Assignment:SO_Tag_Predictor

In [1]:

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

from skmultilearn.adapt import mlknn
from skmultilearn.problem_transform import ClassifierChain
from skmultilearn.problem_transform import BinaryRelevance
from skmultilearn.problem_transform import LabelPowerset

from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from datetime import datetime
```

Stack Overflow: Tag Prediction

1. Business Problem

1.1 Description

Description

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

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

Problem Statemtent

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

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

1.2 Source / useful links

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

1.3 Real World / Business Objectives and Constraints

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

2. Machine Learning problem

2.1 Data

2.1.1 Data Overview

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

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

Test.csv contains the same columns but without the Tags, which y
ou are to predict.
```

Size of Train.csv - 6.75GB

Size of Test.csv - 2GB

Number of rows in Train.csv = 6034195

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

Data Field Explaination

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

Id - Unique identifier for each question

Title - The question's title

Body - The body of the question

Tags - The tags associated with the question in a space-seperated format (all lowercase, should not contain tabs '\t' or ampersands '&')

2.1.2 Example Data point

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

Body :

```
#include<
iostream>\n
#include<
stdlib.h>\n\n
using namespace std;\n\n
int main()\n
{\n
         int n,a[n],x,c,u[n],m[n],e[n][4];\n

         cout<<"Enter the number of variables";\n
cin>>n;\n\n
         cout<<"Enter the Lower, and Upper Limits
of the variables";\n
         for(int y=1; y<n+1; y++)\n
         {\n
            cin>>m[y];\n
            cin>>u[y];\n
         }\n
         for(x=1; x<n+1; x++)\n
         {\n
            a[x] = (m[x] + u[x])/2;\n
         }\n
         c=(n*4)-4;\n
         for(int a1=1; a1<n+1; a1++)\n
         {\n\n
            e[a1][0] = m[a1];\n
            e[a1][1] = m[a1]+1;\n
            e[a1][2] = u[a1]-1;\n
            e[a1][3] = u[a1];\n
```

```
                    }\n
                    for(int i=1; i<n+1; i++)\n
                    {\n
                       for(int l=1; l<=i; l++)\n
                       {\n
                           if(l!=1)\n
                           {\n
                               cout<<a[l]<<"\\t";\n

                           }\n
                       }\n
                       for(int j=0; j<4; j++)\n
                       {\n
                           cout<<e[i][j];\n
                           for(int k=0; k<n-(i+1); k++)\n

                           {\n
                               cout<<a[k]<<"\\t";\n

                           }\n
                           cout<<"\\n";\n
                       }\n
                    }    \n\n
                    system("PAUSE");\n
                    return 0;    \n
           }\n


\n\n


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

```
            1           50              50\n
            2           50              50\n
            99          50              50\n
            100         50              50\n
            50          1               50\n
            50          2               50\n
            50          99              50\n
            50          100             50\n
            50          50              1\n
            50          50              2\n
            50          50              99\n
            50          50              100\n


\n\n


if the no of inputs is 3 and their ranges are\n
        1,100\n
        1,100\n
        1,100\n
        (could be varied too)
\n\n


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

Tags : 'c++ c'

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

2.2.1 Type of Machine Learning Problem

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

2.2.2 Performance metric

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

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

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

'Micro f1 score':
Calculate metrics globally by counting the total true positives, false negatives and false positives. This is a better metric when we have class imbalance.

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

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

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

3. Exploratory Data Analysis

3.1 Data Loading and Cleaning

3.1.1 Using Pandas with SQLite to Load the data

In [2]:
```
#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
```

In [4]:
```
#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_dat = pd.read_sql_query("""SELECT Title, Body, Tags, COUNT(*) a
s cnt_dup  FROM  no_dup_train GROUP BY Title, Body, Tags""", con)
    #Always remember to close the database
    con.close()
```

```
    # Let's now drop unwanted column.
    #Printing first 5 columns from our data frame

    print("Time taken to run this cell :", datetime.now() - start)
else:
    print("Please download the train.db file from drive or run the abov
e cells to genarate train.db file")
```

```
Time taken to run this cell : 0:18:46.522385
```

In [5]:
```
tag_dat.head()
```

Out[5]:

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

In [7]:
```
# number of times each question appeared in our database
```

```
tag_dat.cnt_dup.value_counts()
```

Out[7]:
```
1    4206315
Name: cnt_dup, dtype: int64
```

In [10]:
```
start = datetime.now()
tag_dat["tag_count"] = tag_dat["Tags"].apply(lambda text: len(text.spli
t(" ")))
# adding a new feature number of tags per question
print("Time taken to run this cell :", datetime.now() - start)
tag_dat.head()
```

```
Time taken to run this cell : 0:00:08.028043
```

Out[10]:

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

In [11]:
```
# distribution of number of tags per question
```

```
tag_dat.tag_count.value_counts()
```

Out[11]:
```
3    1206157
2    1111706
4     814996
1     568298
5     505158
Name: tag_count, dtype: int64
```

In [3]:
```
#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""", c
on)
    #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 abov
e cells to genarate train.db file")
```

```
Time taken to run this cell : 0:01:08.424259
```

In [4]:
```
tag_data.head()
```

Out[4]:

	Tags
1	c# silverlight data-binding
2	c# silverlight data-binding columns
3	jsp jstl

	Tags
4	java jdbc
5	facebook api facebook-php-sdk

3.2 Analysis of Tags

3.2.1 Total number of unique tags

In [5]:
```
# 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 [6]:
```
print("Number of data points :", tag_dtm.shape[0])
print("Number of unique tags :", tag_dtm.shape[1])
```

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

In [7]:
```
#'get_feature_name()' gives us the vocabulary.
tags = vectorizer.get_feature_names()
#Lets look at the tags we have.
print("Some of the tags we have :", tags[:10])
```

```
Some of the tags we have : ['.a', '.app', '.asp.net-mvc', '.aspxauth',
'.bash-profile', '.class-file', '.cs-file', '.doc', '.drv', '.ds-stor
e']
```

3.2.3 Number of times a tag appeared

In [8]:
```
# https://stackoverflow.com/questions/15115765/how-to-access-sparse-matrix-elements
#Lets now store the document term matrix in a dictionary.
freqs = tag_dtm.sum(axis=0).A1
result = dict(zip(tags, freqs))
```

In [9]:
```
#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[9]:

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

In [10]:
```
tag_df_sorted = tag_df.sort_values(['Counts'], ascending=False)
tag_counts = tag_df_sorted['Counts'].values
```

Top ten frequent tags occured

In [11]:
```
tag_df_sorted.head(10)
```

Out[11]:

	Tags	Counts
4337	c#	331505
18069	java	299414
27249	php	284103
18157	javascript	265423
1234	android	235436
18608	jquery	221533
4346	c++	143935
29101	python	134137
17643	iphone	128681
2215	asp.net	125651

Top ten rare tags occured

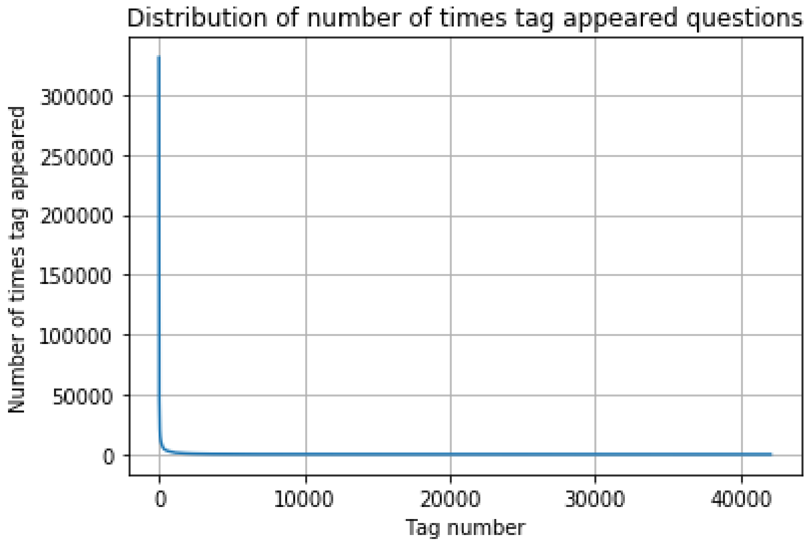
In [12]:
```
tag_df_sorted.tail(10)
```

Out[12]:

	Tags	Counts
29974	rdkit	1
11662	expired-film	1
29953	rcswitch	1
29948	rcharts	1
29942	rc.exe	1
29936	rbindlist	1

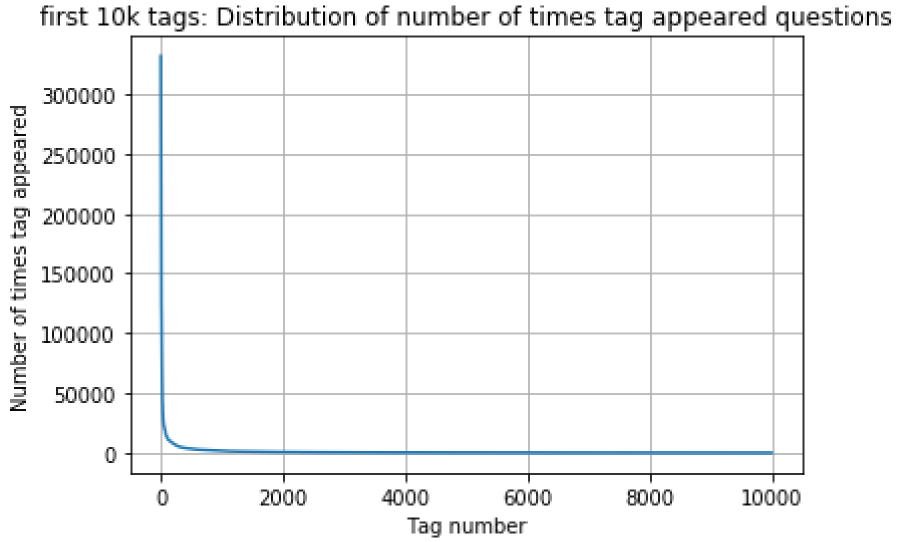
	Tags	Counts
29934	rbga	1
29930	rbar	1
2925	azureus	1
42047	zzt-oop	1

In [13]:
```
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 [14]:
```
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])
```

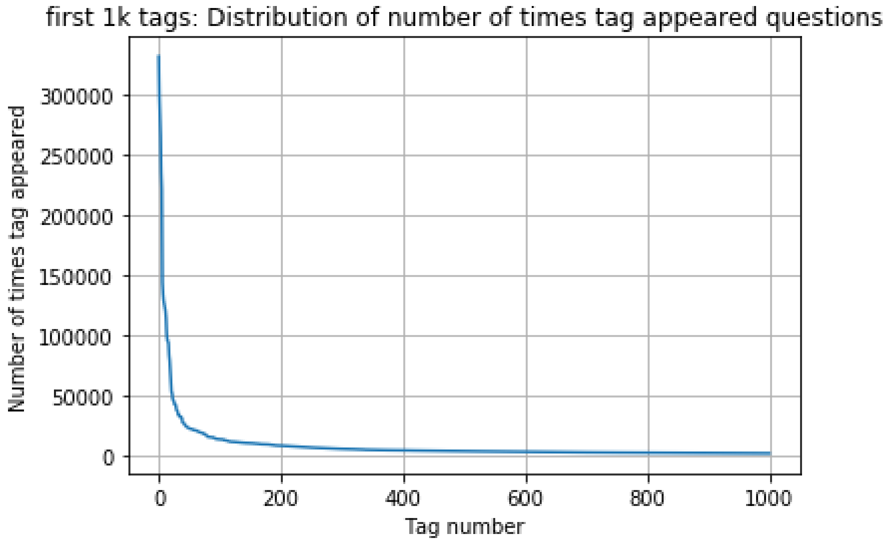



```
400 [331505  44829  22429  17728  13364  11162  10029   9148   8054   7
151
   6466   5865   5370   4983   4526   4281   4144   3929   3750   3593
   3453   3299   3123   2989   2891   2738   2647   2527   2431   2331
   2259   2186   2097   2020   1959   1900   1828   1770   1723   1673
   1631   1574   1532   1479   1448   1406   1365   1328   1300   1266
   1245   1222   1197   1181   1158   1139   1121   1101   1076   1056
   1038   1023   1006    983    966    952    938    926    911    891
    882    869    856    841    830    816    804    789    779    770
    752    743    733    725    712    702    688    678    671    658
    650    643    634    627    616    607    598    589    583    577
    568    559    552    545    540    533    526    518    512    506
    500    495    490    485    480    477    469    465    457    450
    447    442    437    432    426    422    418    413    408    403
    398    393    388    385    381    378    374    370    367    365
    361    357    354    350    347    344    342    339    336    332
    330    326    323    319    315    312    309    307    304    301
    299    296    293    291    289    286    284    281    278    276
    275    272    270    268    265    262    260    258    256    254
```

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

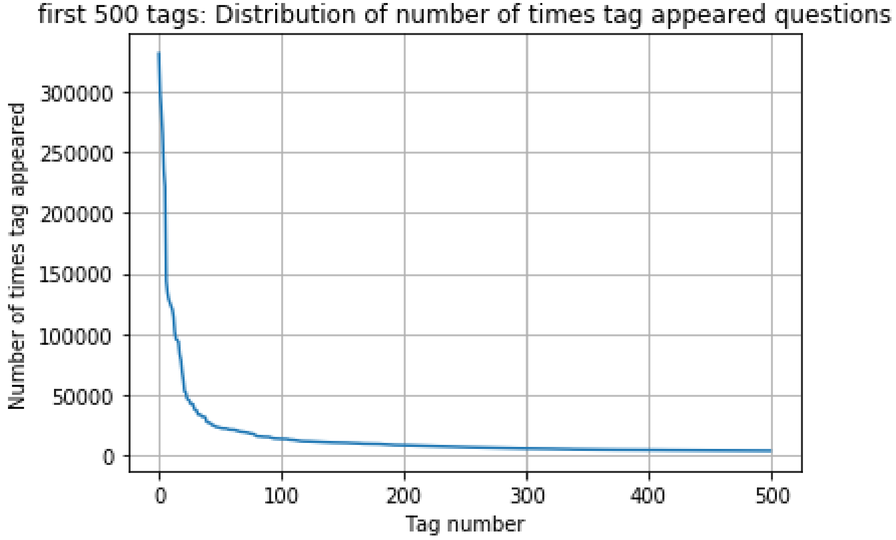
In [15]:

```
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  24
537
  22429  21820  20957  19758  18905  17728  15533  15097  14884  13703
  13364  13157  12407  11658  11228  11162  10863  10600  10350  10224
  10029   9884   9719   9411   9252   9148   9040   8617   8361   8163
   8054   7867   7702   7564   7274   7151   7052   6847   6656   6553
   6466   6291   6183   6093   5971   5865   5760   5577   5490   5411
   5370   5283   5207   5107   5066   4983   4891   4785   4658   4549
   4526   4487   4429   4335   4310   4281   4239   4228   4195   4159
   4144   4088   4050   4002   3957   3929   3874   3849   3818   3797
   3750   3703   3685   3658   3615   3593   3564   3521   3505   3483
   3453   3427   3396   3363   3326   3299   3272   3232   3196   3168
   3123   3094   3073   3050   3012   2989   2984   2953   2934   2903
   2891   2844   2819   2784   2754   2738   2726   2708   2681   2669
   2647   2621   2604   2594   2556   2527   2510   2482   2460   2444
   2431   2409   2395   2380   2363   2331   2312   2297   2290   2281
   2259   2246   2222   2211   2198   2186   2162   2142   2132   2107
   2097   2078   2057   2045   2036   2020   2011   1994   1971   1965
   1959   1952   1940   1932   1912   1900   1879   1865   1855   1841
   1828   1821   1813   1801   1782   1770   1760   1747   1741   1734
   1723   1707   1697   1688   1683   1673   1665   1656   1646   1639]
```

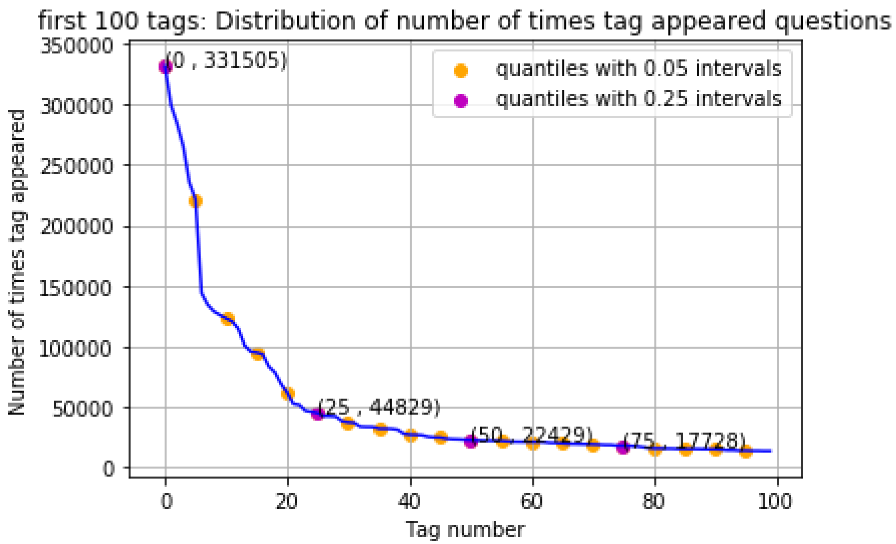
In [16]:
```
plt.plot(tag_counts[0:500])
plt.title('first 500 tags: Distribution of number of times tag appeared
 questions')
plt.grid()
plt.xlabel("Tag number")
plt.ylabel("Number of times tag appeared")
plt.show()
print(len(tag_counts[0:500:5]), tag_counts[0:500:5])
```




```
100 [331505 221533 122769  95160  62023  44829  37170  31897  26925  24
537
  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 [17]:
```
plt.plot(tag_counts[0:100], c='b')
```

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

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  245
37
  22429  21820  20957  19758  18905  17728  15533  15097  14884  13703]
```

In [18]:
```
# 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
```

Tag analysis using bi-grams

In [5]:
```
# 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(),ngram_rang
e=(1,2))
# 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 [6]:
```
print("Number of data points :", tag_dtm.shape[0])
print("Number of unique tags :", tag_dtm.shape[1])
```

```
Number of data points : 4206314
Number of unique tags : 1539526
```

In [7]:
```
#'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', '.a .lib', '.a mvs', '.a stm32f4disco
very', '.app', '.asp.net-mvc', '.aspxauth', '.aspxauth aspxcombobox',
'.bash-profile', '.bash-profile .profile']
```

In [8]:
```
# https://stackoverflow.com/questions/15115765/how-to-access-sparse-mat
rix-elements
#Lets now store the document term matrix in a dictionary.
freqs = tag_dtm.sum(axis=0).A1
result = dict(zip(tags, freqs))
```

In [9]:
```
#Saving this dictionary to csv files.
if not os.path.isfile('tag_counts_dict_.csv'):
    with open('tag_counts_dict_.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_.csv", names=['Tags', 'Counts'])
tag_df.head()
```

Out[9]:

	Tags	Counts
0	.a	18
1	.a .lib	2
2	.a mvs	1
3	.a stm32f4discovery	1
4	.app	37

In [10]:
```
tag_df_sorted = tag_df.sort_values(['Counts'], ascending=False)
tag_counts = tag_df_sorted['Counts'].values
```

Top ten most frequent tags

In [14]:
```
tag_df_sorted.head(10)
```

Out[14]:

	Tags	Counts
166872	c#	331505
681896	java	299414
996383	php	284103
691486	javascript	265423
46586	android	235436
710247	jquery	221533
174410	c++	143935
1057420	python	134137
669944	iphone	128681
89747	asp.net	125651

Top ten rare tags

In [15]:
```
tag_df_sorted.tail(10)
```

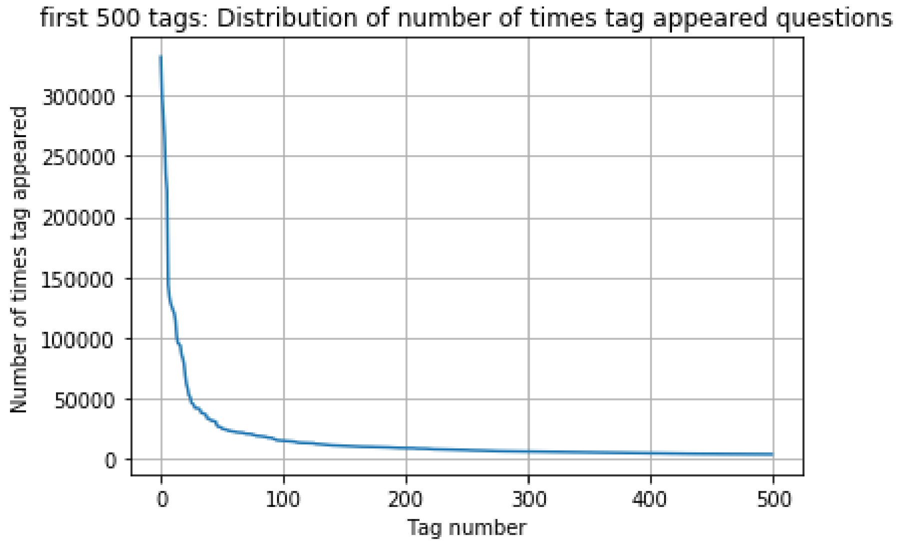
Out[15]:

	Tags	Counts
741248	language microsoft-office	1
741250	language microsoft-outlook-2003	1
741251	language midlet	1
741252	language mixed	1
741253	language mkmapview	1
741254	language mkreversegeocoder	1

	Tags	Counts
741255	language mo	1
741256	language monodroid	1
741257	language mozilla	1
1539525	zzt-oop	1

In [11]:

```
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  69168  46769  42006  37868  33373  31
069
  25375  23585  22747  21942  21006  20180  19161  18432  17640  15671
  15380  15087  14108  13532  13364  13157  12407  11897  11454  11208
```

```
 11023  10741  10525  10288  10205  10029   9981   9787   9523   9265
  9159   9079   8833   8698   8545   8197   8065   7933   7828   7684
  7528   7265   7115   7044   6856   6704   6628   6513   6447   6291
  6207   6139   6039   5971   5866   5817   5757   5669   5574   5498
  5421   5399   5360   5283   5209   5128   5083   5020   4988   4939
  4890   4785   4663   4593   4548   4526   4501   4447   4392   4335
  4314   4289   4273   4241   4233   4201   4184   4157   4142   4088]
```

In [13]:
```
# 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)))
```

```
178 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. Both in uni-gram and bi-gram the top ten tags are same.
5. 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 [20]:
```
#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.
```

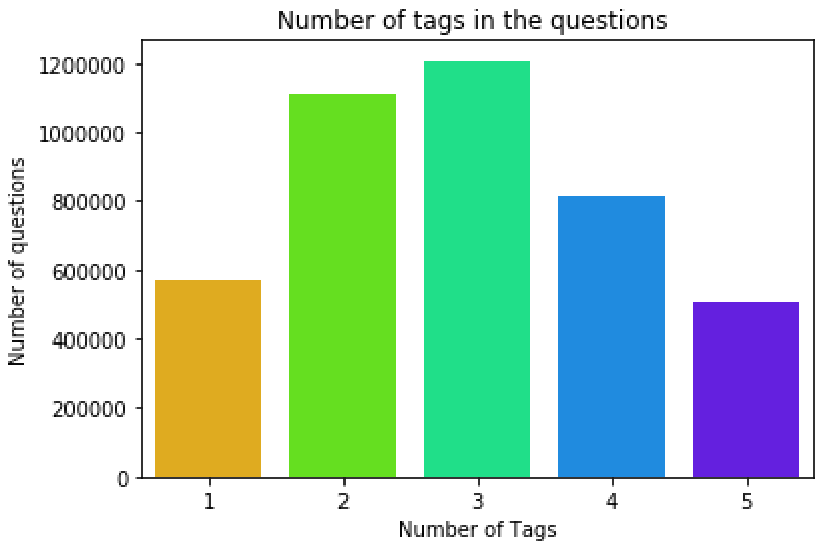
```
tag_quest_count=[int(j) for i in tag_quest_count for j in i]
print ('We have total {} datapoints.'.format(len(tag_quest_count)))

print(tag_quest_count[:5])
```

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

In [21]:
```
print( "Maximum number of tags per question: %d"%max(tag_quest_count))
print( "Minimum number of tags per question: %d"%min(tag_quest_count))
print( "Avg. number of tags per question: %f"% ((sum(tag_quest_count)*
1.0)/len(tag_quest_count)))
```

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

In [22]:
```
sns.countplot(tag_quest_count, palette='gist_rainbow')
plt.title("Number of tags in the questions ")
plt.xlabel("Number of Tags")
plt.ylabel("Number of questions")
plt.show()
```



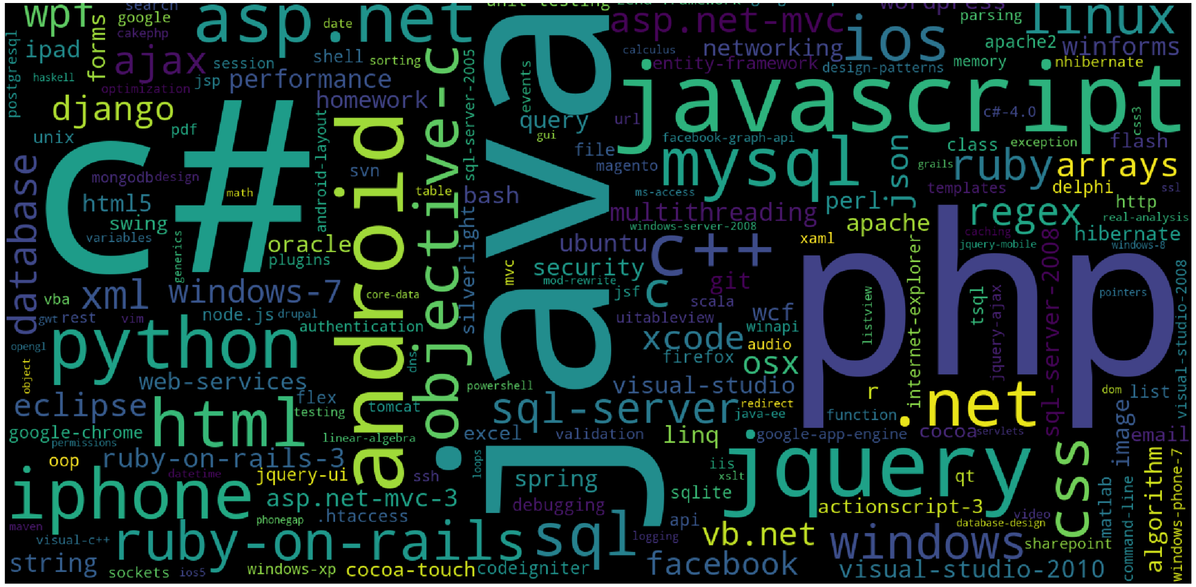
Observations:

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

3.2.5 Most Frequent Tags

In [23]:

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

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

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

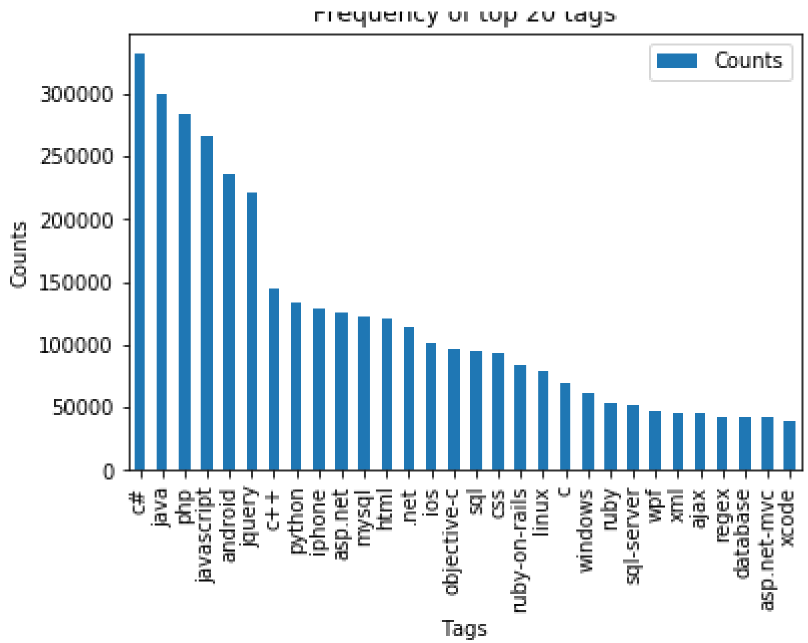



```
Time taken to run this cell : 0:00:07.360298
```

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

3.2.6 The top 20 tags

In [21]:
```python
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

1. Separate Code from Body
2. Remove Spcial characters from Question title and description (not in code)
3. **Give more weightage to title : Add title three times to the question**
4. Remove stop words (Except 'C')

5. Remove HTML Tags
6. Convert all the characters into small letters
7. Use SnowballStemmer to stem the words

In [22]:
```
def striphtml(data):
    cleanr = re.compile('<.*?>')
    cleantext = re.sub(cleanr, ' ', str(data))
    return cleantext
stop_words = set(stopwords.words('english'))
stemmer = SnowballStemmer("english")
```

In [23]:
```
#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 (qu
estion text NOT NULL, code text, tags text, words_pre integer, words_po
st integer, is_code integer);"""
create_database_table("Titlemoreweight.db", sql_create_table)
```

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

In [30]:

```
# 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 LIMI
```

```
T 500001;")
        # for selecting random points
        #reader.execute("SELECT Title, Body, Tags From no_dup_train ORD
ER BY RANDOM() LIMIT 500001;")

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

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

In [31]:

```
#http://www.bernzilla.com/2008/05/13/selecting-a-random-row-from-an-sql
ite-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.DOT
ALL))

    question=re.sub('<code>(.*?)</code>', '', question, flags=re.MULTIL
INE|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)+" "+questio
n+" "+str(tags)
#     else:
#         question=str(title)+" "+str(title)+" "+str(title)+" "+questio
n

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

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

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

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

```
print( "Avg. length of questions(Title+Body) before processing: %d"%no_
dup_avg_len_pre)
print( "Avg. length of questions(Title+Body) after processing: %d"%no_d
up_avg_len_post)
print ("Percent of questions containing code: %d"%((questions_with_code
*100.0)/questions_proccesed))

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

```
number of questions completed= 100000
number of questions completed= 200000
number of questions completed= 300000
number of questions completed= 400000
number of questions completed= 500000
Avg. length of questions(Title+Body) before processing: 1239
Avg. length of questions(Title+Body) after processing: 424
Percent of questions containing code: 57
Time taken to run this cell : 0:36:56.759771
```

In [32]:

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

Sample quesitons after preprocessing of data

In [33]:

```
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 1
0")
        print("Questions after preprocessed")
        print('='*100)
        reader.fetchone()
```

```
        for row in reader:
            print(row)
            print('-'*100)
conn_r.commit()
conn_r.close()
```

```
Questions after preprocessed
========================================================================
=============================
('dynam datagrid bind silverlight dynam datagrid bind silverlight dynam
datagrid bind silverlight bind datagrid dynam code wrote code debug cod
e block seem bind correct grid come column form come grid column althou
gh necessari bind nthank repli advance..',)
------------------------------------------------------------------------
-----------------------------
('java.lang.noclassdeffounderror javax servlet jsp tagext taglibraryval
id java.lang.noclassdeffounderror javax servlet jsp tagext taglibraryva
lid java.lang.noclassdeffounderror javax servlet jsp tagext taglibraryv
alid follow guid link instal jstl got follow error tri launch jsp page
java.lang.noclassdeffounderror javax servlet jsp tagext taglibraryvalid
taglib declar instal jstl 1.1 tomcat webapp tri project work also tri v
ersion 1.2 jstl still messag caus solv',)
------------------------------------------------------------------------
-----------------------------
('java.sql.sqlexcept microsoft odbc driver manag invalid descriptor ind
ex java.sql.sqlexcept microsoft odbc driver manag invalid descriptor in
dex java.sql.sqlexcept microsoft odbc driver manag invalid descriptor i
ndex use follow code display caus solv',)
------------------------------------------------------------------------
-----------------------------
('better way updat feed fb php sdk better way updat feed fb php sdk bet
ter way updat feed fb php sdk novic facebook api read mani tutori still
confused.i find post feed api method like correct second way use curl s
ometh like way better',)
------------------------------------------------------------------------
-----------------------------
('btnadd click event open two window record ad btnadd click event open
two window record ad btnadd click event open two window record ad open
window search.aspx use code hav add button search.aspx nwhen insert rec
ord btnadd click event open anoth window nafter insert record close win
```

```
dow',)
----------------------------------------------------------------------
-----------------------------
('sql inject issu prevent correct form submiss php sql inject issu prev
ent correct form submiss php sql inject issu prevent correct form submi
ss php check everyth think make sure input field safe type sql inject g
ood news safe bad news one tag mess form submiss place even touch life
figur exact html use templat file forgiv okay entir php script get exec
ut see data post none forum field post problem use someth titl field no
ne data get post current use print post see submit noth work flawless s
tatement though also mention script work flawless local machin use host
come across problem state list input test mess',)
----------------------------------------------------------------------
-----------------------------
('countabl subaddit lebesgu measur countabl subaddit lebesgu measur cou
ntabl subaddit lebesgu measur let lbrace rbrace sequenc set sigma -alge
bra mathcal want show left bigcup right leq sum left right countabl add
it measur defin set sigma algebra mathcal think use monoton properti so
mewher proof start appreci littl help nthank ad han answer make follow
addit construct given han answer clear bigcup bigcup cap emptyset neq l
eft bigcup right left bigcup right sum left right also construct subset
monoton left right leq left right final would sum leq sum result follo
w',)
----------------------------------------------------------------------
-----------------------------
('hql equival sql queri hql equival sql queri hql equival sql queri hql
queri replac name class properti name error occur hql error',)
----------------------------------------------------------------------
-----------------------------
('undefin symbol architectur i386 objc class skpsmtpmessag referenc err
or undefin symbol architectur i386 objc class skpsmtpmessag referenc er
ror undefin symbol architectur i386 objc class skpsmtpmessag referenc e
rror import framework send email applic background import framework i.e
skpsmtpmessag somebodi suggest get error collect2 ld return exit status
import framework correct sorc taken framework follow mfmailcomposeviewc
ontrol question lock field updat answer drag drop folder project click
copi nthat',)
----------------------------------------------------------------------
-----------------------------
```

Saving Preprocessed data to a Database

In [3]:
```
write_db = 'Titlemoreweight.db'
if os.path.isfile(write_db):
    conn_r = create_connection(write_db)
    if conn_r is not None:
        preprocessed_data = pd.read_sql_query("""SELECT question, Tags
 FROM QuestionsProcessed""", conn_r)
conn_r.commit()
conn_r.close()
```

In [4]:
```
preprocessed_data.head()
```

Out[4]:

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

In [5]:
```
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
```

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

Converting string Tags to multilable output variables

In [6]:
```
vectorizer = CountVectorizer(tokenizer = lambda x: x.split(), binary='t
rue')
multilabel_y = vectorizer.fit_transform(preprocessed_data['tags'])
```

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

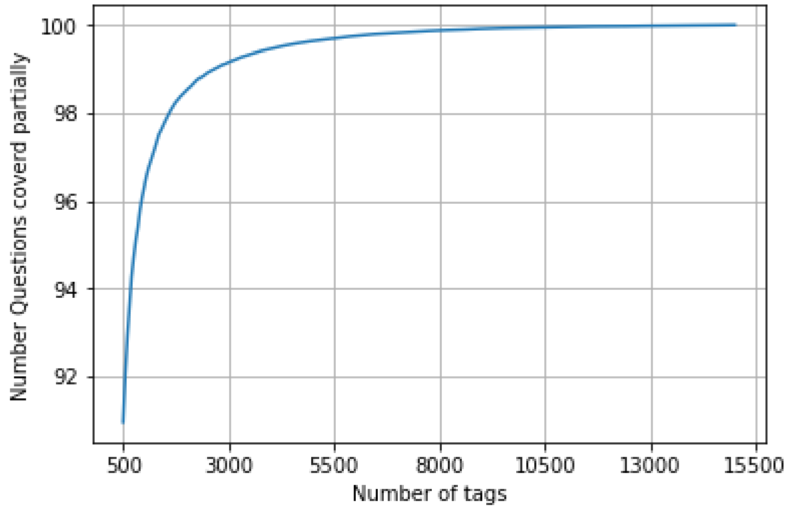
In [7]:
```
def tags_to_choose(n):
    t = multilabel_y.sum(axis=0).tolist()[0]
    sorted_tags_i = sorted(range(len(t)), key=lambda i: t[i], reverse=T
rue)
    multilabel_yn=multilabel_y[:,sorted_tags_i[:n]]
    return multilabel_yn

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

Selecting 500 Tags

In [8]:
```
questions_explained = []
total_tags=multilabel_y.shape[1]
total_qs=preprocessed_data.shape[0]
for i in range(500, total_tags, 100):
```

```
    questions_explained.append(np.round(((total_qs-questions_explained_
fn(i))/total_qs)*100,3))
```

In [9]:
```
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, mini
mun is 500(it covers 90% of the tags)
print("with ",5500,"tags we are covering ",questions_explained[50],"% o
f 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 [10]:
```
multilabel_yx = tags_to_choose(500)
print("number of questions that are not covered :", questions_explained
```

```
_fn(500),"out of ", total_qs)
```

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

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

In [11]:
```
total_size=preprocessed_data.shape[0]
train_size=int(0.80*total_size)

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

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

In [12]:
```
print(x_train.shape)
print(x_test.shape)
```

```
(400000, 2)
(100000, 2)
```

In [13]:
```
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)
```

4.3 Featurizing data

In [14]:
```
start = datetime.now()
vectorizer = CountVectorizer(max_features=70000,tokenizer = lambda x: x
.split(),ngram_range=(1,2))
x_train_multilabel = vectorizer.fit_transform(x_train['question'])
x_test_multilabel = vectorizer.transform(x_test['question'])
print("Time taken to run this cell :", datetime.now() - start)
```

```
Time taken to run this cell : 0:02:05.475311
```

In [15]:
```
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, 70000) Y : (400000, 500)
Dimensions of test data X: (100000, 70000) Y: (100000, 500)
```

4.4 Applying Logistic Regression with OneVsRest Classifier

In [16]:
```
start = datetime.now()
classifier_1 = OneVsRestClassifier(LogisticRegression(penalty='l1'))
classifier_1.fit(x_train_multilabel, y_train)
predictions_1 = classifier_1.predict(x_test_multilabel)
print("Accuracy :",metrics.accuracy_score(y_test, predictions_1))
print("Hamming loss ",metrics.hamming_loss(y_test,predictions_1))


precision = precision_score(y_test, predictions_1, average='micro')
recall = recall_score(y_test, predictions_1, average='micro')
f1 = f1_score(y_test, predictions_1, average='micro')

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

precision = precision_score(y_test, predictions_1, average='macro')
recall = recall_score(y_test, predictions_1, average='macro')
f1 = f1_score(y_test, predictions_1, average='macro')

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

```
print (metrics.classification_report(y_test, predictions_1))
print("Time taken to run this cell :", datetime.now() - start)
```

```
Accuracy : 0.21294
Hamming loss  0.00312752
Micro-average quality numbers
Precision: 0.5696, Recall: 0.4107, F1-measure: 0.4772
Macro-average quality numbers
Precision: 0.4532, Recall: 0.3364, F1-measure: 0.3825
              precision    recall  f1-score   support

           0       0.89      0.74      0.81      5519
           1       0.52      0.40      0.45      8190
           2       0.64      0.47      0.55      6529
           3       0.68      0.53      0.59      3231
           4       0.66      0.49      0.56      6430
           5       0.61      0.42      0.50      2879
           6       0.74      0.56      0.64      5086
           7       0.75      0.62      0.68      4533
           8       0.35      0.18      0.24      3000
           9       0.69      0.59      0.64      2765
          10       0.43      0.30      0.35      3051
          11       0.59      0.44      0.50      3009
          12       0.49      0.35      0.41      2630
          13       0.54      0.38      0.44      1426
          14       0.81      0.63      0.71      2548
          15       0.48      0.30      0.37      2371
          16       0.52      0.29      0.37       873
          17       0.80      0.65      0.72      2151
          18       0.44      0.30      0.35      2204
          19       0.55      0.43      0.48       831
          20       0.69      0.50      0.58      1860
          21       0.25      0.17      0.21      2023
          22       0.40      0.29      0.34      1513
          23       0.79      0.59      0.67      1207
          24       0.44      0.32      0.37       506
          25       0.52      0.37      0.43       425
          26       0.58      0.44      0.50       793
          27       0.53      0.40      0.46      1291
```

28	0.60	0.43	0.50	1208
29	0.31	0.17	0.22	406
30	0.45	0.24	0.31	504
31	0.21	0.15	0.17	732
32	0.47	0.35	0.40	441
33	0.52	0.35	0.42	1645
34	0.46	0.29	0.36	1058
35	0.72	0.57	0.64	946
36	0.47	0.29	0.36	644
37	0.90	0.71	0.79	136
38	0.51	0.36	0.42	570
39	0.63	0.34	0.44	766
40	0.52	0.43	0.47	1132
41	0.34	0.28	0.31	174
42	0.65	0.53	0.58	210
43	0.65	0.45	0.54	433
44	0.60	0.49	0.54	626
45	0.58	0.39	0.47	852
46	0.64	0.49	0.55	534
47	0.31	0.24	0.27	350
48	0.65	0.52	0.58	496
49	0.76	0.64	0.69	785
50	0.19	0.12	0.15	475
51	0.27	0.22	0.24	305
52	0.20	0.10	0.13	251
53	0.54	0.41	0.47	914
54	0.39	0.26	0.31	728
55	0.16	0.08	0.11	258
56	0.35	0.29	0.32	821
57	0.34	0.20	0.25	541
58	0.59	0.35	0.44	748
59	0.89	0.71	0.79	724
60	0.36	0.19	0.25	660
61	0.43	0.26	0.32	235
62	0.87	0.72	0.79	718
63	0.78	0.69	0.74	468
64	0.42	0.28	0.34	191
65	0.28	0.19	0.23	429
66	0.21	0.12	0.15	415
67	0.67	0.51	0.58	274

67	0.67	0.51	0.58	274
68	0.73	0.55	0.63	510
69	0.60	0.48	0.54	466
70	0.25	0.16	0.19	305
71	0.33	0.23	0.27	247
72	0.71	0.52	0.60	401
73	0.90	0.81	0.85	86
74	0.63	0.44	0.52	120
75	0.83	0.70	0.76	129
76	0.14	0.05	0.08	473
77	0.38	0.31	0.34	143
78	0.68	0.46	0.55	347
79	0.49	0.28	0.36	479
80	0.43	0.36	0.39	279
81	0.50	0.29	0.37	461
82	0.16	0.08	0.10	298
83	0.71	0.51	0.59	396
84	0.39	0.36	0.37	184
85	0.43	0.30	0.36	573
86	0.29	0.14	0.19	325
87	0.50	0.42	0.46	273
88	0.42	0.32	0.36	135
89	0.29	0.19	0.23	232
90	0.49	0.40	0.44	409
91	0.50	0.33	0.39	420
92	0.69	0.58	0.63	408
93	0.53	0.51	0.52	241
94	0.20	0.09	0.13	211
95	0.30	0.18	0.23	277
96	0.21	0.11	0.15	410
97	0.80	0.52	0.63	501
98	0.65	0.61	0.63	136
99	0.45	0.36	0.40	239
100	0.34	0.20	0.25	324
101	0.85	0.71	0.77	277
102	0.88	0.76	0.82	613
103	0.34	0.23	0.27	157
104	0.21	0.12	0.15	295
105	0.69	0.48	0.56	334
106	0.66	0.36	0.46	335

106	0.66	0.36	0.46	335
107	0.67	0.58	0.62	389
108	0.51	0.33	0.40	251
109	0.56	0.48	0.52	317
110	0.30	0.11	0.16	187
111	0.45	0.17	0.25	140
112	0.61	0.48	0.54	154
113	0.49	0.28	0.36	332
114	0.43	0.31	0.36	323
115	0.41	0.32	0.36	344
116	0.65	0.55	0.59	370
117	0.40	0.28	0.33	313
118	0.75	0.75	0.75	874
119	0.36	0.29	0.32	293
120	0.15	0.09	0.11	200
121	0.67	0.50	0.57	463
122	0.27	0.14	0.19	119
123	0.14	0.04	0.06	256
124	0.85	0.70	0.77	195
125	0.29	0.17	0.22	138
126	0.70	0.51	0.59	376
127	0.15	0.07	0.10	122
128	0.13	0.06	0.08	252
129	0.41	0.36	0.38	144
130	0.32	0.21	0.26	150
131	0.20	0.10	0.13	210
132	0.49	0.33	0.39	361
133	0.85	0.65	0.74	453
134	0.81	0.77	0.79	124
135	0.15	0.12	0.13	91
136	0.55	0.39	0.46	128
137	0.46	0.42	0.44	218
138	0.34	0.21	0.26	243
139	0.30	0.19	0.23	149
140	0.70	0.54	0.61	318
141	0.20	0.12	0.15	159
142	0.56	0.40	0.46	274
143	0.81	0.81	0.81	362
144	0.44	0.29	0.35	118
145	0.45	0.41	0.43	164

145	0.45	0.41	0.43	164
146	0.52	0.39	0.44	461
147	0.70	0.42	0.53	159
148	0.38	0.21	0.27	166
149	0.90	0.60	0.72	346
150	0.48	0.21	0.29	350
151	0.86	0.67	0.76	55
152	0.69	0.51	0.59	387
153	0.38	0.32	0.35	150
154	0.32	0.15	0.20	281
155	0.23	0.18	0.20	202
156	0.72	0.65	0.68	130
157	0.22	0.11	0.14	245
158	0.88	0.70	0.78	177
159	0.46	0.39	0.42	130
160	0.37	0.23	0.28	336
161	0.78	0.65	0.71	220
162	0.18	0.10	0.13	229
163	0.80	0.49	0.61	316
164	0.64	0.41	0.50	283
165	0.51	0.38	0.43	197
166	0.55	0.54	0.55	101
167	0.35	0.22	0.27	231
168	0.39	0.32	0.35	370
169	0.41	0.25	0.31	258
170	0.28	0.17	0.21	101
171	0.33	0.27	0.30	89
172	0.45	0.36	0.40	193
173	0.47	0.33	0.38	309
174	0.28	0.15	0.19	172
175	0.82	0.75	0.78	95
176	0.85	0.62	0.72	346
177	0.81	0.60	0.69	322
178	0.54	0.49	0.52	232
179	0.25	0.12	0.16	125
180	0.50	0.41	0.45	145
181	0.27	0.21	0.24	77
182	0.19	0.12	0.14	182
183	0.51	0.36	0.42	257

184	0.22	0.13	0.17	216
185	0.29	0.21	0.24	242
186	0.33	0.21	0.25	165
187	0.69	0.57	0.63	263
188	0.22	0.11	0.15	174
189	0.61	0.43	0.50	136
190	0.80	0.60	0.69	202
191	0.31	0.21	0.25	134
192	0.57	0.44	0.50	230
193	0.23	0.19	0.21	90
194	0.59	0.56	0.57	185
195	0.18	0.09	0.12	156
196	0.14	0.09	0.11	160
197	0.30	0.17	0.22	266
198	0.27	0.15	0.20	284
199	0.18	0.08	0.11	145
200	0.86	0.78	0.82	212
201	0.49	0.27	0.35	317
202	0.69	0.63	0.66	427
203	0.21	0.14	0.17	232
204	0.39	0.30	0.34	217
205	0.49	0.49	0.49	527
206	0.22	0.10	0.13	124
207	0.44	0.35	0.39	103
208	0.78	0.54	0.64	287
209	0.19	0.11	0.14	193
210	0.58	0.39	0.47	220
211	0.45	0.24	0.31	140
212	0.18	0.11	0.14	161
213	0.46	0.53	0.49	72
214	0.61	0.44	0.51	396
215	0.63	0.40	0.49	134
216	0.46	0.27	0.34	400
217	0.32	0.25	0.28	75
218	0.93	0.78	0.85	219
219	0.57	0.41	0.48	210
220	0.85	0.69	0.76	298
221	0.88	0.72	0.80	266
222	0.66	0.46	0.54	290
223	0.15	0.05	0.08	128

223	0.15	0.05	0.08	128
224	0.69	0.47	0.56	159
225	0.40	0.34	0.37	164
226	0.46	0.37	0.41	144
227	0.54	0.38	0.45	276
228	0.10	0.05	0.06	235
229	0.13	0.06	0.08	216
230	0.34	0.22	0.27	228
231	0.71	0.56	0.63	64
232	0.22	0.15	0.18	103
233	0.63	0.40	0.49	216
234	0.43	0.22	0.29	116
235	0.47	0.34	0.39	77
236	0.85	0.76	0.80	67
237	0.32	0.19	0.24	218
238	0.24	0.17	0.20	139
239	0.12	0.03	0.05	94
240	0.40	0.30	0.34	77
241	0.33	0.14	0.19	167
242	0.63	0.40	0.49	86
243	0.27	0.24	0.26	58
244	0.52	0.39	0.45	269
245	0.17	0.11	0.13	112
246	0.92	0.82	0.87	255
247	0.21	0.21	0.21	58
248	0.14	0.07	0.10	81
249	0.06	0.02	0.03	131
250	0.38	0.26	0.31	93
251	0.58	0.36	0.44	154
252	0.11	0.05	0.07	129
253	0.41	0.34	0.37	83
254	0.23	0.12	0.15	191
255	0.11	0.06	0.08	219
256	0.14	0.08	0.10	130
257	0.42	0.32	0.36	93
258	0.63	0.52	0.57	217
259	0.27	0.18	0.22	141
260	0.71	0.24	0.36	143
261	0.44	0.20	0.28	219
262	0.51	0.34	0.41	107

262	0.51	0.34	0.41	107
263	0.35	0.23	0.28	236
264	0.26	0.19	0.22	119
265	0.39	0.28	0.33	72
266	0.09	0.04	0.06	70
267	0.36	0.26	0.30	107
268	0.53	0.48	0.50	169
269	0.29	0.16	0.21	129
270	0.70	0.56	0.62	159
271	0.74	0.54	0.62	190
272	0.44	0.33	0.38	248
273	0.86	0.75	0.80	264
274	0.85	0.66	0.74	105
275	0.16	0.11	0.13	104
276	0.05	0.03	0.03	115
277	0.76	0.61	0.68	170
278	0.70	0.48	0.57	145
279	0.85	0.75	0.80	230
280	0.59	0.40	0.48	80
281	0.65	0.55	0.59	217
282	0.70	0.53	0.60	175
283	0.28	0.19	0.23	269
284	0.55	0.43	0.48	74
285	0.67	0.54	0.60	206
286	0.84	0.70	0.77	227
287	0.62	0.42	0.50	130
288	0.21	0.09	0.13	129
289	0.15	0.11	0.13	80
290	0.18	0.13	0.15	99
291	0.60	0.39	0.47	208
292	0.24	0.12	0.16	67
293	0.77	0.55	0.64	109
294	0.37	0.28	0.32	140
295	0.25	0.17	0.20	241
296	0.24	0.12	0.17	72
297	0.23	0.16	0.19	107
298	0.62	0.61	0.61	61
299	0.79	0.57	0.66	77
300	0.15	0.11	0.12	111
301	0.03	0.01	0.01	126

301	0.03	0.01	0.01	126
302	0.16	0.12	0.14	73
303	0.56	0.44	0.49	176
304	0.90	0.83	0.86	230
305	0.83	0.71	0.77	156
306	0.43	0.36	0.39	146
307	0.22	0.13	0.17	98
308	0.06	0.03	0.04	78
309	0.44	0.17	0.25	94
310	0.57	0.38	0.45	162
311	0.71	0.50	0.59	116
312	0.46	0.37	0.41	57
313	0.25	0.09	0.13	65
314	0.43	0.38	0.41	138
315	0.51	0.32	0.39	195
316	0.38	0.32	0.35	69
317	0.23	0.18	0.20	134
318	0.48	0.40	0.43	148
319	0.79	0.57	0.66	161
320	0.19	0.17	0.18	104
321	0.68	0.62	0.65	156
322	0.55	0.46	0.50	134
323	0.55	0.46	0.50	232
324	0.22	0.16	0.19	92
325	0.34	0.20	0.25	197
326	0.08	0.06	0.06	126
327	0.14	0.06	0.08	115
328	0.93	0.71	0.81	198
329	0.42	0.29	0.34	125
330	0.53	0.26	0.35	81
331	0.32	0.14	0.19	94
332	0.31	0.20	0.24	56
333	0.17	0.09	0.12	260
334	0.17	0.13	0.15	60
335	0.24	0.12	0.16	110
336	0.60	0.56	0.58	71
337	0.10	0.08	0.09	66
338	0.40	0.45	0.43	150
339	0.07	0.06	0.06	54
340	0.78	0.57	0.66	195

340	0.78	0.57	0.66	195
341	0.67	0.53	0.59	79
342	0.41	0.53	0.46	38
343	0.52	0.37	0.43	43
344	0.36	0.29	0.32	68
345	0.65	0.36	0.46	73
346	0.12	0.07	0.09	116
347	0.70	0.53	0.61	111
348	0.26	0.19	0.22	63
349	0.83	0.73	0.78	104
350	0.56	0.55	0.55	44
351	0.28	0.28	0.28	40
352	0.75	0.62	0.68	136
353	0.38	0.26	0.31	54
354	0.25	0.12	0.16	134
355	0.53	0.42	0.47	120
356	0.45	0.32	0.37	228
357	0.54	0.40	0.46	269
358	0.58	0.39	0.47	80
359	0.78	0.64	0.70	140
360	0.32	0.22	0.26	125
361	0.87	0.74	0.80	169
362	0.12	0.09	0.10	56
363	0.83	0.77	0.80	154
364	0.22	0.21	0.21	58
365	0.27	0.18	0.22	71
366	0.92	0.67	0.77	54
367	0.15	0.11	0.13	116
368	0.25	0.19	0.21	54
369	0.09	0.06	0.07	71
370	0.23	0.11	0.15	61
371	0.30	0.10	0.15	71
372	0.49	0.42	0.45	52
373	0.58	0.46	0.51	150
374	0.28	0.23	0.25	93
375	0.10	0.06	0.07	67
376	0.04	0.01	0.02	76
377	0.46	0.37	0.41	106
378	0.10	0.03	0.05	86

379	0.23	0.21	0.22	14
380	0.81	0.57	0.67	122
381	0.08	0.05	0.06	104
382	0.26	0.15	0.19	66
383	0.50	0.41	0.45	110
384	0.17	0.07	0.10	155
385	0.41	0.38	0.40	50
386	0.22	0.12	0.16	64
387	0.23	0.12	0.16	93
388	0.46	0.33	0.39	102
389	0.10	0.05	0.06	108
390	0.91	0.71	0.79	178
391	0.37	0.24	0.29	115
392	0.63	0.45	0.53	42
393	0.03	0.01	0.01	134
394	0.26	0.17	0.21	112
395	0.35	0.29	0.32	176
396	0.25	0.14	0.18	125
397	0.64	0.46	0.54	224
398	0.79	0.70	0.74	63
399	0.11	0.07	0.08	59
400	0.44	0.40	0.42	63
401	0.43	0.30	0.35	98
402	0.43	0.24	0.31	162
403	0.27	0.19	0.22	83
404	0.64	0.84	0.73	19
405	0.18	0.13	0.15	92
406	0.64	0.39	0.48	41
407	0.50	0.35	0.41	43
408	0.66	0.48	0.56	160
409	0.19	0.12	0.15	50
410	0.00	0.00	0.00	19
411	0.30	0.23	0.26	175
412	0.20	0.14	0.16	72
413	0.27	0.11	0.15	95
414	0.26	0.14	0.19	97
415	0.15	0.10	0.12	48
416	0.46	0.34	0.39	83
417	0.21	0.15	0.18	40
418	0.25	0.14	0.18	91

418	0.25	0.14	0.18	91
419	0.51	0.42	0.46	90
420	0.33	0.30	0.31	37
421	0.07	0.05	0.06	66
422	0.48	0.40	0.43	73
423	0.38	0.30	0.34	56
424	0.85	0.88	0.87	33
425	0.20	0.07	0.10	76
426	0.06	0.02	0.03	81
427	0.93	0.73	0.82	150
428	1.00	0.72	0.84	29
429	0.98	0.95	0.97	389
430	0.58	0.44	0.50	167
431	0.41	0.15	0.22	123
432	0.23	0.15	0.18	39
433	0.33	0.29	0.31	82
434	0.90	0.68	0.78	66
435	0.56	0.47	0.51	93
436	0.50	0.38	0.43	87
437	0.16	0.09	0.12	86
438	0.64	0.48	0.55	104
439	0.49	0.23	0.31	100
440	0.16	0.07	0.10	141
441	0.40	0.43	0.41	110
442	0.24	0.20	0.21	123
443	0.37	0.21	0.27	71
444	0.23	0.11	0.15	109
445	0.41	0.35	0.38	48
446	0.39	0.29	0.33	76
447	0.17	0.21	0.19	38
448	0.59	0.54	0.56	81
449	0.48	0.29	0.36	132
450	0.41	0.35	0.37	81
451	0.67	0.38	0.49	76
452	0.15	0.09	0.11	44
453	0.12	0.02	0.04	44
454	0.75	0.54	0.63	70
455	0.29	0.25	0.27	155
456	0.31	0.26	0.28	43
457	0.30	0.33	0.36	72

457	0.39	0.33	0.36	72
458	0.16	0.11	0.13	62
459	0.44	0.32	0.37	69
460	0.17	0.09	0.12	119
461	0.62	0.33	0.43	79
462	0.29	0.21	0.25	47
463	0.32	0.23	0.27	104
464	0.58	0.39	0.46	106
465	0.38	0.33	0.35	64
466	0.44	0.32	0.37	173
467	0.60	0.48	0.53	107
468	0.43	0.29	0.35	126
469	0.17	0.06	0.09	114
470	0.91	0.81	0.86	140
471	0.62	0.41	0.49	79
472	0.39	0.39	0.39	143
473	0.65	0.40	0.49	158
474	0.26	0.12	0.16	138
475	0.20	0.15	0.17	59
476	0.63	0.48	0.54	88
477	0.70	0.66	0.68	176
478	0.90	0.79	0.84	24
479	0.26	0.17	0.21	92
480	0.69	0.57	0.62	100
481	0.38	0.35	0.37	103
482	0.26	0.16	0.20	74
483	0.70	0.60	0.65	105
484	0.29	0.12	0.17	83
485	0.05	0.04	0.04	82
486	0.30	0.18	0.23	71
487	0.39	0.23	0.28	120
488	0.27	0.10	0.14	105
489	0.55	0.39	0.46	87
490	0.90	0.84	0.87	32
491	0.05	0.03	0.04	69
492	0.16	0.06	0.09	49
493	0.05	0.03	0.04	117
494	0.47	0.41	0.44	61
495	0.94	0.80	0.86	344

```
         496       0.25      0.17      0.20        52
         497       0.49      0.36      0.42       137
         498       0.32      0.16      0.22        98
         499       0.30      0.20      0.24        79

   micro avg       0.57      0.41      0.48    173812
   macro avg       0.45      0.34      0.38    173812
weighted avg       0.55      0.41      0.47    173812
 samples avg       0.44      0.39      0.38    173812

Time taken to run this cell : 8:42:02.612043
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classificati
on.py:1143: UndefinedMetricWarning: Precision and F-score are ill-defin
ed and being set to 0.0 in samples with no predicted labels.
  'precision', 'predicted', average, warn_for)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classificati
on.py:1145: UndefinedMetricWarning: Recall and F-score are ill-defined
and being set to 0.0 in samples with no true labels.
  'recall', 'true', average, warn_for)
```

4.5 Applying Linear SVM with OneVsRest Classifier

In [17]:

```
from sklearn.model_selection import GridSearchCV

model= OneVsRestClassifier(SGDClassifier(loss='hinge'))
parameters = [{'estimator__alpha':[0.0001,0.001,.01,1,10,100]}]
clf = GridSearchCV(model, parameters, cv=3, scoring='accuracy')
clf.fit(x_train_multilabel, y_train)
```

Out[17]:

```
GridSearchCV(cv=3, error_score='raise-deprecating',
       estimator=OneVsRestClassifier(estimator=SGDClassifier(alpha=0.00
01, average=False, class_weight=None,
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=N
one,
       n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',
       power_t=0.5, random_state=None, shuffle=True, tol=None,
```

```
        validation_fraction=0.1, verbose=0, warm_start=False),
           n_jobs=None),
        fit_params=None, iid='warn', n_jobs=None,
        param_grid=[{'estimator__alpha': [0.0001, 0.001, 0.01, 1, 10, 10
0]}],
        pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
        scoring='accuracy', verbose=0)
```

In [19]:
```
optimal_alpha = 0.0001
print("The optimal value of alpha",optimal_alpha)
```

```
The optimal value of alpha 0.0001
```

In [23]:
```
start = datetime.now()
classifier = OneVsRestClassifier(SGDClassifier(loss='hinge', alpha=.000
1, penalty='l1'))
classifier.fit(x_train_multilabel, y_train)
predictions = classifier.predict (x_test_multilabel)


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


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

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

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

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

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

```
Accuracy : 0.1177
Hamming loss  0.0045292
Micro-average quality numbers
Precision: 0.3739, Recall: 0.4490, F1-measure: 0.4080
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classificati
on.py:1143: UndefinedMetricWarning: Precision is ill-defined and being
set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classificati
on.py:1143: UndefinedMetricWarning: F-score is ill-defined and being se
t to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classificati
on.py:1143: UndefinedMetricWarning: Precision and F-score are ill-defin
ed and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
```

```
Macro-average quality numbers
Precision: 0.2921, Recall: 0.3684, F1-measure: 0.3170
              precision    recall  f1-score   support

           0       0.70      0.77      0.74      5519
           1       0.45      0.40      0.42      8190
           2       0.54      0.49      0.52      6529
           3       0.45      0.60      0.52      3231
           4       0.56      0.52      0.54      6430
           5       0.49      0.50      0.50      2879
           6       0.57      0.62      0.59      5086
           7       0.60      0.65      0.63      4533
           8       0.23      0.22      0.23      3000
           9       0.54      0.65      0.59      2765
          10       0.33      0.32      0.32      3051
          11       0.47      0.50      0.48      3009
          12       0.39      0.42      0.40      2630
          13       0.35      0.43      0.38      1426
          14       0.70      0.66      0.68      2548
```

15	0.35	0.36	0.36	2371
16	0.31	0.40	0.35	873
17	0.62	0.71	0.66	2151
18	0.29	0.39	0.33	2204
19	0.28	0.50	0.36	831
20	0.58	0.57	0.57	1860
21	0.17	0.21	0.19	2023
22	0.29	0.35	0.32	1513
23	0.52	0.68	0.59	1207
24	0.27	0.42	0.33	506
25	0.22	0.48	0.30	425
26	0.32	0.48	0.38	793
27	0.41	0.46	0.43	1291
28	0.50	0.44	0.47	1208
29	0.11	0.27	0.16	406
30	0.25	0.35	0.30	504
31	0.18	0.19	0.18	732
32	0.26	0.44	0.33	441
33	0.34	0.36	0.35	1645
34	0.39	0.31	0.34	1058
35	0.54	0.62	0.58	946
36	0.37	0.39	0.38	644
37	0.40	0.77	0.53	136
38	0.29	0.49	0.37	570
39	0.40	0.44	0.42	766
40	0.38	0.45	0.41	1132
41	0.08	0.31	0.13	174
42	0.42	0.66	0.52	210
43	0.43	0.52	0.47	433
44	0.35	0.52	0.42	626
45	0.43	0.42	0.43	852
46	0.49	0.53	0.51	534
47	0.18	0.28	0.22	350
48	0.46	0.57	0.51	496
49	0.58	0.70	0.64	785
50	0.12	0.21	0.16	475
51	0.15	0.23	0.18	305
52	0.12	0.13	0.12	251
53	0.42	0.50	0.46	914

54	0.30	0.24	0.26	728
55	0.11	0.10	0.11	258
56	0.20	0.35	0.26	821
57	0.24	0.19	0.21	541
58	0.39	0.42	0.40	748
59	0.68	0.74	0.71	724
60	0.20	0.24	0.22	660
61	0.23	0.31	0.26	235
62	0.72	0.81	0.76	718
63	0.63	0.72	0.67	468
64	0.19	0.44	0.27	191
65	0.15	0.14	0.14	429
66	0.12	0.16	0.14	415
67	0.40	0.58	0.47	274
68	0.66	0.63	0.65	510
69	0.44	0.56	0.49	466
70	0.14	0.15	0.15	305
71	0.16	0.29	0.21	247
72	0.62	0.53	0.57	401
73	0.26	0.83	0.39	86
74	0.17	0.49	0.26	120
75	0.52	0.80	0.63	129
76	0.09	0.06	0.08	473
77	0.08	0.42	0.14	143
78	0.48	0.60	0.54	347
79	0.40	0.34	0.37	479
80	0.20	0.47	0.28	279
81	0.42	0.28	0.33	461
82	0.07	0.13	0.09	298
83	0.52	0.60	0.56	396
84	0.29	0.38	0.33	184
85	0.19	0.35	0.24	573
86	0.12	0.20	0.15	325
87	0.28	0.39	0.32	273
88	0.22	0.39	0.28	135
89	0.14	0.21	0.17	232
90	0.35	0.48	0.40	409
91	0.26	0.41	0.32	420
92	0.49	0.59	0.53	408

93	0.26	0.54	0.35	241
94	0.10	0.15	0.12	211
95	0.13	0.18	0.15	277
96	0.15	0.08	0.11	410
97	0.74	0.53	0.62	501
98	0.32	0.71	0.44	136
99	0.29	0.39	0.33	239
100	0.18	0.23	0.20	324
101	0.58	0.69	0.63	277
102	0.77	0.80	0.79	613
103	0.30	0.32	0.31	157
104	0.12	0.19	0.15	295
105	0.32	0.53	0.40	334
106	0.28	0.25	0.27	335
107	0.46	0.59	0.52	389
108	0.40	0.42	0.41	251
109	0.36	0.50	0.42	317
110	0.13	0.17	0.15	187
111	0.11	0.26	0.16	140
112	0.18	0.49	0.27	154
113	0.35	0.37	0.36	332
114	0.36	0.14	0.20	323
115	0.26	0.31	0.28	344
116	0.56	0.55	0.56	370
117	0.37	0.35	0.36	313
118	0.67	0.79	0.73	874
119	0.18	0.35	0.24	293
120	0.02	0.01	0.01	200
121	0.52	0.59	0.55	463
122	0.13	0.28	0.18	119
123	0.02	0.02	0.02	256
124	0.67	0.78	0.72	195
125	0.20	0.25	0.22	138
126	0.47	0.57	0.51	376
127	0.04	0.08	0.06	122
128	0.09	0.13	0.10	252
129	0.13	0.21	0.16	144
130	0.14	0.22	0.17	150
131	0.10	0.05	0.07	210

132	0.40	0.38	0.39	361
133	0.80	0.65	0.72	453
134	0.58	0.81	0.67	124
135	0.09	0.15	0.11	91
136	0.09	0.42	0.15	128
137	0.25	0.48	0.33	218
138	0.46	0.32	0.38	243
139	0.14	0.28	0.18	149
140	0.44	0.58	0.50	318
141	0.09	0.17	0.12	159
142	0.43	0.51	0.46	274
143	0.61	0.84	0.71	362
144	0.18	0.39	0.25	118
145	0.27	0.50	0.35	164
146	0.37	0.40	0.38	461
147	0.42	0.52	0.47	159
148	0.16	0.27	0.20	166
149	0.83	0.62	0.71	346
150	0.16	0.16	0.16	350
151	0.27	0.64	0.38	55
152	0.55	0.61	0.58	387
153	0.16	0.26	0.20	150
154	0.23	0.19	0.21	281
155	0.09	0.23	0.13	202
156	0.48	0.72	0.58	130
157	0.13	0.17	0.15	245
158	0.55	0.66	0.60	177
159	0.15	0.39	0.22	130
160	0.25	0.29	0.27	336
161	0.63	0.70	0.66	220
162	0.13	0.22	0.16	229
163	0.45	0.54	0.49	316
164	0.54	0.43	0.48	283
165	0.26	0.36	0.30	197
166	0.22	0.52	0.31	101
167	0.20	0.27	0.23	231
168	0.32	0.34	0.33	370
169	0.20	0.31	0.24	258
170	0.09	0.13	0.11	101

171	0.19	0.34	0.24	89
172	0.29	0.47	0.36	193
173	0.36	0.42	0.39	309
174	0.30	0.16	0.21	172
175	0.47	0.83	0.60	95
176	0.72	0.71	0.72	346
177	0.72	0.57	0.63	322
178	0.37	0.60	0.46	232
179	0.14	0.18	0.15	125
180	0.22	0.37	0.27	145
181	0.05	0.23	0.09	77
182	0.05	0.11	0.07	182
183	0.38	0.49	0.43	257
184	0.17	0.03	0.05	216
185	0.18	0.26	0.21	242
186	0.17	0.27	0.21	165
187	0.52	0.71	0.60	263
188	0.16	0.25	0.20	174
189	0.34	0.40	0.37	136
190	0.81	0.63	0.71	202
191	0.17	0.22	0.19	134
192	0.46	0.55	0.50	230
193	0.15	0.24	0.18	90
194	0.37	0.65	0.47	185
195	0.06	0.12	0.07	156
196	0.07	0.21	0.11	160
197	0.12	0.17	0.14	266
198	0.23	0.21	0.22	284
199	0.17	0.09	0.12	145
200	0.66	0.80	0.72	212
201	0.29	0.31	0.30	317
202	0.53	0.60	0.57	427
203	0.16	0.21	0.18	232
204	0.28	0.33	0.30	217
205	0.42	0.54	0.47	527
206	0.04	0.10	0.06	124
207	0.19	0.25	0.21	103
208	0.62	0.60	0.61	287
209	0.12	0.16	0.13	193

210	0.31	0.44	0.36	220
211	0.21	0.29	0.25	140
212	0.07	0.18	0.10	161
213	0.21	0.49	0.29	72
214	0.53	0.62	0.57	396
215	0.27	0.43	0.33	134
216	0.13	0.16	0.14	400
217	0.24	0.47	0.32	75
218	0.81	0.74	0.78	219
219	0.48	0.50	0.49	210
220	0.67	0.67	0.67	298
221	0.77	0.68	0.73	266
222	0.52	0.60	0.56	290
223	0.07	0.11	0.09	128
224	0.26	0.53	0.35	159
225	0.19	0.42	0.26	164
226	0.26	0.49	0.34	144
227	0.40	0.53	0.46	276
228	0.06	0.08	0.06	235
229	0.17	0.09	0.12	216
230	0.13	0.33	0.19	228
231	0.41	0.69	0.51	64
232	0.05	0.16	0.08	103
233	0.36	0.45	0.40	216
234	0.19	0.30	0.23	116
235	0.27	0.55	0.36	77
236	0.48	0.70	0.57	67
237	0.18	0.20	0.19	218
238	0.10	0.20	0.13	139
239	0.08	0.03	0.05	94
240	0.25	0.43	0.32	77
241	0.18	0.11	0.13	167
242	0.37	0.41	0.39	86
243	0.07	0.31	0.12	58
244	0.28	0.40	0.33	269
245	0.12	0.19	0.15	112
246	0.75	0.78	0.77	255
247	0.15	0.33	0.21	58
248	0.04	0.07	0.05	81

249	0.03	0.06	0.04	131
250	0.14	0.29	0.19	93
251	0.40	0.45	0.42	154
252	0.04	0.05	0.04	129
253	0.30	0.37	0.33	83
254	0.13	0.19	0.16	191
255	0.00	0.00	0.00	219
256	0.04	0.09	0.06	130
257	0.20	0.44	0.28	93
258	0.52	0.52	0.52	217
259	0.10	0.23	0.14	141
260	0.16	0.31	0.21	143
261	0.23	0.27	0.25	219
262	0.36	0.37	0.37	107
263	0.27	0.33	0.30	236
264	0.18	0.22	0.20	119
265	0.16	0.39	0.22	72
266	0.08	0.19	0.11	70
267	0.15	0.24	0.18	107
268	0.45	0.60	0.51	169
269	0.14	0.21	0.17	129
270	0.53	0.57	0.55	159
271	0.40	0.49	0.44	190
272	0.23	0.25	0.24	248
273	0.76	0.80	0.78	264
274	0.66	0.76	0.70	105
275	0.08	0.20	0.11	104
276	0.05	0.06	0.06	115
277	0.48	0.65	0.55	170
278	0.35	0.46	0.40	145
279	0.77	0.73	0.75	230
280	0.28	0.39	0.33	80
281	0.58	0.67	0.62	217
282	0.54	0.61	0.57	175
283	0.22	0.22	0.22	269
284	0.32	0.55	0.40	74
285	0.62	0.55	0.58	206
286	0.84	0.72	0.77	227
287	0.19	0.45	0.27	130

288	0.10	0.11	0.10	129
289	0.03	0.21	0.05	80
290	0.13	0.18	0.15	99
291	0.48	0.49	0.48	208
292	0.07	0.13	0.10	67
293	0.51	0.56	0.54	109
294	0.15	0.32	0.20	140
295	0.13	0.30	0.18	241
296	0.07	0.17	0.10	72
297	0.08	0.17	0.11	107
298	0.32	0.52	0.40	61
299	0.39	0.56	0.46	77
300	0.09	0.14	0.11	111
301	0.00	0.00	0.00	126
302	0.10	0.10	0.10	73
303	0.40	0.56	0.46	176
304	0.62	0.77	0.69	230
305	0.88	0.66	0.75	156
306	0.29	0.47	0.36	146
307	0.18	0.23	0.20	98
308	0.02	0.06	0.03	78
309	0.20	0.20	0.20	94
310	0.39	0.49	0.44	162
311	0.62	0.64	0.63	116
312	0.28	0.47	0.35	57
313	0.05	0.14	0.07	65
314	0.32	0.38	0.35	138
315	0.27	0.21	0.24	195
316	0.26	0.35	0.30	69
317	0.17	0.22	0.19	134
318	0.27	0.38	0.32	148
319	0.56	0.50	0.52	161
320	0.12	0.28	0.17	104
321	0.45	0.56	0.50	156
322	0.40	0.46	0.43	134
323	0.42	0.54	0.47	232
324	0.16	0.17	0.17	92
325	0.18	0.36	0.24	197
326	0.13	0.05	0.07	126

327	0.17	0.05	0.08	115
328	0.81	0.74	0.77	198
329	0.25	0.43	0.32	125
330	0.36	0.33	0.35	81
331	0.07	0.10	0.08	94
332	0.05	0.12	0.07	56
333	0.11	0.11	0.11	260
334	0.12	0.18	0.15	60
335	0.22	0.30	0.25	110
336	0.41	0.58	0.48	71
337	0.06	0.17	0.09	66
338	0.25	0.37	0.30	150
339	0.00	0.00	0.00	54
340	0.67	0.75	0.71	195
341	0.26	0.49	0.34	79
342	0.12	0.37	0.18	38
343	0.19	0.44	0.27	43
344	0.23	0.31	0.26	68
345	0.29	0.48	0.36	73
346	0.00	0.00	0.00	116
347	0.61	0.51	0.56	111
348	0.03	0.13	0.05	63
349	0.42	0.78	0.55	104
350	0.19	0.55	0.28	44
351	0.24	0.30	0.26	40
352	0.31	0.45	0.37	136
353	0.30	0.33	0.31	54
354	0.17	0.10	0.13	134
355	0.31	0.33	0.32	120
356	0.32	0.39	0.35	228
357	0.40	0.37	0.38	269
358	0.25	0.49	0.33	80
359	0.36	0.54	0.43	140
360	0.13	0.22	0.16	125
361	0.79	0.66	0.72	169
362	0.10	0.18	0.12	56
363	0.80	0.81	0.80	154
364	0.24	0.17	0.20	58
365	0.15	0.27	0.19	71

366	0.51	0.78	0.61	54
367	0.18	0.18	0.18	116
368	0.02	0.06	0.03	54
369	0.02	0.08	0.03	71
370	0.03	0.07	0.04	61
371	0.05	0.17	0.07	71
372	0.44	0.58	0.50	52
373	0.29	0.49	0.37	150
374	0.17	0.32	0.23	93
375	0.08	0.10	0.09	67
376	0.05	0.07	0.05	76
377	0.21	0.36	0.27	106
378	0.07	0.05	0.06	86
379	0.06	0.21	0.09	14
380	0.11	0.16	0.13	122
381	0.00	0.00	0.00	104
382	0.15	0.24	0.18	66
383	0.39	0.44	0.41	110
384	0.00	0.00	0.00	155
385	0.11	0.22	0.15	50
386	0.10	0.20	0.13	64
387	0.03	0.04	0.04	93
388	0.27	0.36	0.31	102
389	0.05	0.06	0.05	108
390	0.71	0.67	0.69	178
391	0.26	0.23	0.24	115
392	0.29	0.55	0.38	42
393	0.00	0.00	0.00	134
394	0.07	0.04	0.05	112
395	0.28	0.38	0.32	176
396	0.12	0.16	0.14	125
397	0.52	0.49	0.50	224
398	0.36	0.62	0.46	63
399	0.02	0.07	0.03	59
400	0.28	0.41	0.33	63
401	0.07	0.21	0.11	98
402	0.22	0.20	0.21	162
403	0.19	0.29	0.23	83
404	0.62	0.84	0.71	19

405	0.11	0.20	0.14	92
406	0.26	0.54	0.35	41
407	0.21	0.33	0.26	43
408	0.30	0.39	0.34	160
409	0.19	0.30	0.23	50
410	0.01	0.05	0.02	19
411	0.17	0.17	0.17	175
412	0.10	0.08	0.09	72
413	0.07	0.13	0.09	95
414	0.10	0.16	0.13	97
415	0.15	0.27	0.19	48
416	0.33	0.42	0.37	83
417	0.17	0.20	0.18	40
418	0.07	0.10	0.08	91
419	0.42	0.47	0.44	90
420	0.19	0.35	0.25	37
421	0.04	0.11	0.06	66
422	0.47	0.48	0.47	73
423	0.23	0.34	0.28	56
424	0.29	0.85	0.43	33
425	0.09	0.07	0.08	76
426	0.05	0.09	0.06	81
427	0.54	0.75	0.63	150
428	0.64	0.79	0.71	29
429	0.99	0.65	0.78	389
430	0.46	0.50	0.48	167
431	0.04	0.07	0.05	123
432	0.19	0.38	0.25	39
433	0.28	0.29	0.28	82
434	0.72	0.71	0.72	66
435	0.38	0.48	0.43	93
436	0.52	0.38	0.44	87
437	0.12	0.20	0.15	86
438	0.54	0.68	0.60	104
439	0.19	0.22	0.21	100
440	0.10	0.13	0.11	141
441	0.26	0.42	0.32	110
442	0.20	0.18	0.19	123
443	0.30	0.25	0.27	71

444	0.25	0.19	0.22	109
445	0.15	0.23	0.18	48
446	0.34	0.46	0.39	76
447	0.17	0.32	0.22	38
448	0.58	0.59	0.59	81
449	0.37	0.35	0.36	132
450	0.28	0.37	0.32	81
451	0.45	0.50	0.47	76
452	0.06	0.09	0.07	44
453	0.04	0.05	0.04	44
454	0.39	0.60	0.47	70
455	0.14	0.26	0.18	155
456	0.17	0.33	0.22	43
457	0.20	0.28	0.24	72
458	0.07	0.19	0.11	62
459	0.07	0.16	0.10	69
460	0.00	0.00	0.00	119
461	0.48	0.41	0.44	79
462	0.16	0.21	0.18	47
463	0.10	0.16	0.13	104
464	0.34	0.45	0.39	106
465	0.16	0.23	0.19	64
466	0.43	0.42	0.42	173
467	0.40	0.45	0.42	107
468	0.12	0.27	0.17	126
469	0.05	0.05	0.05	114
470	0.82	0.81	0.82	140
471	0.26	0.33	0.29	79
472	0.26	0.34	0.29	143
473	0.37	0.47	0.41	158
474	0.21	0.11	0.14	138
475	0.05	0.10	0.07	59
476	0.50	0.51	0.51	88
477	0.69	0.77	0.73	176
478	0.64	0.75	0.69	24
479	0.09	0.13	0.11	92
480	0.61	0.61	0.61	100
481	0.20	0.40	0.27	103
482	0.10	0.28	0.14	74

483	0.68	0.65	0.66	105
484	0.08	0.10	0.09	83
485	0.00	0.00	0.00	82
486	0.13	0.18	0.15	71
487	0.27	0.31	0.29	120
488	0.16	0.11	0.13	105
489	0.48	0.49	0.49	87
490	0.48	0.81	0.60	32
491	0.02	0.07	0.03	69
492	0.14	0.06	0.09	49
493	0.08	0.06	0.07	117
494	0.37	0.25	0.29	61
495	0.85	0.42	0.56	344
496	0.10	0.15	0.12	52
497	0.31	0.35	0.33	137
498	0.11	0.17	0.14	98
499	0.24	0.30	0.27	79
micro avg	0.37	0.45	0.41	173812
macro avg	0.29	0.37	0.32	173812
weighted avg	0.40	0.45	0.42	173812
samples avg	0.38	0.42	0.36	173812

Time taken to run this cell : 0:22:03.420731

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classificati
on.py:1143: UndefinedMetricWarning: Precision and F-score are ill-defin
ed and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classificati
on.py:1143: UndefinedMetricWarning: Precision and F-score are ill-defin
ed and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classificati
on.py:1143: UndefinedMetricWarning: Precision and F-score are ill-defin
ed and being set to 0.0 in samples with no predicted labels.
  'precision', 'predicted', average, warn_for)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classificati
on.py:1145: UndefinedMetricWarning: Recall and F-score are ill-defined
and being set to 0.0 in samples with no true labels.
```

```
'recall', 'true', average, warn_for)
```

Conclusion:

In [6]:
```
models = pd.DataFrame({'Model': ["Logisticregression","SGDClassifier"],
 'Precision' : [.57,0.37],'Recall': [.41,.45],'F1-score':[.48,.41],}, c
olumns = ["Model","Precision", "Recall","F1-score"])
models
```

Out[6]:

	Model	Precision	Recall	F1-score
0	Logisticregression	0.57	0.41	0.48
1	SGDClassifier	0.37	0.45	0.41

1.Exploratory data analysis was is done for the dataset.
2.Cleaning and processing of data is done(stopwords removal,special characters removal,stemming etc).
3.Processed data(tags) giving more weightage to title add title three times to the question.
4.Plotted number of tags with covering percentage of questions which tells 500 tags are covering 90% of questions.
5.Spliting the data into train and test which is 80:20 ratio.
6.We used bow of words countvectorizer bi-grams for vectorizing.
7.Performed onevsrestclassifier with logisiticRegression which take lot of time to train the model.
8.Performed hyperparameter tunning using Gridsearch to find correct alpha for SGDclassifier.
9.Performed onevsrestclassifier with SGDclassifier with Hinge loss which is liner SVM.
10.We can see in the performance table that Logistic Regression works better than Linear SVM.