
142	0.20	0.02	0.04	44
143	0.28	0.48	0.35	27
144	0.83	0.47	0.60	32
145	0.00	0.00	0.00	17
146	0.43	0.43	0.43	35
147	0.60	0.53	0.56	64
148	0.10	0.02	0.03	48
149	0.14	0.17	0.15	30
150	0.00	0.00	0.00	32
151	0.18	0.05	0.08	39
152	0.00	0.00	0.00	51
153	0.00	0.00	0.00	52
154	1.00	0.30	0.46	44
155	0.92	0.52	0.67	42
156	0.53	0.61	0.57	64
157	0.15	0.18	0.16	22
158	0.31	0.33	0.32	36
159	0.49	0.56	0.52	57
160	0.00	0.00	0.00	56
161	0.41	0.50	0.45	44
162	0.47	0.56	0.51	39
163	0.51	0.58	0.54	52
164	0.00	0.00	0.00	43
165	0.20	0.05	0.09	37
166	0.00	0.00	0.00	51
167	0.41	0.30	0.35	66
168	0.00	0.00	0.00	38
169	0.33	0.25	0.28	56

170	0.00	0.00	0.00	53
171	0.00	0.00	0.00	42
172	0.48	0.53	0.50	38
173	0.22	0.16	0.18	51
174	0.93	0.67	0.78	57
175	0.41	0.47	0.44	34
176	0.44	0.28	0.34	39
177	0.20	0.03	0.05	39
178	0.25	0.08	0.12	36
179	0.00	0.00	0.00	44
180	0.71	0.64	0.67	47
181	0.05	0.08	0.06	26
182	0.00	0.00	0.00	43
183	0.41	0.33	0.36	40
184	0.00	0.00	0.00	50
185	0.00	0.00	0.00	40
186	0.43	0.27	0.33	45
187	0.81	0.33	0.46	40
188	0.06	0.04	0.05	26
189	0.29	0.24	0.26	50
190	0.22	0.03	0.05	66
191	0.19	0.15	0.16	34
192	0.79	0.46	0.58	24
193	0.15	0.31	0.20	26
194	0.43	0.58	0.49	31
195	0.74	0.40	0.52	63
196	0.78	0.62	0.69	40
197	0.62	0.45	0.52	51
198	0.23	0.28	0.25	40
199	0.04	0.06	0.05	48
200	0.00	0.00	0.00	38
201	0.43	0.40	0.41	45
202	0.33	0.08	0.12	26
203	0.54	0.45	0.49	31
204	0.94	0.64	0.76	53
205	0.25	0.17	0.20	35
206	0.42	0.64	0.51	25
207	0.00	0.00	0.00	39
208	0.67	0.22	0.33	36
209	0.00	0.00	0.00	46
210	0.25	0.02	0.04	42
211	0.88	0.72	0.79	64
212	0.25	0.27	0.26	37
213	0.54	0.49	0.51	43
214	0.21	0.24	0.22	17
215	0.93	0.75	0.83	53
216	0.88	0.59	0.71	59
217	0.12	0.13	0.13	38
218	0.29	0.41	0.34	46
219	0.96	0.67	0.79	33
220	1.00	0.02	0.04	48
221	0.38	0.21	0.27	14
222	0.10	0.10	0.10	21
223	0.00	0.00	0.00	33
224	0.00	0.00	0.00	50
225	0.86	0.46	0.60	26
226	0.57	0.55	0.56	49
227	0.31	0.38	0.34	34
228	0.94	0.76	0.84	21
229	0.41	0.38	0.39	37
230	0.00	0.00	0.00	26
231	0.39	0.39	0.39	28
232	0.00	0.00	0.00	42
233	0.97	0.73	0.83	51
234	0.00	0.00	0.00	40
235	0.17	0.47	0.25	19
236	0.55	0.19	0.29	31
237	0.67	0.75	0.71	32
238	0.49	0.58	0.53	31
239	0.00	0.00	0.00	32
240	0.34	0.38	0.36	26
241	0.00	0.00	0.00	34
242	0.14	0.28	0.18	18
243	1.00	0.06	0.11	35
244	0.31	0.18	0.23	22
245	0.19	0.21	0.20	24
246	0.41	0.35	0.38	40

247	0.20	0.18	0.19	17
248	0.53	0.73	0.61	37
249	0.10	0.17	0.12	24
250	0.00	0.00	0.00	18
251	0.17	0.06	0.09	33
252	0.83	0.66	0.73	44
253	0.00	0.00	0.00	37
254	0.00	0.00	0.00	36
255	0.79	0.30	0.43	37
256	0.00	0.00	0.00	24
257	0.00	0.00	0.00	42
258	0.00	0.00	0.00	20
259	0.53	0.85	0.66	27
260	0.56	0.35	0.43	26
261	0.00	0.00	0.00	30
262	0.95	0.67	0.78	30
263	1.00	0.05	0.10	38
264	0.33	0.03	0.06	32
265	0.57	0.09	0.15	46
266	0.33	0.26	0.29	19
267	1.00	0.06	0.11	18
268	0.00	0.00	0.00	31
269	0.68	0.65	0.67	23
270	0.00	0.00	0.00	39
271	0.00	0.00	0.00	27
272	1.00	0.57	0.73	28
273	0.44	0.42	0.43	36
274	0.00	0.00	0.00	31
275	0.00	0.00	0.00	30
276	0.82	0.88	0.85	16
277	0.21	0.29	0.24	21
278	0.43	0.17	0.24	36
279	0.00	0.00	0.00	28
280	0.00	0.00	0.00	31
281	0.00	0.00	0.00	22
282	0.00	0.00	0.00	27
283	0.20	0.21	0.21	19
284	0.00	0.00	0.00	31
285	0.25	0.05	0.08	21
286	0.00	0.00	0.00	20
287	0.88	0.65	0.75	23
288	0.00	0.00	0.00	13
289	0.00	0.00	0.00	46
290	0.15	0.10	0.12	39
291	0.00	0.00	0.00	31
292	0.50	0.27	0.35	30
293	0.19	0.36	0.25	14
294	0.44	0.55	0.49	31
295	0.00	0.00	0.00	38
296	0.11	0.16	0.13	25
297	0.00	0.00	0.00	28
298	0.43	0.59	0.50	17
299	0.00	0.00	0.00	15
300	1.00	0.29	0.44	7
301	1.00	0.35	0.52	23
302	0.00	0.00	0.00	11
303	0.92	0.63	0.75	19
304	0.00	0.00	0.00	29
305	0.00	0.00	0.00	25
306	0.28	0.20	0.23	25
307	0.76	0.70	0.73	23
308	0.19	0.17	0.18	18
309	0.00	0.00	0.00	17
310	0.43	0.37	0.40	27
311	0.36	0.54	0.43	24
312	0.00	0.00	0.00	30
313	0.38	0.32	0.35	31
314	0.26	0.36	0.30	28
315	0.52	0.47	0.49	30
316	0.25	0.04	0.06	28
317	0.00	0.00	0.00	21
318	0.00	0.00	0.00	8
319	0.00	0.00	0.00	12
320	0.00	0.00	0.00	35
321	0.52	0.43	0.47	28
322	0.17	0.19	0.18	21
323	0.00	0.00	0.00	20

324	0.50	0.30	0.38	23
325	0.00	0.00	0.00	25
326	0.79	0.73	0.76	15
327	0.54	0.52	0.53	27
328	0.53	0.31	0.39	29
329	0.89	0.45	0.60	38
330	0.32	0.22	0.26	27
331	0.18	0.23	0.20	26
332	0.54	0.22	0.31	32
333	0.00	0.00	0.00	17
334	0.00	0.00	0.00	32
335	0.44	0.53	0.48	30
336	0.28	0.56	0.38	16
337	0.00	0.00	0.00	19
338	0.00	0.00	0.00	24
339	0.00	0.00	0.00	33
340	0.25	0.29	0.27	17
341	0.63	0.33	0.44	36
342	0.72	0.81	0.76	16
343	0.00	0.00	0.00	15
344	0.14	0.17	0.15	24
345	0.30	0.43	0.35	21
346	0.00	0.00	0.00	20
347	0.39	0.54	0.45	13
348	0.00	0.00	0.00	19
349	0.31	0.65	0.42	20
350	0.19	0.15	0.17	26
351	0.13	0.18	0.15	22
352	0.50	0.37	0.42	19
353	0.00	0.00	0.00	30
354	0.00	0.00	0.00	25
355	0.54	0.65	0.59	23
356	0.00	0.00	0.00	24
357	0.52	0.55	0.53	29
358	0.00	0.00	0.00	25
359	0.71	0.44	0.54	39
360	0.64	0.30	0.41	23
361	1.00	0.05	0.09	21
362	0.00	0.00	0.00	17
363	0.20	0.33	0.25	9
364	1.00	0.31	0.47	26
365	0.67	0.87	0.75	23
366	0.67	0.33	0.44	30
367	0.29	0.50	0.36	16
368	0.33	0.25	0.29	4
369	0.00	0.00	0.00	34
370	1.00	0.47	0.64	19
371	0.50	0.26	0.34	23
372	0.00	0.00	0.00	34
373	0.24	0.19	0.21	27
374	0.08	0.16	0.10	19
375	0.35	0.39	0.37	23
376	0.63	0.67	0.65	18
377	0.04	0.04	0.04	25
378	0.36	0.33	0.35	24
379	0.00	0.00	0.00	21
380	0.00	0.00	0.00	17
381	0.33	0.05	0.08	21
382	0.11	0.14	0.12	22
383	0.06	0.03	0.04	34
384	0.38	0.15	0.21	20
385	0.00	0.00	0.00	10
386	0.00	0.00	0.00	17
387	0.00	0.00	0.00	20
388	0.41	0.41	0.41	22
389	0.00	0.00	0.00	18
390	0.30	0.29	0.30	24
391	0.00	0.00	0.00	17
392	0.00	0.00	0.00	22
393	0.00	0.00	0.00	4
394	0.50	0.46	0.48	13
395	0.00	0.00	0.00	22
396	0.27	0.35	0.31	17
397	0.00	0.00	0.00	20
398	0.67	0.25	0.36	8
399	0.00	0.00	0.00	15
400	0.00	0.00	0.00	19

401	0.18	0.17	0.17	12
402	0.00	0.00	0.00	19
403	0.33	0.35	0.34	20
404	0.00	0.00	0.00	17
405	0.00	0.00	0.00	15
406	0.33	0.35	0.34	23
407	1.00	0.17	0.29	12
408	0.00	0.00	0.00	21
409	0.00	0.00	0.00	15
410	0.90	0.75	0.82	12
411	0.50	0.35	0.41	20
412	0.43	0.40	0.42	25
413	0.19	0.17	0.18	18
414	0.00	0.00	0.00	32
415	0.90	0.33	0.49	27
416	0.33	0.20	0.25	15
417	0.25	0.11	0.15	9
418	0.10	0.10	0.10	10
419	0.56	0.36	0.44	25
420	0.16	0.43	0.24	14
421	1.00	0.14	0.25	14
422	0.00	0.00	0.00	20
423	0.21	0.42	0.28	19
424	0.00	0.00	0.00	10
425	0.00	0.00	0.00	32
426	0.00	0.00	0.00	36
427	0.00	0.00	0.00	22
428	0.33	0.15	0.21	27
429	0.18	0.12	0.14	17
430	0.57	0.72	0.63	18
431	0.42	0.48	0.44	21
432	0.18	0.23	0.20	13
433	0.00	0.00	0.00	11
434	0.00	0.00	0.00	3
435	0.00	0.00	0.00	26
436	0.38	0.16	0.22	19
437	0.00	0.00	0.00	20
438	1.00	0.06	0.12	16
439	0.17	0.07	0.10	27
440	0.25	0.04	0.06	27
441	0.00	0.00	0.00	23
442	0.50	0.07	0.12	14
443	0.06	0.05	0.06	19
444	0.00	0.00	0.00	21
445	0.00	0.00	0.00	17
446	0.00	0.00	0.00	7
447	0.45	0.50	0.48	20
448	0.80	0.36	0.50	11
449	0.20	0.06	0.09	17
450	0.73	0.70	0.71	23
451	1.00	0.18	0.31	22
452	0.48	0.71	0.57	17
453	0.25	0.15	0.19	13
454	0.28	0.42	0.33	12
455	0.00	0.00	0.00	18
456	0.00	0.00	0.00	19
457	0.00	0.00	0.00	23
458	0.88	0.41	0.56	17
459	0.00	0.00	0.00	8
460	0.09	0.09	0.09	11
461	0.80	0.67	0.73	18
462	0.42	0.77	0.54	13
463	0.00	0.00	0.00	10
464	0.75	0.12	0.21	24
465	0.40	0.12	0.18	17
466	0.55	0.42	0.48	26
467	0.50	0.14	0.22	14
468	0.11	0.43	0.18	7
469	0.47	0.40	0.43	20
470	0.25	0.10	0.14	20
471	0.80	0.29	0.42	14
472	0.15	0.09	0.11	23
473	0.10	0.08	0.09	24
474	1.00	0.24	0.39	25
475	0.00	0.00	0.00	21
476	0.33	0.05	0.09	20
477	0.30	0.21	0.25	14

```
         478       0.00      0.00      0.00         7
         479       0.67      0.35      0.46        23
         480       0.00      0.00      0.00        21
         481       0.92      0.33      0.49        33
         482       0.57      0.18      0.28        22
         483       0.00      0.00      0.00        16
         484       0.00      0.00      0.00        15
         485       0.00      0.00      0.00        17
         486       0.00      0.00      0.00        10
         487       0.12      0.06      0.08        18
         488       0.55      0.46      0.50        13
         489       0.00      0.00      0.00        14
         490       0.33      0.30      0.32        10
         491       0.22      0.65      0.33        17
         492       0.00      0.00      0.00        29
         493       0.00      0.00      0.00        16
         494       1.00      0.67      0.80         9
         495       0.00      0.00      0.00        13
         496       0.86      0.73      0.79        26
         497       1.00      0.29      0.45        17
         498       1.00      0.11      0.20        18
         499       0.73      0.42      0.53        19

   micro avg       0.56      0.31      0.40     35876
   macro avg       0.36      0.25      0.27     35876
weighted avg       0.50      0.31      0.36     35876
 samples avg       0.38      0.31      0.32     35876

Time taken to run this cell : 0:06:47.666859
```

In [1]:

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = [ "Model","vectorizer", "best-alpha", "Precision ","Recall","F1-score","Accuracy"]

x.add_row(["SGD(log)","BOW",0.001,0.5708,0.3176,0.4081,0.18])
x.add_row(["LOGISTIC REGR","BOW",10,0.4322,0.4602,0.4458,0.14])
x.add_row(["SGD(hinge) SVM","BOW",0.001,0.5632,0.3136,0.4029,0.17])
print(x)
```

```
+----------------+------------+------------+------------+--------+----------+----------+
|     Model      | vectorizer | best-alpha | Precision  | Recall | F1-score | Accuracy |
+----------------+------------+------------+------------+--------+----------+----------+
|    SGD(log)    |    BOW     |   0.001    |   0.5708   | 0.3176 |  0.4081  |   0.18   |
| LOGISTIC REGR  |    BOW     |     10     |   0.4322   | 0.4602 |  0.4458  |   0.14   |
| SGD(hinge) SVM |    BOW     |   0.001    |   0.5632   | 0.3136 |  0.4029  |   0.17   |
+----------------+------------+------------+------------+--------+----------+----------+
```

5. Observations

1. In this case study we performed with 0.2 million data points for EDA and 80k points for modeling because of low computational resources
2. In above analysis many of questions contains minimum 2-3 tags
3. Most frequent question tag is C# and it is programming language
4. And we used BOW text vectorizer for implementing models
5. We done hyper-parameter tuning for each model, it takes lot of time for tuning hyper parameter and logistic regression takes more time for computing hyper-parameter than other models
6. we took f1-micro is the performance metric
7. In this case study we see new classifier that is onevsrest classifier which is used when we have multi class labels
8. We done models with SGD with log-loss(logistic regressio) and SGD with hinge-loss(SVM) and pure Logistic regression
9. Out of 3 models SGD Logistic regression gives better results in Precision, Recall, and F1-score, but compare to all model accuracy it gives high accuracy ,How ever remaining 2 models also performed nearer to SGD logistic regression with some low accuracy and high recall ,but SGD logistic regression gives stable precision reasonable recall and f1-scores at last SGD models are performed with similar results and logistic too but low accuracy