Assignment-19

January 16, 2019

In [1]:
```
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import sqlite3
import csv
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from wordcloud import WordCloud
import re
import os
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
import numpy as np
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
import pickle
from sklearn.externals import joblib
```

In [17]:

1 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.
1.2 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.
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:
2.1.2 Example Data point

3. Exploratory Data Analysis

3.1 Data Loading and Cleaning
3.1.1 Using Pandas with SQLite to Load the data

In [2]:
```
import zipfile
archive = zipfile.ZipFile('Train.zip', 'r')
csvfile = archive.open('Train.csv')
```

In [3]:
```
#Creating db file from csv
#Learn SQL: https://www.w3schools.com/sql/default.asp
if not os.path.isfile('train.db'):
```

```
    start = datetime.now()
    disk_engine = create_engine('sqlite:///train.db')
    start = dt.datetime.now()
    chunksize = 180000
    j = 0
    index_start = 1
    for df in pd.read_csv(csvfile, names=['Id', 'Title', 'Body', 'Tags'], chunksize=chun
        df.index += index_start
        j+=1
        print('{} rows'.format(j*chunksize))
        df.to_sql('data', disk_engine, if_exists='append')
        index_start = df.index[-1] + 1
    print("Time taken to run this cell :", datetime.now() - start)
```

3.1.2 Counting the number of rows

```
In [4]: 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 genarat
```

```
Number of rows in the database :
 6034196
Time taken to count the number of rows : 0:00:31.207379
```

3.1.3 Checking for duplicates

```
In [5]: #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 da
            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 tra
```

```
Time taken to run this cell : 0:02:23.193506
```

```
In [6]: df_no_dup.head()
        # we can observe that there are duplicates
```

Out[6]:
```
                                               Title  \
0      Implementing Boundary Value Analysis of S...
1          Dynamic Datagrid Binding in Silverlight?
2          Dynamic Datagrid Binding in Silverlight?
3     java.lang.NoClassDefFoundError: javax/serv...
4     java.sql.SQLException:[Microsoft][ODBC Dri...

                                                Body  \
0  <pre><code>#include&lt;iostream&gt;\n#include&...
1  <p>I should do binding for datagrid dynamicall...
2  <p>I should do binding for datagrid dynamicall...
3  <p>I followed the guide in <a href="http://sta...
4  <p>I use the following code</p>\n\n<pre><code>...

                                Tags  cnt_dup
0                              c++ c        1
1          c# silverlight data-binding        1
2  c# silverlight data-binding columns        1
3                           jsp jstl        1
4                          java jdbc        2
```

In [7]:
```python
print("number of duplicate questions :", num_rows['count(*)'].values[0]- df_no_dup.shape
```

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

In [8]:
```python
# number of times each question appeared in our database
df_no_dup.cnt_dup.value_counts()
```

Out[8]:
```
1    2656284
2    1272336
3     277575
4         90
5         25
6          5
Name: cnt_dup, dtype: int64
```

In [9]:
```python
print(df_no_dup.head())
```

```
                                               Title  \
0      Implementing Boundary Value Analysis of S...
1          Dynamic Datagrid Binding in Silverlight?
2          Dynamic Datagrid Binding in Silverlight?
3     java.lang.NoClassDefFoundError: javax/serv...
4     java.sql.SQLException:[Microsoft][ODBC Dri...

                                                Body  \
0  <pre><code>#include&lt;iostream&gt;\n#include&...
1  <p>I should do binding for datagrid dynamicall...
```

```
2  <p>I should do binding for datagrid dynamicall...
3  <p>I followed the guide in <a href="http://sta...
4  <p>I use the following code</p>\n\n<pre><code>...

                                 Tags  cnt_dup
0                               c++ c        1
1          c# silverlight data-binding        1
2  c# silverlight data-binding columns        1
3                            jsp jstl        1
4                           java jdbc        2
```

In [10]:
```
start = datetime.now()
aa_count=[]
hh=[]
for j in range(len(df_no_dup)):
    tex=df_no_dup['Tags'][j]
    #print(tex)
    if tex is not None:
        #print("heyram")
        #start=datetime.now()
        hh.append(tex)
        text=len(tex.split(" ") )
        #print(text)
        aa_count.append(text)

print(len(aa_count))
aaa=pd.DataFrame(aa_count,columns=['tag_count'])
hhh=pd.DataFrame(hh,columns=['Tags'])
df_no_dup=pd.concat([hhh,aaa],axis=1)
# adding a new feature number of tags per question
print("Time taken to run this cell :", datetime.now() - start)
df_no_dup.head()
np.where(pd.isnull(df_no_dup))
```

```
4206308
Time taken to run this cell : 0:02:22.340723
```

Out[10]:
```
(array([], dtype=int64), array([], dtype=int64))
```

In [11]:
```
df_no_dup=df_no_dup.dropna()
```

In [ ]:

In [12]:
```
start = datetime.now()
df_no_dup["tag_count"] = df_no_dup["Tags"].apply(lambda text: len(text.split(" ")))
# adding a new feature number of tags per question
print("Time taken to run this cell :", datetime.now() - start)
df_no_dup.head()
```

```
Time taken to run this cell : 0:00:03.483702
```

```
Out[12]:                                  Tags  tag_count
         0                              c++ c          2
         1          c# silverlight data-binding          3
         2  c# silverlight data-binding columns          4
         3                            jsp jstl          2
         4                           java jdbc          2
```

```
In [13]: # distribution of number of tags per question
         df_no_dup.tag_count.value_counts()
```

```
Out[13]: 3    1206157
         2    1111706
         4     814996
         1     568291
         5     505158
         Name: tag_count, dtype: int64
```

```
In [14]: #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 [15]: #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 genar
```

```
Time taken to run this cell : 0:00:53.947521
```

3.2 Analysis of Tags

3.2.1 Total number of unique tags

```
In [16]: tag_data=tag_data.dropna()
```

```
In [17]: # Taking only 0.5 million data points
         #tag_data=tag_data[0:10000]
```

```
In [18]: print(tag_data.head())
         print(len(tag_data))
```

```
                                 Tags
1          c# silverlight data-binding
2  c# silverlight data-binding columns
3                             jsp jstl
4                            java jdbc
5        facebook api facebook-php-sdk
4206307
```

```
In [19]: # 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 [20]: print("Number of data points :", tag_dtm.shape[0])
         print("Number of unique tags :", tag_dtm.shape[1])
```

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

```
In [21]: #'get_feature_name()' gives us the vocabulary.
         tags = vectorizer.get_feature_names()
         #Lets look at the tags we have.
         print("Some of the tags we have :", tags[:10])
```

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

3.2.3 Number of times a tag appeared

```
In [22]: # 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))
         #print(result)
```

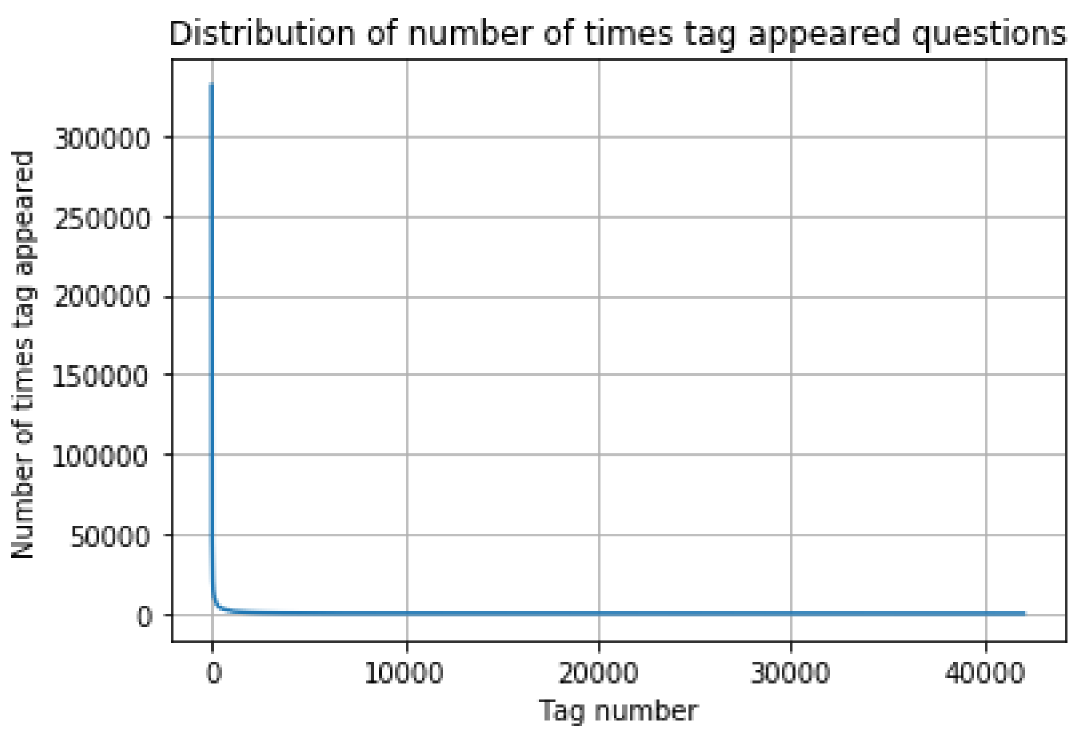
In [23]:
```python
#Saving this dictionary to csv files.
if not os.path.isfile('tag_counts_dict_dtm.csv'):
    with open('tag_counts_dict_dtm.csv', 'w') as csv_file:
        writer = csv.writer(csv_file)
        for key, value in result.items():
            writer.writerow([key, value])
tag_df = pd.read_csv("tag_counts_dict_dtm.csv", names=['Tags', 'Counts'])
tag_df.head()
```

Out[23]:

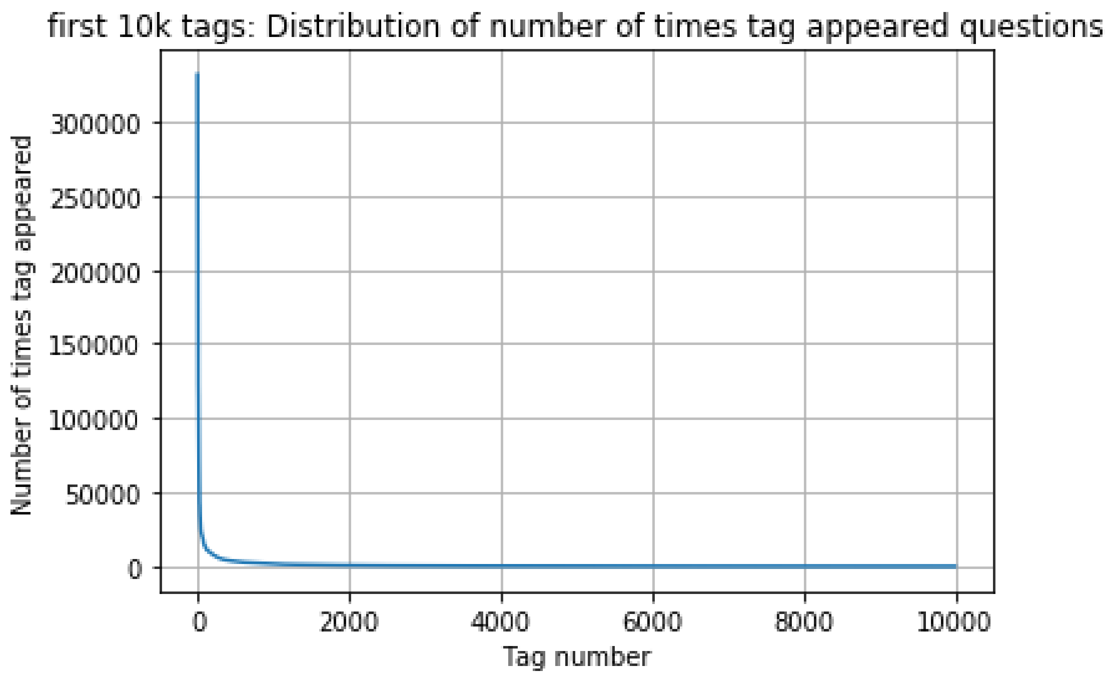
	Tags	Counts
0	.a	18
1	.app	37
2	.asp.net-mvc	1
3	.aspxauth	21
4	.bash-profile	138

In [24]:
```python
tag_df_sorted = tag_df.sort_values(['Counts'], ascending=False)
tag_counts = tag_df_sorted['Counts'].values
```

In [25]:
```python
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()
```



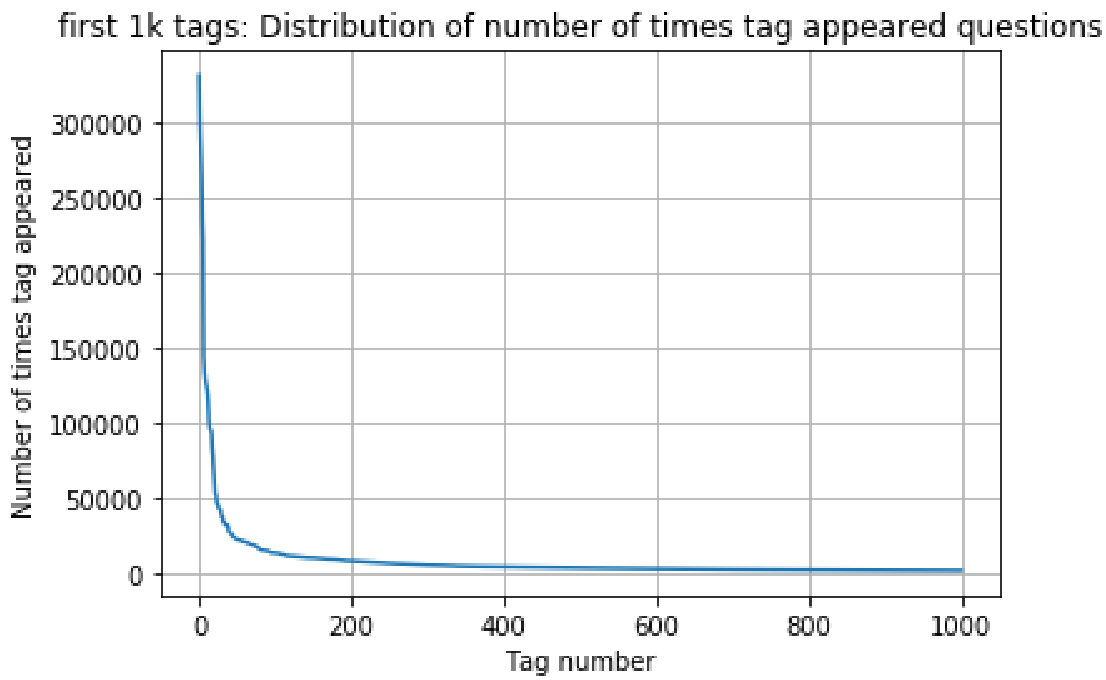
In [26]:
```
plt.plot(tag_counts[0:10000])
plt.title('first 10k tags: Distribution of number of times tag appeared questions')
plt.grid()
plt.xlabel("Tag number")
plt.ylabel("Number of times tag appeared")
plt.show()
print(len(tag_counts[0:10000:25]), tag_counts[0:10000:25])
```




```
400 [331505  44829  22429  17728  13364  11162  10029   9148   8054   7151
   6466   5865   5370   4983   4526   4281   4144   3929   3750   3593
   3453   3299   3123   2986   2891   2738   2647   2527   2431   2331
   2259   2186   2097   2020   1959   1900   1828   1770   1723   1673
   1631   1574   1532   1479   1448   1406   1365   1328   1300   1266
   1245   1222   1197   1181   1158   1139   1121   1101   1076   1056
   1038   1023   1006    983    966    952    938    926    911    891
    882    869    856    841    830    816    804    789    779    770
    752    743    733    725    712    702    688    678    671    658
    650    643    634    627    616    607    598    589    583    577
    568    559    552    545    540    533    526    518    512    506
    500    495    490    485    480    477    469    465    457    450
    447    442    437    432    426    422    418    413    408    403
    398    393    388    385    381    378    374    370    367    365
    361    357    354    350    347    344    342    339    336    332
    330    326    323    319    315    312    309    307    304    301
    299    296    293    291    289    286    284    281    278    276
```

```
 275   272   270   268   265   262   260   258   256   254
 252   250   249   247   245   243   241   239   238   236
 234   233   232   230   228   226   224   222   220   219
 217   215   214   212   210   209   207   205   204   203
 201   200   199   198   196   194   193   192   191   189
 188   186   185   183   182   181   180   179   178   177
 175   174   172   171   170   169   168   167   166   165
 164   162   161   160   159   158   157   156   156   155
 154   153   152   151   150   149   149   148   147   146
 145   144   143   142   142   141   140   139   138   137
 137   136   135   134   134   133   132   131   130   130
 129   128   128   127   126   126   125   124   124   123
 123   122   122   121   120   120   119   118   118   117
 117   116   116   115   115   114   113   113   112   111
 111   110   109   109   108   108   107   106   106   106
 105   105   104   104   103   103   102   102   101   101
 100   100    99    99    98    98    97    97    96    96
  95    95    94    94    93    93    93    92    92    91
  91    90    90    89    89    88    88    87    87    86
  86    86    85    85    84    84    83    83    83    82
  82    82    81    81    80    80    80    79    79    78
  78    78    78    77    77    76    76    76    75    75
  75    74    74    74    73    73    73    73    72    72]
```

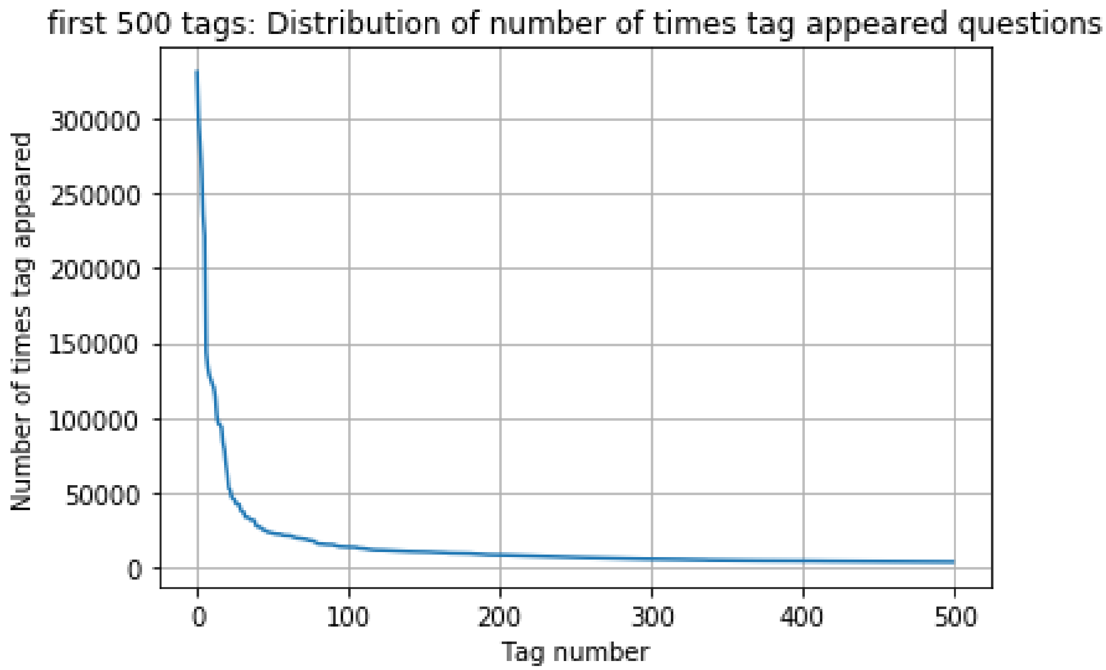
In [27]:
```
plt.plot(tag_counts[0:1000])
plt.title('first 1k tags: Distribution of number of times tag appeared questions')
plt.grid()
plt.xlabel("Tag number")
plt.ylabel("Number of times tag appeared")
plt.show()
print(len(tag_counts[0:1000:5]), tag_counts[0:1000:5])
```




```
200 [331505 221533 122769  95160  62023  44829  37170  31897  26925  24537
  22429  21820  20957  19758  18905  17728  15533  15097  14884  13703
  13364  13157  12407  11658  11228  11162  10863  10600  10350  10224
  10029   9884   9719   9411   9252   9148   9040   8617   8361   8163
   8054   7867   7702   7564   7274   7151   7052   6847   6656   6553
   6466   6291   6183   6093   5971   5865   5760   5577   5490   5411
   5370   5283   5207   5107   5066   4983   4891   4785   4658   4549
   4526   4487   4429   4335   4310   4281   4239   4228   4195   4159
   4144   4088   4050   4002   3957   3929   3874   3849   3818   3797
   3750   3703   3685   3658   3615   3593   3564   3521   3505   3483
   3453   3427   3396   3363   3326   3299   3272   3232   3196   3168
   3123   3094   3073   3050   3012   2986   2983   2953   2934   2903
   2891   2844   2819   2784   2754   2738   2726   2708   2681   2669
   2647   2621   2604   2594   2556   2527   2510   2482   2460   2444
   2431   2409   2395   2380   2363   2331   2312   2297   2290   2281
   2259   2246   2222   2211   2198   2186   2162   2142   2132   2107
   2097   2078   2057   2045   2036   2020   2011   1994   1971   1965
   1959   1952   1940   1932   1912   1900   1879   1865   1855   1841
   1828   1821   1813   1801   1782   1770   1760   1747   1741   1734
   1723   1707   1697   1688   1683   1673   1665   1656   1646   1639]
```

In [28]:
```
plt.plot(tag_counts[0:500])
plt.title('first 500 tags: Distribution of number of times tag appeared questions')
plt.grid()
```

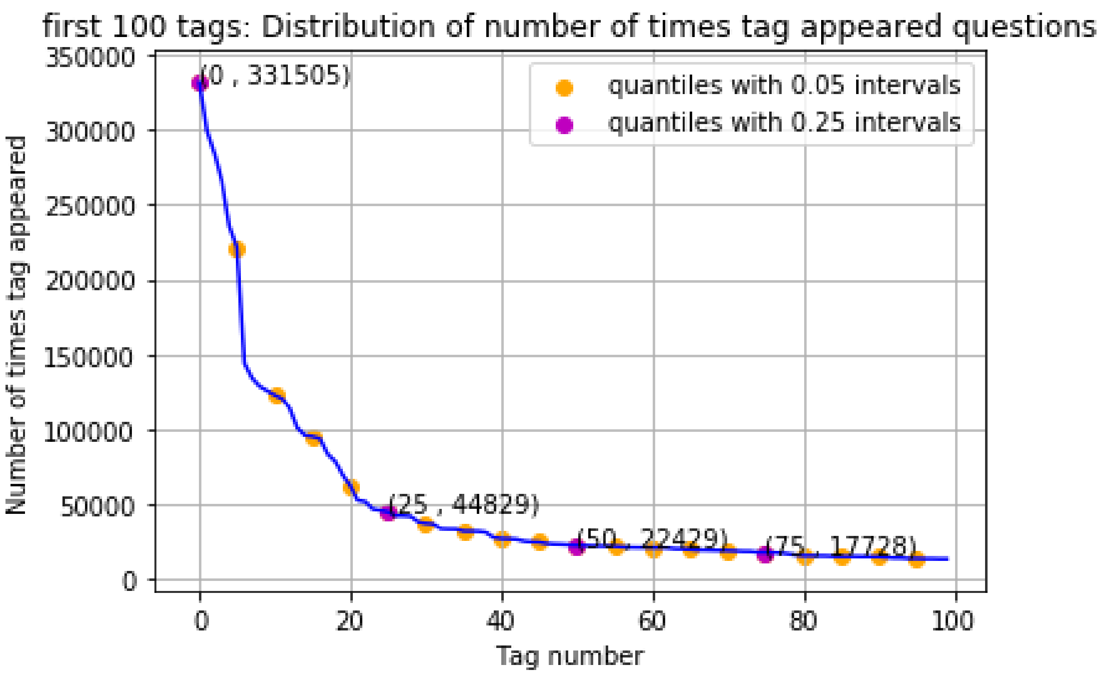
```
plt.xlabel("Tag number")
plt.ylabel("Number of times tag appeared")
plt.show()
print(len(tag_counts[0:500:5]), tag_counts[0:500:5])
```




```
100 [331505 221533 122769  95160  62023  44829  37170  31897  26925  24537
  22429  21820  20957  19758  18905  17728  15533  15097  14884  13703
  13364  13157  12407  11658  11228  11162  10863  10600  10350  10224
  10029   9884   9719   9411   9252   9148   9040   8617   8361   8163
   8054   7867   7702   7564   7274   7151   7052   6847   6656   6553
   6466   6291   6183   6093   5971   5865   5760   5577   5490   5411
   5370   5283   5207   5107   5066   4983   4891   4785   4658   4549
   4526   4487   4429   4335   4310   4281   4239   4228   4195   4159
   4144   4088   4050   4002   3957   3929   3874   3849   3818   3797
   3750   3703   3685   3658   3615   3593   3564   3521   3505   3483]
```

In [29]:
```
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
# quantiles with 0.25 difference
plt.scatter(x=list(range(0,100,25)), y=tag_counts[0:100:25], c='m', label = "quantiles

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

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




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

```
In [30]: # Store tags greater than 10K in one list
         lst_tags_gt_10k = tag_df[tag_df.Counts>10000].Tags
         #Print the length of the list
         print ('{} Tags are used more than 10000 times'.format(len(lst_tags_gt_10k)))
         # Store tags greater than 100K in one list
         lst_tags_gt_100k = tag_df[tag_df.Counts>100000].Tags
         #Print the length of the list.
         print ('{} Tags are used more than 100000 times'.format(len(lst_tags_gt_100k)))
```

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

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

3.2.4 Tags Per Question

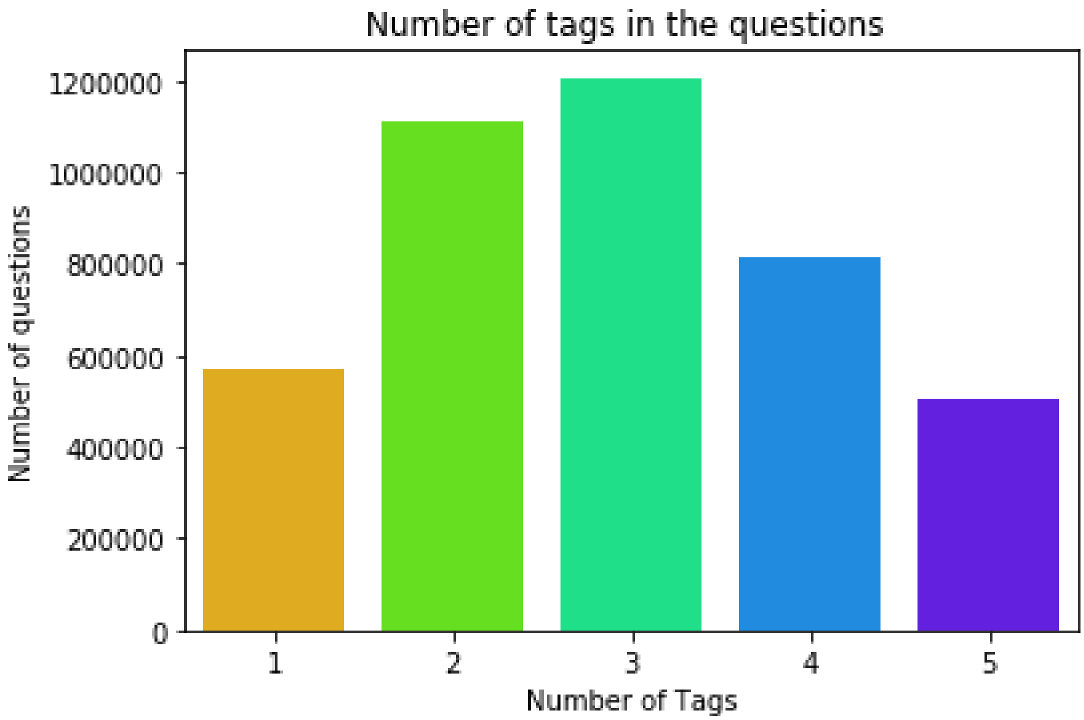
```
In [31]: #Storing the count of tag in each question in list 'tag_count'
         tag_quest_count = tag_dtm.sum(axis=1).tolist()
         #Converting each value in the 'tag_quest_count' to integer.
         tag_quest_count=[int(j) for i in tag_quest_count for j in i]
         print ('We have total {} datapoints.'.format(len(tag_quest_count)))

         print(tag_quest_count[:5])
```

```
We have total 4206307 datapoints.
[3, 4, 2, 2, 3]
```

```
In [32]: 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_ques
```

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

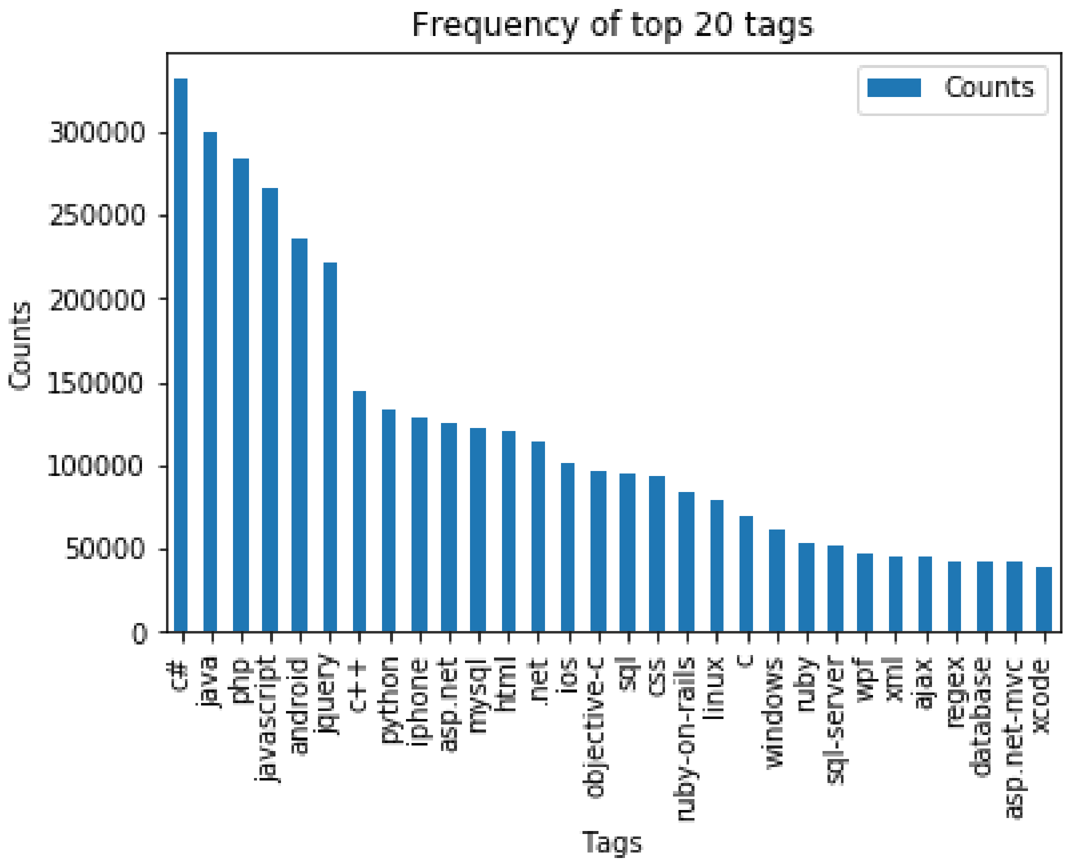
```
In [33]: sns.countplot(tag_quest_count, palette='gist_rainbow')
         plt.title("Number of tags in the questions ")
         plt.xlabel("Number of Tags")
         plt.ylabel("Number of questions")
         plt.show()
```



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 The top 20 tags

```
In [35]: i=np.arange(30)
         tag_df_sorted.head(30).plot(kind='bar')
         plt.title('Frequency of top 20 tags')
         plt.xticks(i, tag_df_sorted['Tags'])
         plt.xlabel('Tags')
         plt.ylabel('Counts')
         plt.show()
```



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

3.3 Cleaning and preprocessing of Questions

3.3.1 Preprocessing

```
<li> Sample 0.5M data points </li>
<li> Separate out code-snippets from Body </li>
<li> Remove Spcial characters from Question title and description (not in code)</li>
<li> Remove stop words (Except 'C') </li>
<li> Remove HTML Tags </li>
<li> Convert all the characters into small letters </li>
<li> Use SnowballStemmer to stem the words </li>
```

In [2]:
```
import nltk
nltk.download('stopwords')
```

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

Out[2]:
```
True
```

In [37]:
```
def striphtml(data):
    cleanr = re.compile('<.*?>')
    cleantext = re.sub(cleanr, ' ', str(data))
    return cleantext
stop_words = set(stopwords.words('english'))
stemmer = SnowballStemmer("english")
```

In [38]:
```
#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
    create_database_table("Processed.db", sql_create_table)
```

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

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

```
In [39]: nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
```

```
Out[39]: True
```

```
In [40]: print("\n")
```

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 [ ]:

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

```
In [41]: sql_create_table = """CREATE TABLE IF NOT EXISTS QuestionsProcessed (question text NOT
         create_database_table("Titlemoreweight.db", sql_create_table)
```

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

```
In [42]: # http://www.sqlitetutorial.net/sqlite-delete/
         # https://stackoverflow.com/questions/2279706/select-random-row-from-a-sqlite-table

         read_db = 'train_no_dup.db'
         write_db = 'Titlemoreweight.db'
         train_datasize = 400000
         if os.path.isfile(read_db):
             conn_r = create_connection(read_db)
             if conn_r is not None:
                 reader =conn_r.cursor()
                 # for selecting first 0.5M rows
                 reader.execute("SELECT Title, Body, Tags From no_dup_train LIMIT 500001;")
                 # for selecting random points
                 #reader.execute("SELECT Title, Body, Tags From no_dup_train ORDER BY RANDOM() L

         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

```
<li> Separate Code from Body </li>
<li> Remove Spcial characters from Question title and description (not in code)</li>
<li> <b> Give more weightage to title : Add title three times to the question </b> </li>

<li> Remove stop words (Except 'C') </li>
<li> Remove HTML Tags </li>
<li> Convert all the characters into small letters </li>
<li> Use SnowballStemmer to stem the words </li>
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [43]:
```
#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
            question=' '.join(str(stemmer.stem(j)) for j in words if j not in stop_words and (l

            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_p
            if (questions_proccesed%100000==0):
                print("number of questions completed=",questions_proccesed)


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

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

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

```
number of questions completed= 100000
number of questions completed= 200000
number of questions completed= 300000
number of questions completed= 400000
number of questions completed= 500000
Avg. length of questions(Title+Body) before processing: 1239
Avg. length of questions(Title+Body) after processing: 424
Percent of questions containing code: 57
Time taken to run this cell : 0:21:37.730850
```

In [44]:
```
# 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 [45]:
```
if os.path.isfile(write_db):
    conn_r = create_connection(write_db)
    if conn_r is not None:
        reader =conn_r.cursor()
        reader.execute("SELECT question From QuestionsProcessed LIMIT 10")
        print("Questions after preprocessed")
        print('='*100)
        reader.fetchone()
        for row in reader:
```

```
                print(row)
                print('-'*100)
        conn_r.commit()
        conn_r.close()
```

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

__ Saving Preprocessed data to a Database __

In [46]:
```
#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 QuestionsPr
conn_r.commit()
conn_r.close()
```

In [47]:
```
preprocessed_data.head()
```

Out[47]:
```
                                            question  \
0  dynam datagrid bind silverlight dynam datagrid...
1  dynam datagrid bind silverlight dynam datagrid...
2  java.lang.noclassdeffounderror javax servlet j...
3  java.sql.sqlexcept microsoft odbc driver manag...
4  better way updat feed fb php sdk better way up...

                              tags
0           c# silverlight data-binding
```

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

```
In [48]: print("number of data points in sample :", preprocessed_data.shape[0])
         print("number of dimensions :", preprocessed_data.shape[1])
```

```
number of data points in sample : 500000
number of dimensions : 2
```

__ Converting string Tags to multilable output variables __

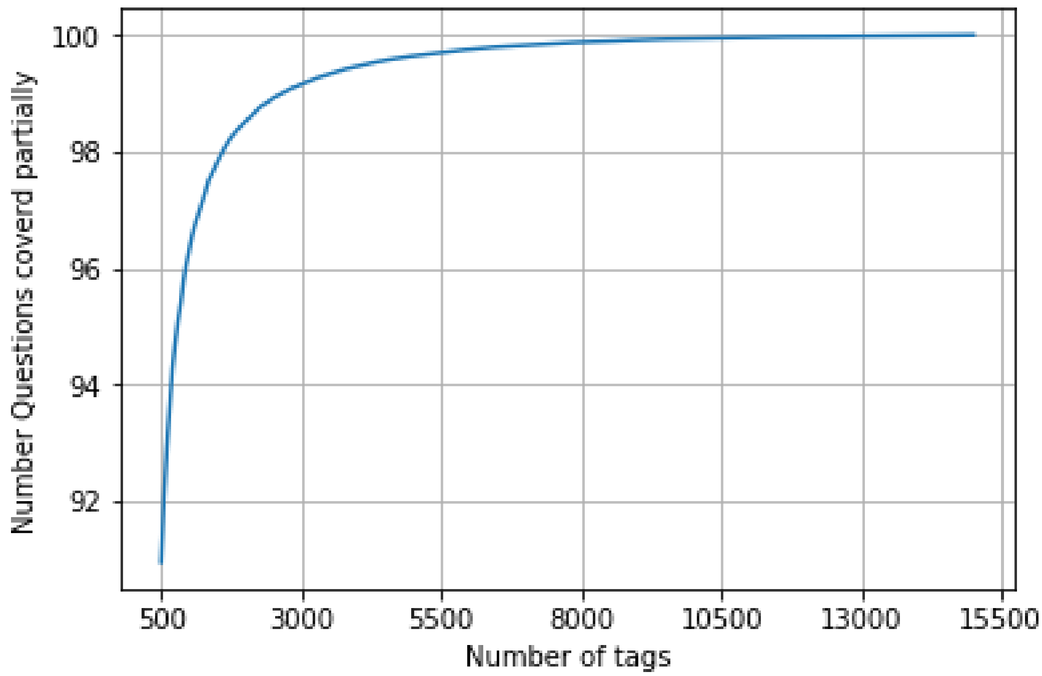
```
In [49]: vectorizer = CountVectorizer(tokenizer = lambda x: x.split(), binary='true')
         multilabel_y = vectorizer.fit_transform(preprocessed_data['tags'])
```

__ Selecting 500 Tags __

```
In [50]: 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 [51]: 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)
```

```
In [52]: 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 co
         print("with ",5500,"tags we are covering ",questions_explained[50],"% of questions")
         print("with ",500,"tags we are covering ",questions_explained[0],"% of questions")
```




```
with  5500 tags we are covering  99.157 % of questions
with  500 tags we are covering  90.956 % of questions
```

```
In [53]: # 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
```

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

```
In [54]: from sklearn.externals import joblib
         joblib.dump(preprocessed_data, 'preprocessed_data.pkl')
```

```
Out[54]: ['preprocessed_data.pkl']
```

```
In [55]: x_train=preprocessed_data.head(train_datasize)
         x_test=preprocessed_data.tail(preprocessed_data.shape[0] - 400000)

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

```
In [56]: print("Number of data points in train data :", y_train.shape)
         print("Number of data points in test data :", y_test.shape)
```

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

4.5.2 Featurizing data with TfIdf vectorizer

```
In [57]: print("a")
```

```
a
```

```
In [58]: start = datetime.now()
         vectorizer = TfidfVectorizer(min_df=0.00009, max_features=200000, smooth_idf=True, norm
                                      tokenizer = lambda x: x.split(), sublinear_tf=False,
                                      ngram_range=(1,4))
         x_train_multilabel = vectorizer.fit_transform(x_train['question'])
         x_test_multilabel = vectorizer.transform(x_test['question'])
         print("Time taken to run this cell :", datetime.now() - start)
```

```
Time taken to run this cell : 0:07:18.075098
```

```
In [59]: print("Dimensions of train data X:",x_train_multilabel.shape, "Y :",y_train.shape)
         print("Dimensions of test data X:",x_test_multilabel.shape,"Y:",y_test.shape)
```

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

```
In [ ]:
```

4.5.3 OneVsRest Classifier with SGDClassifier using TFIDF

```
In [60]: start = datetime.now()
         classifier = OneVsRestClassifier(SGDClassifier(loss='log',
                                                        alpha=0.00001,
                                                        penalty='l1'), n_jobs=-1)
         classifier.fit(x_train_multilabel, y_train)
         predictions = classifier.predict (x_test_multilabel)


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


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

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

    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,

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

```
Accuracy : 0.23625
Hamming loss  0.00278104
Micro-average quality numbers
Precision: 0.7216, Recall: 0.3256, F1-measure: 0.4488
Macro-average quality numbers
Precision: 0.5490, Recall: 0.2571, F1-measure: 0.3342
             precision    recall  f1-score   support

          0       0.94      0.64      0.76      5519
          1       0.68      0.26      0.38      8190
          2       0.81      0.38      0.52      6529
          3       0.81      0.43      0.56      3231
          4       0.81      0.41      0.54      6430
          5       0.82      0.34      0.48      2879
          6       0.87      0.49      0.63      5086
          7       0.88      0.54      0.67      4533
          8       0.61      0.13      0.21      3000
          9       0.81      0.53      0.64      2765
         10       0.59      0.17      0.26      3051
         11       0.70      0.33      0.45      3009
         12       0.65      0.25      0.36      2630
         13       0.71      0.23      0.35      1426
         14       0.90      0.53      0.67      2548
         15       0.68      0.18      0.29      2371
         16       0.64      0.23      0.34       873
         17       0.89      0.60      0.72      2151
         18       0.63      0.23      0.34      2204
         19       0.72      0.40      0.51       831
         20       0.77      0.40      0.53      1860
         21       0.27      0.08      0.12      2023
         22       0.50      0.22      0.31      1513
         23       0.91      0.49      0.64      1207
         24       0.56      0.29      0.38       506
         25       0.68      0.30      0.41       425
         26       0.65      0.40      0.50       793
```

27	0.60	0.33	0.42	1291
28	0.75	0.36	0.48	1208
29	0.41	0.09	0.14	406
30	0.76	0.17	0.28	504
31	0.30	0.11	0.16	732
32	0.57	0.22	0.32	441
33	0.57	0.18	0.27	1645
34	0.72	0.25	0.37	1058
35	0.83	0.55	0.66	946
36	0.66	0.20	0.30	644
37	0.98	0.67	0.79	136
38	0.63	0.35	0.45	570
39	0.85	0.28	0.43	766
40	0.62	0.28	0.38	1132
41	0.46	0.19	0.27	174
42	0.80	0.53	0.64	210
43	0.80	0.41	0.54	433
44	0.66	0.49	0.57	626
45	0.74	0.31	0.44	852
46	0.75	0.43	0.54	534
47	0.32	0.13	0.18	350
48	0.74	0.51	0.60	496
49	0.80	0.61	0.69	785
50	0.16	0.03	0.06	475
51	0.28	0.08	0.13	305
52	0.47	0.04	0.07	251
53	0.68	0.40	0.50	914
54	0.46	0.16	0.23	728
55	0.29	0.02	0.03	258
56	0.47	0.19	0.27	821
57	0.50	0.09	0.15	541
58	0.78	0.28	0.41	748
59	0.94	0.62	0.75	724
60	0.33	0.06	0.11	660
61	0.85	0.19	0.31	235
62	0.91	0.71	0.80	718
63	0.83	0.63	0.71	468
64	0.54	0.32	0.40	191
65	0.36	0.13	0.19	429
66	0.27	0.05	0.08	415
67	0.76	0.47	0.58	274
68	0.82	0.52	0.63	510
69	0.67	0.45	0.54	466
70	0.27	0.06	0.10	305
71	0.46	0.14	0.22	247
72	0.78	0.48	0.59	401
73	0.98	0.73	0.84	86
74	0.73	0.37	0.49	120

75	0.89	0.67	0.77	129
76	0.50	0.00	0.01	473
77	0.35	0.25	0.29	143
78	0.80	0.45	0.57	347
79	0.73	0.23	0.35	479
80	0.54	0.31	0.40	279
81	0.78	0.17	0.28	461
82	0.19	0.01	0.03	298
83	0.77	0.45	0.57	396
84	0.55	0.34	0.42	184
85	0.67	0.20	0.31	573
86	0.47	0.05	0.08	325
87	0.49	0.27	0.35	273
88	0.42	0.21	0.28	135
89	0.30	0.07	0.12	232
90	0.57	0.31	0.40	409
91	0.64	0.25	0.36	420
92	0.75	0.53	0.62	408
93	0.69	0.47	0.56	241
94	0.33	0.04	0.08	211
95	0.33	0.07	0.12	277
96	0.28	0.04	0.07	410
97	0.89	0.32	0.47	501
98	0.78	0.59	0.67	136
99	0.55	0.33	0.41	239
100	0.58	0.14	0.22	324
101	0.93	0.61	0.73	277
102	0.92	0.70	0.79	613
103	0.51	0.17	0.25	157
104	0.23	0.06	0.10	295
105	0.85	0.34	0.49	334
106	0.81	0.14	0.24	335
107	0.76	0.48	0.59	389
108	0.56	0.24	0.33	251
109	0.54	0.41	0.46	317
110	0.68	0.08	0.14	187
111	0.48	0.07	0.12	140
112	0.61	0.28	0.38	154
113	0.63	0.18	0.28	332
114	0.46	0.27	0.34	323
115	0.48	0.21	0.29	344
116	0.76	0.49	0.60	370
117	0.57	0.22	0.32	313
118	0.78	0.68	0.72	874
119	0.47	0.19	0.27	293
120	0.00	0.00	0.00	200
121	0.76	0.48	0.59	463
122	0.38	0.09	0.15	119

123	0.75	0.01	0.02	256
124	0.91	0.69	0.79	195
125	0.41	0.11	0.17	138
126	0.81	0.49	0.61	376
127	0.15	0.03	0.05	122
128	0.15	0.03	0.05	252
129	0.41	0.10	0.16	144
130	0.41	0.08	0.13	150
131	0.17	0.01	0.02	210
132	0.66	0.25	0.37	361
133	0.94	0.54	0.68	453
134	0.89	0.73	0.80	124
135	0.27	0.03	0.06	91
136	0.68	0.27	0.38	128
137	0.58	0.34	0.43	218
138	0.79	0.16	0.26	243
139	0.38	0.19	0.25	149
140	0.76	0.44	0.55	318
141	0.29	0.11	0.16	159
142	0.66	0.35	0.46	274
143	0.87	0.72	0.79	362
144	0.58	0.15	0.24	118
145	0.67	0.37	0.48	164
146	0.59	0.28	0.38	461
147	0.66	0.39	0.49	159
148	0.34	0.14	0.20	166
149	0.99	0.45	0.62	346
150	0.65	0.09	0.15	350
151	0.90	0.64	0.74	55
152	0.79	0.46	0.58	387
153	0.48	0.09	0.16	150
154	0.60	0.12	0.20	281
155	0.27	0.06	0.10	202
156	0.76	0.62	0.68	130
157	0.27	0.07	0.12	245
158	0.88	0.58	0.70	177
159	0.47	0.26	0.34	130
160	0.48	0.12	0.20	336
161	0.91	0.57	0.70	220
162	0.19	0.03	0.06	229
163	0.89	0.40	0.55	316
164	0.75	0.35	0.47	283
165	0.64	0.32	0.43	197
166	0.48	0.25	0.33	101
167	0.47	0.19	0.27	231
168	0.61	0.22	0.32	370
169	0.41	0.17	0.24	258
170	0.30	0.06	0.10	101

171	0.37	0.21	0.27	89
172	0.52	0.37	0.43	193
173	0.41	0.21	0.27	309
174	0.52	0.13	0.21	172
175	0.93	0.72	0.81	95
176	0.94	0.59	0.73	346
177	0.94	0.43	0.59	322
178	0.64	0.46	0.53	232
179	0.35	0.06	0.11	125
180	0.55	0.27	0.36	145
181	0.40	0.10	0.16	77
182	0.20	0.03	0.05	182
183	0.61	0.31	0.41	257
184	0.08	0.01	0.02	216
185	0.35	0.06	0.11	242
186	0.41	0.16	0.23	165
187	0.76	0.56	0.64	263
188	0.34	0.11	0.17	174
189	0.71	0.29	0.42	136
190	0.88	0.49	0.63	202
191	0.42	0.15	0.22	134
192	0.73	0.40	0.52	230
193	0.43	0.18	0.25	90
194	0.58	0.48	0.53	185
195	0.18	0.04	0.06	156
196	0.38	0.07	0.12	160
197	0.61	0.06	0.12	266
198	0.43	0.06	0.11	284
199	0.43	0.06	0.11	145
200	0.94	0.68	0.79	212
201	0.68	0.22	0.33	317
202	0.79	0.54	0.64	427
203	0.31	0.09	0.14	232
204	0.50	0.22	0.31	217
205	0.48	0.42	0.45	527
206	0.13	0.02	0.03	124
207	0.50	0.09	0.15	103
208	0.89	0.48	0.63	287
209	0.28	0.07	0.11	193
210	0.71	0.31	0.44	220
211	0.78	0.18	0.29	140
212	0.17	0.02	0.03	161
213	0.55	0.25	0.34	72
214	0.61	0.45	0.52	396
215	0.86	0.32	0.47	134
216	0.50	0.06	0.10	400
217	0.56	0.25	0.35	75
218	0.96	0.75	0.85	219

219	0.75	0.36	0.48	210
220	0.90	0.59	0.71	298
221	0.97	0.60	0.74	266
222	0.78	0.41	0.54	290
223	0.08	0.01	0.01	128
224	0.78	0.38	0.51	159
225	0.58	0.30	0.39	164
226	0.62	0.35	0.45	144
227	0.58	0.32	0.41	276
228	0.17	0.02	0.03	235
229	0.33	0.02	0.04	216
230	0.35	0.17	0.23	228
231	0.71	0.47	0.57	64
232	0.44	0.07	0.12	103
233	0.69	0.29	0.41	216
234	0.75	0.08	0.14	116
235	0.55	0.36	0.44	77
236	0.96	0.64	0.77	67
237	0.52	0.06	0.10	218
238	0.35	0.09	0.14	139
239	0.17	0.01	0.02	94
240	0.55	0.27	0.37	77
241	0.52	0.09	0.15	167
242	0.83	0.29	0.43	86
243	0.45	0.16	0.23	58
244	0.57	0.17	0.26	269
245	0.18	0.06	0.09	112
246	0.95	0.73	0.83	255
247	0.44	0.19	0.27	58
248	0.25	0.02	0.04	81
249	0.00	0.00	0.00	131
250	0.43	0.22	0.29	93
251	0.66	0.29	0.40	154
252	0.33	0.04	0.07	129
253	0.63	0.33	0.43	83
254	0.36	0.09	0.14	191
255	0.16	0.03	0.05	219
256	0.25	0.03	0.05	130
257	0.46	0.29	0.36	93
258	0.69	0.43	0.53	217
259	0.33	0.11	0.16	141
260	0.95	0.13	0.23	143
261	0.56	0.12	0.20	219
262	0.54	0.27	0.36	107
263	0.40	0.23	0.29	236
264	0.29	0.17	0.21	119
265	0.31	0.11	0.16	72
266	0.00	0.00	0.00	70

267	0.32	0.14	0.19	107
268	0.66	0.41	0.51	169
269	0.30	0.10	0.15	129
270	0.74	0.53	0.62	159
271	0.81	0.30	0.44	190
272	0.62	0.22	0.33	248
273	0.91	0.70	0.79	264
274	0.90	0.66	0.76	105
275	0.57	0.08	0.14	104
276	0.14	0.02	0.03	115
277	0.83	0.59	0.69	170
278	0.65	0.23	0.34	145
279	0.92	0.57	0.71	230
280	0.57	0.42	0.49	80
281	0.68	0.55	0.61	217
282	0.75	0.47	0.58	175
283	0.34	0.05	0.09	269
284	0.65	0.27	0.38	74
285	0.86	0.49	0.62	206
286	0.90	0.60	0.72	227
287	0.85	0.31	0.45	130
288	0.39	0.07	0.12	129
289	0.50	0.03	0.05	80
290	0.14	0.06	0.08	99
291	0.78	0.32	0.45	208
292	0.17	0.01	0.03	67
293	0.82	0.42	0.56	109
294	0.40	0.24	0.30	140
295	0.24	0.08	0.12	241
296	0.24	0.10	0.14	72
297	0.22	0.04	0.06	107
298	0.80	0.39	0.53	61
299	0.93	0.36	0.52	77
300	0.19	0.06	0.10	111
301	0.00	0.00	0.00	126
302	0.00	0.00	0.00	73
303	0.56	0.35	0.43	176
304	0.96	0.70	0.81	230
305	0.97	0.59	0.73	156
306	0.51	0.36	0.42	146
307	0.29	0.08	0.13	98
308	0.00	0.00	0.00	78
309	0.71	0.05	0.10	94
310	0.76	0.35	0.48	162
311	0.81	0.53	0.64	116
312	0.48	0.26	0.34	57
313	0.80	0.06	0.11	65
314	0.51	0.36	0.42	138

315	0.53	0.21	0.30	195
316	0.46	0.26	0.33	69
317	0.34	0.10	0.15	134
318	0.49	0.33	0.40	148
319	0.85	0.44	0.58	161
320	0.22	0.14	0.17	104
321	0.85	0.53	0.65	156
322	0.60	0.31	0.41	134
323	0.57	0.38	0.45	232
324	0.44	0.18	0.26	92
325	0.47	0.28	0.35	197
326	0.12	0.02	0.04	126
327	0.50	0.04	0.08	115
328	0.98	0.64	0.78	198
329	0.63	0.31	0.42	125
330	0.83	0.19	0.30	81
331	0.50	0.09	0.15	94
332	1.00	0.02	0.04	56
333	0.13	0.03	0.04	260
334	0.18	0.03	0.06	60
335	0.32	0.09	0.14	110
336	0.63	0.41	0.50	71
337	0.13	0.03	0.05	66
338	0.44	0.31	0.36	150
339	0.00	0.00	0.00	54
340	0.85	0.54	0.66	195
341	0.89	0.20	0.33	79
342	0.38	0.16	0.22	38
343	0.67	0.37	0.48	43
344	0.53	0.24	0.33	68
345	0.67	0.38	0.49	73
346	0.27	0.03	0.05	116
347	0.88	0.34	0.49	111
348	0.29	0.10	0.14	63
349	0.82	0.59	0.69	104
350	0.64	0.48	0.55	44
351	0.73	0.20	0.31	40
352	0.98	0.40	0.57	136
353	0.42	0.20	0.27	54
354	0.36	0.04	0.07	134
355	0.51	0.28	0.36	120
356	0.55	0.25	0.34	228
357	0.66	0.28	0.39	269
358	0.69	0.36	0.48	80
359	0.86	0.43	0.57	140
360	0.40	0.15	0.22	125
361	0.89	0.63	0.74	169
362	0.11	0.04	0.05	56

363	0.94	0.66	0.77	154
364	0.33	0.05	0.09	58
365	0.26	0.13	0.17	71
366	1.00	0.65	0.79	54
367	0.29	0.03	0.06	116
368	0.00	0.00	0.00	54
369	0.00	0.00	0.00	71
370	0.20	0.03	0.06	61
371	0.55	0.08	0.15	71
372	0.65	0.46	0.54	52
373	0.78	0.36	0.49	150
374	0.34	0.13	0.19	93
375	0.19	0.04	0.07	67
376	0.00	0.00	0.00	76
377	0.74	0.16	0.26	106
378	0.27	0.03	0.06	86
379	0.33	0.07	0.12	14
380	1.00	0.40	0.57	122
381	0.19	0.03	0.05	104
382	0.32	0.09	0.14	66
383	0.46	0.27	0.34	110
384	0.00	0.00	0.00	155
385	0.40	0.08	0.13	50
386	0.24	0.11	0.15	64
387	0.43	0.06	0.11	93
388	0.61	0.27	0.38	102
389	0.07	0.01	0.02	108
390	0.96	0.66	0.78	178
391	0.62	0.17	0.27	115
392	0.77	0.40	0.53	42
393	0.00	0.00	0.00	134
394	0.50	0.02	0.03	112
395	0.42	0.12	0.19	176
396	0.50	0.08	0.14	125
397	0.70	0.23	0.35	224
398	0.88	0.56	0.68	63
399	0.00	0.00	0.00	59
400	0.48	0.35	0.40	63
401	0.50	0.18	0.27	98
402	0.57	0.16	0.25	162
403	0.41	0.14	0.21	83
404	0.73	0.84	0.78	19
405	0.29	0.07	0.11	92
406	0.86	0.15	0.25	41
407	0.62	0.30	0.41	43
408	0.80	0.32	0.46	160
409	0.17	0.10	0.13	50
410	0.00	0.00	0.00	19

411	0.39	0.10	0.16	175
412	0.29	0.06	0.09	72
413	0.56	0.05	0.10	95
414	0.16	0.03	0.05	97
415	0.30	0.15	0.20	48
416	0.44	0.28	0.34	83
417	0.50	0.07	0.13	40
418	0.37	0.08	0.13	91
419	0.52	0.28	0.36	90
420	0.29	0.22	0.25	37
421	0.00	0.00	0.00	66
422	0.61	0.34	0.44	73
423	0.48	0.25	0.33	56
424	0.93	0.82	0.87	33
425	0.00	0.00	0.00	76
426	0.25	0.05	0.08	81
427	0.99	0.68	0.81	150
428	0.95	0.66	0.78	29
429	0.99	0.65	0.78	389
430	0.64	0.36	0.46	167
431	0.48	0.08	0.14	123
432	0.45	0.33	0.38	39
433	0.29	0.16	0.20	82
434	1.00	0.65	0.79	66
435	0.63	0.45	0.53	93
436	0.52	0.25	0.34	87
437	0.26	0.06	0.10	86
438	0.73	0.47	0.57	104
439	0.62	0.13	0.21	100
440	0.25	0.01	0.01	141
441	0.42	0.25	0.31	110
442	0.40	0.13	0.20	123
443	0.50	0.13	0.20	71
444	0.44	0.06	0.11	109
445	0.42	0.21	0.28	48
446	0.43	0.25	0.32	76
447	0.26	0.13	0.18	38
448	0.69	0.54	0.61	81
449	0.57	0.16	0.25	132
450	0.46	0.26	0.33	81
451	0.88	0.29	0.44	76
452	0.00	0.00	0.00	44
453	0.00	0.00	0.00	44
454	0.94	0.41	0.57	70
455	0.48	0.07	0.12	155
456	0.43	0.14	0.21	43
457	0.52	0.21	0.30	72
458	0.29	0.08	0.13	62

```
        459       0.64      0.13      0.22        69
        460       0.07      0.01      0.01       119
        461       0.77      0.13      0.22        79
        462       0.69      0.23      0.35        47
        463       0.26      0.05      0.08       104
        464       0.65      0.34      0.45       106
        465       0.54      0.11      0.18        64
        466       0.57      0.28      0.38       173
        467       0.79      0.35      0.48       107
        468       0.82      0.11      0.20       126
        469       0.00      0.00      0.00       114
        470       0.94      0.79      0.86       140
        471       0.91      0.27      0.41        79
        472       0.39      0.28      0.33       143
        473       0.68      0.30      0.41       158
        474       0.38      0.07      0.11       138
        475       0.00      0.00      0.00        59
        476       0.57      0.32      0.41        88
        477       0.86      0.57      0.68       176
        478       0.94      0.71      0.81        24
        479       0.09      0.01      0.02        92
        480       0.82      0.50      0.62       100
        481       0.49      0.17      0.26       103
        482       0.52      0.23      0.32        74
        483       0.83      0.57      0.68       105
        484       0.29      0.02      0.04        83
        485       0.25      0.02      0.04        82
        486       0.38      0.11      0.17        71
        487       0.43      0.18      0.26       120
        488       0.20      0.01      0.02       105
        489       0.72      0.30      0.42        87
        490       1.00      0.81      0.90        32
        491       0.00      0.00      0.00        69
        492       0.00      0.00      0.00        49
        493       0.00      0.00      0.00       117
        494       0.50      0.16      0.25        61
        495       0.99      0.52      0.68       344
        496       0.37      0.19      0.25        52
        497       0.62      0.19      0.29       137
        498       0.29      0.04      0.07        98
        499       0.72      0.16      0.27        79

avg / total       0.67      0.33      0.43    173812

Time taken to run this cell : 0:05:30.994191
```

```
In [61]: joblib.dump(classifier, 'lr_with_more_title_weight.pkl')
```

Out[61]: ['lr_with_more_title_weight.pkl']

ASSIGNMENT

```
<li>  bag of words upto 4 grams and compute the micro f1 score with Logistic regression(OvR) </l
<li> Perform hyperparam tuning on alpha (or lambda) for Logistic regression to improve the perfo
<li>  OneVsRestClassifier  with Linear-SVM (SGDClassifier with loss-hinge)</li>
```

1.1 Featurizing Using Bag of Words

In [ ]:

In [2]:
```
alpha=[10**-3,10**-2,10**-1]
```

In [63]:
```
start = datetime.now()
vectorizer = CountVectorizer(min_df=0.00009, max_features=200000,  \
                             tokenizer = lambda x: x.split(),  ngram_range=(1,4))
```

In [64]:
```
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 : 0:07:24.935906
```

In [65]:
```
print("Dimensions of train data X:",x_train_multilabel.shape, "Y :",y_train.shape)
print("Dimensions of test data X:",x_test_multilabel.shape,"Y:",y_test.shape)
```

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

Dump and load train and test data into joblib

In [66]:
```
joblib.dump(x_train_multilabel, 'x_train_BOW.pkl')
joblib.dump(x_test_multilabel, 'x_test_BOW.pkl')
joblib.dump(y_train, 'y_train.pkl')
joblib.dump(y_test, 'y_test.pkl')
```

Out[66]: ['y_test.pkl']

In [3]:
```
x_train_multilabel = joblib.load('x_train_BOW.pkl')
y_train = joblib.load('y_train.pkl')
```

In [15]:
```
x_test_multilabel = joblib.load('x_test_BOW.pkl')
y_test = joblib.load('y_test.pkl')
```

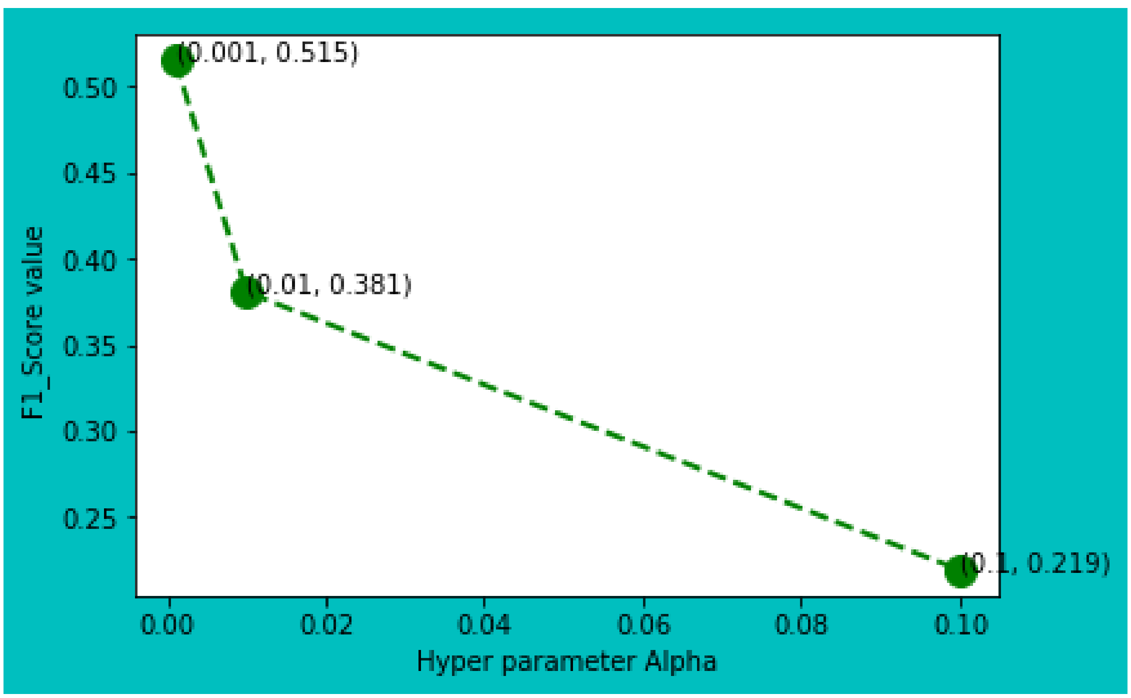
2 OneVsRestClassifier with Logistic regression

(alpha tuning using Gridsearch)

2.1 OneVsRestClassifier with SGDClassifier(penalty=l2, loss=log)==> {Logistic regression}

In [9]:
```
start = datetime.now()
import warnings
warnings.filterwarnings('ignore')

# hp1={'estimator__C':alpha}

cv_scores = []
for i in alpha:
    print(i)
    hp1={'estimator__alpha':[i],
         'estimator__loss':['log'],
         'estimator__penalty':['l2']}
    print(hp1)
    classifier = OneVsRestClassifier(SGDClassifier())

    model11 =GridSearchCV(classifier,hp1,
                          cv=3, scoring='f1_micro',n_jobs=-1)
    print("Gridsearchcv")
    best_model1=model11.fit(x_train_multilabel, y_train)
    print('fit model')
    Train_model_score=best_model1.score(x_train_multilabel,
                                        y_train)
#print("best_model1")
    cv_scores.append(Train_model_score.mean())

fscore = [x for x in cv_scores]

# determining best alpha
optimal_alpha21 = alpha[fscore.index(max(fscore))]
print('\n The optimal value of alpha with penalty=l2 and loss= log is %d.' % optimal_alp

# Plots
fig4 = plt.figure( facecolor='c', edgecolor='k')
plt.plot(alpha, fscore,color='green', marker='o', linestyle='dashed',
linewidth=2, markersize=12)

for xy in zip(alpha, np.round(fscore,3)):
    plt.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')

plt.xlabel('Hyper parameter Alpha')
plt.ylabel('F1_Score value ')
plt.show()

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

```
0.001
{'estimator__alpha': [0.001], 'estimator__loss': ['log'], 'estimator__penalty': ['l2']}
Gridsearchcv
fit model
0.01
{'estimator__alpha': [0.01], 'estimator__loss': ['log'], 'estimator__penalty': ['l2']}
Gridsearchcv
fit model
0.1
{'estimator__alpha': [0.1], 'estimator__loss': ['log'], 'estimator__penalty': ['l2']}
Gridsearchcv
fit model

 The optimal value of alpha with penalty=l1 and loss= log is 0.
```




```
Time taken to run this cell : 1:59:14.455889
```

In [10]:
```python
print(optimal_alpha21)
```

```
0.001
```

In [ ]:

In [11]:
```python
start = datetime.now()
best_model1 = OneVsRestClassifier(SGDClassifier(loss='log', alpha=optimal_alpha21,
                                                penalty='l2'), n_jobs=-1)
best_model1.fit(x_train_multilabel, y_train)
```

Out[11]:
```
OneVsRestClassifier(estimator=SGDClassifier(alpha=0.001, average=False, class_weight=No
       eta0=0.0, fit_intercept=True, l1_ratio=0.15,
       learning_rate='optimal', loss='log', max_iter=None, n_iter=None,
       n_jobs=1, penalty='l2', power_t=0.5, random_state=None,
       shuffle=True, tol=None, verbose=0, warm_start=False),
          n_jobs=-1)
```

In [12]:
```python
joblib.dump(best_model1, 'best_model1_LR.pkl')
```

Out[12]:
```
['best_model1_LR.pkl']
```

In [13]:
```python
best_model1=joblib.load('best_model1_LR.pkl')
```

In [16]:
```python
predictions = best_model1.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-averasge quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall,

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,

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

```
Accuracy : 0.2117
Hamming loss  0.00296836
Micro-averasge quality numbers
Precision: 0.6491, Recall: 0.3179, F1-measure: 0.4268
Macro-average quality numbers
Precision: 0.4948, Recall: 0.2353, F1-measure: 0.3058
             precision    recall  f1-score   support

          0       0.95      0.64      0.76      5519
```

1	0.68	0.27	0.39	8190
2	0.80	0.37	0.51	6529
3	0.82	0.42	0.55	3231
4	0.80	0.43	0.56	6430
5	0.80	0.35	0.49	2879
6	0.88	0.47	0.62	5086
7	0.87	0.56	0.68	4533
8	0.60	0.14	0.23	3000
9	0.81	0.57	0.67	2765
10	0.59	0.21	0.31	3051
11	0.71	0.33	0.45	3009
12	0.63	0.27	0.38	2630
13	0.73	0.27	0.39	1426
14	0.90	0.49	0.63	2548
15	0.63	0.13	0.22	2371
16	0.63	0.25	0.36	873
17	0.85	0.62	0.72	2151
18	0.63	0.26	0.37	2204
19	0.72	0.41	0.53	831
20	0.78	0.40	0.53	1860
21	0.28	0.14	0.18	2023
22	0.44	0.31	0.37	1513
23	0.91	0.47	0.62	1207
24	0.49	0.36	0.41	506
25	0.60	0.29	0.40	425
26	0.59	0.42	0.49	793
27	0.57	0.38	0.46	1291
28	0.70	0.32	0.44	1208
29	0.36	0.09	0.14	406
30	0.58	0.14	0.23	504
31	0.28	0.15	0.20	732
32	0.57	0.27	0.37	441
33	0.51	0.30	0.37	1645
34	0.71	0.23	0.35	1058
35	0.83	0.58	0.68	946
36	0.60	0.22	0.32	644
37	0.98	0.63	0.77	136
38	0.60	0.45	0.51	570
39	0.85	0.22	0.34	766
40	0.60	0.31	0.40	1132
41	0.46	0.22	0.30	174
42	0.69	0.43	0.53	210
43	0.76	0.39	0.52	433
44	0.65	0.47	0.55	626
45	0.65	0.31	0.42	852
46	0.71	0.43	0.53	534
47	0.27	0.23	0.25	350
48	0.72	0.50	0.59	496

49	0.79	0.64	0.71	785
50	0.20	0.13	0.16	475
51	0.28	0.15	0.19	305
52	0.34	0.06	0.11	251
53	0.67	0.38	0.49	914
54	0.43	0.22	0.29	728
55	0.00	0.00	0.00	258
56	0.38	0.27	0.32	821
57	0.39	0.12	0.19	541
58	0.80	0.24	0.37	748
59	0.95	0.57	0.71	724
60	0.27	0.07	0.11	660
61	0.85	0.19	0.31	235
62	0.88	0.69	0.78	718
63	0.83	0.55	0.66	468
64	0.49	0.44	0.47	191
65	0.25	0.18	0.21	429
66	0.26	0.14	0.19	415
67	0.68	0.46	0.55	274
68	0.84	0.47	0.61	510
69	0.65	0.42	0.51	466
70	0.26	0.13	0.18	305
71	0.37	0.17	0.23	247
72	0.75	0.41	0.53	401
73	0.90	0.65	0.76	86
74	0.71	0.34	0.46	120
75	0.90	0.62	0.73	129
76	0.46	0.01	0.02	473
77	0.36	0.35	0.35	143
78	0.75	0.38	0.51	347
79	0.69	0.21	0.32	479
80	0.49	0.39	0.44	279
81	0.75	0.11	0.19	461
82	0.20	0.08	0.12	298
83	0.71	0.41	0.52	396
84	0.46	0.37	0.41	184
85	0.45	0.27	0.34	573
86	0.24	0.09	0.13	325
87	0.46	0.24	0.32	273
88	0.32	0.25	0.28	135
89	0.25	0.16	0.20	232
90	0.49	0.40	0.44	409
91	0.62	0.34	0.44	420
92	0.75	0.46	0.57	408
93	0.51	0.48	0.49	241
94	0.31	0.10	0.16	211
95	0.27	0.18	0.22	277
96	0.29	0.07	0.11	410

97	0.88	0.16	0.27	501
98	0.79	0.57	0.66	136
99	0.49	0.29	0.37	239
100	0.47	0.18	0.26	324
101	0.90	0.50	0.64	277
102	0.90	0.64	0.75	613
103	0.44	0.20	0.27	157
104	0.21	0.15	0.17	295
105	0.67	0.36	0.47	334
106	0.78	0.05	0.10	335
107	0.75	0.49	0.59	389
108	0.53	0.34	0.41	251
109	0.48	0.40	0.43	317
110	0.47	0.09	0.14	187
111	0.35	0.06	0.10	140
112	0.43	0.25	0.32	154
113	0.58	0.14	0.22	332
114	0.42	0.29	0.35	323
115	0.41	0.19	0.26	344
116	0.72	0.45	0.55	370
117	0.54	0.19	0.29	313
118	0.80	0.46	0.58	874
119	0.34	0.24	0.28	293
120	0.13	0.04	0.05	200
121	0.75	0.42	0.54	463
122	0.36	0.24	0.29	119
123	0.25	0.00	0.01	256
124	0.91	0.62	0.74	195
125	0.39	0.20	0.26	138
126	0.79	0.51	0.62	376
127	0.17	0.06	0.09	122
128	0.20	0.08	0.11	252
129	0.39	0.10	0.16	144
130	0.41	0.07	0.12	150
131	0.16	0.03	0.06	210
132	0.58	0.22	0.32	361
133	0.94	0.39	0.55	453
134	0.89	0.66	0.76	124
135	0.25	0.01	0.02	91
136	0.53	0.30	0.39	128
137	0.46	0.33	0.39	218
138	0.38	0.08	0.13	243
139	0.33	0.24	0.28	149
140	0.68	0.32	0.44	318
141	0.18	0.15	0.17	159
142	0.65	0.39	0.49	274
143	0.85	0.61	0.71	362
144	0.48	0.20	0.29	118

145	0.58	0.37	0.45	164
146	0.57	0.29	0.38	461
147	0.66	0.45	0.53	159
148	0.35	0.16	0.22	166
149	0.97	0.31	0.47	346
150	0.61	0.07	0.12	350
151	0.88	0.42	0.57	55
152	0.72	0.46	0.56	387
153	0.39	0.06	0.10	150
154	0.52	0.06	0.11	281
155	0.29	0.16	0.21	202
156	0.73	0.55	0.63	130
157	0.28	0.11	0.15	245
158	0.89	0.47	0.62	177
159	0.43	0.28	0.34	130
160	0.49	0.25	0.33	336
161	0.85	0.50	0.63	220
162	0.18	0.10	0.13	229
163	0.90	0.28	0.43	316
164	0.71	0.28	0.41	283
165	0.54	0.28	0.37	197
166	0.31	0.20	0.24	101
167	0.39	0.24	0.30	231
168	0.44	0.21	0.28	370
169	0.42	0.28	0.33	258
170	0.23	0.09	0.13	101
171	0.46	0.25	0.32	89
172	0.39	0.34	0.36	193
173	0.41	0.28	0.34	309
174	0.50	0.12	0.19	172
175	0.90	0.75	0.82	95
176	0.93	0.43	0.59	346
177	0.95	0.24	0.39	322
178	0.57	0.43	0.49	232
179	0.54	0.06	0.10	125
180	0.43	0.21	0.28	145
181	0.47	0.19	0.28	77
182	0.13	0.07	0.09	182
183	0.55	0.35	0.43	257
184	0.13	0.06	0.08	216
185	0.29	0.14	0.19	242
186	0.28	0.19	0.23	165
187	0.77	0.46	0.58	263
188	0.31	0.16	0.21	174
189	0.78	0.33	0.46	136
190	0.94	0.36	0.52	202
191	0.40	0.15	0.22	134
192	0.63	0.31	0.41	230

193	0.31	0.18	0.23	90
194	0.59	0.52	0.56	185
195	0.08	0.04	0.05	156
196	0.23	0.07	0.11	160
197	0.10	0.02	0.03	266
198	0.38	0.10	0.16	284
199	0.15	0.03	0.06	145
200	0.93	0.52	0.67	212
201	0.49	0.23	0.31	317
202	0.73	0.43	0.54	427
203	0.25	0.14	0.18	232
204	0.40	0.25	0.31	217
205	0.48	0.38	0.42	527
206	0.10	0.04	0.06	124
207	0.34	0.16	0.21	103
208	0.81	0.34	0.48	287
209	0.25	0.11	0.15	193
210	0.69	0.25	0.37	220
211	0.64	0.06	0.12	140
212	0.08	0.05	0.06	161
213	0.55	0.29	0.38	72
214	0.60	0.43	0.50	396
215	0.77	0.17	0.28	134
216	0.36	0.07	0.12	400
217	0.44	0.25	0.32	75
218	0.97	0.50	0.66	219
219	0.79	0.28	0.41	210
220	0.93	0.37	0.53	298
221	0.96	0.41	0.58	266
222	0.70	0.29	0.41	290
223	0.22	0.05	0.09	128
224	0.75	0.36	0.49	159
225	0.36	0.22	0.27	164
226	0.56	0.34	0.42	144
227	0.54	0.41	0.46	276
228	0.07	0.02	0.03	235
229	0.23	0.03	0.05	216
230	0.36	0.25	0.30	228
231	0.67	0.45	0.54	64
232	0.15	0.07	0.09	103
233	0.72	0.20	0.31	216
234	0.60	0.13	0.21	116
235	0.57	0.43	0.49	77
236	0.91	0.60	0.72	67
237	0.56	0.05	0.08	218
238	0.15	0.09	0.12	139
239	0.19	0.03	0.05	94
240	0.39	0.16	0.22	77

241	0.47	0.10	0.17	167
242	0.77	0.23	0.36	86
243	0.48	0.19	0.27	58
244	0.45	0.22	0.29	269
245	0.17	0.06	0.09	112
246	0.96	0.54	0.69	255
247	0.39	0.21	0.27	58
248	0.36	0.06	0.11	81
249	0.03	0.01	0.01	131
250	0.30	0.23	0.26	93
251	0.57	0.28	0.38	154
252	0.20	0.05	0.09	129
253	0.55	0.35	0.43	83
254	0.22	0.10	0.14	191
255	0.14	0.07	0.09	219
256	0.07	0.02	0.03	130
257	0.41	0.31	0.35	93
258	0.63	0.35	0.45	217
259	0.24	0.11	0.15	141
260	0.89	0.12	0.21	143
261	0.53	0.11	0.18	219
262	0.42	0.32	0.36	107
263	0.32	0.32	0.32	236
264	0.21	0.19	0.20	119
265	0.32	0.24	0.27	72
266	0.18	0.09	0.12	70
267	0.26	0.13	0.17	107
268	0.61	0.33	0.43	169
269	0.22	0.15	0.18	129
270	0.70	0.50	0.58	159
271	0.48	0.17	0.25	190
272	0.57	0.21	0.31	248
273	0.93	0.43	0.59	264
274	0.88	0.50	0.64	105
275	0.09	0.03	0.04	104
276	0.09	0.02	0.03	115
277	0.86	0.51	0.64	170
278	0.63	0.19	0.29	145
279	0.88	0.30	0.45	230
280	0.54	0.33	0.41	80
281	0.68	0.47	0.56	217
282	0.74	0.38	0.50	175
283	0.37	0.11	0.17	269
284	0.61	0.30	0.40	74
285	0.86	0.36	0.51	206
286	0.92	0.43	0.58	227
287	0.77	0.25	0.38	130
288	0.28	0.06	0.10	129

289	0.17	0.06	0.09	80
290	0.15	0.12	0.14	99
291	0.83	0.21	0.34	208
292	0.37	0.10	0.16	67
293	0.78	0.33	0.46	109
294	0.32	0.33	0.33	140
295	0.17	0.14	0.15	241
296	0.23	0.19	0.21	72
297	0.28	0.12	0.17	107
298	0.67	0.43	0.52	61
299	0.86	0.39	0.54	77
300	0.18	0.09	0.12	111
301	0.00	0.00	0.00	126
302	0.33	0.01	0.03	73
303	0.53	0.40	0.46	176
304	0.96	0.46	0.62	230
305	0.94	0.40	0.57	156
306	0.43	0.36	0.39	146
307	0.28	0.11	0.16	98
308	0.08	0.04	0.05	78
309	0.33	0.02	0.04	94
310	0.56	0.31	0.40	162
311	0.67	0.37	0.48	116
312	0.47	0.25	0.32	57
313	0.67	0.03	0.06	65
314	0.46	0.30	0.37	138
315	0.48	0.24	0.32	195
316	0.41	0.33	0.37	69
317	0.19	0.08	0.11	134
318	0.41	0.30	0.35	148
319	0.70	0.29	0.41	161
320	0.18	0.22	0.20	104
321	0.81	0.43	0.56	156
322	0.56	0.31	0.40	134
323	0.49	0.41	0.45	232
324	0.37	0.18	0.25	92
325	0.34	0.26	0.30	197
326	0.09	0.02	0.04	126
327	0.29	0.04	0.08	115
328	0.97	0.31	0.47	198
329	0.53	0.32	0.40	125
330	0.57	0.10	0.17	81
331	0.22	0.06	0.10	94
332	0.33	0.02	0.03	56
333	0.12	0.09	0.10	260
334	0.67	0.07	0.12	60
335	0.28	0.17	0.21	110
336	0.65	0.42	0.51	71

337	0.11	0.06	0.08	66
338	0.46	0.33	0.38	150
339	0.00	0.00	0.00	54
340	0.89	0.33	0.49	195
341	0.75	0.19	0.30	79
342	0.33	0.32	0.32	38
343	0.57	0.30	0.39	43
344	0.50	0.21	0.29	68
345	0.60	0.38	0.47	73
346	0.07	0.03	0.04	116
347	0.93	0.23	0.36	111
348	0.23	0.08	0.12	63
349	0.89	0.39	0.55	104
350	0.54	0.30	0.38	44
351	0.50	0.15	0.23	40
352	1.00	0.18	0.31	136
353	0.48	0.28	0.35	54
354	0.27	0.04	0.08	134
355	0.48	0.26	0.34	120
356	0.42	0.23	0.30	228
357	0.53	0.22	0.31	269
358	0.69	0.30	0.42	80
359	0.65	0.25	0.36	140
360	0.37	0.18	0.24	125
361	0.88	0.33	0.48	169
362	0.12	0.05	0.07	56
363	0.95	0.47	0.63	154
364	0.33	0.05	0.09	58
365	0.22	0.20	0.21	71
366	1.00	0.37	0.54	54
367	0.19	0.05	0.08	116
368	0.25	0.02	0.03	54
369	0.12	0.04	0.06	71
370	0.10	0.03	0.05	61
371	0.40	0.06	0.10	71
372	0.61	0.33	0.42	52
373	0.60	0.17	0.27	150
374	0.39	0.23	0.29	93
375	0.33	0.06	0.10	67
376	0.00	0.00	0.00	76
377	0.66	0.18	0.28	106
378	0.17	0.01	0.02	86
379	0.20	0.07	0.11	14
380	0.94	0.14	0.24	122
381	0.11	0.05	0.07	104
382	0.19	0.08	0.11	66
383	0.49	0.26	0.34	110
384	0.20	0.01	0.02	155

385	0.22	0.04	0.07	50
386	0.22	0.17	0.19	64
387	0.19	0.03	0.06	93
388	0.54	0.20	0.29	102
389	0.10	0.02	0.03	108
390	0.95	0.32	0.48	178
391	0.58	0.16	0.25	115
392	0.50	0.21	0.30	42
393	0.00	0.00	0.00	134
394	0.06	0.01	0.02	112
395	0.45	0.21	0.29	176
396	0.19	0.02	0.04	125
397	0.69	0.21	0.32	224
398	0.85	0.27	0.41	63
399	0.00	0.00	0.00	59
400	0.42	0.29	0.34	63
401	0.23	0.16	0.19	98
402	0.35	0.07	0.12	162
403	0.33	0.20	0.25	83
404	0.76	0.68	0.72	19
405	0.19	0.12	0.15	92
406	0.60	0.22	0.32	41
407	0.74	0.33	0.45	43
408	0.66	0.18	0.28	160
409	0.28	0.22	0.25	50
410	0.00	0.00	0.00	19
411	0.28	0.15	0.20	175
412	0.29	0.06	0.09	72
413	0.40	0.04	0.08	95
414	0.17	0.10	0.13	97
415	0.20	0.10	0.14	48
416	0.43	0.29	0.35	83
417	0.11	0.03	0.04	40
418	0.25	0.10	0.14	91
419	0.42	0.28	0.34	90
420	0.18	0.11	0.14	37
421	0.10	0.05	0.06	66
422	0.54	0.37	0.44	73
423	0.41	0.20	0.27	56
424	0.95	0.58	0.72	33
425	0.05	0.01	0.02	76
426	0.19	0.06	0.09	81
427	1.00	0.32	0.48	150
428	1.00	0.52	0.68	29
429	1.00	0.07	0.13	389
430	0.62	0.20	0.31	167
431	0.32	0.06	0.10	123
432	0.37	0.26	0.30	39

433	0.42	0.29	0.35	82
434	1.00	0.42	0.60	66
435	0.60	0.39	0.47	93
436	0.55	0.20	0.29	87
437	0.24	0.05	0.08	86
438	0.81	0.34	0.48	104
439	0.55	0.11	0.18	100
440	0.36	0.04	0.06	141
441	0.36	0.32	0.34	110
442	0.26	0.17	0.21	123
443	0.00	0.00	0.00	71
444	0.21	0.03	0.05	109
445	0.22	0.12	0.16	48
446	0.33	0.20	0.25	76
447	0.17	0.13	0.15	38
448	0.69	0.49	0.58	81
449	0.51	0.20	0.29	132
450	0.49	0.28	0.36	81
451	0.80	0.16	0.26	76
452	0.00	0.00	0.00	44
453	0.12	0.02	0.04	44
454	0.73	0.31	0.44	70
455	0.22	0.10	0.14	155
456	0.33	0.21	0.26	43
457	0.40	0.22	0.29	72
458	0.17	0.06	0.09	62
459	0.50	0.12	0.19	69
460	0.05	0.03	0.03	119
461	0.72	0.23	0.35	79
462	0.31	0.11	0.16	47
463	0.21	0.07	0.10	104
464	0.59	0.36	0.45	106
465	0.62	0.12	0.21	64
466	0.58	0.24	0.34	173
467	0.66	0.23	0.34	107
468	0.48	0.08	0.14	126
469	0.00	0.00	0.00	114
470	0.95	0.51	0.67	140
471	0.62	0.06	0.11	79
472	0.30	0.22	0.26	143
473	0.50	0.18	0.27	158
474	0.27	0.05	0.09	138
475	0.07	0.03	0.04	59
476	0.62	0.28	0.39	88
477	0.85	0.42	0.56	176
478	0.93	0.54	0.68	24
479	0.18	0.04	0.07	92
480	0.83	0.30	0.44	100

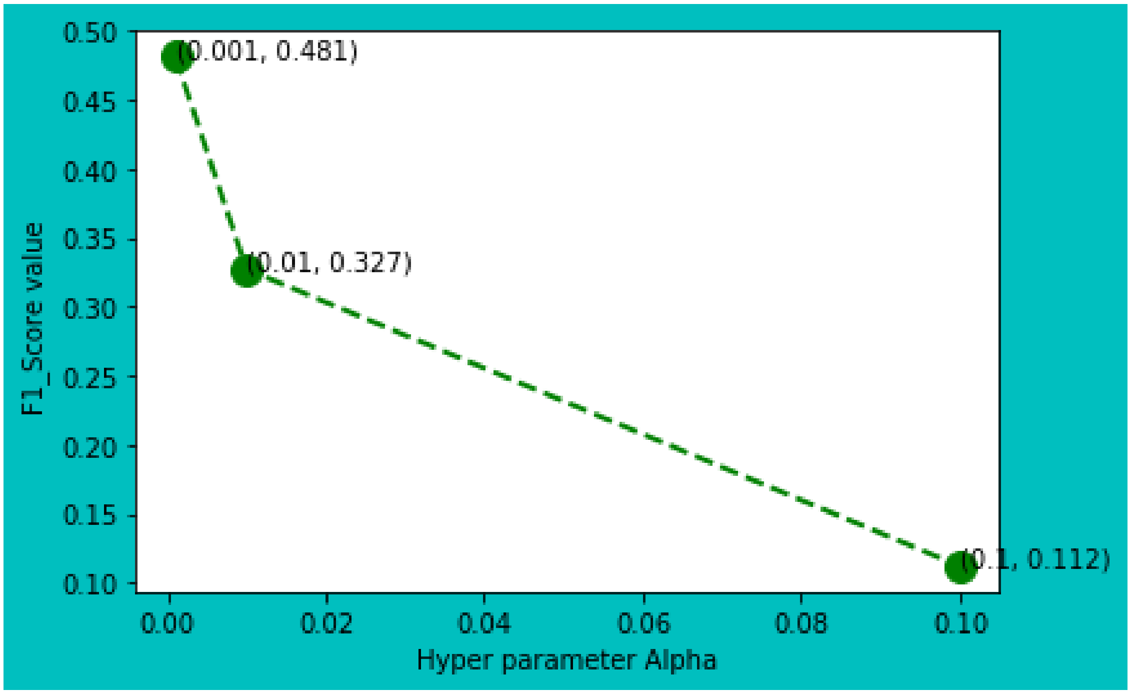
```
        481       0.41      0.19      0.26       103
        482       0.30      0.27      0.28        74
        483       0.82      0.30      0.44       105
        484       0.03      0.01      0.02        83
        485       0.10      0.02      0.04        82
        486       0.35      0.15      0.22        71
        487       0.36      0.20      0.26       120
        488       0.20      0.02      0.03       105
        489       0.62      0.23      0.34        87
        490       0.95      0.59      0.73        32
        491       0.00      0.00      0.00        69
        492       0.25      0.02      0.04        49
        493       0.06      0.01      0.01       117
        494       0.43      0.05      0.09        61
        495       1.00      0.08      0.15       344
        496       0.31      0.15      0.21        52
        497       0.57      0.12      0.19       137
        498       0.42      0.05      0.09        98
        499       0.71      0.06      0.12        79

avg / total       0.64      0.32      0.41    173812

Time taken to run this cell : 0:15:15.119457
```

2.2 OneVsRestClassifier with Logistic regression(penalty=l1)

In [17]:
```
start = datetime.now()
import warnings
warnings.filterwarnings('ignore')

# hp1={'estimator__C':alpha}

cv_scores = []
for i in alpha:
    print(i)
    hp1={'estimator__alpha':[i],
         'estimator__loss':['log'],
         'estimator__penalty':['l1']}
    print(hp1)
    classifier = OneVsRestClassifier(SGDClassifier())

    model11 =GridSearchCV(classifier,hp1,
                          cv=3, scoring='f1_micro',n_jobs=-1)
    print("Gridsearchcv")
    best_model1=model11.fit(x_train_multilabel, y_train)
    print('fit model')
    Train_model_score=best_model1.score(x_train_multilabel,
```

```
                                                y_train)
        #print("best_model1")
            cv_scores.append(Train_model_score.mean())

        fscore = [x for x in cv_scores]

        # determining best alpha
        optimal_alpha22 = alpha[fscore.index(max(fscore))]
        print('\n The optimal value of alpha with penalty=l1 and loss= log is %d.' % optimal_al

        # Plots
        fig4 = plt.figure( facecolor='c', edgecolor='k')
        plt.plot(alpha, fscore,color='green', marker='o', linestyle='dashed',
        linewidth=2, markersize=12)

        for xy in zip(alpha, np.round(fscore,3)):
            plt.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')

        plt.xlabel('Hyper parameter Alpha')
        plt.ylabel('F1_Score value ')
        plt.show()

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

```
0.001
{'estimator__alpha': [0.001], 'estimator__loss': ['log'], 'estimator__penalty': ['l1']}
Gridsearchcv
fit model
0.01
{'estimator__alpha': [0.01], 'estimator__loss': ['log'], 'estimator__penalty': ['l1']}
Gridsearchcv
fit model
0.1
{'estimator__alpha': [0.1], 'estimator__loss': ['log'], 'estimator__penalty': ['l1']}
Gridsearchcv
fit model

 The optimal value of alpha with penalty=l1 and loss= log is 0.
```




```
Time taken to run this cell : 2:56:17.727412
```

In [18]:
```python
start = datetime.now()
best_model2 = OneVsRestClassifier(SGDClassifier(loss='log', alpha=optimal_alpha22,
                                                 penalty='l1'), n_jobs=-1)
best_model2.fit(x_train_multilabel, y_train)
```

Out[18]:
```
OneVsRestClassifier(estimator=SGDClassifier(alpha=0.001, average=False, class_weight=No
       eta0=0.0, fit_intercept=True, l1_ratio=0.15,
       learning_rate='optimal', loss='log', max_iter=None, n_iter=None,
       n_jobs=1, penalty='l1', power_t=0.5, random_state=None,
       shuffle=True, tol=None, verbose=0, warm_start=False),
          n_jobs=-1)
```

In [19]:
```python
joblib.dump(best_model2, 'best_model2_LR.pkl')
```

Out[19]:
```
['best_model2_LR.pkl']
```

In [ ]:

In [20]:
```python
best_model2=joblib.load('best_model2_LR.pkl')
```

2.3 Logistic regression with l1 penalty

In [21]:
```python
start = datetime.now()
#classifier = OneVsRestClassifier(LogisticRegression(penalty='l1'), n_jobs=-1)
```

```
#classifier.fit(x_train_multilabel, y_train)
predictions = best_model2.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,

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,

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

```
Accuracy : 0.1879
Hamming loss  0.00319694
Micro-average quality numbers
Precision: 0.5718, Recall: 0.3201, F1-measure: 0.4104
Macro-average quality numbers
Precision: 0.4113, Recall: 0.2385, F1-measure: 0.2830
             precision    recall  f1-score   support

          0       0.68      0.68      0.68      5519
          1       0.57      0.20      0.29      8190
          2       0.75      0.33      0.46      6529
          3       0.76      0.40      0.52      3231
          4       0.70      0.42      0.53      6430
          5       0.62      0.39      0.48      2879
          6       0.72      0.55      0.62      5086
          7       0.83      0.60      0.69      4533
          8       0.48      0.14      0.22      3000
          9       0.75      0.48      0.59      2765
         10       0.57      0.14      0.23      3051
         11       0.66      0.37      0.48      3009
         12       0.61      0.22      0.32      2630
         13       0.54      0.14      0.22      1426
         14       0.81      0.61      0.70      2548
         15       0.64      0.12      0.20      2371
         16       0.49      0.28      0.35       873
         17       0.74      0.68      0.71      2151
         18       0.63      0.22      0.33      2204
```

19	0.62	0.42	0.50	831
20	0.70	0.51	0.59	1860
21	0.24	0.11	0.15	2023
22	0.34	0.25	0.28	1513
23	0.90	0.45	0.60	1207
24	0.47	0.33	0.39	506
25	0.67	0.32	0.43	425
26	0.46	0.41	0.44	793
27	0.54	0.31	0.39	1291
28	0.62	0.32	0.42	1208
29	0.26	0.09	0.14	406
30	0.50	0.26	0.35	504
31	0.26	0.14	0.18	732
32	0.47	0.35	0.40	441
33	0.35	0.11	0.17	1645
34	0.51	0.34	0.41	1058
35	0.72	0.59	0.65	946
36	0.48	0.29	0.36	644
37	0.61	0.77	0.68	136
38	0.56	0.43	0.49	570
39	0.76	0.36	0.48	766
40	0.53	0.27	0.35	1132
41	0.33	0.22	0.27	174
42	0.47	0.51	0.49	210
43	0.62	0.51	0.56	433
44	0.57	0.47	0.52	626
45	0.39	0.28	0.33	852
46	0.66	0.38	0.48	534
47	0.20	0.24	0.22	350
48	0.52	0.60	0.55	496
49	0.79	0.59	0.67	785
50	0.16	0.15	0.16	475
51	0.24	0.12	0.16	305
52	0.16	0.09	0.11	251
53	0.59	0.39	0.47	914
54	0.43	0.18	0.25	728
55	0.00	0.00	0.00	258
56	0.37	0.14	0.20	821
57	0.38	0.14	0.20	541
58	0.54	0.33	0.41	748
59	0.87	0.67	0.76	724
60	0.23	0.09	0.13	660
61	0.63	0.29	0.39	235
62	0.89	0.68	0.77	718
63	0.84	0.49	0.62	468
64	0.49	0.46	0.47	191
65	0.19	0.16	0.17	429
66	0.17	0.10	0.12	415

67	0.66	0.51	0.58	274
68	0.84	0.50	0.63	510
69	0.63	0.44	0.52	466
70	0.20	0.18	0.19	305
71	0.38	0.17	0.23	247
72	0.71	0.41	0.52	401
73	0.93	0.78	0.85	86
74	0.69	0.31	0.43	120
75	0.77	0.79	0.78	129
76	0.04	0.01	0.02	473
77	0.30	0.31	0.31	143
78	0.77	0.41	0.54	347
79	0.55	0.23	0.33	479
80	0.35	0.32	0.33	279
81	0.80	0.11	0.20	461
82	0.13	0.04	0.07	298
83	0.70	0.40	0.51	396
84	0.37	0.33	0.35	184
85	0.30	0.18	0.23	573
86	0.11	0.01	0.02	325
87	0.51	0.23	0.32	273
88	0.27	0.21	0.24	135
89	0.19	0.15	0.17	232
90	0.48	0.35	0.40	409
91	0.51	0.36	0.42	420
92	0.63	0.60	0.62	408
93	0.58	0.47	0.52	241
94	0.23	0.09	0.12	211
95	0.14	0.19	0.16	277
96	0.14	0.13	0.13	410
97	0.82	0.15	0.25	501
98	0.69	0.63	0.66	136
99	0.49	0.25	0.33	239
100	0.34	0.09	0.14	324
101	0.54	0.50	0.52	277
102	0.82	0.75	0.78	613
103	0.45	0.18	0.26	157
104	0.17	0.09	0.12	295
105	0.60	0.40	0.48	334
106	0.07	0.01	0.01	335
107	0.77	0.47	0.59	389
108	0.18	0.25	0.21	251
109	0.42	0.29	0.34	317
110	0.52	0.06	0.11	187
111	0.24	0.15	0.18	140
112	0.12	0.03	0.04	154
113	0.40	0.39	0.40	332
114	0.40	0.20	0.27	323

115	0.35	0.09	0.14	344
116	0.59	0.48	0.53	370
117	0.50	0.17	0.26	313
118	0.79	0.53	0.63	874
119	0.36	0.16	0.22	293
120	0.01	0.01	0.01	200
121	0.75	0.42	0.54	463
122	0.27	0.32	0.29	119
123	0.00	0.00	0.00	256
124	0.87	0.73	0.79	195
125	0.34	0.17	0.22	138
126	0.62	0.50	0.55	376
127	0.20	0.07	0.11	122
128	0.17	0.06	0.09	252
129	0.50	0.02	0.04	144
130	0.09	0.02	0.03	150
131	0.10	0.01	0.02	210
132	0.43	0.08	0.14	361
133	0.89	0.55	0.68	453
134	0.83	0.70	0.76	124
135	0.00	0.00	0.00	91
136	0.18	0.30	0.23	128
137	0.44	0.29	0.35	218
138	0.09	0.00	0.01	243
139	0.32	0.21	0.25	149
140	0.68	0.29	0.41	318
141	0.10	0.14	0.12	159
142	0.69	0.30	0.42	274
143	0.78	0.79	0.79	362
144	0.50	0.24	0.32	118
145	0.61	0.39	0.48	164
146	0.57	0.25	0.35	461
147	0.61	0.42	0.50	159
148	0.35	0.13	0.19	166
149	0.92	0.56	0.70	346
150	0.42	0.01	0.03	350
151	0.79	0.62	0.69	55
152	0.75	0.44	0.56	387
153	0.00	0.00	0.00	150
154	0.40	0.19	0.26	281
155	0.25	0.12	0.17	202
156	0.62	0.65	0.64	130
157	0.30	0.11	0.16	245
158	0.83	0.48	0.61	177
159	0.42	0.24	0.31	130
160	0.46	0.17	0.25	336
161	0.81	0.60	0.69	220
162	0.10	0.03	0.05	229

163	0.85	0.46	0.59	316
164	0.38	0.23	0.29	283
165	0.56	0.28	0.37	197
166	0.12	0.08	0.10	101
167	0.34	0.22	0.27	231
168	0.31	0.12	0.17	370
169	0.27	0.28	0.27	258
170	0.12	0.05	0.07	101
171	0.37	0.21	0.27	89
172	0.28	0.45	0.35	193
173	0.38	0.31	0.34	309
174	0.46	0.15	0.23	172
175	0.87	0.73	0.79	95
176	0.72	0.71	0.71	346
177	0.92	0.34	0.50	322
178	0.52	0.43	0.47	232
179	0.57	0.03	0.06	125
180	0.42	0.19	0.26	145
181	0.10	0.22	0.13	77
182	0.15	0.04	0.07	182
183	0.53	0.35	0.42	257
184	0.13	0.04	0.06	216
185	0.26	0.08	0.12	242
186	0.29	0.17	0.21	165
187	0.72	0.53	0.61	263
188	0.28	0.11	0.16	174
189	0.63	0.09	0.15	136
190	0.94	0.51	0.66	202
191	0.31	0.23	0.26	134
192	0.79	0.36	0.49	230
193	0.21	0.16	0.18	90
194	0.55	0.51	0.53	185
195	0.09	0.04	0.06	156
196	0.00	0.00	0.00	160
197	0.00	0.00	0.00	266
198	0.44	0.07	0.12	284
199	0.14	0.07	0.09	145
200	0.91	0.59	0.72	212
201	0.25	0.04	0.07	317
202	0.57	0.65	0.61	427
203	0.16	0.17	0.16	232
204	0.26	0.17	0.20	217
205	0.45	0.35	0.39	527
206	0.07	0.02	0.03	124
207	0.00	0.00	0.00	103
208	0.77	0.59	0.67	287
209	0.15	0.09	0.11	193
210	0.46	0.21	0.29	220

211	0.00	0.00	0.00	140
212	0.08	0.18	0.11	161
213	0.50	0.18	0.27	72
214	0.60	0.50	0.54	396
215	0.87	0.25	0.39	134
216	0.00	0.00	0.00	400
217	0.43	0.33	0.38	75
218	0.90	0.80	0.85	219
219	0.70	0.38	0.49	210
220	0.90	0.32	0.47	298
221	0.96	0.52	0.67	266
222	0.82	0.29	0.43	290
223	0.19	0.04	0.06	128
224	0.77	0.32	0.45	159
225	0.43	0.29	0.34	164
226	0.51	0.36	0.42	144
227	0.44	0.40	0.42	276
228	0.02	0.00	0.01	235
229	0.12	0.00	0.01	216
230	0.32	0.20	0.25	228
231	0.66	0.45	0.54	64
232	0.08	0.04	0.05	103
233	0.74	0.27	0.40	216
234	0.00	0.00	0.00	116
235	0.46	0.35	0.40	77
236	0.94	0.67	0.78	67
237	0.00	0.00	0.00	218
238	0.09	0.04	0.05	139
239	0.24	0.04	0.07	94
240	0.45	0.32	0.38	77
241	0.33	0.01	0.01	167
242	0.07	0.19	0.10	86
243	0.12	0.14	0.13	58
244	0.25	0.13	0.18	269
245	0.11	0.04	0.05	112
246	0.96	0.61	0.74	255
247	0.25	0.24	0.25	58
248	0.09	0.05	0.06	81
249	0.00	0.00	0.00	131
250	0.12	0.14	0.13	93
251	0.30	0.32	0.31	154
252	0.07	0.02	0.03	129
253	0.41	0.35	0.38	83
254	0.23	0.09	0.13	191
255	0.12	0.01	0.02	219
256	0.07	0.01	0.01	130
257	0.37	0.31	0.34	93
258	0.44	0.65	0.53	217

259	0.18	0.06	0.09	141
260	0.94	0.10	0.19	143
261	0.47	0.08	0.14	219
262	0.38	0.34	0.36	107
263	0.30	0.35	0.32	236
264	0.22	0.23	0.22	119
265	0.15	0.14	0.14	72
266	0.20	0.07	0.11	70
267	0.13	0.21	0.16	107
268	0.67	0.34	0.45	169
269	0.22	0.16	0.18	129
270	0.49	0.65	0.56	159
271	0.42	0.13	0.19	190
272	0.45	0.10	0.16	248
273	0.89	0.74	0.81	264
274	0.86	0.56	0.68	105
275	0.13	0.05	0.07	104
276	0.03	0.04	0.04	115
277	0.85	0.50	0.63	170
278	0.43	0.16	0.23	145
279	0.60	0.40	0.48	230
280	0.57	0.42	0.49	80
281	0.60	0.71	0.65	217
282	0.77	0.49	0.60	175
283	0.00	0.00	0.00	269
284	0.53	0.22	0.31	74
285	0.67	0.66	0.66	206
286	0.84	0.52	0.64	227
287	0.83	0.27	0.41	130
288	0.28	0.12	0.17	129
289	0.20	0.01	0.02	80
290	0.15	0.09	0.11	99
291	0.76	0.23	0.35	208
292	0.26	0.12	0.16	67
293	0.50	0.26	0.34	109
294	0.24	0.24	0.24	140
295	0.16	0.13	0.14	241
296	0.17	0.12	0.14	72
297	0.29	0.11	0.16	107
298	0.71	0.20	0.31	61
299	0.53	0.35	0.42	77
300	0.16	0.05	0.08	111
301	0.00	0.00	0.00	126
302	0.06	0.01	0.02	73
303	0.50	0.43	0.46	176
304	0.82	0.66	0.73	230
305	0.84	0.73	0.78	156
306	0.41	0.34	0.37	146

307	0.16	0.05	0.08	98
308	0.25	0.01	0.02	78
309	0.40	0.02	0.04	94
310	0.67	0.25	0.37	162
311	0.59	0.65	0.62	116
312	0.47	0.26	0.34	57
313	0.00	0.00	0.00	65
314	0.49	0.35	0.41	138
315	0.36	0.26	0.30	195
316	0.25	0.42	0.32	69
317	0.00	0.00	0.00	134
318	0.33	0.26	0.29	148
319	0.70	0.20	0.32	161
320	0.13	0.14	0.14	104
321	0.73	0.47	0.58	156
322	0.45	0.23	0.31	134
323	0.57	0.30	0.39	232
324	0.06	0.17	0.09	92
325	0.25	0.09	0.13	197
326	0.00	0.00	0.00	126
327	0.33	0.01	0.02	115
328	0.99	0.45	0.62	198
329	0.49	0.26	0.34	125
330	0.60	0.04	0.07	81
331	0.12	0.02	0.04	94
332	0.00	0.00	0.00	56
333	0.03	0.00	0.01	260
334	0.00	0.00	0.00	60
335	0.21	0.14	0.17	110
336	0.49	0.46	0.48	71
337	0.12	0.06	0.08	66
338	0.44	0.33	0.37	150
339	0.00	0.00	0.00	54
340	0.86	0.48	0.62	195
341	0.00	0.00	0.00	79
342	0.25	0.34	0.29	38
343	0.37	0.23	0.29	43
344	0.33	0.01	0.03	68
345	0.54	0.44	0.48	73
346	0.00	0.00	0.00	116
347	0.71	0.48	0.57	111
348	0.12	0.05	0.07	63
349	0.89	0.49	0.63	104
350	0.71	0.34	0.46	44
351	0.00	0.00	0.00	40
352	0.93	0.40	0.56	136
353	0.40	0.39	0.40	54
354	0.14	0.07	0.10	134

355	0.28	0.11	0.16	120
356	0.28	0.16	0.20	228
357	0.57	0.09	0.15	269
358	0.66	0.34	0.45	80
359	0.75	0.15	0.25	140
360	0.10	0.19	0.13	125
361	0.88	0.43	0.57	169
362	0.10	0.05	0.07	56
363	0.86	0.59	0.70	154
364	0.00	0.00	0.00	58
365	0.12	0.11	0.12	71
366	0.97	0.54	0.69	54
367	0.14	0.07	0.09	116
368	0.00	0.00	0.00	54
369	0.00	0.00	0.00	71
370	0.03	0.07	0.04	61
371	0.00	0.00	0.00	71
372	0.72	0.44	0.55	52
373	0.67	0.36	0.47	150
374	0.38	0.19	0.26	93
375	0.25	0.01	0.03	67
376	0.00	0.00	0.00	76
377	0.91	0.09	0.17	106
378	0.50	0.01	0.02	86
379	0.14	0.07	0.10	14
380	1.00	0.25	0.39	122
381	0.03	0.01	0.01	104
382	0.24	0.18	0.21	66
383	0.44	0.24	0.31	110
384	0.00	0.00	0.00	155
385	0.08	0.02	0.03	50
386	0.22	0.19	0.20	64
387	0.00	0.00	0.00	93
388	0.53	0.21	0.30	102
389	0.33	0.01	0.02	108
390	0.83	0.70	0.76	178
391	0.53	0.14	0.22	115
392	0.92	0.29	0.44	42
393	0.00	0.00	0.00	134
394	0.00	0.00	0.00	112
395	0.25	0.03	0.06	176
396	0.00	0.00	0.00	125
397	0.44	0.24	0.31	224
398	0.64	0.48	0.55	63
399	0.00	0.00	0.00	59
400	0.33	0.25	0.29	63
401	0.10	0.02	0.03	98
402	0.36	0.06	0.10	162

403	0.29	0.14	0.19	83
404	0.63	0.89	0.74	19
405	0.13	0.08	0.10	92
406	0.33	0.15	0.20	41
407	0.56	0.23	0.33	43
408	0.80	0.05	0.09	160
409	0.22	0.16	0.18	50
410	0.00	0.00	0.00	19
411	0.32	0.14	0.20	175
412	0.08	0.01	0.02	72
413	0.50	0.02	0.04	95
414	0.08	0.06	0.07	97
415	0.18	0.25	0.21	48
416	0.38	0.25	0.30	83
417	0.00	0.00	0.00	40
418	0.19	0.07	0.10	91
419	0.38	0.26	0.31	90
420	0.27	0.24	0.26	37
421	0.04	0.03	0.03	66
422	0.57	0.27	0.37	73
423	0.34	0.20	0.25	56
424	0.65	0.85	0.74	33
425	0.00	0.00	0.00	76
426	0.00	0.00	0.00	81
427	0.99	0.50	0.66	150
428	0.95	0.66	0.78	29
429	0.00	0.00	0.00	389
430	0.65	0.22	0.32	167
431	0.00	0.00	0.00	123
432	0.38	0.23	0.29	39
433	0.35	0.22	0.27	82
434	0.18	0.47	0.26	66
435	0.51	0.29	0.37	93
436	0.14	0.01	0.02	87
437	0.25	0.03	0.06	86
438	0.66	0.37	0.47	104
439	0.02	0.01	0.01	100
440	0.33	0.01	0.01	141
441	0.29	0.23	0.26	110
442	0.22	0.09	0.13	123
443	0.00	0.00	0.00	71
444	0.36	0.05	0.08	109
445	0.23	0.12	0.16	48
446	0.36	0.18	0.24	76
447	0.04	0.03	0.03	38
448	0.66	0.43	0.52	81
449	0.47	0.06	0.11	132
450	0.39	0.30	0.34	81

451	0.89	0.11	0.19	76
452	0.00	0.00	0.00	44
453	0.00	0.00	0.00	44
454	0.88	0.30	0.45	70
455	0.11	0.01	0.01	155
456	0.22	0.16	0.19	43
457	0.31	0.15	0.21	72
458	0.23	0.11	0.15	62
459	1.00	0.09	0.16	69
460	0.25	0.03	0.06	119
461	0.68	0.16	0.27	79
462	0.17	0.02	0.04	47
463	0.11	0.01	0.02	104
464	0.37	0.33	0.35	106
465	0.00	0.00	0.00	64
466	0.55	0.20	0.29	173
467	0.66	0.48	0.55	107
468	0.50	0.01	0.02	126
469	0.00	0.00	0.00	114
470	0.94	0.72	0.81	140
471	0.00	0.00	0.00	79
472	0.32	0.27	0.29	143
473	0.56	0.23	0.32	158
474	1.00	0.01	0.01	138
475	0.04	0.05	0.05	59
476	0.58	0.39	0.46	88
477	0.81	0.45	0.58	176
478	0.92	0.50	0.65	24
479	0.00	0.00	0.00	92
480	0.78	0.28	0.41	100
481	0.44	0.04	0.07	103
482	0.22	0.22	0.22	74
483	0.76	0.45	0.56	105
484	0.05	0.01	0.02	83
485	0.11	0.01	0.02	82
486	0.33	0.03	0.05	71
487	0.39	0.21	0.27	120
488	0.00	0.00	0.00	105
489	0.60	0.17	0.27	87
490	1.00	0.75	0.86	32
491	0.00	0.00	0.00	69
492	0.00	0.00	0.00	49
493	0.00	0.00	0.00	117
494	0.80	0.07	0.12	61
495	0.98	0.62	0.76	344
496	0.16	0.10	0.12	52
497	0.71	0.04	0.07	137
498	0.00	0.00	0.00	98

```
        499       0.35      0.23      0.28        79

avg / total       0.55      0.32      0.39    173812

Time taken to run this cell : 0:00:19.236854
```

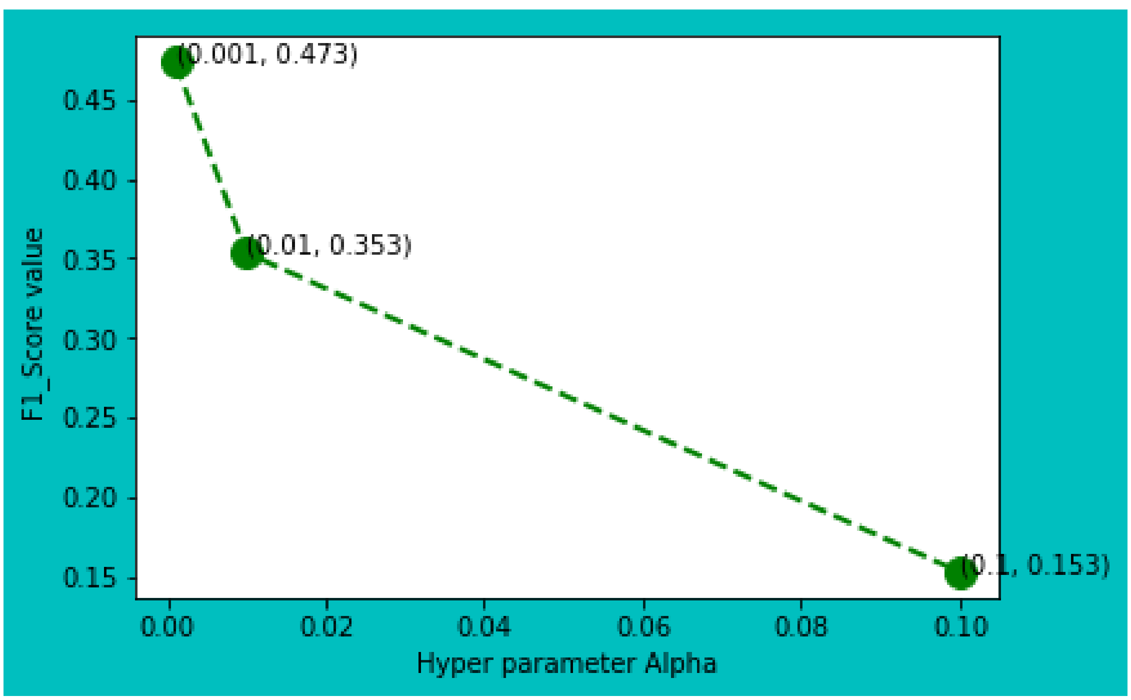
In [ ]:

2.4 OneVsRestClassifier with Linear-SVM (SGDClassifier with loss-hinge)

In [22]:
```
start = datetime.now()
import warnings
warnings.filterwarnings('ignore')

# hp1={'estimator__C':alpha}

cv_scores = []
for i in alpha:
    print(i)
    hp1={'estimator__alpha':[i],
         'estimator__loss':['hinge'],
         'estimator__penalty':['l1']}
    print(hp1)
    classifier = OneVsRestClassifier(SGDClassifier())

    model11 =GridSearchCV(classifier,hp1,
                          cv=3, scoring='f1_micro',n_jobs=-1)
    print("Gridsearchcv")
    best_model1=model11.fit(x_train_multilabel, y_train)
    print('fit model')
    Train_model_score=best_model1.score(x_train_multilabel,
                                        y_train)
#print("best_model1")
    cv_scores.append(Train_model_score.mean())

fscore = [x for x in cv_scores]

# determining best alpha
optimal_alpha23 = alpha[fscore.index(max(fscore))]
print('\n The optimal value of alpha with penalty=l1 and loss= log is %d.' % optimal_al

# Plots
fig4 = plt.figure( facecolor='c', edgecolor='k')
plt.plot(alpha, fscore,color='green', marker='o', linestyle='dashed',
linewidth=2, markersize=12)

for xy in zip(alpha, np.round(fscore,3)):
```

```
        plt.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')

    plt.xlabel('Hyper parameter Alpha')
    plt.ylabel('F1_Score value ')
    plt.show()

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

```
0.001
{'estimator__alpha': [0.001], 'estimator__loss': ['hinge'], 'estimator__penalty': ['l1']}
Gridsearchcv
fit model
0.01
{'estimator__alpha': [0.01], 'estimator__loss': ['hinge'], 'estimator__penalty': ['l1']}
Gridsearchcv
fit model
0.1
{'estimator__alpha': [0.1], 'estimator__loss': ['hinge'], 'estimator__penalty': ['l1']}
Gridsearchcv
fit model

 The optimal value of alpha with penalty=l1 and loss= log is 0.
```




```
Time taken to run this cell : 2:18:49.138029
```

2.5 OneVsRestClassifier with SGDClassifier for optimal alpha with hinge loss

In [23]:
```
start = datetime.now()
classifier2 = OneVsRestClassifier(SGDClassifier(loss='hinge',
                                                alpha=optimal_alpha23,
                                                penalty='l1'))
classifier2=classifier2.fit(x_train_multilabel, y_train)
```

In [24]:
```
joblib.dump(classifier2, 'classifier2.pkl')
```

Out[24]: ['classifier2.pkl']

In [25]:
```
classifier2=joblib.load('classifier2.pkl')
```

In [26]:
```
predictions = classifier2.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-averasge quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall,

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,

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

```
Accuracy : 0.17585
Hamming loss  0.00330166
Micro-averasge quality numbers
Precision: 0.5428, Recall: 0.3186, F1-measure: 0.4015
Macro-average quality numbers
Precision: 0.3193, Recall: 0.2399, F1-measure: 0.2547
             precision    recall  f1-score   support

          0       0.67      0.68      0.68      5519
          1       0.45      0.21      0.29      8190
          2       0.70      0.38      0.49      6529
```

3	0.65	0.43	0.52	3231
4	0.83	0.33	0.47	6430
5	0.58	0.41	0.48	2879
6	0.78	0.57	0.65	5086
7	0.82	0.59	0.68	4533
8	0.44	0.16	0.24	3000
9	0.60	0.59	0.59	2765
10	0.20	0.01	0.02	3051
11	0.65	0.37	0.47	3009
12	0.54	0.29	0.37	2630
13	0.27	0.20	0.23	1426
14	0.77	0.64	0.70	2548
15	0.59	0.14	0.22	2371
16	0.38	0.32	0.35	873
17	0.73	0.69	0.71	2151
18	0.49	0.27	0.35	2204
19	0.55	0.43	0.48	831
20	0.74	0.47	0.57	1860
21	0.27	0.01	0.02	2023
22	0.34	0.02	0.03	1513
23	0.73	0.62	0.67	1207
24	0.00	0.00	0.00	506
25	0.52	0.33	0.41	425
26	0.52	0.36	0.42	793
27	0.52	0.37	0.43	1291
28	0.49	0.40	0.44	1208
29	0.14	0.18	0.16	406
30	0.69	0.25	0.37	504
31	0.00	0.00	0.00	732
32	0.37	0.39	0.38	441
33	0.02	0.00	0.00	1645
34	0.58	0.32	0.41	1058
35	0.66	0.57	0.61	946
36	0.52	0.29	0.37	644
37	0.59	0.82	0.68	136
38	0.48	0.41	0.44	570
39	0.70	0.31	0.43	766
40	0.56	0.08	0.14	1132
41	0.29	0.25	0.27	174
42	0.58	0.63	0.60	210
43	0.61	0.53	0.57	433
44	0.47	0.53	0.50	626
45	0.45	0.28	0.35	852
46	0.61	0.39	0.48	534
47	0.00	0.00	0.00	350
48	0.56	0.62	0.59	496
49	0.71	0.69	0.70	785
50	0.05	0.00	0.00	475

51	0.00	0.00	0.00	305
52	0.06	0.01	0.01	251
53	0.44	0.54	0.48	914
54	0.00	0.00	0.00	728
55	0.03	0.00	0.01	258
56	0.00	0.00	0.00	821
57	0.36	0.06	0.11	541
58	0.68	0.24	0.35	748
59	0.80	0.74	0.77	724
60	0.22	0.09	0.13	660
61	0.62	0.28	0.38	235
62	0.84	0.83	0.83	718
63	0.63	0.68	0.65	468
64	0.47	0.44	0.45	191
65	0.12	0.19	0.14	429
66	0.00	0.00	0.00	415
67	0.63	0.65	0.64	274
68	0.74	0.63	0.68	510
69	0.51	0.49	0.50	466
70	0.00	0.00	0.00	305
71	0.14	0.26	0.18	247
72	0.62	0.52	0.56	401
73	0.88	0.78	0.83	86
74	0.26	0.41	0.32	120
75	0.84	0.75	0.79	129
76	0.00	0.00	0.00	473
77	0.23	0.43	0.30	143
78	0.73	0.51	0.60	347
79	0.57	0.36	0.44	479
80	0.23	0.41	0.30	279
81	0.62	0.13	0.22	461
82	0.03	0.04	0.03	298
83	0.63	0.50	0.56	396
84	0.36	0.33	0.34	184
85	0.30	0.11	0.16	573
86	0.37	0.04	0.07	325
87	0.53	0.21	0.30	273
88	0.30	0.35	0.32	135
89	0.00	0.00	0.00	232
90	0.30	0.42	0.35	409
91	0.60	0.29	0.39	420
92	0.64	0.58	0.61	408
93	0.42	0.59	0.49	241
94	0.00	0.00	0.00	211
95	0.00	0.00	0.00	277
96	0.00	0.00	0.00	410
97	0.84	0.15	0.25	501
98	0.56	0.68	0.62	136

99	0.44	0.24	0.31	239
100	0.08	0.15	0.11	324
101	0.67	0.61	0.64	277
102	0.85	0.69	0.76	613
103	0.25	0.20	0.22	157
104	0.00	0.00	0.00	295
105	0.72	0.37	0.49	334
106	0.00	0.00	0.00	335
107	0.54	0.60	0.57	389
108	0.33	0.21	0.26	251
109	0.39	0.42	0.40	317
110	0.00	0.00	0.00	187
111	0.17	0.15	0.16	140
112	0.09	0.05	0.07	154
113	0.49	0.31	0.38	332
114	0.00	0.00	0.00	323
115	0.19	0.16	0.17	344
116	0.58	0.61	0.59	370
117	0.42	0.15	0.22	313
118	0.69	0.73	0.71	874
119	0.41	0.16	0.23	293
120	0.00	0.00	0.00	200
121	0.60	0.49	0.54	463
122	0.00	0.00	0.00	119
123	0.00	0.00	0.00	256
124	0.80	0.82	0.81	195
125	0.30	0.05	0.09	138
126	0.56	0.57	0.56	376
127	0.00	0.00	0.00	122
128	0.02	0.00	0.01	252
129	0.00	0.00	0.00	144
130	0.42	0.18	0.25	150
131	0.00	0.00	0.00	210
132	0.62	0.02	0.04	361
133	0.80	0.64	0.71	453
134	0.68	0.76	0.71	124
135	0.00	0.00	0.00	91
136	0.51	0.14	0.22	128
137	0.36	0.36	0.36	218
138	0.60	0.10	0.17	243
139	0.00	0.00	0.00	149
140	0.61	0.31	0.41	318
141	0.07	0.18	0.10	159
142	0.58	0.30	0.39	274
143	0.76	0.66	0.70	362
144	0.32	0.31	0.32	118
145	0.41	0.49	0.45	164
146	0.41	0.46	0.44	461

147	0.57	0.60	0.59	159
148	0.18	0.05	0.08	166
149	0.94	0.51	0.66	346
150	0.30	0.05	0.08	350
151	0.81	0.64	0.71	55
152	0.59	0.53	0.56	387
153	0.58	0.05	0.09	150
154	0.36	0.11	0.17	281
155	0.11	0.07	0.09	202
156	0.50	0.72	0.59	130
157	0.00	0.00	0.00	245
158	0.64	0.49	0.55	177
159	0.40	0.29	0.34	130
160	0.25	0.25	0.25	336
161	0.60	0.69	0.64	220
162	0.00	0.00	0.00	229
163	0.79	0.46	0.58	316
164	0.69	0.27	0.39	283
165	0.30	0.47	0.37	197
166	0.38	0.05	0.09	101
167	0.00	0.00	0.00	231
168	0.26	0.23	0.24	370
169	0.30	0.26	0.28	258
170	0.05	0.01	0.02	101
171	0.31	0.18	0.23	89
172	0.21	0.30	0.24	193
173	0.36	0.38	0.37	309
174	0.18	0.19	0.18	172
175	0.66	0.75	0.70	95
176	0.68	0.60	0.64	346
177	0.86	0.39	0.54	322
178	0.51	0.54	0.53	232
179	0.00	0.00	0.00	125
180	0.37	0.34	0.36	145
181	0.19	0.21	0.20	77
182	0.00	0.00	0.00	182
183	0.39	0.49	0.43	257
184	0.07	0.08	0.08	216
185	0.00	0.00	0.00	242
186	0.00	0.00	0.00	165
187	0.60	0.58	0.59	263
188	0.17	0.20	0.18	174
189	0.00	0.00	0.00	136
190	0.80	0.57	0.66	202
191	0.00	0.00	0.00	134
192	0.68	0.43	0.53	230
193	0.30	0.23	0.26	90
194	0.37	0.54	0.44	185

195	0.00	0.00	0.00	156
196	0.00	0.00	0.00	160
197	0.00	0.00	0.00	266
198	0.00	0.00	0.00	284
199	0.07	0.03	0.04	145
200	0.82	0.76	0.79	212
201	0.00	0.00	0.00	317
202	0.55	0.55	0.55	427
203	0.09	0.02	0.03	232
204	0.00	0.00	0.00	217
205	0.43	0.42	0.42	527
206	0.00	0.00	0.00	124
207	0.24	0.15	0.18	103
208	0.51	0.43	0.47	287
209	0.00	0.00	0.00	193
210	0.48	0.19	0.27	220
211	0.67	0.21	0.32	140
212	0.00	0.00	0.00	161
213	0.37	0.14	0.20	72
214	0.56	0.43	0.48	396
215	0.67	0.29	0.41	134
216	0.06	0.01	0.02	400
217	0.32	0.36	0.34	75
218	0.87	0.74	0.80	219
219	0.79	0.30	0.44	210
220	0.91	0.36	0.51	298
221	0.46	0.69	0.55	266
222	0.44	0.34	0.38	290
223	0.12	0.12	0.12	128
224	0.46	0.48	0.47	159
225	0.53	0.29	0.38	164
226	0.34	0.44	0.38	144
227	0.45	0.25	0.32	276
228	0.00	0.00	0.00	235
229	0.00	0.00	0.00	216
230	0.00	0.00	0.00	228
231	0.69	0.64	0.67	64
232	0.07	0.12	0.09	103
233	0.46	0.34	0.39	216
234	0.33	0.02	0.03	116
235	0.36	0.71	0.48	77
236	0.86	0.73	0.79	67
237	0.00	0.00	0.00	218
238	0.07	0.03	0.04	139
239	0.00	0.00	0.00	94
240	0.47	0.25	0.32	77
241	0.42	0.05	0.09	167
242	0.40	0.43	0.42	86

243	0.05	0.02	0.03	58
244	0.00	0.00	0.00	269
245	0.13	0.12	0.12	112
246	0.73	0.79	0.76	255
247	0.27	0.21	0.24	58
248	0.00	0.00	0.00	81
249	0.00	0.00	0.00	131
250	0.12	0.31	0.17	93
251	0.00	0.00	0.00	154
252	0.00	0.00	0.00	129
253	0.31	0.36	0.33	83
254	0.21	0.12	0.15	191
255	0.00	0.00	0.00	219
256	0.00	0.00	0.00	130
257	0.32	0.25	0.28	93
258	0.58	0.50	0.53	217
259	0.00	0.00	0.00	141
260	0.74	0.20	0.31	143
261	0.53	0.14	0.22	219
262	0.41	0.22	0.29	107
263	0.27	0.33	0.29	236
264	0.11	0.19	0.14	119
265	0.00	0.00	0.00	72
266	0.20	0.11	0.15	70
267	0.23	0.06	0.09	107
268	0.44	0.44	0.44	169
269	0.00	0.00	0.00	129
270	0.53	0.62	0.57	159
271	0.20	0.16	0.18	190
272	0.00	0.00	0.00	248
273	0.84	0.74	0.78	264
274	0.58	0.63	0.61	105
275	0.14	0.06	0.08	104
276	0.00	0.00	0.00	115
277	0.88	0.12	0.22	170
278	0.41	0.31	0.35	145
279	0.83	0.30	0.45	230
280	0.39	0.46	0.42	80
281	0.54	0.64	0.58	217
282	0.63	0.70	0.66	175
283	0.00	0.00	0.00	269
284	0.45	0.43	0.44	74
285	0.60	0.47	0.53	206
286	0.83	0.71	0.77	227
287	0.77	0.26	0.39	130
288	0.16	0.12	0.14	129
289	0.00	0.00	0.00	80
290	0.00	0.00	0.00	99

291	0.51	0.20	0.28	208
292	0.10	0.03	0.05	67
293	1.00	0.01	0.02	109
294	0.00	0.00	0.00	140
295	0.12	0.20	0.15	241
296	0.10	0.12	0.11	72
297	0.20	0.14	0.16	107
298	0.61	0.18	0.28	61
299	0.81	0.17	0.28	77
300	0.00	0.00	0.00	111
301	0.00	0.00	0.00	126
302	0.00	0.00	0.00	73
303	0.31	0.42	0.36	176
304	0.87	0.71	0.78	230
305	0.93	0.58	0.72	156
306	0.34	0.35	0.35	146
307	0.00	0.00	0.00	98
308	0.00	0.00	0.00	78
309	0.48	0.21	0.29	94
310	0.21	0.41	0.28	162
311	0.71	0.51	0.59	116
312	0.34	0.46	0.39	57
313	0.00	0.00	0.00	65
314	0.34	0.34	0.34	138
315	0.30	0.32	0.31	195
316	0.28	0.48	0.35	69
317	0.00	0.00	0.00	134
318	0.23	0.41	0.29	148
319	0.78	0.38	0.51	161
320	0.00	0.00	0.00	104
321	0.57	0.69	0.62	156
322	0.49	0.32	0.39	134
323	0.47	0.28	0.35	232
324	0.00	0.00	0.00	92
325	0.00	0.00	0.00	197
326	0.00	0.00	0.00	126
327	0.00	0.00	0.00	115
328	0.96	0.34	0.50	198
329	0.27	0.38	0.31	125
330	0.67	0.15	0.24	81
331	0.00	0.00	0.00	94
332	0.00	0.00	0.00	56
333	0.00	0.00	0.00	260
334	0.00	0.00	0.00	60
335	0.13	0.19	0.16	110
336	0.32	0.56	0.41	71
337	0.00	0.00	0.00	66
338	0.35	0.25	0.29	150

339	0.00	0.00	0.00	54
340	0.60	0.46	0.52	195
341	1.00	0.03	0.05	79
342	0.38	0.08	0.13	38
343	0.47	0.21	0.29	43
344	0.00	0.00	0.00	68
345	0.37	0.47	0.41	73
346	0.08	0.05	0.06	116
347	0.72	0.23	0.35	111
348	0.00	0.00	0.00	63
349	0.62	0.65	0.64	104
350	0.50	0.43	0.46	44
351	0.00	0.00	0.00	40
352	0.29	0.38	0.33	136
353	0.35	0.31	0.33	54
354	0.00	0.00	0.00	134
355	0.82	0.12	0.20	120
356	0.29	0.14	0.19	228
357	0.62	0.06	0.10	269
358	0.33	0.54	0.41	80
359	0.31	0.33	0.32	140
360	0.00	0.00	0.00	125
361	0.87	0.39	0.54	169
362	0.08	0.05	0.06	56
363	0.82	0.64	0.72	154
364	0.00	0.00	0.00	58
365	0.07	0.23	0.11	71
366	0.97	0.54	0.69	54
367	0.00	0.00	0.00	116
368	0.00	0.00	0.00	54
369	0.00	0.00	0.00	71
370	0.00	0.00	0.00	61
371	0.45	0.07	0.12	71
372	0.41	0.50	0.45	52
373	0.27	0.18	0.22	150
374	0.24	0.32	0.27	93
375	0.00	0.00	0.00	67
376	0.00	0.00	0.00	76
377	0.16	0.07	0.09	106
378	0.00	0.00	0.00	86
379	0.00	0.00	0.00	14
380	1.00	0.03	0.06	122
381	0.00	0.00	0.00	104
382	0.16	0.12	0.14	66
383	0.21	0.26	0.24	110
384	0.00	0.00	0.00	155
385	0.00	0.00	0.00	50
386	0.21	0.16	0.18	64

387	0.00	0.00	0.00	93
388	0.33	0.38	0.35	102
389	0.00	0.00	0.00	108
390	0.85	0.70	0.77	178
391	0.54	0.24	0.34	115
392	0.46	0.43	0.44	42
393	0.00	0.00	0.00	134
394	0.00	0.00	0.00	112
395	0.00	0.00	0.00	176
396	0.00	0.00	0.00	125
397	0.52	0.48	0.50	224
398	0.59	0.37	0.45	63
399	0.00	0.00	0.00	59
400	0.32	0.46	0.38	63
401	0.00	0.00	0.00	98
402	0.00	0.00	0.00	162
403	0.04	0.22	0.06	83
404	0.65	0.79	0.71	19
405	0.00	0.00	0.00	92
406	0.15	0.27	0.19	41
407	0.36	0.28	0.32	43
408	0.04	0.03	0.03	160
409	0.00	0.00	0.00	50
410	0.00	0.00	0.00	19
411	0.25	0.12	0.16	175
412	0.00	0.00	0.00	72
413	0.20	0.11	0.14	95
414	0.00	0.00	0.00	97
415	0.00	0.00	0.00	48
416	0.27	0.36	0.31	83
417	0.00	0.00	0.00	40
418	0.00	0.00	0.00	91
419	0.27	0.22	0.25	90
420	0.29	0.46	0.35	37
421	0.00	0.00	0.00	66
422	0.44	0.36	0.39	73
423	0.37	0.25	0.30	56
424	0.88	0.88	0.88	33
425	0.00	0.00	0.00	76
426	0.00	0.00	0.00	81
427	0.96	0.73	0.83	150
428	0.58	0.76	0.66	29
429	0.00	0.00	0.00	389
430	0.47	0.18	0.26	167
431	0.00	0.00	0.00	123
432	0.29	0.31	0.30	39
433	0.28	0.34	0.31	82
434	0.95	0.55	0.69	66

435	0.47	0.44	0.46	93
436	0.00	0.00	0.00	87
437	0.18	0.07	0.10	86
438	0.35	0.61	0.45	104
439	0.00	0.00	0.00	100
440	0.00	0.00	0.00	141
441	0.29	0.35	0.31	110
442	0.00	0.00	0.00	123
443	0.53	0.11	0.19	71
444	0.14	0.02	0.03	109
445	0.30	0.29	0.29	48
446	0.42	0.21	0.28	76
447	0.00	0.00	0.00	38
448	0.49	0.51	0.50	81
449	0.00	0.00	0.00	132
450	0.47	0.38	0.42	81
451	0.60	0.33	0.42	76
452	0.00	0.00	0.00	44
453	0.00	0.00	0.00	44
454	0.45	0.49	0.47	70
455	0.00	0.00	0.00	155
456	0.00	0.00	0.00	43
457	0.09	0.36	0.14	72
458	0.00	0.00	0.00	62
459	0.00	0.00	0.00	69
460	0.00	0.00	0.00	119
461	0.00	0.00	0.00	79
462	0.00	0.00	0.00	47
463	0.00	0.00	0.00	104
464	0.00	0.00	0.00	106
465	0.00	0.00	0.00	64
466	0.31	0.26	0.28	173
467	0.67	0.21	0.31	107
468	0.00	0.00	0.00	126
469	0.00	0.00	0.00	114
470	0.88	0.59	0.71	140
471	0.00	0.00	0.00	79
472	0.35	0.43	0.39	143
473	0.69	0.11	0.20	158
474	0.00	0.00	0.00	138
475	0.00	0.00	0.00	59
476	0.43	0.62	0.51	88
477	0.65	0.63	0.64	176
478	0.85	0.71	0.77	24
479	0.08	0.10	0.09	92
480	0.25	0.20	0.22	100
481	0.00	0.00	0.00	103
482	0.00	0.00	0.00	74

```
        483       0.70      0.54      0.61       105
        484       0.00      0.00      0.00        83
        485       0.00      0.00      0.00        82
        486       0.24      0.10      0.14        71
        487       0.28      0.53      0.36       120
        488       0.00      0.00      0.00       105
        489       0.62      0.37      0.46        87
        490       1.00      0.81      0.90        32
        491       0.00      0.00      0.00        69
        492       0.00      0.00      0.00        49
        493       0.00      0.00      0.00       117
        494       0.33      0.07      0.11        61
        495       0.00      0.00      0.00       344
        496       0.00      0.00      0.00        52
        497       0.00      0.00      0.00       137
        498       0.29      0.05      0.09        98
        499       0.00      0.00      0.00        79

avg / total       0.47      0.32      0.36    173812

Time taken to run this cell : 0:13:46.246785
```

3 Observation

In [10]:
```python
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Sr.No", "MODEL","FEATURIZATION","PENALTY" ,"ALPHA",'LOSS','MICRO_F1_S
```

In [11]:
```python
x.add_row(["1", 'OneVsRest+SGD Classifier', "Tf-idf","l1",0.0001,"log",0.4488])
x.add_row(["2", 'OneVsRest+SGD(log)=LR', "Bag-of-words","l2",0.001,"log",0.4268])
x.add_row(["3", 'OneVsRest+SGD(log)=LR', "Bag-of-words","l1",0.001,"log",0.4104])
x.add_row(["4", 'OneVsRest+SGD Classifier', "Bag-of-words","l1",0.001,"Hinge",0.4028])
```

In [12]:
```python
print(x)
```

```
+-------+--------------------------+---------------+---------+--------+-------+----------------+
| Sr.No |          MODEL           | FEATURIZATION | PENALTY | ALPHA  |  LOSS | MICRO_F1_SCORE |
+-------+--------------------------+---------------+---------+--------+-------+----------------+
|   1   | OneVsRest+SGD Classifier |     Tf-idf    |    l1   | 0.0001 |  log  |     0.4488     |
|   2   |  OneVsRest+SGD(log)=LR   |  Bag-of-words |    l2   | 0.001  |  log  |     0.4268     |
|   3   |  OneVsRest+SGD(log)=LR   |  Bag-of-words |    l1   | 0.001  |  log  |     0.4104     |
|   4   | OneVsRest+SGD Classifier |  Bag-of-words |    l1   | 0.001  | Hinge |     0.4028     |
+-------+--------------------------+---------------+---------+--------+-------+----------------+
```

- The objective's result is shown as above.

- Model {bag of words upto 4 grams and computed the micro f1 score with Logistic regression(OvR)} performs 42.68% on tag prediction which is not higher than the result obtained with model{ TF-IDF with alpha=00.0001 ,n_grams=(1,3)}
- The performance of model with various alpha value is shown in graph.