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 pickle
import os
from sqlalchemy import create_engine # database connection
import datetime as dt
import nltk
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 sklearn.model_selection import GridSearchCV
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 lower case, 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 variable
s";\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?
```

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 [2]:
```
#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 [2]:
```
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 : 0:01:48.245190
```

3.1.3 Checking for duplicates

In [ ]:
```
#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_d
up 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 ge
narate train.db file")
```

In [0]:
```
df_no_dup.head()
# we can observe that there are duplicates
```

Out[0]:

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

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

```
number of duplicate questions : 1827881 ( 30.2920389063 % )
```

In [0]:
```
# number of times each question appeared in our database
df_no_dup.cnt_dup.value_counts()
```

Out[0]:
```
1    2656284
2    1272336
3     277575
4         90
5         25
6          5
Name: cnt_dup, dtype: int64
```

In [0]:
```
start = datetime.now()
df_no_dup["tag_count"] = df_no_dup["Tags"].apply(lambda text: len(text.split("
")))
# adding a new feature number of tags per question
print("Time taken to run this cell :", datetime.now() - start)
df_no_dup.head()
```

```
Time taken to run this cell : 0:00:03.169523
```

Out[0]:

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

In [0]:
```
# distribution of number of tags per question
df_no_dup.tag_count.value_counts()
```

Out[0]:
```
3    1206157
2    1111706
4     814996
1     568298
5     505158
Name: tag_count, dtype: int64
```

In [3]:
```
#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 [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_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 : 0:00:44.371760
```

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 string
s.
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-store']
```

3.2.3 Number of times a tag appeared

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

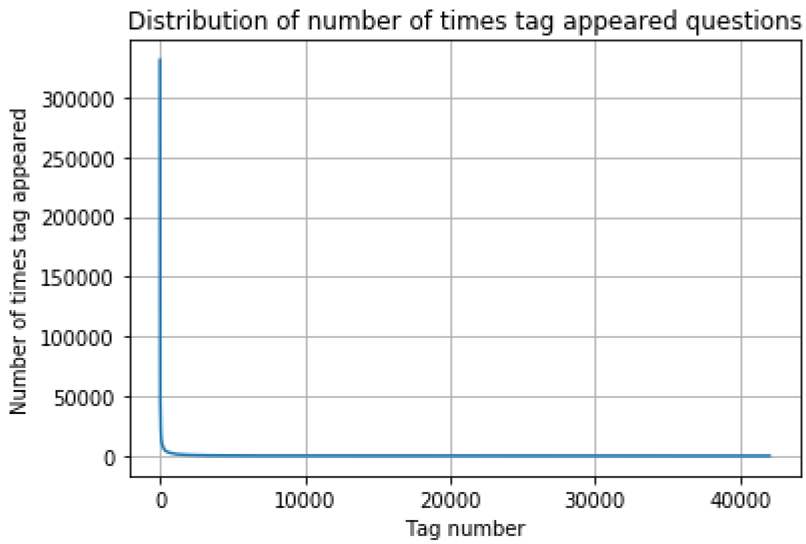
In [9]:
```
#Saving this dictionary to csv files.
if not os.path.isfile('tag_counts_dict_dtm.csv'):
    with open('tag_counts_dict_dtm.csv', 'w') as csv_file:
        writer = csv.writer(csv_file)
        for key, value in result.items():
            writer.writerow([key, value])
tag_df = pd.read_csv("tag_counts_dict_dtm.csv", names=['Tags', 'Counts'])
tag_df.head()
```

Out[9]:

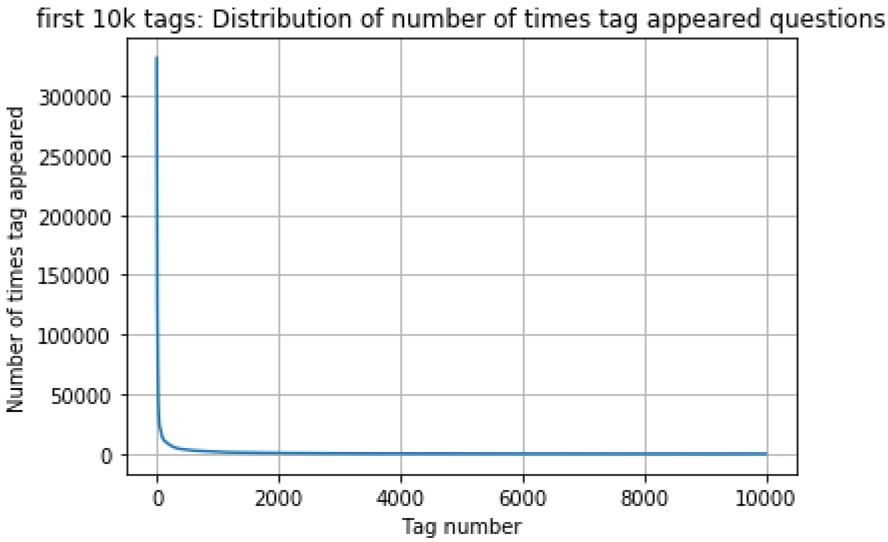
	Tags	Counts
0	.a	18
1	.app	37
2	.asp.net-mvc	1
3	.aspxauth	21
4	.bash-profile	138

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

In [11]:
```
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()
```

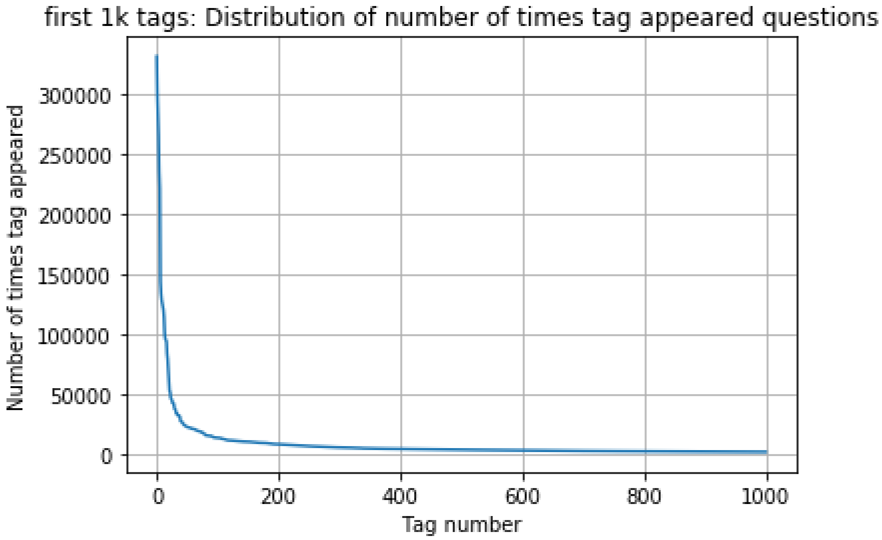


In [12]:
```
plt.plot(tag_counts[0:10000])
plt.title('first 10k tags: Distribution of number of times tag appeared questi
ons')
plt.grid()
plt.xlabel("Tag number")
plt.ylabel("Number of times tag appeared")
plt.show()
print(len(tag_counts[0:10000:25]), tag_counts[0:10000:25])
```


first 10k tags: Distribution of number of times tag appeared questions
Number of times tag appeared
300000
250000
200000
150000
100000
50000
0
0
2000
4000
6000
8000
10000
Tag number

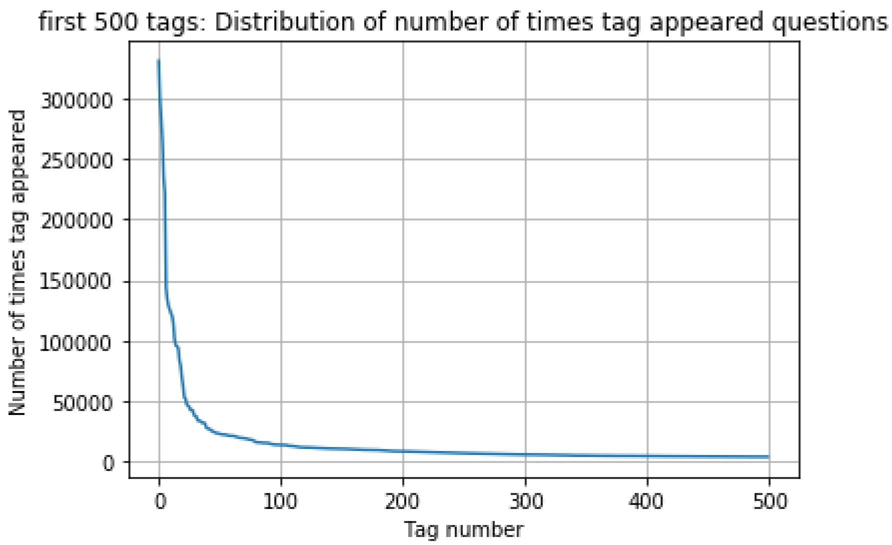
```
400 [331505  44829  22429  17728  13364  11162  10029   9148   8054   7151
   6466   5865   5370   4983   4526   4281   4144   3929   3750   3593
   3453   3299   3123   2989   2891   2738   2647   2527   2431   2331
   2259   2186   2097   2020   1959   1900   1828   1770   1723   1673
   1631   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 [13]:
```
plt.plot(tag_counts[0:1000])
plt.title('first 1k tags: Distribution of number of times tag appeared questio
ns')
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])
```



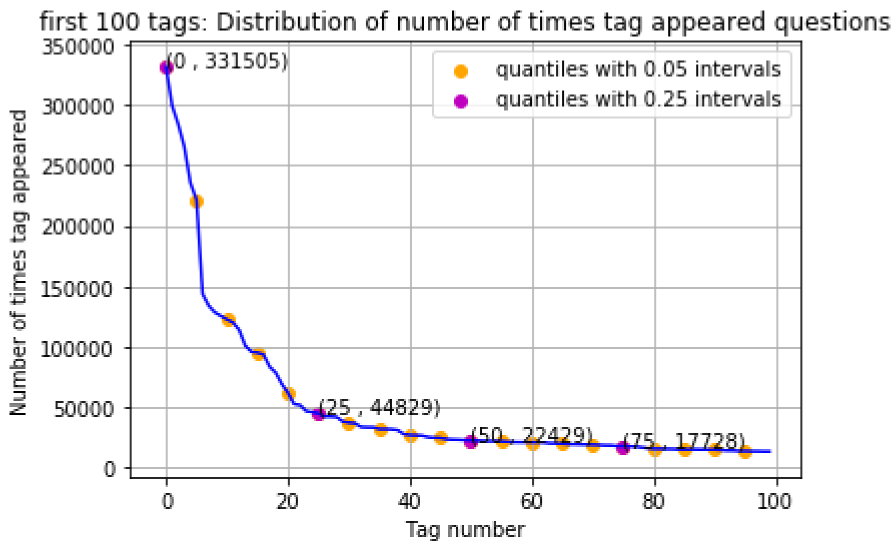

```
200 [331505 221533 122769  95160  62023  44829  37170  31897  26925  24537
  22429  21820  20957  19758  18905  17728  15533  15097  14884  13703
  13364  13157  12407  11658  11228  11162  10863  10600  10350  10224
  10029   9884   9719   9411   9252   9148   9040   8617   8361   8163
   8054   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 [14]:
```
plt.plot(tag_counts[0:500])
plt.title('first 500 tags: Distribution of number of times tag appeared questi
ons')
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 [15]:
```
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 = "q
uantiles 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 questi
ons')
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 [16]:
```
# 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

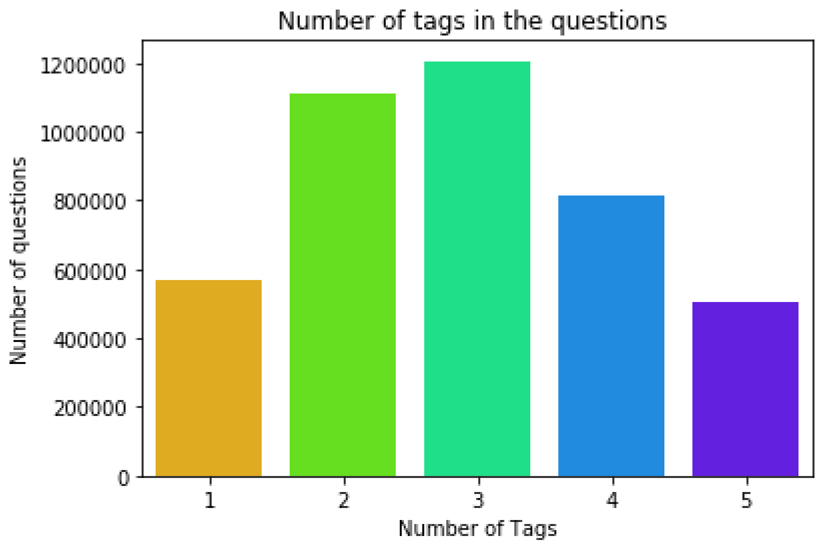
In [17]:
```
#Storing the count of tag in each question in list 'tag_count'
tag_quest_count = tag_dtm.sum(axis=1).tolist()
#Converting list of lists into single list, we will get [[3], [4], [2], [2],
 [3]] and we are converting this to [3, 4, 2, 2, 3]
tag_quest_count=[int(j) for i in tag_quest_count for j in i]
print ('We have total {} datapoints.'.format(len(tag_quest_count)))

print(tag_quest_count[:5])
```

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

In [18]:
```
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 [19]:
```
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()
```



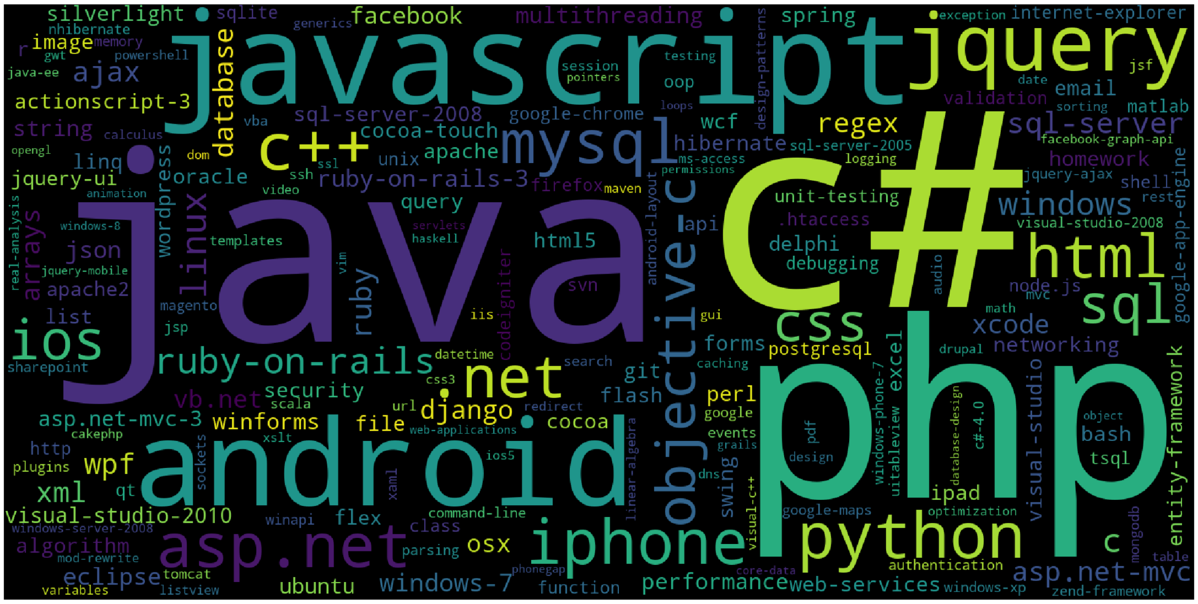
Observations:

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 [20]:

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

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

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

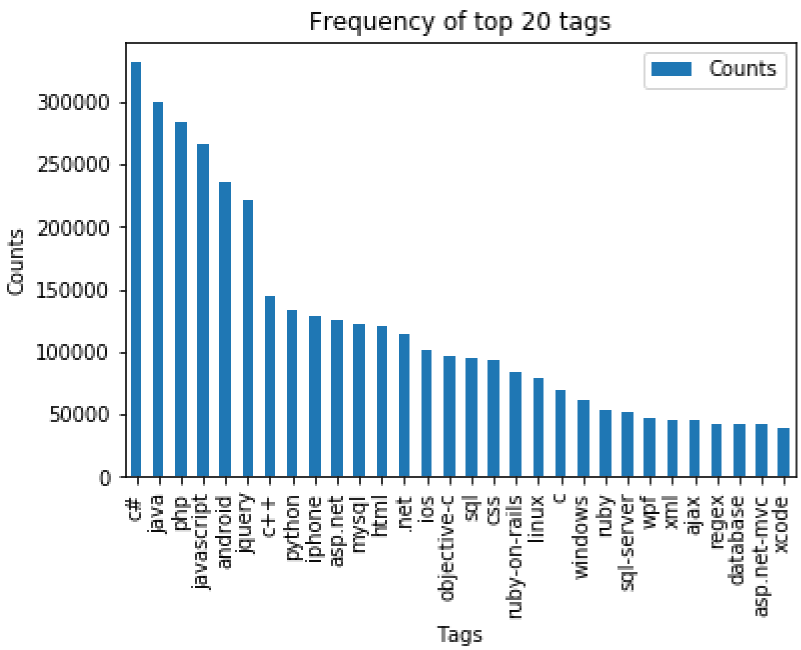



```
Time taken to run this cell : 0:00:04.827276
```

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]:
```
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 500k 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 [2]:
```
def striphtml(data):
    cleanr = re.compile('<.*?>')
    cleantext = re.sub(cleanr, ' ', str(data))
    return cleantext
stop_words = set(stopwords.words('english'))
stemmer = SnowballStemmer("english")
```

In [3]:
```
#http://www.sqlitetutorial.net/sqlite-python/create-tables/
def create_connection(db_file):
    """ create a database connection to the SQLite database
        specified by db_file
    :param db_file: database file
    :return: Connection object or None
    """
    try:
        conn = sqlite3.connect(db_file)
        return conn
    except Error as e:
        print(e)

    return None

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

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

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

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

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

4. Machine Learning Models

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

In [4]:
```
sql_create_table = """CREATE TABLE IF NOT EXISTS QuestionsProcessed (question
 text NOT NULL, code text, tags text, words_pre integer, words_post integer, i
s_code integer);"""
create_database_table("Titlemoreweight.db", sql_create_table)
```

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

In [5]:
```
# 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 10000
1;")
        # for selecting random points
        #reader.execute("SELECT Title, Body, Tags From no_dup_train ORDER BY R
ANDOM() 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 database:
QuestionsProcessed
Cleared All the rows
```

4.5.1 Preprocessing of questions

1. Separate Code from Body
2. Remove Spcial characters from Question title and description (not in code)
3. **Give more weightage to title : Add title three times to the question**
4. Remove stop words (Except 'C')
5. Remove HTML Tags
6. Convert all the characters into small letters
7. Use SnowballStemmer to stem the words

In [6]:
```
#http://www.bernzilla.com/2008/05/13/selecting-a-random-row-from-an-sqlite-tab
le/
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+" "+s
tr(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 th
e letter 'c'
    question=' '.join(str(stemmer.stem(j)) for j in words if j not in stop_wor
ds 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_pr
e,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 : 0:04:00.585418
```

In [7]:

```
# never forget to close the conections or else we will end up with database lo
cks
conn_r.commit()
conn_w.commit()
conn_r.close()
conn_w.close()
```

Sample quesitons after preprocessing of data

In [8]:
```
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 5")
        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 datag
rid bind silverlight bind datagrid dynam code wrote code debug code block see
m bind correct grid come column form come grid column although necessari bind
nthank repli advance..',)
----------------------------------------------------------------------------
-----------------------
('java.lang.noclassdeffounderror javax servlet jsp tagext taglibraryvalid jav
a.lang.noclassdeffounderror javax servlet jsp tagext taglibraryvalid java.lan
g.noclassdeffounderror javax servlet jsp tagext taglibraryvalid follow guid l
ink instal jstl got follow error tri launch jsp page java.lang.noclassdeffoun
derror javax servlet jsp tagext taglibraryvalid taglib declar instal jstl 1.1
tomcat webapp tri project work also tri version 1.2 jstl still messag caus so
lv',)
----------------------------------------------------------------------------
-----------------------
('java.sql.sqlexcept microsoft odbc driver manag invalid descriptor index jav
a.sql.sqlexcept microsoft odbc driver manag invalid descriptor index java.sq
l.sqlexcept microsoft odbc driver manag invalid descriptor index use follow c
ode display caus solv',)
----------------------------------------------------------------------------
-----------------------
('better way updat feed fb php sdk better way updat feed fb php sdk better wa
y updat feed fb php sdk novic facebook api read mani tutori still confused.i
find post feed api method like correct second way use curl someth like way be
tter',)
----------------------------------------------------------------------------
-----------------------
```

Saving Preprocessed data to a Database

In [9]:
```
#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 [10]:
```
preprocessed_data.head()
```

Out[10]:

	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 [11]:
```
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

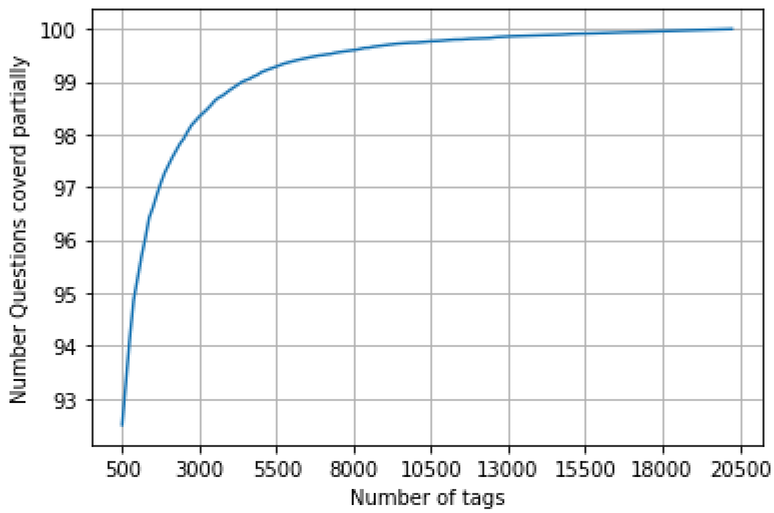
In [12]:
```
vectorizer = CountVectorizer(tokenizer = lambda x: x.split(), binary='true')
multilabel_y = vectorizer.fit_transform(preprocessed_data['tags'])
```

Selecting 500 Tags

In [13]:
```
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 [14]:
```
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 [15]:
```
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 quest
ions")
print("with ",500,"tags we are covering ",questions_explained[0],"% of questio
ns")
```




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

In [16]:
```
# 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 [17]:
```
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 [18]:
```
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)
```

In [19]:
```
x_train.to_pickle("x_train")
x_test.to_pickle("x_test")

with open("y_train.txt", "wb") as fp:   #Pickling
    pickle.dump(y_train, fp)

with open("y_test.txt", "wb") as fp:   #Pickling
    pickle.dump(y_test, fp)
```

In [2]:
```
x_train = pd.read_pickle('x_train')
x_test = pd.read_pickle('x_test')

with open("y_train.txt", "rb") as fp:   # Unpickling
    y_train = pickle.load(fp)

with open("y_test.txt", "rb") as fp:   # Unpickling
    y_test = pickle.load(fp)
```

In [20]:
```
print(x_train.shape,y_train.shape)
print(x_test.shape,y_test.shape)
```

```
(70000, 2) (70000, 500)
(30000, 2) (30000, 500)
```

4.5.2 Featurizing data with TfIdf vectorizer

In [23]:
```
vectorizer = CountVectorizer(max_features =10000,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'])
```

In [24]:
```
print("Dimensions of train data X:",x_train_multilabel.shape, "Y :",y_train.sh
ape)
print("Dimensions of test data X:",x_test_multilabel.shape,"Y:",y_test.shape)
```

```
Dimensions of train data X: (70000, 10000) Y : (70000, 500)
Dimensions of test data X: (30000, 10000) Y: (30000, 500)
```

4.5.3 Applying Logistic Regression with OneVsRest Classifier

In [25]:
```
start = datetime.now()
parameter = {"estimator__alpha" : [0.00001,0.0001,0.001,0.01,0.1,1,10,100]}
classifier = OneVsRestClassifier(SGDClassifier(loss='log', penalty='l1'))
gridsearch = GridSearchCV(classifier, parameter, scoring = 'f1_micro', verbose
=3, n_jobs=-1)
gridsearch.fit(x_train_multilabel, y_train)
print("Time taken to run this cell :", datetime.now() - start)
```

```
Fitting 3 folds for each of 8 candidates, totalling 24 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  24 out of  24 | elapsed: 40.8min finished

Time taken to run this cell : 0:47:08.836903
```

In [26]:
```
gridsearch.best_params_
```

Out[26]:
```
{'estimator__alpha': 0.001}
```

In [27]:
```
classifier = OneVsRestClassifier(SGDClassifier(loss='log',alpha=0.001 , penalt
y='l1', n_jobs=-1))
classifier.fit(x_train_multilabel, y_train)
```

Out[27]:
```
OneVsRestClassifier(estimator=SGDClassifier(alpha=0.001, 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='lo
g',
                                            max_iter=1000, n_iter_no_change=
5,
                                            n_jobs=-1, penalty='l1',
                                            power_t=0.5, random_state=None,
                                            shuffle=True, tol=0.001,
                                            validation_fraction=0.1, verbose=
0,
                                            warm_start=False),
                    n_jobs=None)
```

In [28]:
```
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 : 0.146
Hamming loss  0.0036076
Micro-average quality numbers
Precision: 0.5923, Recall: 0.3288, F1-measure: 0.4229
Macro-average quality numbers
Precision: 0.3598, Recall: 0.2032, F1-measure: 0.2412
              precision    recall  f1-score   support

           0       0.87      0.81      0.84      6668
           1       0.55      0.10      0.17      3659
           2       0.35      0.10      0.15       971
           3       0.74      0.57      0.65      1506
           4       0.67      0.44      0.53      1649
           5       0.79      0.50      0.61      1113
           6       0.72      0.33      0.45      1482
           7       0.81      0.54      0.64       980
           8       0.84      0.58      0.68      1520
           9       0.74      0.45      0.56      1041
          10       0.68      0.43      0.52       861
          11       0.60      0.40      0.48       386
          12       0.63      0.46      0.53        37
          13       0.77      0.30      0.44       917
          14       0.35      0.16      0.22       519
          15       0.67      0.42      0.51       656
          16       0.65      0.18      0.29       794
          17       0.69      0.08      0.14       700
          18       0.57      0.45      0.50       363
          19       0.85      0.59      0.70       541
          20       0.44      0.27      0.33       540
          21       0.74      0.61      0.67       362
          22       0.77      0.48      0.59       551
          23       0.59      0.26      0.36       309
          24       0.48      0.23      0.31       331
          25       0.41      0.20      0.27       424
          26       0.62      0.28      0.39       465
          27       0.27      0.09      0.14       386
          28       0.37      0.28      0.32       107
          29       0.28      0.15      0.20       195
          30       0.09      0.12      0.10       758
          31       0.88      0.47      0.61        15
          32       0.60      0.62      0.61       323
          33       0.52      0.27      0.36       279
          34       0.69      0.20      0.31       275
          35       0.77      0.51      0.61       268
          36       0.20      0.01      0.02        76
          37       0.31      0.25      0.28       269
          38       0.60      0.44      0.51       255
          39       0.18      0.16      0.17       249
          40       0.29      0.11      0.16        66
          41       0.40      0.13      0.20       209
          42       0.33      0.18      0.23        72
          43       0.00      0.00      0.00       430
          44       0.38      0.22      0.27       279
          45       0.36      0.38      0.37       240
          46       0.47      0.20      0.28       157
          47       0.73      0.56      0.63       249
          48       0.32      0.12      0.18       198
```

49	0.57	0.19	0.28	171
50	0.71	0.74	0.73	200
51	0.74	0.49	0.59	85
52	0.32	0.43	0.37	175
53	0.23	0.20	0.21	114
54	0.24	0.09	0.13	223
55	0.49	0.28	0.36	122
56	0.68	0.57	0.62	168
57	0.25	0.02	0.03	176
58	0.30	0.16	0.21	140
59	0.38	0.20	0.26	191
60	0.89	0.73	0.80	152
61	0.23	0.08	0.11	208
62	0.16	0.05	0.08	136
63	0.58	0.48	0.53	158
64	0.79	0.23	0.35	203
65	0.62	0.46	0.52	105
66	0.28	0.33	0.30	58
67	0.51	0.57	0.54	128
68	0.27	0.09	0.14	158
69	0.14	0.07	0.09	248
70	0.00	0.00	0.00	201
71	0.41	0.24	0.30	89
72	0.34	0.37	0.35	157
73	0.14	0.03	0.06	29
74	0.02	0.02	0.02	58
75	0.41	0.22	0.29	158
76	0.61	0.71	0.66	110
77	0.31	0.45	0.37	33
78	0.38	0.06	0.10	210
79	0.55	0.55	0.55	169
80	0.57	0.27	0.36	15
81	0.31	0.48	0.38	214
82	0.24	0.15	0.19	65
83	0.28	0.20	0.23	156
84	0.60	0.42	0.50	59
85	0.49	0.44	0.46	55
86	0.27	0.17	0.21	36
87	0.14	0.38	0.21	29
88	0.57	0.54	0.55	54
89	0.66	0.74	0.70	137
90	0.12	0.08	0.09	103
91	0.27	0.15	0.20	79
92	0.43	0.31	0.36	84
93	0.59	0.39	0.47	133
94	0.00	0.00	0.00	318
95	0.63	0.47	0.54	51
96	0.53	0.29	0.38	82
97	0.19	0.07	0.10	75
98	0.00	0.00	0.00	120
99	0.14	0.28	0.19	18
100	0.49	0.47	0.48	196
101	0.67	0.34	0.45	208
102	0.50	0.06	0.10	122
103	0.14	0.02	0.03	62
104	0.36	0.05	0.08	88
105	0.66	0.35	0.46	65

106	0.12	0.16	0.14	115
107	0.22	0.07	0.11	29
108	0.53	0.18	0.27	109
109	0.48	0.19	0.27	73
110	0.07	0.06	0.06	102
111	0.51	0.47	0.49	180
112	0.00	0.00	0.00	292
113	0.57	0.80	0.66	54
114	0.10	0.03	0.04	120
115	0.39	0.22	0.28	107
116	0.38	0.06	0.10	52
117	0.34	0.15	0.21	72
118	0.81	0.47	0.60	139
119	0.53	0.18	0.26	57
120	0.45	0.23	0.30	44
121	0.36	0.14	0.20	85
122	0.55	0.55	0.55	82
123	0.07	0.04	0.05	100
124	0.67	1.00	0.80	4
125	0.29	0.56	0.38	9
126	0.23	0.07	0.10	46
127	0.29	0.13	0.18	54
128	0.87	0.73	0.79	195
129	0.67	0.44	0.53	54
130	0.06	0.02	0.03	96
131	0.47	0.71	0.57	35
132	0.11	0.03	0.05	58
133	0.00	0.00	0.00	36
134	0.41	0.33	0.37	36
135	0.64	0.72	0.67	39
136	0.00	0.00	0.00	97
137	0.26	0.27	0.27	70
138	0.50	0.06	0.11	17
139	0.29	0.13	0.18	119
140	0.92	0.54	0.68	101
141	0.30	0.23	0.26	115
142	0.31	0.18	0.23	94
143	0.52	0.60	0.56	84
144	0.68	0.30	0.41	64
145	0.00	0.00	0.00	61
146	0.09	0.03	0.05	132
147	0.60	0.25	0.36	119
148	0.71	0.48	0.58	62
149	0.25	0.20	0.23	83
150	0.10	0.06	0.07	72
151	0.24	0.17	0.20	23
152	0.33	0.08	0.13	76
153	0.18	0.11	0.14	18
154	0.10	0.12	0.11	17
155	0.04	0.04	0.04	24
156	0.38	0.10	0.16	136
157	0.37	0.13	0.19	129
158	1.00	0.01	0.01	143
159	0.66	0.53	0.59	107
160	0.46	0.24	0.32	78
161	0.17	0.22	0.19	73
162	0.12	0.01	0.02	106

163	0.08	0.01	0.01	126
164	0.47	0.29	0.36	63
165	0.00	0.00	0.00	229
166	1.00	0.04	0.08	115
167	1.00	0.11	0.20	46
168	0.64	0.13	0.22	69
169	0.57	0.49	0.52	70
170	0.86	0.46	0.60	54
171	0.00	0.00	0.00	43
172	0.38	0.24	0.29	76
173	0.21	0.33	0.26	12
174	0.33	0.11	0.16	76
175	0.64	0.52	0.57	91
176	0.94	0.39	0.55	157
177	0.42	0.24	0.31	41
178	0.00	0.00	0.00	0
179	1.00	1.00	1.00	1
180	0.59	0.24	0.34	55
181	0.16	0.06	0.09	62
182	1.00	0.50	0.67	2
183	0.44	0.23	0.30	80
184	0.00	0.00	0.00	206
185	0.57	0.28	0.37	86
186	0.35	0.29	0.32	66
187	0.88	0.59	0.71	59
188	0.82	0.54	0.65	68
189	0.04	0.01	0.02	108
190	0.23	0.19	0.21	85
191	0.52	0.30	0.38	86
192	0.31	0.24	0.27	46
193	0.75	0.17	0.27	18
194	0.53	0.55	0.54	74
195	0.55	0.22	0.31	55
196	0.63	0.58	0.60	38
197	0.49	0.18	0.26	95
198	0.00	0.00	0.00	16
199	0.20	0.05	0.08	39
200	0.00	0.00	0.00	58
201	0.33	0.07	0.12	55
202	0.00	0.00	0.00	58
203	0.00	0.00	0.00	66
204	0.90	0.44	0.59	64
205	0.00	0.00	0.00	10
206	0.08	0.09	0.09	66
207	0.13	0.12	0.13	73
208	0.14	0.22	0.17	54
209	0.00	0.00	0.00	61
210	0.46	0.50	0.48	12
211	0.29	0.17	0.22	59
212	0.43	0.23	0.30	26
213	0.33	0.10	0.15	105
214	0.70	0.32	0.44	50
215	0.00	0.00	0.00	65
216	0.75	0.15	0.25	79
217	0.43	0.18	0.26	55
218	0.00	0.00	0.00	3
219	0.17	0.06	0.09	62

220	0.80	0.05	0.09	81
221	0.13	0.15	0.14	34
222	0.50	0.05	0.09	64
223	0.91	0.33	0.48	61
224	0.08	0.06	0.06	18
225	0.83	0.50	0.62	10
226	0.72	0.72	0.72	99
227	0.57	0.31	0.40	13
228	0.54	0.19	0.28	74
229	0.71	0.60	0.65	50
230	0.27	0.30	0.28	74
231	0.00	0.00	0.00	4
232	0.44	0.42	0.43	26
233	0.00	0.00	0.00	146
234	0.65	0.54	0.59	61
235	0.67	0.15	0.25	13
236	0.35	0.14	0.20	49
237	0.73	0.44	0.55	90
238	0.00	0.00	0.00	58
239	0.67	0.17	0.27	24
240	0.94	0.45	0.61	64
241	0.85	0.69	0.76	75
242	0.54	0.52	0.53	63
243	0.59	0.43	0.50	76
244	0.42	0.32	0.36	63
245	0.00	0.00	0.00	41
246	0.00	0.00	0.00	162
247	0.25	0.05	0.08	22
248	0.76	0.50	0.60	52
249	0.78	0.37	0.50	19
250	0.75	0.52	0.62	23
251	0.81	0.39	0.52	57
252	0.25	0.17	0.20	36
253	0.05	0.05	0.05	41
254	0.00	0.00	0.00	10
255	0.14	0.05	0.07	22
256	0.29	0.50	0.36	8
257	0.33	0.21	0.26	62
258	0.22	0.05	0.08	43
259	0.50	0.30	0.37	87
260	0.00	0.00	0.00	56
261	0.00	0.00	0.00	3
262	0.50	0.30	0.37	20
263	0.00	0.00	0.00	15
264	0.00	0.00	0.00	50
265	0.31	0.20	0.24	25
266	0.14	0.11	0.12	47
267	0.48	0.24	0.32	97
268	0.82	0.50	0.62	36
269	0.60	0.27	0.37	56
270	0.53	0.50	0.51	38
271	0.00	0.00	0.00	58
272	0.25	0.12	0.17	8
273	0.05	0.04	0.04	27
274	0.00	0.00	0.00	123
275	0.67	0.20	0.31	69
276	0.83	0.31	0.45	112

277	0.00	0.00	0.00	31
278	0.00	0.00	0.00	29
279	0.43	0.16	0.23	38
280	0.12	0.04	0.06	50
281	0.81	0.65	0.72	20
282	0.96	0.53	0.69	45
283	0.00	0.00	0.00	15
284	0.00	0.00	0.00	74
285	0.57	0.09	0.15	46
286	0.30	0.10	0.15	29
287	0.08	0.02	0.03	54
288	0.92	0.33	0.49	33
289	0.00	0.00	0.00	26
290	0.85	0.54	0.66	41
291	0.16	0.21	0.18	24
292	0.33	0.03	0.05	40
293	0.42	0.30	0.35	33
294	0.12	0.03	0.05	31
295	0.08	0.02	0.03	47
296	0.21	0.09	0.13	33
297	0.00	0.00	0.00	45
298	0.00	0.00	0.00	59
299	0.17	0.02	0.04	51
300	0.17	0.22	0.20	49
301	0.58	0.18	0.28	38
302	0.86	0.43	0.57	28
303	0.29	0.12	0.17	16
304	0.00	0.00	0.00	32
305	0.55	0.25	0.34	24
306	0.20	0.02	0.04	44
307	1.00	0.17	0.29	6
308	0.00	0.00	0.00	48
309	0.78	0.43	0.55	49
310	0.17	0.03	0.05	38
311	0.20	0.08	0.11	62
312	0.00	0.00	0.00	27
313	0.00	0.00	0.00	49
314	0.44	0.17	0.24	24
315	0.33	0.02	0.03	59
316	1.00	0.10	0.18	10
317	0.35	0.25	0.29	67
318	0.00	0.00	0.00	12
319	0.00	0.00	0.00	14
320	0.00	0.00	0.00	12
321	0.00	0.00	0.00	9
322	0.73	0.35	0.47	23
323	0.75	0.45	0.57	33
324	0.51	0.54	0.53	57
325	0.50	0.08	0.14	25
326	0.00	0.00	0.00	44
327	0.25	0.04	0.06	27
328	0.00	0.00	0.00	34
329	1.00	0.29	0.44	7
330	0.47	0.41	0.44	22
331	0.17	0.16	0.17	25
332	0.00	0.00	0.00	106
333	0.59	0.15	0.25	84

334	0.06	0.03	0.04	36
335	1.00	0.15	0.27	13
336	0.00	0.00	0.00	37
337	0.50	0.05	0.10	38
338	0.95	0.43	0.59	44
339	0.00	0.00	0.00	34
340	0.29	0.50	0.37	40
341	0.73	0.35	0.47	23
342	0.00	0.00	0.00	11
343	0.00	0.00	0.00	12
344	0.13	0.12	0.12	25
345	0.00	0.00	0.00	1
346	0.00	0.00	0.00	41
347	0.24	0.11	0.15	46
348	0.29	0.11	0.15	19
349	0.43	0.08	0.13	38
350	0.68	0.39	0.50	33
351	0.25	0.06	0.09	53
352	0.00	0.00	0.00	49
353	0.46	0.22	0.30	27
354	0.06	0.03	0.04	31
355	1.00	0.33	0.50	12
356	0.00	0.00	0.00	33
357	0.54	0.62	0.58	24
358	0.35	0.18	0.24	34
359	0.59	0.30	0.40	33
360	0.00	0.00	0.00	47
361	0.58	0.28	0.38	39
362	0.88	0.55	0.68	38
363	0.17	0.12	0.14	17
364	0.00	0.00	0.00	33
365	0.00	0.00	0.00	26
366	1.00	0.05	0.10	19
367	0.00	0.00	0.00	98
368	0.43	0.32	0.36	38
369	0.00	0.00	0.00	28
370	0.40	0.13	0.20	15
371	0.10	0.05	0.06	22
372	0.00	0.00	0.00	12
373	0.04	0.17	0.06	6
374	0.20	0.03	0.06	31
375	0.40	0.11	0.17	38
376	0.00	0.00	0.00	42
377	0.00	0.00	0.00	23
378	0.50	0.25	0.33	4
379	0.00	0.00	0.00	37
380	0.40	0.33	0.36	6
381	1.00	0.17	0.29	18
382	0.38	0.15	0.21	40
383	0.00	0.00	0.00	53
384	0.40	0.32	0.36	25
385	0.23	0.11	0.15	53
386	0.88	0.50	0.64	14
387	0.00	0.00	0.00	88
388	0.00	0.00	0.00	16
389	0.00	0.00	0.00	8
390	0.00	0.00	0.00	37

391	0.91	0.38	0.54	52
392	0.00	0.00	0.00	17
393	0.83	0.14	0.23	37
394	0.00	0.00	0.00	19
395	0.00	0.00	0.00	9
396	0.25	0.07	0.11	14
397	0.79	0.38	0.51	29
398	0.71	0.13	0.22	38
399	0.96	0.63	0.76	38
400	0.00	0.00	0.00	36
401	0.57	0.14	0.23	56
402	0.86	0.60	0.71	20
403	0.25	0.09	0.13	11
404	0.65	0.56	0.60	27
405	0.94	0.28	0.43	57
406	0.00	0.00	0.00	95
407	0.15	0.08	0.11	25
408	0.00	0.00	0.00	11
409	0.00	0.00	0.00	27
410	0.42	0.45	0.43	11
411	0.08	0.06	0.07	53
412	0.38	0.26	0.31	31
413	0.08	0.07	0.07	29
414	0.00	0.00	0.00	27
415	0.17	0.23	0.20	30
416	0.12	0.06	0.09	31
417	0.33	0.20	0.25	10
418	0.00	0.00	0.00	23
419	0.50	0.33	0.40	6
420	0.73	0.50	0.59	22
421	0.00	0.00	0.00	1
422	0.00	0.00	0.00	59
423	0.00	0.00	0.00	38
424	0.00	0.00	0.00	76
425	0.25	0.05	0.09	19
426	0.00	0.00	0.00	15
427	0.94	0.35	0.52	48
428	0.60	0.21	0.32	28
429	0.45	0.45	0.45	40
430	0.00	0.00	0.00	29
431	0.00	0.00	0.00	43
432	0.00	0.00	0.00	19
433	0.00	0.00	0.00	34
434	0.00	0.00	0.00	0
435	0.00	0.00	0.00	2
436	0.15	0.05	0.08	40
437	0.37	0.18	0.25	38
438	0.77	0.38	0.51	26
439	0.00	0.00	0.00	36
440	0.00	0.00	0.00	27
441	0.46	0.63	0.53	19
442	0.73	0.52	0.61	21
443	0.25	0.06	0.09	35
444	0.25	0.22	0.24	18
445	0.64	0.28	0.39	25
446	0.79	0.45	0.57	49
447	0.07	0.11	0.09	71

```
         448       0.00      0.00      0.00        19
         449       0.00      0.00      0.00        55
         450       0.00      0.00      0.00        52
         451       0.00      0.00      0.00        25
         452       0.90      0.45      0.60        40
         453       0.00      0.00      0.00        14
         454       0.17      0.33      0.22        15
         455       0.00      0.00      0.00        18
         456       0.00      0.00      0.00         6
         457       0.00      0.00      0.00        22
         458       0.00      0.00      0.00        18
         459       0.83      0.17      0.29        29
         460       0.00      0.00      0.00        24
         461       0.17      0.07      0.10        14
         462       0.50      0.08      0.13        26
         463       0.00      0.00      0.00        22
         464       0.19      0.23      0.20        40
         465       0.33      0.07      0.12        41
         466       0.22      0.10      0.13        42
         467       0.15      0.08      0.10        51
         468       0.44      0.11      0.17        37
         469       0.00      0.00      0.00         5
         470       0.00      0.00      0.00        19
         471       0.92      0.26      0.40        43
         472       0.05      0.04      0.04        55
         473       1.00      0.03      0.07        29
         474       1.00      0.42      0.59        24
         475       0.00      0.00      0.00        68
         476       0.20      0.03      0.05        38
         477       0.19      0.23      0.21        22
         478       0.00      0.00      0.00        53
         479       0.00      0.00      0.00        26
         480       0.00      0.00      0.00        64
         481       0.00      0.00      0.00        26
         482       0.40      0.29      0.33         7
         483       0.29      0.15      0.20        13
         484       0.67      0.09      0.15        23
         485       0.62      0.28      0.38        29
         486       0.38      0.13      0.19        23
         487       0.00      0.00      0.00        31
         488       0.00      0.00      0.00        30
         489       0.24      0.14      0.18        36
         490       0.00      0.00      0.00        16
         491       0.00      0.00      0.00        39
         492       0.07      0.09      0.08        11
         493       0.20      0.12      0.15        25
         494       0.00      0.00      0.00        15
         495       1.00      0.22      0.36         9
         496       0.35      0.42      0.38        19
         497       0.00      0.00      0.00        72
         498       0.33      0.16      0.21        19
         499       0.00      0.00      0.00        32

   micro avg       0.59      0.33      0.42     60294
   macro avg       0.36      0.20      0.24     60294
weighted avg       0.53      0.33      0.38     60294
 samples avg       0.44      0.33      0.35     60294
```

```
Time taken to run this cell : 0:55:40.746100
```

In [29]:
```
start = datetime.now()
parameter = {"estimator__alpha" : [0.00001,0.0001,0.001,0.01,0.1,1,10,100]}
sgdclassifier = OneVsRestClassifier(SGDClassifier(loss='hinge'))
gridsearch = GridSearchCV(sgdclassifier, parameter, scoring = 'f1_micro', verb
ose=3, n_jobs=-1)
gridsearch.fit(x_train_multilabel, y_train)
gridsearch.best_params_
```

```
Fitting 3 folds for each of 8 candidates, totalling 24 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  24 out of  24 | elapsed: 13.3min finished
```

Out[29]:
```
{'estimator__alpha': 0.001}
```

In [30]:
```
sgdclassifier = OneVsRestClassifier(SGDClassifier(loss='hinge',alpha=0.001, n_
jobs=-1))
sgdclassifier.fit(x_train_multilabel, y_train)
```

Out[30]:
```
OneVsRestClassifier(estimator=SGDClassifier(alpha=0.001, 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=1000,
                                            n_iter_no_change=5, n_jobs=-1,
                                            penalty='l2', power_t=0.5,
                                            random_state=None, shuffle=True,
                                            tol=0.001, validation_fraction=0.
1,
                                            verbose=0, warm_start=False),
                    n_jobs=None)
```

In [31]:
```
predictions = sgdclassifier.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 : 0.19733333333333333
Hamming loss  0.0030884666666666665
Micro-average quality numbers
Precision: 0.7475, Recall: 0.3499, F1-measure: 0.4766
Macro-average quality numbers
Precision: 0.5022, Recall: 0.2283, F1-measure: 0.2887
              precision    recall  f1-score   support

           0       0.88      0.84      0.86      6668
           1       0.63      0.14      0.23      3659
           2       0.48      0.09      0.15       971
           3       0.75      0.63      0.68      1506
           4       0.78      0.47      0.58      1649
           5       0.86      0.50      0.63      1113
           6       0.77      0.37      0.50      1482
           7       0.81      0.58      0.68       980
           8       0.90      0.57      0.70      1520
           9       0.73      0.67      0.70      1041
          10       0.79      0.48      0.60       861
          11       0.61      0.38      0.47       386
          12       0.70      0.43      0.53        37
          13       0.81      0.31      0.45       917
          14       0.50      0.16      0.25       519
          15       0.75      0.39      0.52       656
          16       0.74      0.19      0.30       794
          17       0.66      0.15      0.24       700
          18       0.80      0.60      0.68       363
          19       0.91      0.55      0.68       541
          20       0.54      0.21      0.30       540
          21       0.85      0.64      0.73       362
          22       0.83      0.40      0.54       551
          23       0.65      0.29      0.41       309
          24       0.71      0.28      0.40       331
          25       0.57      0.21      0.31       424
          26       0.75      0.22      0.34       465
          27       0.17      0.00      0.01       386
          28       0.52      0.22      0.31       107
          29       0.62      0.18      0.28       195
          30       0.65      0.18      0.28       758
          31       0.75      0.40      0.52        15
          32       0.73      0.62      0.67       323
          33       0.69      0.18      0.28       279
          34       0.90      0.25      0.40       275
          35       0.80      0.63      0.70       268
          36       0.50      0.03      0.05        76
          37       0.50      0.06      0.11       269
          38       0.73      0.49      0.59       255
          39       0.46      0.16      0.24       249
          40       0.23      0.05      0.08        66
          41       0.61      0.08      0.14       209
          42       0.49      0.28      0.35        72
          43       0.31      0.03      0.06       430
          44       0.53      0.19      0.28       279
          45       0.56      0.19      0.29       240
          46       0.73      0.26      0.38       157
          47       0.90      0.40      0.56       249
          48       0.68      0.14      0.23       198
```

49	0.60	0.31	0.41	171
50	0.88	0.65	0.75	200
51	0.83	0.73	0.77	85
52	0.66	0.41	0.50	175
53	0.28	0.21	0.24	114
54	0.42	0.07	0.12	223
55	0.65	0.27	0.38	122
56	0.67	0.61	0.64	168
57	0.00	0.00	0.00	176
58	0.49	0.13	0.20	140
59	0.52	0.06	0.11	191
60	0.93	0.68	0.78	152
61	0.25	0.01	0.02	208
62	0.31	0.07	0.12	136
63	0.79	0.37	0.50	158
64	0.75	0.27	0.40	203
65	0.67	0.41	0.51	105
66	0.58	0.38	0.46	58
67	0.62	0.49	0.55	128
68	0.12	0.01	0.01	158
69	0.00	0.00	0.00	248
70	0.66	0.21	0.32	201
71	0.53	0.43	0.47	89
72	0.50	0.10	0.17	157
73	0.20	0.07	0.10	29
74	0.00	0.00	0.00	58
75	0.64	0.19	0.29	158
76	0.81	0.60	0.69	110
77	0.70	0.42	0.53	33
78	0.57	0.02	0.04	210
79	0.63	0.72	0.67	169
80	0.62	0.33	0.43	15
81	0.51	0.27	0.35	214
82	0.38	0.15	0.22	65
83	0.36	0.08	0.13	156
84	0.68	0.58	0.62	59
85	0.67	0.58	0.62	55
86	0.37	0.19	0.25	36
87	0.65	0.38	0.48	29
88	0.86	0.56	0.67	54
89	0.85	0.70	0.77	137
90	1.00	0.02	0.04	103
91	0.20	0.01	0.02	79
92	0.89	0.29	0.43	84
93	0.68	0.49	0.57	133
94	0.72	0.42	0.53	318
95	0.81	0.49	0.61	51
96	0.69	0.41	0.52	82
97	0.67	0.03	0.05	75
98	0.00	0.00	0.00	120
99	0.82	0.50	0.62	18
100	0.64	0.44	0.53	196
101	0.72	0.41	0.53	208
102	0.50	0.02	0.03	122
103	0.00	0.00	0.00	62
104	1.00	0.01	0.02	88
105	0.86	0.37	0.52	65

106	0.33	0.13	0.19	115
107	0.80	0.14	0.24	29
108	0.75	0.11	0.19	109
109	0.62	0.32	0.42	73
110	0.82	0.14	0.24	102
111	0.53	0.28	0.36	180
112	0.00	0.00	0.00	292
113	0.93	0.80	0.86	54
114	0.00	0.00	0.00	120
115	0.58	0.07	0.12	107
116	0.82	0.17	0.29	52
117	0.33	0.06	0.10	72
118	0.77	0.46	0.58	139
119	0.78	0.32	0.45	57
120	0.91	0.23	0.36	44
121	0.48	0.12	0.19	85
122	0.69	0.63	0.66	82
123	0.00	0.00	0.00	100
124	0.80	1.00	0.89	4
125	1.00	0.56	0.71	9
126	0.32	0.13	0.18	46
127	0.50	0.02	0.04	54
128	0.89	0.67	0.77	195
129	0.68	0.52	0.59	54
130	0.00	0.00	0.00	96
131	0.66	0.60	0.63	35
132	0.00	0.00	0.00	58
133	0.33	0.03	0.05	36
134	0.68	0.36	0.47	36
135	0.92	0.56	0.70	39
136	0.00	0.00	0.00	97
137	0.35	0.50	0.41	70
138	0.43	0.18	0.25	17
139	0.37	0.08	0.14	119
140	0.91	0.50	0.65	101
141	0.42	0.28	0.34	115
142	0.68	0.18	0.29	94
143	0.60	0.52	0.56	84
144	0.68	0.39	0.50	64
145	0.20	0.05	0.08	61
146	1.00	0.01	0.02	132
147	0.58	0.26	0.36	119
148	0.83	0.55	0.66	62
149	0.48	0.19	0.28	83
150	0.31	0.19	0.24	72
151	0.47	0.39	0.43	23
152	0.53	0.11	0.18	76
153	0.33	0.33	0.33	18
154	1.00	0.06	0.11	17
155	0.80	0.17	0.28	24
156	0.50	0.16	0.24	136
157	0.55	0.36	0.43	129
158	0.55	0.21	0.30	143
159	0.69	0.48	0.56	107
160	0.59	0.31	0.40	78
161	0.59	0.14	0.22	73
162	0.00	0.00	0.00	106

163	0.00	0.00	0.00	126
164	0.63	0.35	0.45	63
165	0.00	0.00	0.00	229
166	0.85	0.10	0.17	115
167	0.73	0.24	0.36	46
168	0.65	0.22	0.33	69
169	0.66	0.44	0.53	70
170	0.88	0.43	0.57	54
171	0.00	0.00	0.00	43
172	0.57	0.41	0.48	76
173	0.40	0.33	0.36	12
174	0.25	0.01	0.03	76
175	0.68	0.56	0.61	91
176	0.84	0.48	0.61	157
177	0.65	0.32	0.43	41
178	0.00	0.00	0.00	0
179	1.00	1.00	1.00	1
180	0.70	0.35	0.46	55
181	0.20	0.02	0.03	62
182	0.00	0.00	0.00	2
183	0.65	0.33	0.43	80
184	0.00	0.00	0.00	206
185	0.62	0.19	0.29	86
186	0.44	0.27	0.34	66
187	0.94	0.56	0.70	59
188	0.81	0.62	0.70	68
189	0.80	0.07	0.14	108
190	0.44	0.09	0.16	85
191	0.85	0.26	0.39	86
192	0.72	0.39	0.51	46
193	0.62	0.28	0.38	18
194	0.75	0.28	0.41	74
195	0.75	0.16	0.27	55
196	0.81	0.58	0.68	38
197	0.67	0.23	0.34	95
198	0.00	0.00	0.00	16
199	0.20	0.03	0.05	39
200	1.00	0.02	0.03	58
201	0.36	0.16	0.22	55
202	0.55	0.10	0.17	58
203	0.00	0.00	0.00	66
204	0.98	0.64	0.77	64
205	0.00	0.00	0.00	10
206	0.00	0.00	0.00	66
207	0.50	0.16	0.25	73
208	0.25	0.02	0.03	54
209	0.00	0.00	0.00	61
210	0.28	0.42	0.33	12
211	0.38	0.10	0.16	59
212	0.65	0.42	0.51	26
213	0.27	0.06	0.09	105
214	0.62	0.40	0.49	50
215	0.73	0.12	0.21	65
216	0.70	0.35	0.47	79
217	0.42	0.20	0.27	55
218	0.00	0.00	0.00	3
219	0.33	0.05	0.08	62

220	0.83	0.06	0.11	81
221	0.75	0.09	0.16	34
222	0.00	0.00	0.00	64
223	1.00	0.31	0.47	61
224	0.67	0.11	0.19	18
225	0.86	0.60	0.71	10
226	0.73	0.84	0.78	99
227	0.86	0.46	0.60	13
228	0.44	0.09	0.16	74
229	0.81	0.68	0.74	50
230	0.31	0.07	0.11	74
231	0.00	0.00	0.00	4
232	0.30	0.12	0.17	26
233	0.40	0.01	0.03	146
234	0.88	0.48	0.62	61
235	1.00	0.08	0.14	13
236	0.83	0.10	0.18	49
237	0.74	0.48	0.58	90
238	0.00	0.00	0.00	58
239	0.83	0.21	0.33	24
240	0.94	0.50	0.65	64
241	0.87	0.64	0.74	75
242	0.55	0.59	0.57	63
243	0.81	0.50	0.62	76
244	0.54	0.41	0.47	63
245	1.00	0.02	0.05	41
246	0.00	0.00	0.00	162
247	1.00	0.09	0.17	22
248	0.86	0.60	0.70	52
249	0.82	0.47	0.60	19
250	0.61	0.48	0.54	23
251	0.76	0.33	0.46	57
252	1.00	0.06	0.11	36
253	0.00	0.00	0.00	41
254	0.00	0.00	0.00	10
255	0.25	0.05	0.08	22
256	0.57	0.50	0.53	8
257	0.79	0.18	0.29	62
258	0.71	0.12	0.20	43
259	0.70	0.43	0.53	87
260	0.00	0.00	0.00	56
261	0.00	0.00	0.00	3
262	0.64	0.35	0.45	20
263	0.00	0.00	0.00	15
264	0.00	0.00	0.00	50
265	0.47	0.28	0.35	25
266	0.25	0.04	0.07	47
267	0.57	0.53	0.55	97
268	0.87	0.72	0.79	36
269	0.65	0.39	0.49	56
270	0.62	0.63	0.62	38
271	0.00	0.00	0.00	58
272	0.25	0.12	0.17	8
273	0.00	0.00	0.00	27
274	0.00	0.00	0.00	123
275	0.58	0.32	0.41	69
276	0.91	0.53	0.67	112

277	0.50	0.06	0.11	31
278	0.00	0.00	0.00	29
279	0.89	0.21	0.34	38
280	0.75	0.18	0.29	50
281	0.93	0.70	0.80	20
282	1.00	0.64	0.78	45
283	0.67	0.13	0.22	15
284	0.58	0.20	0.30	74
285	0.69	0.20	0.31	46
286	0.00	0.00	0.00	29
287	0.00	0.00	0.00	54
288	0.90	0.55	0.68	33
289	0.00	0.00	0.00	26
290	0.85	0.56	0.68	41
291	0.30	0.12	0.18	24
292	0.75	0.07	0.14	40
293	0.52	0.45	0.48	33
294	0.14	0.03	0.05	31
295	0.00	0.00	0.00	47
296	1.00	0.03	0.06	33
297	0.62	0.11	0.19	45
298	0.00	0.00	0.00	59
299	0.00	0.00	0.00	51
300	0.53	0.16	0.25	49
301	0.78	0.18	0.30	38
302	0.86	0.43	0.57	28
303	0.33	0.19	0.24	16
304	0.40	0.06	0.11	32
305	0.67	0.17	0.27	24
306	0.60	0.07	0.12	44
307	0.50	0.17	0.25	6
308	0.00	0.00	0.00	48
309	0.61	0.45	0.52	49
310	0.00	0.00	0.00	38
311	0.00	0.00	0.00	62
312	0.00	0.00	0.00	27
313	0.00	0.00	0.00	49
314	0.42	0.21	0.28	24
315	0.00	0.00	0.00	59
316	1.00	0.10	0.18	10
317	0.38	0.22	0.28	67
318	0.75	0.25	0.38	12
319	0.00	0.00	0.00	14
320	0.50	0.08	0.14	12
321	0.44	0.44	0.44	9
322	0.82	0.39	0.53	23
323	0.95	0.55	0.69	33
324	0.66	0.54	0.60	57
325	0.00	0.00	0.00	25
326	0.17	0.02	0.04	44
327	0.33	0.07	0.12	27
328	1.00	0.09	0.16	34
329	0.50	0.14	0.22	7
330	0.64	0.41	0.50	22
331	0.20	0.04	0.07	25
332	0.00	0.00	0.00	106
333	0.66	0.23	0.34	84

334	0.00	0.00	0.00	36
335	0.75	0.23	0.35	13
336	0.00	0.00	0.00	37
337	0.67	0.05	0.10	38
338	0.95	0.48	0.64	44
339	0.00	0.00	0.00	34
340	0.39	0.28	0.32	40
341	0.92	0.52	0.67	23
342	0.25	0.09	0.13	11
343	0.83	0.42	0.56	12
344	0.38	0.20	0.26	25
345	0.00	0.00	0.00	1
346	0.71	0.12	0.21	41
347	0.38	0.07	0.11	46
348	0.00	0.00	0.00	19
349	0.93	0.34	0.50	38
350	0.61	0.33	0.43	33
351	0.00	0.00	0.00	53
352	0.00	0.00	0.00	49
353	0.60	0.22	0.32	27
354	0.00	0.00	0.00	31
355	0.71	0.42	0.53	12
356	0.75	0.09	0.16	33
357	0.70	0.67	0.68	24
358	0.48	0.29	0.36	34
359	0.71	0.52	0.60	33
360	0.00	0.00	0.00	47
361	0.62	0.33	0.43	39
362	0.85	0.58	0.69	38
363	0.29	0.29	0.29	17
364	0.89	0.24	0.38	33
365	0.00	0.00	0.00	26
366	0.33	0.05	0.09	19
367	0.17	0.02	0.04	98
368	0.57	0.45	0.50	38
369	1.00	0.39	0.56	28
370	0.00	0.00	0.00	15
371	0.50	0.05	0.08	22
372	0.00	0.00	0.00	12
373	0.40	0.33	0.36	6
374	0.00	0.00	0.00	31
375	0.36	0.11	0.16	38
376	0.00	0.00	0.00	42
377	0.00	0.00	0.00	23
378	0.20	0.25	0.22	4
379	0.00	0.00	0.00	37
380	0.25	0.33	0.29	6
381	1.00	0.39	0.56	18
382	0.64	0.17	0.27	40
383	0.00	0.00	0.00	53
384	0.50	0.28	0.36	25
385	0.33	0.06	0.10	53
386	0.90	0.64	0.75	14
387	0.42	0.22	0.29	88
388	0.00	0.00	0.00	16
389	0.00	0.00	0.00	8
390	0.67	0.05	0.10	37

391	0.97	0.54	0.69	52
392	0.00	0.00	0.00	17
393	0.58	0.19	0.29	37
394	0.00	0.00	0.00	19
395	0.00	0.00	0.00	9
396	0.00	0.00	0.00	14
397	0.89	0.59	0.71	29
398	0.50	0.32	0.39	38
399	0.96	0.68	0.80	38
400	0.50	0.06	0.10	36
401	0.45	0.09	0.15	56
402	0.92	0.60	0.73	20
403	0.50	0.09	0.15	11
404	0.64	0.52	0.57	27
405	0.93	0.70	0.80	57
406	0.00	0.00	0.00	95
407	0.00	0.00	0.00	25
408	0.50	0.18	0.27	11
409	1.00	0.04	0.07	27
410	0.62	0.45	0.53	11
411	0.00	0.00	0.00	53
412	0.73	0.26	0.38	31
413	0.43	0.10	0.17	29
414	1.00	0.04	0.07	27
415	0.11	0.03	0.05	30
416	0.00	0.00	0.00	31
417	1.00	0.20	0.33	10
418	0.00	0.00	0.00	23
419	1.00	0.17	0.29	6
420	0.57	0.59	0.58	22
421	0.00	0.00	0.00	1
422	0.00	0.00	0.00	59
423	0.00	0.00	0.00	38
424	0.00	0.00	0.00	76
425	1.00	0.05	0.10	19
426	0.00	0.00	0.00	15
427	0.81	0.73	0.77	48
428	0.44	0.14	0.22	28
429	0.54	0.33	0.41	40
430	0.88	0.24	0.38	29
431	0.00	0.00	0.00	43
432	0.80	0.21	0.33	19
433	0.33	0.03	0.05	34
434	0.00	0.00	0.00	0
435	0.00	0.00	0.00	2
436	0.00	0.00	0.00	40
437	0.50	0.16	0.24	38
438	0.82	0.54	0.65	26
439	0.00	0.00	0.00	36
440	1.00	0.04	0.07	27
441	0.56	0.47	0.51	19
442	0.69	0.52	0.59	21
443	0.50	0.06	0.10	35
444	0.00	0.00	0.00	18
445	0.79	0.44	0.56	25
446	0.82	0.57	0.67	49
447	0.00	0.00	0.00	71

448	0.20	0.05	0.08	19
449	0.83	0.09	0.16	55
450	0.00	0.00	0.00	52
451	0.00	0.00	0.00	25
452	1.00	0.42	0.60	40
453	0.00	0.00	0.00	14
454	0.50	0.13	0.21	15
455	1.00	0.06	0.11	18
456	0.00	0.00	0.00	6
457	0.00	0.00	0.00	22
458	0.00	0.00	0.00	18
459	0.92	0.38	0.54	29
460	0.00	0.00	0.00	24
461	0.67	0.29	0.40	14
462	0.50	0.15	0.24	26
463	1.00	0.05	0.09	22
464	0.95	0.45	0.61	40
465	0.57	0.10	0.17	41
466	0.43	0.07	0.12	42
467	0.60	0.06	0.11	51
468	0.00	0.00	0.00	37
469	0.00	0.00	0.00	5
470	0.50	0.11	0.17	19
471	0.79	0.26	0.39	43
472	0.00	0.00	0.00	55
473	0.47	0.24	0.32	29
474	1.00	0.62	0.77	24
475	0.61	0.68	0.64	68
476	0.25	0.11	0.15	38
477	0.53	0.36	0.43	22
478	0.20	0.02	0.03	53
479	0.33	0.04	0.07	26
480	0.00	0.00	0.00	64
481	0.33	0.08	0.12	26
482	0.75	0.43	0.55	7
483	0.00	0.00	0.00	13
484	0.75	0.39	0.51	23
485	0.80	0.14	0.24	29
486	0.60	0.26	0.36	23
487	0.75	0.19	0.31	31
488	0.57	0.13	0.22	30
489	1.00	0.03	0.05	36
490	0.33	0.06	0.11	16
491	0.00	0.00	0.00	39
492	0.18	0.18	0.18	11
493	0.35	0.28	0.31	25
494	0.50	0.07	0.12	15
495	0.83	0.56	0.67	9
496	0.00	0.00	0.00	19
497	0.00	0.00	0.00	72
498	0.62	0.26	0.37	19
499	0.00	0.00	0.00	32
micro avg	0.75	0.35	0.48	60294
macro avg	0.50	0.23	0.29	60294
weighted avg	0.63	0.35	0.42	60294
samples avg	0.51	0.36	0.39	60294

```
Time taken to run this cell : 0:17:55.810791
```

In [8]:
```
from prettytable import PrettyTable
x = PrettyTable()
columns=["Model", "Alpha", "Precision", "Recall", "Micro-F1"]
x.add_column(columns[0],["Logistic Regression","Linear-SVM"])
x.add_column(columns[1],[0.001,0.001])
x.add_column(columns[2],[0.5923,0.7475])
x.add_column(columns[3],[0.3288,0.3499])
x.add_column(columns[4],[0.4229,0.4766])
print(x)
```

```
+---------------------+-------+-----------+--------+----------+
|        Model        | Alpha | Precision | Recall | Micro-F1 |
+---------------------+-------+-----------+--------+----------+
| Logistic Regression | 0.001 |   0.5923  | 0.3288 |  0.4229  |
|      Linear-SVM     | 0.001 |   0.7475  | 0.3499 |  0.4766  |
+---------------------+-------+-----------+--------+----------+
```

Conclusion:

1.Since using 500k datapoints resulted in dead kernel repeatedly due to heavy computation, 100k datapoints were used. 2.On using both unigram and 4-grams BoW, it is observed that 4-grams has higher performance with large margin, thus this model was selected. 3.Linear-SVM has higher performance than Logistic Regression on terms of precision. 4.With better computational power, significantly better models could be developed with high amount of data.

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)