```
In [2]: import sys
        print(sys.executable)

        C:\Users\Himanshu Pc\Anaconda3\python.exe
```

```
In [3]: C:\Users\Himanshu Pc\Anaconda3\python

          File "<ipython-input-3-f6de5d0c6db6>", line 1
            C:\Users\Himanshu Pc\Anaconda3\python
                                                ^
        SyntaxError: unexpected character after line continuation character
```

```
In [ ]: !pip install scikit-multilearn
```

In [ ]:

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

Stack Overflow: Tag Prediction

1. Business Problem

1.1 Description

Description

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

Stack Overflow is something which every programmer use one way or another. Each month, over 50 million developers come to Stack Overflow to learn, share their knowledge, and build their careers. It features questions and answers on a wide range of topics in computer programming. The website serves as a platform for users to ask and answer questions, and, through membership and active participation, to vote questions and answers up or down and edit questions and answers in a fashion similar to a wiki or Digg. As of April 2014 Stack

Overflow has over 4,000,000 registered users, and it exceeded 10,000,000 questions in late August 2015. Based on the type of tags assigned to questions, the top eight most discussed topics on the site are: Java, JavaScript, C#, PHP, Android, jQuery, Python and HTML.

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

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

1.2 Source / useful links

Data Source : https://www.kaggle.com/c/facebook-recruiting-iii-keyword-extraction/data (https://www.kaggle.com/c/facebook-recruiting-iii-keyword-extraction/data)
Youtube : https://youtu.be/nNDqbUhtIRg (https://youtu.be/nNDqbUhtIRg)
Research paper : https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/tagging-1.pdf (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 (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 (https://www.kaggle.com/c/facebook-recruiting-iii-keyword-extraction/data)
All of the data is in 2 files: Train and Test.

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

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

Size of Train.csv - 6.75GB

Size of Test.csv - 2GB

Number of rows in Train.csv = 6034195

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

Data Field Explaination

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

Id - Unique identifier for each question

Title - The question's title

Body - The body of the question

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

2.1.2 Example Data point

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

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

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


\n\n

        <p>The answer should come in the form of a table like</p>\n\n
        <pre><code>
        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
        </code></pre>\n\n
        <p>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)</p>\n\n
        <p>The output is not coming,can anyone correct the code or tell me what\'s wrong?</p>\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 (https://www.kaggle.com/wiki/MeanFScore)
http://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html (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 (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 [4]:

```
#Creating db file from csv
#Learn SQL: https://www.w3schools.com/sql/default.asp
if not os.path.isfile('train.db'):
    start = datetime.now()
    disk_engine = create_engine('sqlite:///train.db')
    start = dt.datetime.now()
    chunksize = 180000
    j = 0
    index_start = 1
    for df in pd.read_csv('Train.csv', names=['Id', 'Title', 'Body', 'Tags'], chunksize=chunksize, iterator=True, encoding='utf-8', ):
        df.index += index_start
        j+=1
        print('{} rows'.format(j*chunksize))
        df.to_sql('data', disk_engine, if_exists='append')
        index_start = df.index[-1] + 1
    print("Time taken to run this cell :", datetime.now() - start)
```

3.1.2 Counting the number of rows

In [5]:

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

```
Number of rows in the database :
 6034196
Time taken to count the number of rows : 0:00:03.269985
```

3.1.3 Checking for duplicates

In [6]:

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

```
Time taken to run this cell : 0:04:31.775120
```

In [7]:

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

Out[7]:

	Title	Body	Tags	cnt_dup
0	Implementing Boundary Value Analysis of S...	<pre><code>#include&lt;iostream&gt;\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...	<p>I use the following code</p>\n\n<pre><code>...	java jdbc	2

In [8]:

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

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

In [9]:

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

Out[9]:

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

In [10]:

```
df_no_dup["Tags"] = df_no_dup["Tags"].dropna()
```

In [11]:

```
df_no_dup = df_no_dup[df_no_dup['Tags'].notnull()]
```

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 : 0:00:02.399735
```

Out[12]:

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

```
Time taken to run this cell : 0:00:22.249194
```

3.2 Analysis of Tags

3.2.1 Total number of unique tags

In [16]:

```
# 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 [17]:

```
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 [18]:

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

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

3.2.3 Number of times a tag appeared

In [19]:

```
# 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 [20]:

```
#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[20]:

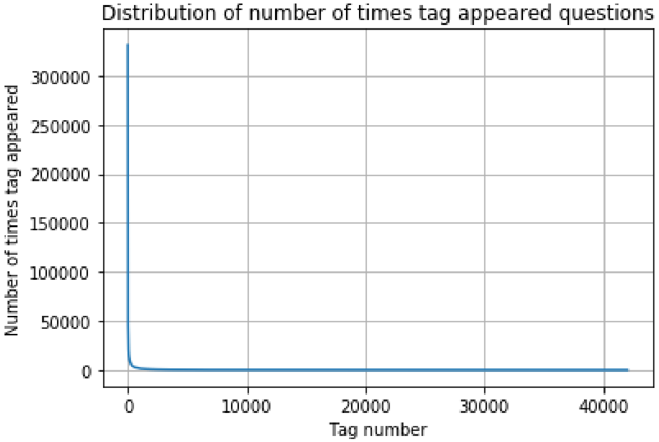
	Tags	Counts
0	.a	18
1	.app	37
2	.asp.net-mvc	1
3	.aspxauth	21
4	.bash-profile	138

In [21]:

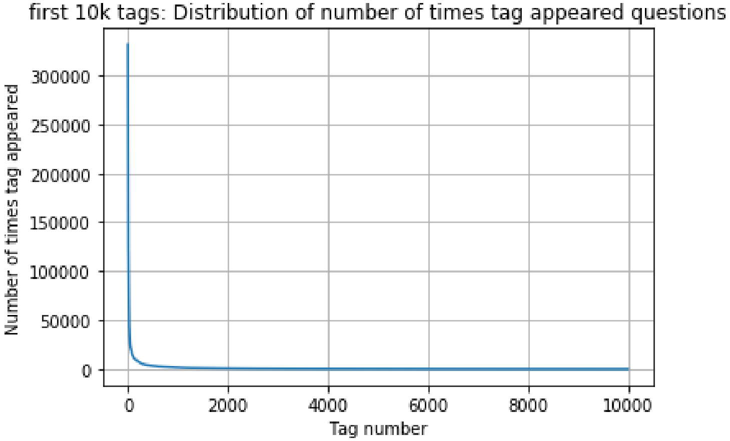
```
tag_df_sorted = tag_df.sort_values(['Counts'], ascending=False)
tag_counts = tag_df_sorted['Counts'].values
```

In [22]:

```
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()
```



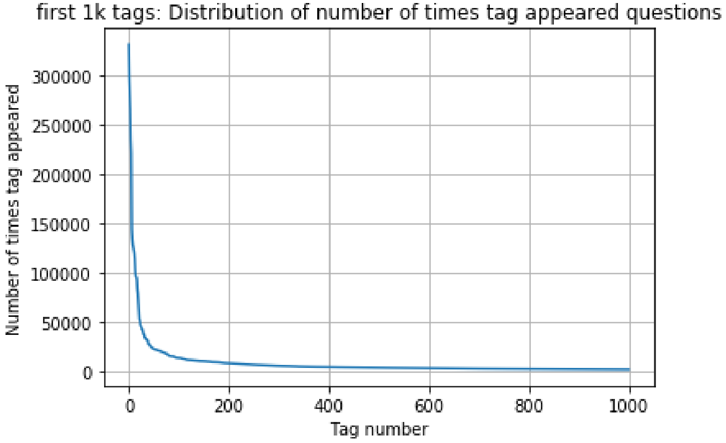
```
In [23]: plt.plot(tag_counts[0:10000])
         plt.title('first 10k tags: Distribution of number of times tag appeared questions')
         plt.grid()
         plt.xlabel("Tag number")
         plt.ylabel("Number of times tag appeared")
         plt.show()
         print(len(tag_counts[0:10000:25]), 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   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 [24]:

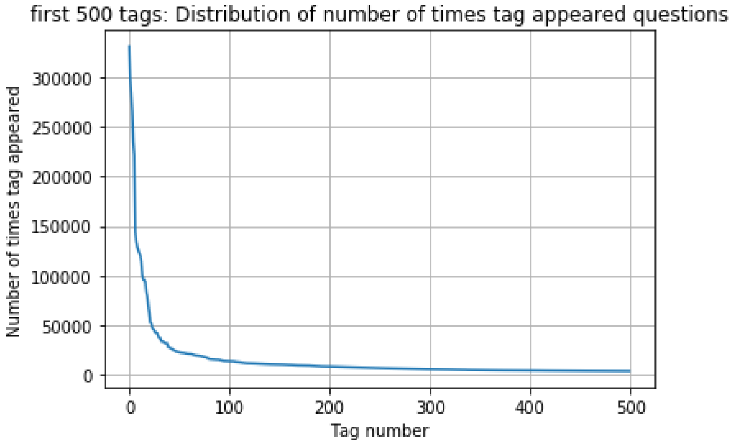
```
plt.plot(tag_counts[0:1000])
plt.title('first 1k tags: Distribution of number of times tag appeared questions')
plt.grid()
plt.xlabel("Tag number")
plt.ylabel("Number of times tag appeared")
plt.show()
print(len(tag_counts[0:1000:5]), tag_counts[0:1000:5])
```



```
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   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 [25]:

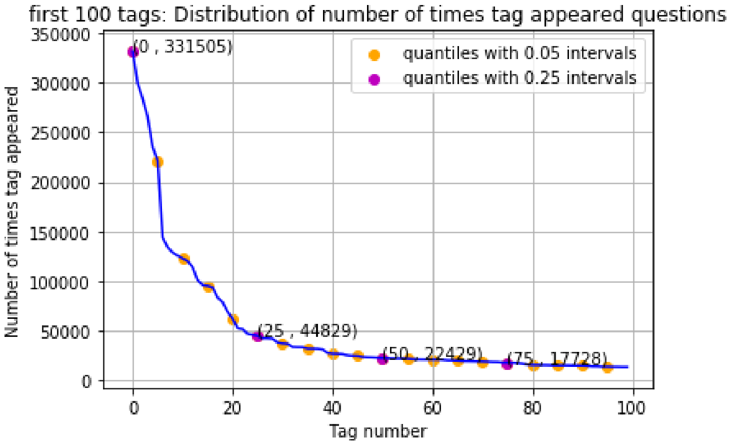
```
plt.plot(tag_counts[0:500])
plt.title('first 500 tags: Distribution of number of times tag appeared questions')
plt.grid()
plt.xlabel("Tag number")
plt.ylabel("Number of times tag appeared")
plt.show()
print(len(tag_counts[0:500:5]), tag_counts[0:500:5])
```



```
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 [26]:

```
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', label = "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  24537
  22429  21820  20957  19758  18905  17728  15533  15097  14884  13703]
```

In [27]:

```
# 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 [28]:

```
#Storing the count of tag in each question in list 'tag_count'
tag_quest_count = tag_dtm.sum(axis=1).tolist()
#Converting list of lists into single list, we will get [[3], [4], [2], [2], [3]] and we are converting this to [3, 4, 2, 2, 3]
tag_quest_count=[int(j) for i in tag_quest_count for j in i]
print ('We have total {} datapoints.'.format(len(tag_quest_count)))

print(tag_quest_count[:5])
```

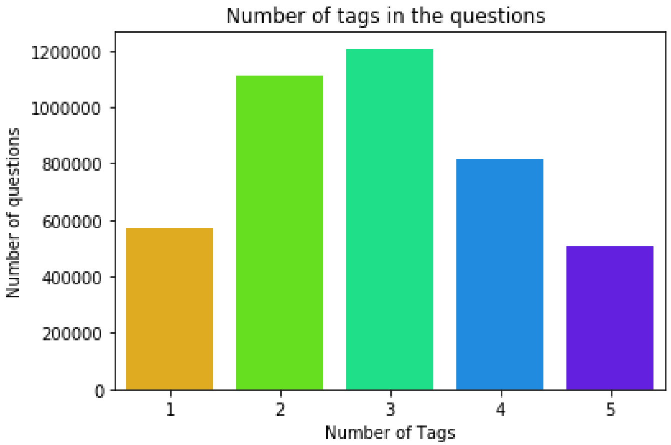
```
We have total 4206307 datapoints.
[3, 4, 2, 2, 3]
```

In [29]:

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

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

In [30]:
```
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()
```



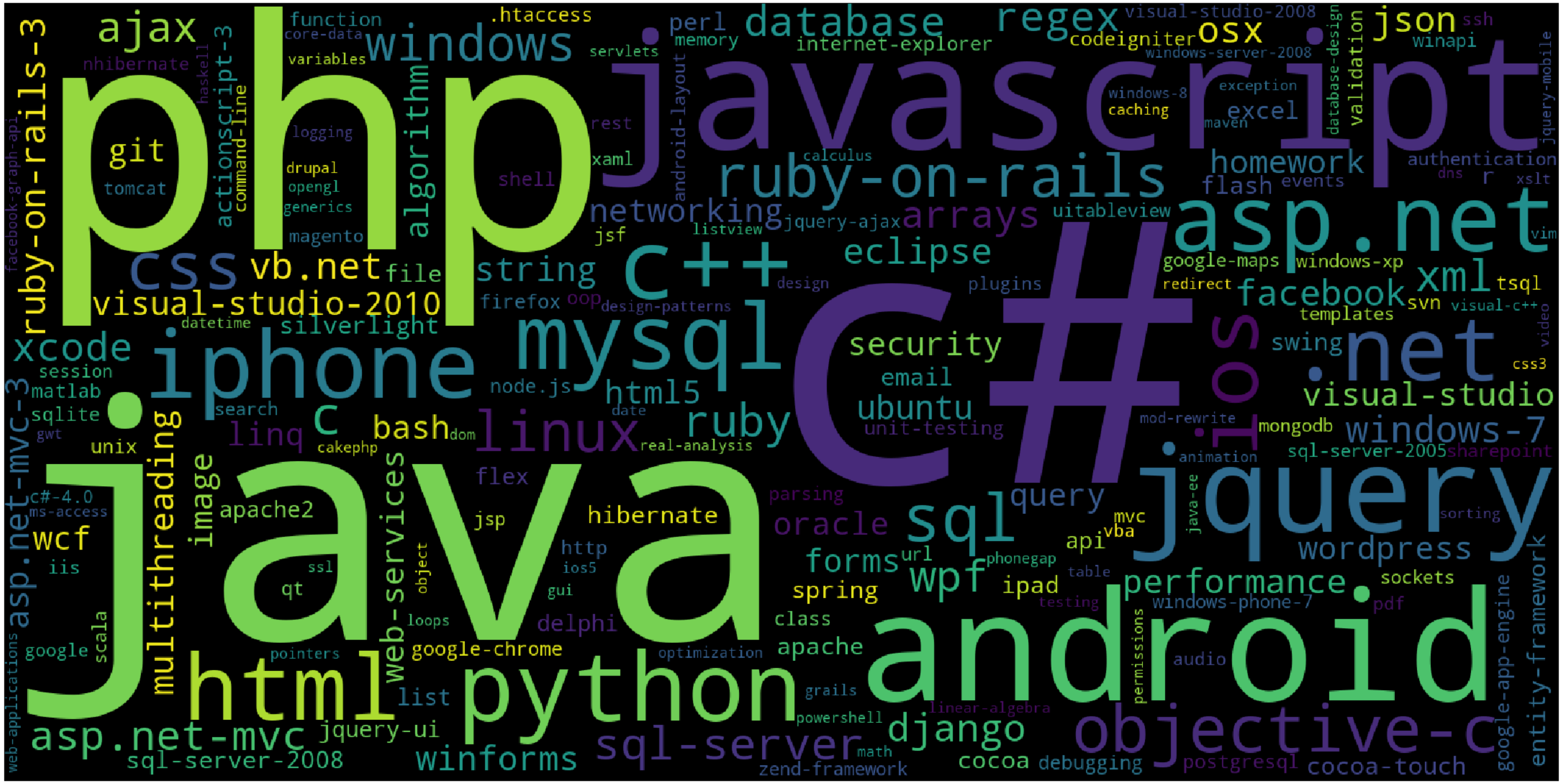
Observations:

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 [31]:

```
# 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)
```



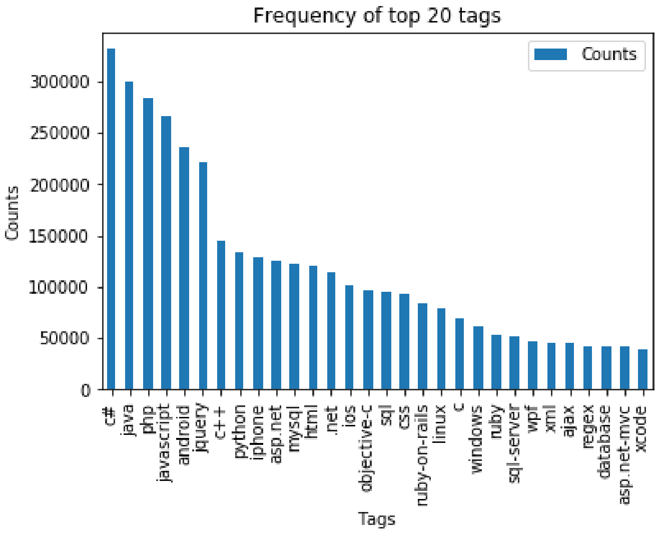
```
Time taken to run this cell : 0:00:03.983340
```

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 [32]:
```
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. Sample 1M data points
2. Separate out code-snippets from Body
3. Remove Spcial characters from Question title and description (not in code)
4. Remove stop words (Except 'C')
5. Remove HTML Tags
6. Convert all the characters into small letters
7. Use SnowballStemmer to stem the words

In [33]:

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

In [34]:

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

    return None

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

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

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

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

```
Tables in the databse:
```

```
QuestionsProcessed
```

In [35]:

```
# http://www.sqlitetutorial.net/sqlite-delete/
# https://stackoverflow.com/questions/2279706/select-random-row-from-a-sqlite-table
start = datetime.now()
read_db = 'train_no_dup.db'
write_db = 'Processed.db'
if os.path.isfile(read_db):
    conn_r = create_connection(read_db)
    if conn_r is not None:
        reader =conn_r.cursor()
        reader.execute("SELECT Title, Body, Tags From no_dup_train ORDER BY RANDOM() LIMIT 1000000;")

if os.path.isfile(write_db):
    conn_w = create_connection(write_db)
    if conn_w is not None:
        tables = checkTableExists(conn_w)
        writer =conn_w.cursor()
        if tables != 0:
            writer.execute("DELETE FROM QuestionsProcessed WHERE 1")
            print("Cleared All the rows")
print("Time taken to run this cell :", datetime.now() - start)
```

```
Tables in the databse:
QuestionsProcessed
Cleared All the rows
Time taken to run this cell : 0:10:20.080935
```

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

In [36]:

```
#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], row[2]

    if '<code>' in question:
        questions_with_code+=1
        is_code = 1
    x = len(question)+len(title)
    len_pre+=x

    code = str(re.findall(r'<code>(.*?)</code>', question, flags=re.DOTALL))

    question=re.sub('<code>(.*?)</code>', '', question, flags=re.MULTILINE|re.DOTALL)
    question=striphtml(question.encode('utf-8'))

    title=title.encode('utf-8')

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

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

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

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

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

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

```
number of questions completed= 100000
number of questions completed= 200000
number of questions completed= 300000
number of questions completed= 400000
number of questions completed= 500000
number of questions completed= 600000
number of questions completed= 700000
number of questions completed= 800000
number of questions completed= 900000
Avg. length of questions(Title+Body) before processing: 1173
Avg. length of questions(Title+Body) after processing: 327
Percent of questions containing code: 57
Time taken to run this cell : 0:23:20.925792
```

In [37]:

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

In [38]:
```
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
====================================================================================================
('count stuck thread weblog log show stuck thread mark weblog case observ mani stuck thread server get slower respons time eat memori want make count
stuck thread health index auto report robot count log file command api help count stuck thread summar solut wlst sampl code viccari',)
----------------------------------------------------------------------------------------------------
('languag best built graphic support architectur student design rather programm look program languag librari best support interact graphic exampl las
t week idea traffic intersect program would insert incom outgo lane connect node use mous show requir path would use calcul averag throughput use bes
t traffic light scheme time want draw shape fit room predefin floor space optim shape javascript canva process also rebol heard anyth concis easier a
vail window platform',)
----------------------------------------------------------------------------------------------------
('find internet time ntp set window window xp get internet time set select ntp server forc updat via date time properti control panel anyon know foun
d window edit nas point abl attach domain nmi problem pc tri sync time window com block firewal end chang registri key point intern ntp server time s
ync fine know domain comput sync domain control machin actual log domain time',)
----------------------------------------------------------------------------------------------------
('display grand total multipl tabl total one report report tabl use differ dataset neach tabl total row need way produc recap tabl tabl total previou
s tabl nthe dataset complet differ could realli hard merg one grand dataset would make recap tabl implement easi way refer total tabl report mayb par
amet would hold valu tabl total without run queri multipl time tabl paramet could extens',)
----------------------------------------------------------------------------------------------------
('startact failur use alarmmanag work applic need set alarm wake applic basic timer inform user time run want bring main activ front perform specif a
ction came across follow problem one implement alarmreceiv main activ brought front implement thing work expect alarmreceiv need defin static lead pr
oblem case content onrec execut fact first implement onrec strang call twice method defin custom widget mytimerwidget extend linearlayout part layout
main activ would realli like know first setalarm fail bring activ front manifest contain follow',)
----------------------------------------------------------------------------------------------------
('pass arraylist customobject function accept paramet arraylist object java write generic java android function accept one paramet use applic regardl
ess type arraylist element function tri pass paramet function get error cast best approach handl situat thank',)
----------------------------------------------------------------------------------------------------
('need defin visual studio version includ secur string function avoid crt secur deprec back tri use visual studio compil mfc program use librari writ
ten visual studio surpris got bunch warn deprec use secur version various string function updat relev function librari use secur function compil fine
later tri compil system visual studio got nag secur function exist decid creat hybrid approach would allow compil program use librari either environ
make use secur function avail alias old one first consid check function see secur version exist xe work requir separ work everi function xe tri figur
way determin secur function exist know introduc visual studio someth use follow check found noth use',)
----------------------------------------------------------------------------------------------------
('bsp tree work singl transpar object tri implement three dimension bsp tree render singl object cube box cylind etc transpar far understand work fig
ur everyth read refer bsp tree use either two dimens multipl object wonder general misunderstood bsp tree appli rather error code look lot thing onli
n code seem bretton wade ftp ftp sgi com bspfaq faq bspfaq html anybodi sampl bsp code singl object transpar particular would wonder thank',)
----------------------------------------------------------------------------------------------------
('macro oper std string comparison c bit code help convert enum string vice versa wrote macro make look better simpler call way unfortun compil vs wa
y give hint compil oper use',)
----------------------------------------------------------------------------------------------------
```

In [39]:

```
#Taking 1 Million entries to a dataframe.
write_db = 'Processed.db'
if os.path.isfile(write_db):
    conn_r = create_connection(write_db)
    if conn_r is not None:
        preprocessed_data = pd.read_sql_query("""SELECT question, Tags FROM QuestionsProcessed""", conn_r)
conn_r.commit()
conn_r.close()
```

In [40]:

```
preprocessed_data.head()
```

Out[40]:

	question	tags
0	infinit loop came across question forum given ...	c++ while-loops infinite-loop
1	count stuck thread weblog log show stuck threa...	weblogic-10.x
2	languag best built graphic support architectur...	programming-languages
3	find internet time ntp set window window xp ge...	windows-7 ntp
4	display grand total multipl tabl total one rep...	reporting-services ssrs-2008

In [41]:

```
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 : 999999
number of dimensions : 2
```

4. Machine Learning Models

4.1 Converting tags for multilabel problems

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

In [42]:

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

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

In [43]:

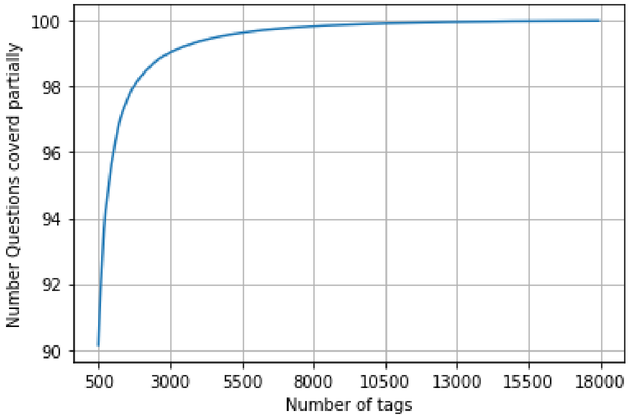
```
def tags_to_choose(n):
    t = multilabel_y.sum(axis=0).tolist()[0]
    sorted_tags_i = sorted(range(len(t)), key=lambda i: t[i], reverse=True)
    multilabel_yn=multilabel_y[:,sorted_tags_i[:n]]
    return multilabel_yn

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

In [44]:

```
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 [45]:

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



```
with  5500 tags we are covering  99.03 % of questions
```

In [46]:

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

```
number of questions that are not covered : 9695 out of  999999
```

In [47]:

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

```
Number of tags in sample : 35428
number of tags taken : 5500 ( 15.524443942644236 %)
```

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

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

In [48]:

```
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 [49]:

```
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 : (799999, 5500)
Number of data points in test data : (200000, 5500)
```

4.3 Featurizing data

In [50]:

```
start = datetime.now()
vectorizer = TfidfVectorizer(min_df=0.00009, max_features=200000, smooth_idf=True, norm="l2", \
                             tokenizer = lambda x: x.split(), sublinear_tf=False, ngram_range=(1,3))
x_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 : 0:05:59.651410
```

In [51]:

```
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: (799999, 88340) Y : (799999, 5500)
Dimensions of test data X: (200000, 88340) Y: (200000, 5500)
```

In [52]:

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

# train
classifier.fit(x_train_multilabel, y_train)

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

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

Out[52]:

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

4.4 Applying Logistic Regression with OneVsRest Classifier

In [53]:
```
# this will be taking so much time try not to run it, download the lr_with_equal_weight.pkl file and use to predict
# This takes about 6-7 hours to run.
""""
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("macro f1 score :",metrics.f1_score(y_test, predictions, average = 'macro'))
print("micro f1 scoore :",metrics.f1_score(y_test, predictions, average = 'micro'))
print("hamming loss :",metrics.hamming_loss(y_test,predictions))
print("Precision recall report :\n",metrics.classification_report(y_test, predictions))
""""
```

```
  File "<ipython-input-53-1dba43333fdc>", line 13
    """"

^
SyntaxError: EOL while scanning string literal
```

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

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

In [54]:
```
sql_create_table = """CREATE TABLE IF NOT EXISTS QuestionsProcessed (question text NOT NULL, code text, tags text, words_pre integer, words_post integ
create_database_table("Titlemoreweight.db", sql_create_table)
```

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

In [55]:

```python
# 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 = 70000
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 100001;")
        # for selecting random points
        #reader.execute("SELECT Title, Body, Tags From no_dup_train ORDER BY RANDOM() LIMIT 500001;")

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

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

4.5.1 Preprocessing of questions

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

```
    <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 [56]:

```
#http://www.bernzilla.com/2008/05/13/selecting-a-random-row-from-an-sqlite-table/
start = datetime.now()
preprocessed_data_list=[]
reader.fetchone()
questions_with_code=0
len_pre=0
len_post=0
questions_proccesed = 0
for row in reader:

    is_code = 0

    title, question, tags = row[0], row[1], str(row[2])

    if '<code>' in question:
        questions_with_code+=1
        is_code = 1
    x = len(question)+len(title)
    len_pre+=x

    code = str(re.findall(r'<code>(.*?)</code>', question, flags=re.DOTALL))

    question=re.sub('<code>(.*?)</code>', '', question, flags=re.MULTILINE|re.DOTALL)
    question=striphtml(question.encode('utf-8'))

    title=title.encode('utf-8')

    # adding title three time to the data to increase its weight
    # add tags string to the training data

    question=str(title)+" "+str(title)+" "+str(title)+" "+question

#     if questions_proccesed<=train_datasize:
#         question=str(title)+" "+str(title)+" "+str(title)+" "+question+" "+str(tags)
#     else:
#         question=str(title)+" "+str(title)+" "+str(title)+" "+question

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

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

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

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

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

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

```
number of questions completed= 100000
Avg. length of questions(Title+Body) before processing: 1232
Avg. length of questions(Title+Body) after processing: 441
Percent of questions containing code: 57
Time taken to run this cell : 0:03:11.897956
```

In [57]:

```
# 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 [58]:

```
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 silverlight bind datagrid dynam code wrote code debug code bloc
k seem bind correct grid come column form come grid column although necessari bind nthank repli advance..',)
----------------------------------------------------------------------------------------------------
('java.lang.noclassdeffounderror javax servlet jsp tagext taglibraryvalid java.lang.noclassdeffounderror javax servlet jsp tagext taglibraryvalid jav
a.lang.noclassdeffounderror javax servlet jsp tagext taglibraryvalid follow guid link instal jstl got follow error tri launch jsp page java.lang.nocl
assdeffounderror javax servlet jsp tagext taglibraryvalid taglib declar instal jstl 1.1 tomcat webapp tri project work also tri version 1.2 jstl stil
l messag caus solv',)
----------------------------------------------------------------------------------------------------
('java.sql.sqlexcept microsoft odbc driver manag invalid descriptor index java.sql.sqlexcept microsoft odbc driver manag invalid descriptor index jav
a.sql.sqlexcept microsoft odbc driver manag invalid descriptor index use follow code display caus solv',)
----------------------------------------------------------------------------------------------------
('better way updat feed fb php sdk better way updat feed fb php sdk better way updat feed fb php sdk novic facebook api read mani tutori still confus
ed.i find post feed api method like correct second way use curl someth like way better',)
----------------------------------------------------------------------------------------------------
('btnadd click event open two window record ad btnadd click event open two window record ad btnadd click event open two window record ad open window
search.aspx use code hav add button search.aspx nwhen insert record btnadd click event open anoth window nafter insert record close window',)
----------------------------------------------------------------------------------------------------
('sql inject issu prevent correct form submiss php sql inject issu prevent correct form submiss php sql inject issu prevent correct form submiss php
check everyth think make sure input field safe type sql inject good news safe bad news one tag mess form submiss place even touch life figur exact ht
ml use templat file forgiv okay entir php script get execut see data post none forum field post problem use someth titl field none data get post curr
ent use print post see submit noth work flawless statement though also mention script work flawless local machin use host come across problem state l
ist input test mess',)
----------------------------------------------------------------------------------------------------
('countabl subaddit lebesgu measur countabl subaddit lebesgu measur countabl subaddit lebesgu measur let lbrace rbrace sequenc set sigma -algebra mat
hcal want show left bigcup right leq sum left right countabl addit measur defin set sigma algebra mathcal think use monoton properti somewher proof s
tart appreci littl help nthank ad han answer make follow addit construct given han answer clear bigcup bigcup cap emptyset neq left bigcup right left
bigcup right sum left right also construct subset monoton left right leq left right final would sum leq sum result follow',)
----------------------------------------------------------------------------------------------------
('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 error undefin symbol architectur i386 objc class skpsmtpmessag referenc error und
efin symbol architectur i386 objc class skpsmtpmessag referenc error import framework send email applic background import framework i.e skpsmtpmessag
somebodi suggest get error collect2 ld return exit status import framework correct sorc taken framework follow mfmailcomposeviewcontrol question lock
field updat answer drag drop folder project click copi nthat',)
----------------------------------------------------------------------------------------------------
```

__ Saving Preprocessed data to a Database __

In [59]:

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

In [60]:

```
preprocessed_data.head()
```

Out[60]:

	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 [61]:

```
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 : 100000
number of dimensions : 2
```

__ Converting string Tags to multilable output variables __

In [62]:

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

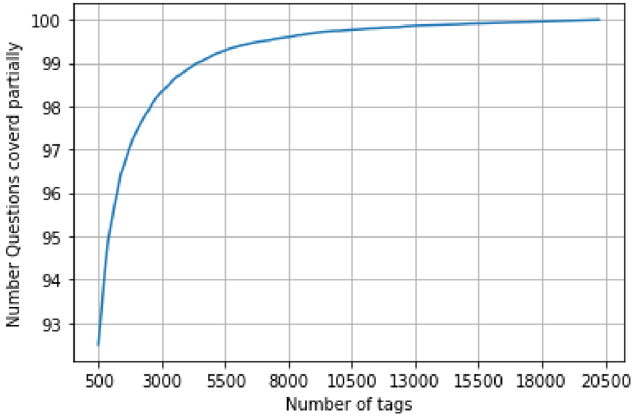
__ Selecting 500 Tags __

In [63]:

```
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 [64]:

```
fig, ax = plt.subplots()
ax.plot(questions_explained)
xlabel = list(500+np.array(range(-50,450,50))*50)
ax.set_xticklabels(xlabel)
plt.xlabel("Number of tags")
plt.ylabel("Number Questions coverd partially")
plt.grid()
plt.show()
# you can choose any number of tags based on your computing power, minimun is 500(it covers 90% of the tags)
print("with ",5500,"tags we are covering ",questions_explained[50],"% of questions")
print("with ",500,"tags we are covering ",questions_explained[0],"% of questions")
```



```
with  5500 tags we are covering  99.481 % of questions
with  500 tags we are covering  92.5 % of questions
```

In [65]:

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

```
number of questions that are not covered : 7500 out of  100000
```

In [66]:

```
x_train=preprocessed_data.head(train_datasize)
x_test=preprocessed_data.tail(preprocessed_data.shape[0] - 70000)

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

In [67]:

```
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 : (70000, 500)
Number of data points in test data : (30000, 500)
```

4.5.2 Featurizing data with TfIdf vectorizer

In [68]:

```
start = datetime.now()
vectorizer = TfidfVectorizer(min_df=0.00009, max_features=200000, smooth_idf=True, norm="l2", \
                             tokenizer = lambda x: x.split(), sublinear_tf=False, ngram_range=(1,3))
x_train_multilabel = vectorizer.fit_transform(x_train['question'])
x_test_multilabel = vectorizer.transform(x_test['question'])
print("Time taken to run this cell :", datetime.now() - start)
```

```
Time taken to run this cell : 0:00:38.297420
```

In [69]:

```
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: (70000, 101047) Y : (70000, 500)
Dimensions of test data X: (30000, 101047) Y: (30000, 500)
```

4.5.3 Applying Logistic Regression with OneVsRest Classifier

```
In [70]: 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, f1))

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

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

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

```
Accuracy : 0.18893333333333334
Hamming loss  0.0031336666666666665
Micro-average quality numbers
Precision: 0.7294, Recall: 0.3504, F1-measure: 0.4734
Macro-average quality numbers
Precision: 0.5290, Recall: 0.2362, F1-measure: 0.3057
             precision    recall  f1-score   support

          0       0.88      0.83      0.85      6668
          1       0.66      0.17      0.27      3659
          2       0.52      0.08      0.15       971
          3       0.73      0.55      0.63      1506
          4       0.80      0.44      0.57      1649
          5       0.86      0.50      0.63      1113
          6       0.78      0.38      0.51      1482
          7       0.86      0.56      0.68       980
          8       0.91      0.59      0.72      1520
          9       0.74      0.45      0.56      1041
         10       0.78      0.50      0.61       861
```

In [72]:

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

Out[72]:

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

In [73]:

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


precision = precision_score(y_test, predictions_2, average='micro')
recall = recall_score(y_test, predictions_2, average='micro')
f1 = f1_score(y_test, predictions_2, average='micro')

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

precision = precision_score(y_test, predictions_2, average='macro')
recall = recall_score(y_test, predictions_2, average='macro')
f1 = f1_score(y_test, predictions_2, average='macro')

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

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

```
Accuracy : 0.19003333333333333
Hamming loss  0.0031274666666666665
Micro-average quality numbers
Precision: 0.7264, Recall: 0.3561, F1-measure: 0.4779
Macro-average quality numbers
Precision: 0.5392, Recall: 0.2482, F1-measure: 0.3179
             precision    recall  f1-score   support

          0       0.87      0.83      0.85      6668
          1       0.66      0.15      0.25      3659
          2       0.52      0.08      0.13       971
          3       0.74      0.54      0.62      1506
          4       0.79      0.45      0.57      1649
          5       0.86      0.49      0.62      1113
          6       0.77      0.37      0.50      1482
          7       0.85      0.56      0.67       980
          8       0.92      0.58      0.71      1520
          9       0.72      0.49      0.58      1041
         10       0.79      0.48      0.60       861
         11       0.62      0.34      0.44       286
```

5. Assignments

1. Use bag of words upto 4 grams and compute the micro f1 score with Logistic regression(OvR)
2. Perform hyperparam tuning on alpha (or lambda) for Logistic regression to improve the performance using GridSearch
3. Try OneVsRestClassifier with Linear-SVM (SGDClassifier with loss-hinge)

Task 1 : Use bag of words upto 4 grams and compute the micro f1 score with Logistic regression(OvR)

Featurizing using Bag of Words

In [95]:

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

```
Time taken to run this cell : 0:01:03.887487
```

In [96]:

```
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: (70000, 102491) Y : (70000, 500)
Dimensions of test data X: (30000, 102491) Y: (30000, 500)
```

Applying Logistic Regression on BoW Features with OneVsRest Classifier

In [87]:

```
start = datetime.now()
classifier = OneVsRestClassifier(LogisticRegression(penalty='l1'), n_jobs=-1)
classifier.fit(x_train_multilabel, y_train)
predictions = classifier.predict (x_test_multilabel)


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


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

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

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

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

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

```
Accuracy : 0.17146666666666666
Hamming loss  0.0034852
Micro-average quality numbers
Precision: 0.5971, Recall: 0.4086, F1-measure: 0.4852
Macro-average quality numbers
Precision: 0.4253, Recall: 0.2964, F1-measure: 0.3386
             precision    recall  f1-score   support

          0       0.86      0.82      0.84      6668
          1       0.46      0.28      0.35      3659
          2       0.23      0.12      0.16       971
          3       0.68      0.56      0.62      1506
          4       0.65      0.50      0.57      1649
          5       0.72      0.51      0.59      1113
          6       0.63      0.43      0.52      1482
          7       0.69      0.58      0.63       980
          8       0.85      0.63      0.73      1520
          9       0.74      0.68      0.71      1041
         10       0.70      0.57      0.63       861
         11       0.50      0.35      0.41       386
```

Task 2 : Perform hyperparam tuning on alpha (or lambda) for Logistic regression to improve the performance using GridSearch

In [97]:

```
start = datetime.now()

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV

param = dict(estimator__C=[0.001, 0.01, 0.1, 1])

model = GridSearchCV(OneVsRestClassifier(LogisticRegression()), param_grid=param, verbose=5, n_jobs=-1)
model.fit(x_train_multilabel, y_train)

print('The best hyper parameter is ', model.best_params_)
print('The best score is ', model.best_score_)
print("Time taken to run this cell :", datetime.now() - start)
```

```
Fitting 3 folds for each of 4 candidates, totalling 12 fits

[Parallel(n_jobs=-1)]: Done   8 out of  12 | elapsed: 60.2min remaining: 30.1min
[Parallel(n_jobs=-1)]: Done  12 out of  12 | elapsed: 94.5min finished

The best hyper parameter is  {'estimator__C': 0.1}
The best score is  0.1745142857142857
Time taken to run this cell : 2:04:05.033169
```

In [98]:

```
best_c = model.best_params_['estimator__C']
```

In [99]:
```
start = datetime.now()
classifier = OneVsRestClassifier(LogisticRegression(C=best_c))
classifier.fit(x_train_multilabel, y_train)
predictions = classifier.predict (x_test_multilabel)


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


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

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

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

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

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

```
Accuracy : 0.1806
Hamming loss  0.003238266666666667
Micro-average quality numbers
Precision: 0.6941, Recall: 0.3475, F1-measure: 0.4632
Macro-average quality numbers
Precision: 0.5058, Recall: 0.2200, F1-measure: 0.2901
             precision    recall  f1-score   support

          0       0.87      0.83      0.85      6668
          1       0.56      0.23      0.33      3659
          2       0.41      0.09      0.15       971
          3       0.72      0.56      0.63      1506
          4       0.75      0.48      0.58      1649
          5       0.84      0.47      0.60      1113
          6       0.74      0.39      0.51      1482
          7       0.81      0.56      0.66       980
          8       0.90      0.58      0.71      1520
          9       0.76      0.43      0.55      1041
         10       0.78      0.52      0.62       861
         11       0.58      0.32      0.41       386
```

Task 3 : Try OneVsRestClassifier with Linear-SVM (SGDClassifier with loss-hinge)

In [100]:

```
start = datetime.now()
classifier = OneVsRestClassifier(SGDClassifier(loss='hinge', alpha=0.00001, penalty='l1'))
classifier.fit(x_train_multilabel, y_train)
predictions = classifier.predict (x_test_multilabel)


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


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

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

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

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

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

```
Accuracy : 0.09273333333333333
Hamming loss  0.006357
Micro-average quality numbers
Precision: 0.3042, Recall: 0.4519, F1-measure: 0.3637
Macro-average quality numbers
Precision: 0.1890, Recall: 0.3356, F1-measure: 0.2302
             precision    recall  f1-score   support

          0       0.81      0.81      0.81      6668
          1       0.37      0.35      0.36      3659
          2       0.13      0.16      0.14       971
          3       0.52      0.55      0.53      1506
          4       0.46      0.56      0.50      1649
          5       0.51      0.58      0.54      1113
          6       0.54      0.49      0.51      1482
          7       0.49      0.61      0.54       980
          8       0.70      0.74      0.72      1520
          9       0.66      0.66      0.66      1041
         10       0.51      0.59      0.55       861
         11       0.17      0.41      0.24       386
         12       0.13      0.51      0.21        37
         13       0.50      0.43      0.46       917
         14       0.27      0.31      0.29       519
```

```
15        0.39      0.52      0.45       656
16        0.40      0.33      0.36       794
17        0.37      0.33      0.35       700
18        0.47      0.71      0.56       363
19        0.53      0.66      0.59       541
20        0.30      0.51      0.37       540
21        0.56      0.61      0.58       362
22        0.56      0.52      0.53       551
23        0.23      0.31      0.27       309
24        0.29      0.47      0.36       331
25        0.29      0.36      0.32       424
26        0.26      0.37      0.31       465
27        0.15      0.18      0.16       386
28        0.12      0.41      0.19       107
29        0.18      0.27      0.21       195
30        0.34      0.51      0.40       758
31        0.06      0.47      0.11        15
32        0.43      0.62      0.51       323
33        0.24      0.28      0.26       279
34        0.32      0.42      0.36       275
35        0.40      0.59      0.48       268
36        0.04      0.09      0.06        76
37        0.12      0.25      0.16       269
38        0.44      0.49      0.46       255
39        0.24      0.42      0.30       249
40        0.08      0.23      0.12        66
41        0.14      0.24      0.18       209
42        0.28      0.38      0.32        72
43        0.29      0.50      0.37       430
44        0.18      0.30      0.22       279
45        0.28      0.40      0.33       240
46        0.24      0.39      0.30       157
47        0.43      0.63      0.51       249
48        0.13      0.19      0.15       198
49        0.21      0.39      0.28       171
50        0.49      0.70      0.58       200
51        0.53      0.82      0.65        85
52        0.27      0.45      0.34       175
53        0.13      0.26      0.17       114
54        0.07      0.20      0.11       223
55        0.21      0.36      0.27       122
56        0.33      0.62      0.43       168
57        0.04      0.07      0.05       176
58        0.13      0.29      0.17       140
59        0.18      0.17      0.17       191
60        0.50      0.75      0.60       152
61        0.19      0.16      0.17       208
62        0.10      0.17      0.13       136
63        0.39      0.37      0.38       158
64        0.28      0.46      0.35       203
65        0.29      0.44      0.35       105
```

```
 66     0.30    0.64    0.41     58
 67     0.28    0.58    0.38    128
 68     0.08    0.10    0.09    158
 69     0.18    0.26    0.21    248
 70     0.23    0.38    0.29    201
 71     0.22    0.52    0.31     89
 72     0.29    0.41    0.34    157
 73     0.14    0.38    0.20     29
 74     0.02    0.07    0.03     58
 75     0.18    0.30    0.23    158
 76     0.42    0.55    0.48    110
 77     0.28    0.55    0.37     33
 78     0.12    0.19    0.15    210
 79     0.42    0.60    0.50    169
 80     0.05    0.27    0.09     15
 81     0.24    0.38    0.29    214
 82     0.14    0.29    0.19     65
 83     0.13    0.23    0.16    156
 84     0.29    0.56    0.38     59
 85     0.39    0.71    0.50     55
 86     0.06    0.19    0.09     36
 87     0.33    0.55    0.41     29
 88     0.22    0.65    0.33     54
 89     0.43    0.77    0.55    137
 90     0.15    0.25    0.19    103
 91     0.12    0.20    0.15     79
 92     0.14    0.29    0.19     84
 93     0.23    0.56    0.33    133
 94     0.62    0.73    0.67    318
 95     0.23    0.55    0.32     51
 96     0.23    0.34    0.28     82
 97     0.06    0.15    0.08     75
 98     0.05    0.04    0.04    120
 99     0.19    0.44    0.27     18
100     0.34    0.46    0.39    196
101     0.40    0.54    0.46    208
102     0.10    0.19    0.13    122
103     0.01    0.05    0.02     62
104     0.10    0.17    0.12     88
105     0.34    0.42    0.37     65
106     0.15    0.17    0.16    115
107     0.05    0.17    0.08     29
108     0.16    0.24    0.19    109
109     0.23    0.32    0.27     73
110     0.10    0.30    0.15    102
111     0.42    0.44    0.43    180
112     0.77    0.25    0.38    292
113     0.49    0.72    0.58     54
114     0.09    0.08    0.09    120
115     0.21    0.40    0.28    107
116     0.12    0.27    0.17     52
```

```
117         0.06        0.19        0.09         72
118         0.34        0.55        0.42        139
119         0.33        0.42        0.37         57
120         0.37        0.50        0.42         44
121         0.13        0.24        0.17         85
122         0.40        0.56        0.47         82
123         0.03        0.05        0.03        100
124         0.15        1.00        0.27          4
125         0.13        0.67        0.22          9
126         0.06        0.15        0.08         46
127         0.08        0.19        0.11         54
128         0.75        0.66        0.70        195
129         0.26        0.52        0.35         54
130         0.07        0.16        0.09         96
131         0.41        0.71        0.52         35
132         0.04        0.10        0.05         58
133         0.06        0.17        0.09         36
134         0.23        0.39        0.29         36
135         0.32        0.62        0.42         39
136         0.00        0.00        0.00         97
137         0.15        0.44        0.22         70
138         0.11        0.24        0.15         17
139         0.09        0.26        0.13        119
140         0.43        0.60        0.50        101
141         0.27        0.38        0.32        115
142         0.20        0.29        0.24         94
143         0.34        0.54        0.41         84
144         0.27        0.53        0.36         64
145         0.04        0.07        0.05         61
146         0.10        0.19        0.13        132
147         0.26        0.34        0.29        119
148         0.35        0.61        0.44         62
149         0.09        0.29        0.14         83
150         0.09        0.18        0.12         72
151         0.09        0.48        0.16         23
152         0.09        0.17        0.12         76
153         0.22        0.50        0.31         18
154         0.11        0.24        0.15         17
155         0.08        0.21        0.11         24
156         0.28        0.28        0.28        136
157         0.28        0.38        0.32        129
158         0.16        0.31        0.21        143
159         0.39        0.66        0.49        107
160         0.20        0.42        0.27         78
161         0.11        0.40        0.17         73
162         0.04        0.10        0.06        106
163         0.10        0.13        0.11        126
164         0.34        0.41        0.37         63
165         0.00        0.00        0.00        229
166         0.36        0.37        0.37        115
167         0.12        0.22        0.16         46
```

```
168        0.18       0.30       0.23         69
169        0.27       0.54       0.36         70
170        0.32       0.33       0.32         54
171        0.01       0.05       0.02         43
172        0.31       0.45       0.37         76
173        0.11       0.50       0.17         12
174        0.09       0.16       0.11         76
175        0.32       0.54       0.40         91
176        0.51       0.63       0.57        157
177        0.20       0.44       0.27         41
178        0.00       0.00       0.00          0
179        0.04       1.00       0.07          1
180        0.16       0.42       0.23         55
181        0.04       0.10       0.06         62
182        0.00       0.00       0.00          2
183        0.21       0.41       0.28         80
184        0.11       0.00       0.01        206
185        0.26       0.26       0.26         86
186        0.23       0.47       0.31         66
187        0.42       0.66       0.52         59
188        0.35       0.65       0.46         68
189        0.12       0.16       0.14        108
190        0.15       0.21       0.17         85
191        0.32       0.27       0.29         86
192        0.14       0.50       0.22         46
193        0.25       0.33       0.29         18
194        0.30       0.68       0.42         74
195        0.14       0.40       0.21         55
196        0.21       0.61       0.31         38
197        0.24       0.38       0.30         95
198        0.04       0.19       0.06         16
199        0.10       0.21       0.14         39
200        0.10       0.14       0.11         58
201        0.09       0.24       0.13         55
202        0.08       0.24       0.12         58
203        0.10       0.14       0.12         66
204        0.44       0.64       0.52         64
205        0.00       0.00       0.00         10
206        0.03       0.27       0.06         66
207        0.12       0.18       0.15         73
208        0.05       0.09       0.07         54
209        0.16       0.26       0.20         61
210        0.10       0.33       0.15         12
211        0.08       0.15       0.10         59
212        0.15       0.46       0.22         26
213        0.17       0.30       0.22        105
214        0.22       0.48       0.30         50
215        0.09       0.18       0.12         65
216        0.24       0.42       0.31         79
217        0.14       0.27       0.19         55
218        0.05       0.33       0.09          3
```

219	0.05	0.13	0.08	62
220	0.16	0.12	0.14	81
221	0.12	0.29	0.17	34
222	0.05	0.11	0.07	64
223	0.16	0.39	0.23	61
224	0.05	0.22	0.08	18
225	0.38	0.60	0.46	10
226	0.50	0.75	0.60	99
227	0.21	0.62	0.31	13
228	0.10	0.26	0.14	74
229	0.50	0.76	0.60	50
230	0.11	0.15	0.13	74
231	0.00	0.00	0.00	4
232	0.20	0.31	0.24	26
233	0.14	0.31	0.19	146
234	0.29	0.46	0.35	61
235	0.05	0.38	0.10	13
236	0.07	0.16	0.10	49
237	0.45	0.46	0.45	90
238	0.11	0.17	0.14	58
239	0.05	0.17	0.08	24
240	0.46	0.58	0.51	64
241	0.44	0.68	0.54	75
242	0.30	0.49	0.37	63
243	0.42	0.50	0.46	76
244	0.27	0.46	0.34	63
245	0.06	0.07	0.07	41
246	0.73	0.37	0.49	162
247	0.07	0.27	0.11	22
248	0.41	0.60	0.49	52
249	0.11	0.53	0.18	19
250	0.23	0.57	0.32	23
251	0.21	0.51	0.30	57
252	0.18	0.28	0.22	36
253	0.04	0.07	0.05	41
254	0.04	0.10	0.05	10
255	0.01	0.05	0.02	22
256	0.17	0.62	0.27	8
257	0.19	0.29	0.23	62
258	0.13	0.30	0.18	43
259	0.32	0.56	0.41	87
260	0.01	0.02	0.02	56
261	0.00	0.00	0.00	3
262	0.11	0.40	0.17	20
263	0.04	0.13	0.06	15
264	0.03	0.16	0.05	50
265	0.16	0.36	0.22	25
266	0.08	0.23	0.12	47
267	0.41	0.62	0.49	97
268	0.30	0.81	0.44	36
269	0.30	0.54	0.38	56

270	0.26	0.55	0.35	38
271	0.02	0.07	0.03	58
272	0.17	0.50	0.26	8
273	0.04	0.07	0.05	27
274	0.08	0.19	0.12	123
275	0.16	0.38	0.22	69
276	0.49	0.72	0.58	112
277	0.02	0.06	0.03	31
278	0.04	0.03	0.04	29
279	0.10	0.29	0.15	38
280	0.25	0.32	0.28	50
281	0.39	0.55	0.46	20
282	0.54	0.71	0.62	45
283	0.14	0.40	0.21	15
284	0.24	0.32	0.28	74
285	0.12	0.15	0.13	46
286	0.05	0.10	0.07	29
287	0.03	0.06	0.04	54
288	0.30	0.58	0.39	33
289	0.01	0.04	0.02	26
290	0.45	0.54	0.49	41
291	0.06	0.17	0.09	24
292	0.14	0.30	0.19	40
293	0.20	0.52	0.29	33
294	0.06	0.26	0.10	31
295	0.02	0.04	0.03	47
296	0.04	0.18	0.06	33
297	0.08	0.22	0.12	45
298	0.07	0.17	0.10	59
299	0.07	0.12	0.09	51
300	0.12	0.18	0.14	49
301	0.11	0.55	0.18	38
302	0.27	0.57	0.37	28
303	0.14	0.31	0.20	16
304	0.07	0.22	0.11	32
305	0.09	0.29	0.14	24
306	0.10	0.18	0.13	44
307	0.08	0.50	0.14	6
308	0.01	0.04	0.02	48
309	0.38	0.47	0.42	49
310	0.01	0.05	0.02	38
311	0.16	0.18	0.17	62
312	0.04	0.11	0.06	27
313	0.05	0.04	0.04	49
314	0.14	0.29	0.19	24
315	0.13	0.07	0.09	59
316	0.08	0.30	0.12	10
317	0.14	0.34	0.20	67
318	0.13	0.50	0.21	12
319	0.00	0.00	0.00	14
320	0.04	0.17	0.07	12

321	0.17	0.67	0.27	9
322	0.24	0.39	0.30	23
323	0.25	0.64	0.36	33
324	0.39	0.49	0.43	57
325	0.04	0.20	0.07	25
326	0.03	0.07	0.05	44
327	0.03	0.19	0.06	27
328	0.11	0.24	0.15	34
329	0.05	0.14	0.07	7
330	0.20	0.41	0.27	22
331	0.05	0.08	0.06	25
332	0.85	0.67	0.75	106
333	0.44	0.50	0.47	84
334	0.02	0.03	0.02	36
335	0.13	0.46	0.21	13
336	0.00	0.00	0.00	37
337	0.13	0.29	0.18	38
338	0.50	0.77	0.61	44
339	0.05	0.18	0.08	34
340	0.18	0.38	0.25	40
341	0.33	0.57	0.41	23
342	0.02	0.09	0.03	11
343	0.20	0.75	0.32	12
344	0.09	0.28	0.14	25
345	0.00	0.00	0.00	1
346	0.06	0.20	0.10	41
347	0.06	0.17	0.09	46
348	0.03	0.11	0.04	19
349	0.12	0.45	0.18	38
350	0.16	0.33	0.21	33
351	0.10	0.38	0.16	53
352	0.00	0.00	0.00	49
353	0.23	0.37	0.29	27
354	0.10	0.13	0.11	31
355	0.10	0.50	0.17	12
356	0.09	0.21	0.13	33
357	0.33	0.67	0.44	24
358	0.20	0.35	0.26	34
359	0.27	0.61	0.37	33
360	0.08	0.19	0.11	47
361	0.24	0.38	0.29	39
362	0.49	0.55	0.52	38
363	0.08	0.35	0.12	17
364	0.10	0.27	0.15	33
365	0.10	0.19	0.13	26
366	0.09	0.26	0.14	19
367	0.08	0.01	0.02	98
368	0.26	0.39	0.32	38
369	0.28	0.54	0.37	28
370	0.05	0.27	0.08	15
371	0.05	0.27	0.09	22

372	0.04	0.17	0.07	12
373	0.06	0.33	0.10	6
374	0.07	0.23	0.10	31
375	0.06	0.13	0.09	38
376	0.00	0.00	0.00	42
377	0.05	0.13	0.07	23
378	0.07	0.50	0.12	4
379	0.00	0.00	0.00	37
380	0.08	0.50	0.14	6
381	0.08	0.39	0.13	18
382	0.20	0.50	0.29	40
383	0.02	0.06	0.03	53
384	0.12	0.40	0.18	25
385	0.18	0.30	0.22	53
386	0.24	0.79	0.37	14
387	0.31	0.47	0.37	88
388	0.02	0.12	0.03	16
389	0.09	0.25	0.13	8
390	0.02	0.14	0.04	37
391	0.58	0.63	0.61	52
392	0.02	0.06	0.03	17
393	0.28	0.68	0.39	37
394	0.00	0.00	0.00	19
395	0.04	0.11	0.06	9
396	0.02	0.07	0.03	14
397	0.37	0.62	0.46	29
398	0.28	0.50	0.36	38
399	0.75	0.87	0.80	38
400	0.05	0.06	0.05	36
401	0.20	0.20	0.20	56
402	0.56	0.75	0.64	20
403	0.00	0.00	0.00	11
404	0.38	0.56	0.45	27
405	0.58	0.86	0.70	57
406	0.00	0.00	0.00	95
407	0.07	0.12	0.09	25
408	0.12	0.27	0.16	11
409	0.08	0.19	0.11	27
410	0.10	0.55	0.17	11
411	0.13	0.23	0.17	53
412	0.29	0.32	0.31	31
413	0.20	0.34	0.25	29
414	0.05	0.15	0.08	27
415	0.09	0.23	0.13	30
416	0.05	0.13	0.07	31
417	0.14	0.30	0.19	10
418	0.02	0.09	0.04	23
419	0.19	0.50	0.27	6
420	0.28	0.41	0.33	22
421	0.00	0.00	0.00	1
422	0.04	0.07	0.05	59

423	0.02	0.08	0.04	38
424	0.14	0.07	0.09	76
425	0.10	0.16	0.12	19
426	0.02	0.13	0.03	15
427	0.42	0.77	0.54	48
428	0.17	0.50	0.25	28
429	0.24	0.45	0.31	40
430	0.20	0.31	0.25	29
431	0.00	0.00	0.00	43
432	0.18	0.26	0.21	19
433	0.01	0.03	0.02	34
434	0.00	0.00	0.00	0
435	0.00	0.00	0.00	2
436	0.10	0.15	0.12	40
437	0.14	0.37	0.20	38
438	0.27	0.58	0.37	26
439	0.03	0.17	0.05	36
440	0.10	0.19	0.13	27
441	0.07	0.47	0.13	19
442	0.27	0.57	0.37	21
443	0.13	0.14	0.14	35
444	0.07	0.17	0.09	18
445	0.14	0.44	0.22	25
446	0.62	0.59	0.60	49
447	0.15	0.15	0.15	71
448	0.05	0.21	0.08	19
449	0.26	0.25	0.26	55
450	0.06	0.10	0.07	52
451	0.01	0.08	0.02	25
452	0.22	0.35	0.27	40
453	0.01	0.14	0.03	14
454	0.14	0.33	0.19	15
455	0.02	0.06	0.03	18
456	0.04	0.33	0.07	6
457	0.05	0.14	0.07	22
458	0.03	0.11	0.04	18
459	0.30	0.55	0.39	29
460	0.02	0.04	0.02	24
461	0.11	0.36	0.16	14
462	0.08	0.19	0.11	26
463	0.12	0.27	0.16	22
464	0.38	0.50	0.43	40
465	0.11	0.17	0.13	41
466	0.15	0.21	0.17	42
467	0.24	0.35	0.29	51
468	0.11	0.19	0.14	37
469	0.09	0.40	0.15	5
470	0.07	0.32	0.11	19
471	0.31	0.51	0.39	43
472	0.05	0.09	0.07	55
473	0.17	0.62	0.27	29

```
        474       0.47      0.75      0.58        24
        475       0.49      0.72      0.59        68
        476       0.14      0.21      0.16        38
        477       0.19      0.50      0.28        22
        478       0.11      0.15      0.12        53
        479       0.04      0.08      0.06        26
        480       0.04      0.14      0.06        64
        481       0.06      0.19      0.09        26
        482       0.10      0.57      0.17         7
        483       0.05      0.08      0.06        13
        484       0.27      0.48      0.34        23
        485       0.26      0.31      0.29        29
        486       0.22      0.35      0.27        23
        487       0.15      0.16      0.16        31
        488       0.14      0.33      0.19        30
        489       0.21      0.31      0.25        36
        490       0.06      0.19      0.10        16
        491       0.01      0.03      0.01        39
        492       0.05      0.27      0.08        11
        493       0.22      0.48      0.30        25
        494       0.02      0.07      0.03        15
        495       0.11      0.67      0.19         9
        496       0.06      0.16      0.08        19
        497       0.12      0.11      0.12        72
        498       0.10      0.37      0.15        19
        499       0.16      0.34      0.22        32

avg / total       0.38      0.45      0.40     60294

Time taken to run this cell : 0:02:50.620132
```

Conclusion & Observation

In [102]:

```
# http://zetcode.com/python/prettytable/
from prettytable import PrettyTable
#If you get a ModuleNotFoundError error , install prettytable using: pip3 install prettytable
x = PrettyTable()
x.field_names = ["Model", "Vectorizer", "F1-Micro", "F1_Macro"]
x.add_row(["Logistic Regression", "TFIDF", 0.477, 0.317])
x.add_row(["Logistic Regression", "BOW", 0.485, 0.338])
x.add_row(["Logistic Regression(Hyperparameter Tuned)", "BOW", 0.463, 0.290])
x.add_row(["Linear SVM", "BOW", 0.363, 0.230])
print(x)
```

```
+-------------------------------------------+------------+----------+----------+
|                   Model                   | Vectorizer | F1-Micro | F1_Macro |
+-------------------------------------------+------------+----------+----------+
|            Logistic Regression            |   TFIDF    |  0.477   |  0.317   |
|            Logistic Regression            |    BOW     |  0.485   |  0.338   |
| Logistic Regression(Hyperparameter Tuned) |    BOW     |  0.463   |   0.29   |
|                 Linear SVM                |    BOW     |  0.363   |   0.23   |
+-------------------------------------------+------------+----------+----------+
```

1. We reduced the number of Tags to 500 to preserve 90% variance of the total data.

2. For Logisic Regression with TFIDF vectorizer with upto 4 grams we were able to achieve a s F!-Micro score of 0.477 & a F1-Macro score of 0.317.

3. But we got the best score for Logistic Regression with BoW having F1-Micro score of 0.485 & a F1-Macro score of 0.338.

4. By using Linear svm we got a F1 Micro & Macro score of 0.363 & 0.23 respectively

In [ ]: