In [8]:
```
!pip install kaggle
from google.colab import files
files.upload()
```

```
Requirement already satisfied: kaggle in /usr/local/lib/python3.6/di
st-packages (1.5.3)
Requirement already satisfied: urllib3<1.25,>=1.21.1 in /usr/local/l
ib/python3.6/dist-packages (from kaggle) (1.22)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.6
/dist-packages (from kaggle) (1.11.0)
Requirement already satisfied: certifi in /usr/local/lib/python3.6/d
ist-packages (from kaggle) (2019.3.9)
Requirement already satisfied: python-dateutil in /usr/local/lib/pyt
hon3.6/dist-packages (from kaggle) (2.5.3)
Requirement already satisfied: requests in /usr/local/lib/python3.6/
dist-packages (from kaggle) (2.18.4)
Requirement already satisfied: tqdm in /usr/local/lib/python3.6/dist
-packages (from kaggle) (4.28.1)
Requirement already satisfied: python-slugify in /usr/local/lib/pyth
on3.6/dist-packages (from kaggle) (3.0.1)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /usr/local/l
ib/python3.6/dist-packages (from requests->kaggle) (3.0.4)
Requirement already satisfied: idna<2.7,>=2.5 in /usr/local/lib/pyth
on3.6/dist-packages (from requests->kaggle) (2.6)
Requirement already satisfied: text-unidecode==1.2 in /usr/local/lib
/python3.6/dist-packages (from python-slugify->kaggle) (1.2)
```

Choose Files no files selected

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving kaggle.json to kaggle.json
```

Out[8]:
```
{'kaggle.json': b'{"username":"rohitbohra2994","key":"b7947860e2ad40
758c4dac7cbd51c8ff"}'}
```

In [10]:
```
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/

# This permissions change avoids a warning on Kaggle tool startup.
!chmod 600 ~/.kaggle/kaggle.json

!kaggle datasets download -d pankajkarki/stackoverflow

!ls
```

```
Downloading stackoverflow.zip to /content
 97% 465M/478M [00:16<00:00, 54.5MB/s]
100% 478M/478M [00:16<00:00, 30.9MB/s]
kaggle.json  sample_data  stackoverflow.zip
```

In [11]:
```
!unzip stackoverflow.zip
```

```
Archive:  stackoverflow.zip
  inflating: Processed.db
  inflating: Titlemoreweight.db
```

In [0]:
```
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import sqlite3
import csv
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from wordcloud import WordCloud
import re
import os
from sqlalchemy import create_engine # database connection
import datetime as dt
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem.snowball import SnowballStemmer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.multiclass import OneVsRestClassifier
from sklearn.linear_model import SGDClassifier
from sklearn import metrics
from sklearn.metrics import f1_score,precision_score,recall_score
from sklearn import svm
from sklearn.linear_model import LogisticRegression
from skmultilearn.adapt import MLkNN
from skmultilearn.problem_transform import ClassifierChain
from skmultilearn.problem_transform import BinaryRelevance
from skmultilearn.problem_transform import LabelPowerset
from sklearn.naive_bayes import GaussianNB
from datetime import datetime
```

In [6]:
```
!pip install scikit-multilearn
```

```
Collecting scikit-multilearn
  Downloading https://files.pythonhosted.org/packages/bb/1f/e6ff649c72a1cdf2c7a1d31eb21705110ce1c5d3e7e26b2cc300e1637272/scikit_multilearn-0.2.0-py3-none-any.whl (https://files.pythonhosted.org/packages/bb/1f/e6ff649c72a1cdf2c7a1d31eb21705110ce1c5d3e7e26b2cc300e1637272/scikit_multilearn-0.2.0-py3-none-any.whl) (89kB)
    100% |████████████████████████████████| 92kB 4.2MB/s
Installing collected packages: scikit-multilearn
Successfully installed scikit-multilearn-0.2.0
```

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/ (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 Limit
s of the variables";\n
                for(int y=1; y<n+1; y++)\n
                {\n
                   cin>>m[y];\n
                   cin>>u[y];\n
                }\n
                for(x=1; x<n+1; x++)\n
                {\n
                   a[x] = (m[x] + u[x])/2;\n
                }\n
                c=(n*4)-4;\n
                for(int a1=1; a1<n+1; a1++)\n
                {\n\n
                   e[a1][0] = m[a1];\n
                   e[a1][1] = m[a1]+1;\n
                   e[a1][2] = u[a1]-1;\n
                   e[a1][3] = u[a1];\n
                }\n
                for(int i=1; i<n+1; i++)\n
                {\n
                   for(int l=1; l<=i; l++)\n
                   {\n
                       if(l!=1)\n
                       {\n
                           cout<<a[l]<<"\\t";\n
                       }\n
                   }\n
                   for(int j=0; j<4; j++)\n
                   {\n
                       cout<<e[i][j];\n
                       for(int k=0; k<n-(i+1); k++)\n
                       {\n
                           cout<<a[k]<<"\\t";\n
```

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


\n\n


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



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


\n\n


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


The output is not coming,can anyone correct the code or tell m
e what\'s wrong?
\n'
Tags : 'c++ c'
```

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

2.2.1 Type of Machine Learning Problem

It is a multi-label classification problem
Multi-label Classification: Multilabel classification assigns to each sample a set of target labels. This can be thought as predicting properties of a data-point that are not mutually exclusive, such as topics that are relevant for a document. A question on Stackoverflow might be about any of C, Pointers, FileIO and/or memory-management at the same time or none of these.
Credit: http://scikit-learn.org/stable/modules/multiclass.html (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 [0]:
```
#Creating db file from csv
#Learn SQL: https://www.w3schools.com/sql/default.asp
if not os.path.isfile('train.db'):
    start = datetime.now()
    disk_engine = create_engine('sqlite:///train.db')
    start = dt.datetime.now()
    chunksize = 180000
    j = 0
    index_start = 1
    for df in pd.read_csv('Train.csv', names=['Id', 'Title', 'Body', '
        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 [0]:
```
if os.path.isfile('train.db'):
    start = datetime.now()
    con = sqlite3.connect('train.db')
    num_rows = pd.read_sql_query("""SELECT count(*) FROM data""", con)
    #Always remember to close the database
    print("Number of rows in the database :","\n",num_rows['count(*)']
    con.close()
    print("Time taken to count the number of rows :", datetime.now() -
else:
    print("Please download the train.db file from drive or run the abo
```

```
Number of rows in the database :
 6034196
Time taken to count the number of rows : 0:01:15.750352
```

3.1.3 Checking for duplicates

In [0]:
```
#Learn SQl: https://www.w3schools.com/sql/default.asp
if os.path.isfile('train.db'):
    start = datetime.now()
    con = sqlite3.connect('train.db')
    df_no_dup = pd.read_sql_query('SELECT Title, Body, Tags, COUNT(*) a
    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 firs
```

```
Time taken to run this cell : 0:04:33.560122
```

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

Out[6]:

	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?	I should do binding for datagrid dynamicall...	c# silverlight data-binding	1
2	Dynamic Datagrid Binding in Silverlight?	I should do binding for datagrid dynamicall...	c# silverlight data-binding columns	1
3	java.lang.NoClassDefFoundError: javax/serv...	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
```

```
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[8]:
```
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
# 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[9]:

	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?	I should do binding for datagrid dynamicall...	c# silverlight data-binding	1
2	Dynamic Datagrid Binding in Silverlight?	I should do binding for datagrid dynamicall...	c# silverlight data-binding columns	1
3	java.lang.NoClassDefFoundError: javax/serv...	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[10]:
```
3    1206157
2    1111706
4     814996
1     568298
5     505158
Name: tag_count, dtype: int64
```

In [0]:
```
#Creating a new database with no duplicates
if not os.path.isfile('train_no_dup.db'):
    disk_dup = create_engine("sqlite:///train_no_dup.db")
    no_dup = pd.DataFrame(df_no_dup, columns=['Title', 'Body', 'Tags']
    no_dup.to_sql('no_dup_train',disk_dup)
```

In [0]:
```
#This method seems more appropriate to work with this much data.
#creating the connection with database file.
if os.path.isfile('train_no_dup.db'):
    start = datetime.now()
    con = sqlite3.connect('train_no_dup.db')
    tag_data = pd.read_sql_query("""SELECT Tags FROM no_dup_train""", 
    #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 abo
```

```
Time taken to run this cell : 0:00:52.992676
```

3.2 Analysis of Tags

3.2.1 Total number of unique tags

In [0]:
```
# Importing & Initializing the "CountVectorizer" object, which
#is scikit-learn's bag of words tool.

#by default 'split()' will tokenize each tag using space.
vectorizer = CountVectorizer(tokenizer = lambda x: x.split())
# fit_transform() does two functions: First, it fits the model
# and learns the vocabulary; second, it transforms our training data
# into feature vectors. The input to fit_transform should be a list of
tag_dtm = vectorizer.fit_transform(tag_data['Tags'])
```

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

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

In [0]:
```
#'get_feature_name()' gives us the vocabulary.
tags = vectorizer.get_feature_names()
#Lets look at the tags we have.
print("Some of the tags we have :", tags[:10])
```

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

3.2.3 Number of times a tag appeared

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

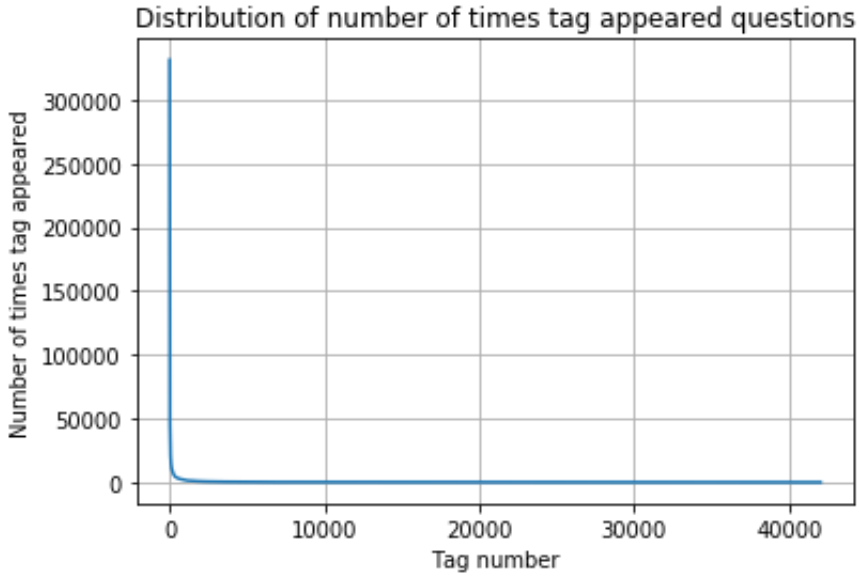
In [0]:
```
#Saving this dictionary to csv files.
if not os.path.isfile('tag_counts_dict_dtm.csv'):
    with open('tag_counts_dict_dtm.csv', 'w') as csv_file:
        writer = csv.writer(csv_file)
        for key, value in result.items():
            writer.writerow([key, value])
tag_df = pd.read_csv("tag_counts_dict_dtm.csv", names=['Tags', 'Counts
tag_df.head()
```

Out[17]:

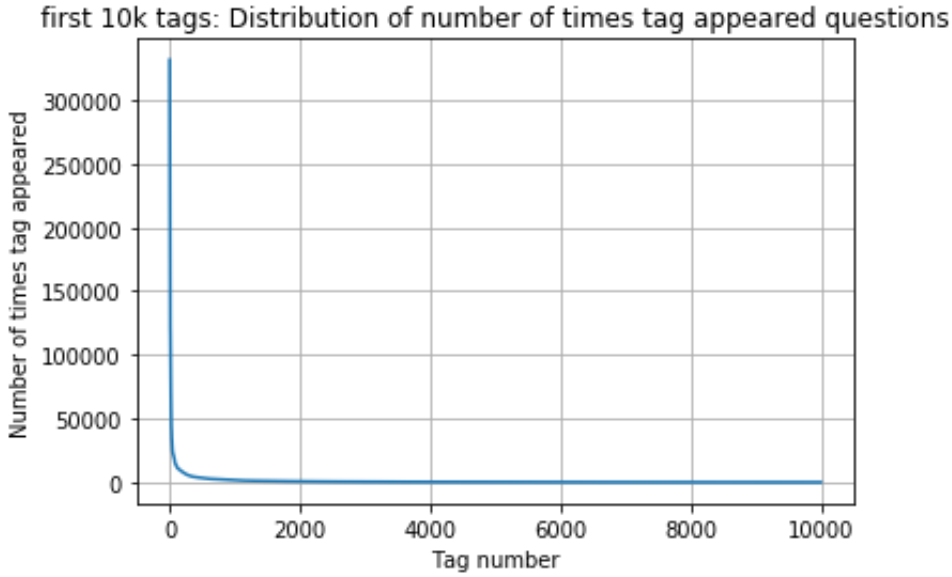
	Tags	Counts
0	.a	18
1	.app	37
2	.asp.net-mvc	1
3	.aspxauth	21
4	.bash-profile	138

In [0]:
```
tag_df_sorted = tag_df.sort_values(['Counts'], ascending=False)
tag_counts = tag_df_sorted['Counts'].values
```

In [0]:
```
plt.plot(tag_counts)
plt.title("Distribution of number of times tag appeared questions")
plt.grid()
plt.xlabel("Tag number")
plt.ylabel("Number of times tag appeared")
plt.show()
```



In [0]:
```
plt.plot(tag_counts[0:10000])
plt.title('first 10k tags: Distribution of number of times tag appeare
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])
```

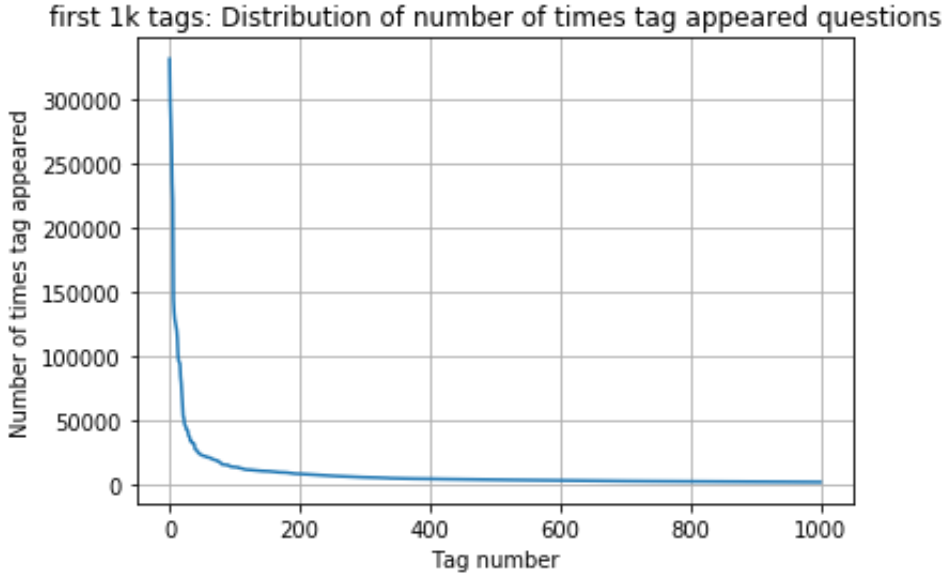



```
400 [331505  44829  22429  17728  13364  11162  10029   9148   8054
7151
   6466   5865   5370   4983   4526   4281   4144   3929   3750   35
```

```
93
    3453    3299    3123    2989    2891    2738    2647    2527    2431    23
31
    2259    2186    2097    2020    1959    1900    1828    1770    1723    16
73
    1631    1574    1532    1479    1448    1406    1365    1328    1300    12
66
    1245    1222    1197    1181    1158    1139    1121    1101    1076    10
56
    1038    1023    1006     983     966     952     938     926     911     8
91
     882     869     856     841     830     816     804     789     779     7
70
     752     743     733     725     712     702     688     678     671     6
58
     650     643     634     627     616     607     598     589     583     5
77
     568     559     552     545     540     533     526     518     512     5
06
     500     495     490     485     480     477     469     465     457     4
50
     447     442     437     432     426     422     418     413     408     4
03
     398     393     388     385     381     378     374     370     367     3
65
     361     357     354     350     347     344     342     339     336     3
32
     330     326     323     319     315     312     309     307     304     3
01
     299     296     293     291     289     286     284     281     278     2
76
     275     272     270     268     265     262     260     258     256     2
54
     252     250     249     247     245     243     241     239     238     2
36
     234     233     232     230     228     226     224     222     220     2
19
     217     215     214     212     210     209     207     205     204     2
03
     201     200     199     198     196     194     193     192     191     1
89
     188     186     185     183     182     181     180     179     178     1
77
     175     174     172     171     170     169     168     167     166     1
65
     164     162     161     160     159     158     157     156     156     1
55
     154     153     152     151     150     149     149     148     147     1
46
     145     144     143     142     142     141     140     139     138     1
37
     137     136     135     134     134     133     132     131     130     1
30
```

```
    129    128    128    127    126    126    125    124    124    1
23
    123    122    122    121    120    120    119    118    118    1
17
    117    116    116    115    115    114    113    113    112    1
11
    111    110    109    109    108    108    107    106    106    1
06
    105    105    104    104    103    103    102    102    101    1
01
    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]
```

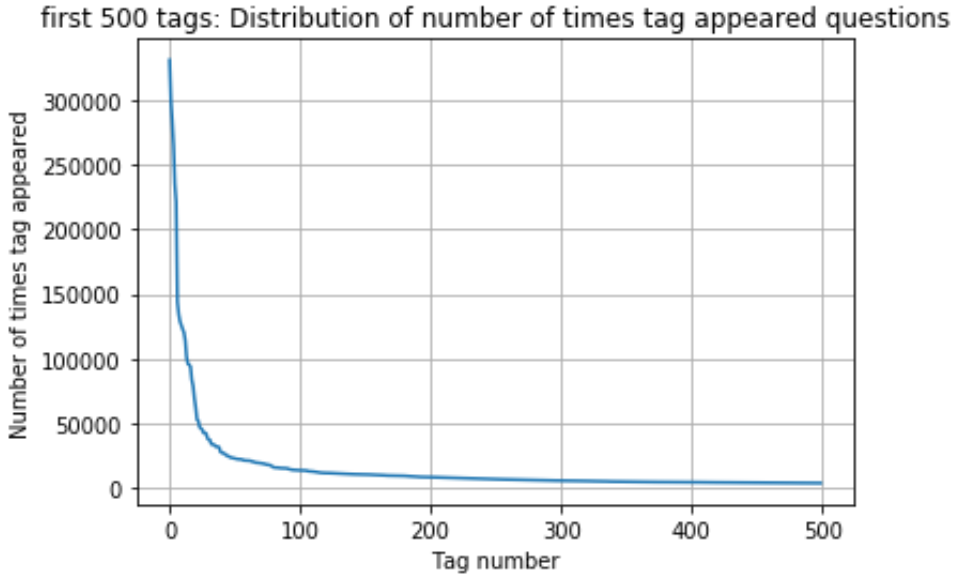
In [0]:
```
plt.plot(tag_counts[0:1000])
plt.title('first 1k tags: Distribution of number of times tag appeared
plt.grid()
plt.xlabel("Tag number")
plt.ylabel("Number of times tag appeared")
plt.show()
print(len(tag_counts[0:1000:5]), tag_counts[0:1000:5])
```




```
200 [331505 221533 122769  95160  62023  44829  37170  31897  26925
24537
  22429  21820  20957  19758  18905  17728  15533  15097  14884  137
03
```

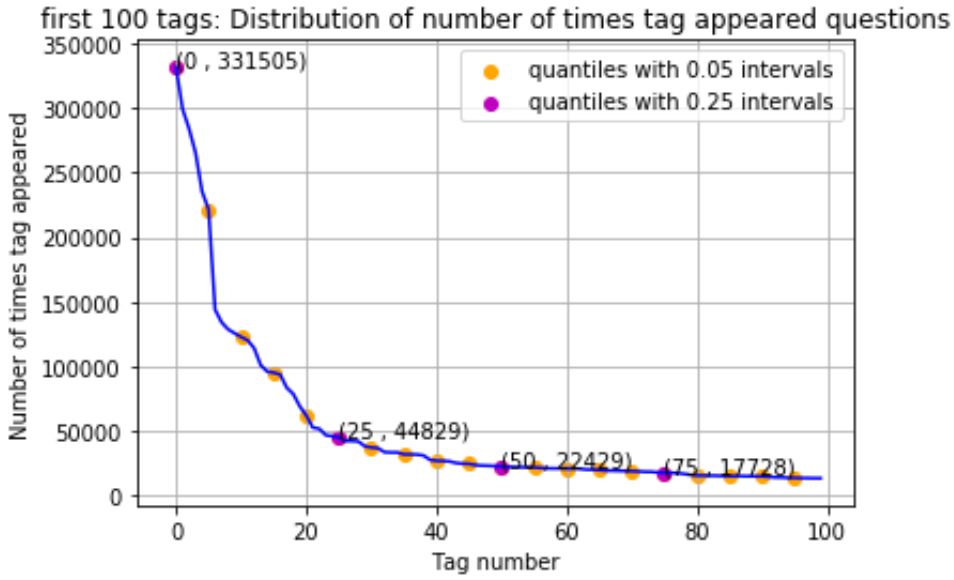
```
[[13364 13157 12407 11658 11228 11162 10863 10600 10350 10224
 [10029  9884  9719  9411  9252  9148  9040  8617  8361  8163
 [ 8054  7867  7702  7564  7274  7151  7052  6847  6656  6553
 [ 6466  6291  6183  6093  5971  5865  5760  5577  5490  5411
 [ 5370  5283  5207  5107  5066  4983  4891  4785  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 [0]:
```
plt.plot(tag_counts[0:500])
plt.title('first 500 tags: Distribution of number of times tag appeare
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  137
03
  13364  13157  12407  11658  11228  11162  10863  10600  10350  102
24
  10029   9884   9719   9411   9252   9148   9040   8617   8361   81
63
   8054   7867   7702   7564   7274   7151   7052   6847   6656   65
53
   6466   6291   6183   6093   5971   5865   5760   5577   5490   54
11
   5370   5283   5207   5107   5066   4983   4891   4785   4658   45
49
   4526   4487   4429   4335   4310   4281   4239   4228   4195   41
59
   4144   4088   4050   4002   3957   3929   3874   3849   3818   37
97
   3750   3703   3685   3658   3615   3593   3564   3521   3505   34
83]
```

In [0]:
```
plt.plot(tag_counts[0:100], c='b')
plt.scatter(x=list(range(0,100,5)), y=tag_counts[0:100:5], c='orange',
# quantiles with 0.25 difference
plt.scatter(x=list(range(0,100,25)), y=tag_counts[0:100:25], c='m', lal

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, 

plt.title('first 100 tags: Distribution of number of times tag appeare
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  137
03]
```

In [0]:
```
# Store tags greater than 10K in one list
lst_tags_gt_10k = tag_df[tag_df.Counts>10000].Tags
#Print the length of the list
print ('{} Tags are used more than 10000 times'.format(len(lst_tags_gt_
# 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_g
```

```
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

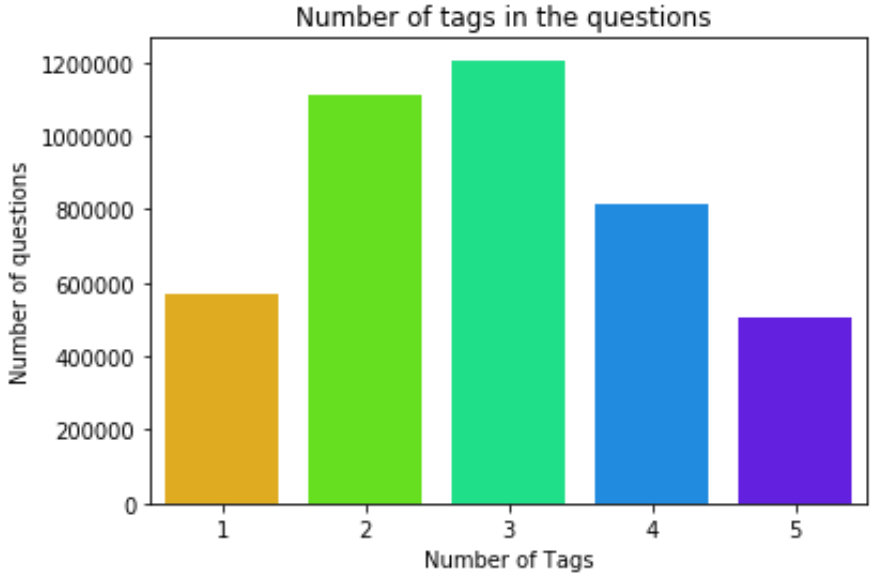
In [0]:
```
#Storing the count of tag in each question in list 'tag_count'
tag_quest_count = tag_dtm.sum(axis=1).tolist()
#Converting list of lists into single list, we will get [[3], [4], [2]
tag_quest_count=[int(j) for i in tag_quest_count for j in i]
print ('We have total {} datapoints.'.format(len(tag_quest_count)))

print(tag_quest_count[:5])
```

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

In [0]:
```
print( "Maximum number of tags per question: %d"%max(tag_quest_count))
print( "Minimum number of tags per question: %d"%min(tag_quest_count))
print( "Avg. number of tags per question: %f"% ((sum(tag_quest_count)*
```

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

In [0]:
```
sns.countplot(tag_quest_count, palette='gist_rainbow')
plt.title("Number of tags in the questions ")
plt.xlabel("Number of Tags")
plt.ylabel("Number of questions")
plt.show()
```

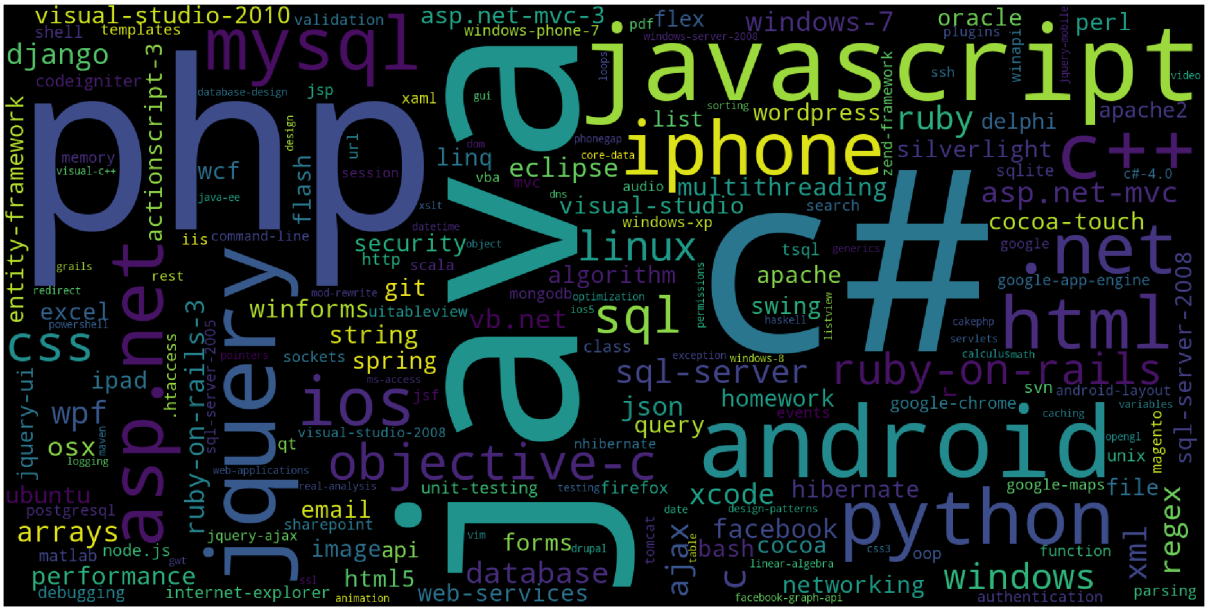


Observations:

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

3.2.5 Most Frequent Tags

In [0]:
```
# Ploting word cloud
start = datetime.now()

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

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



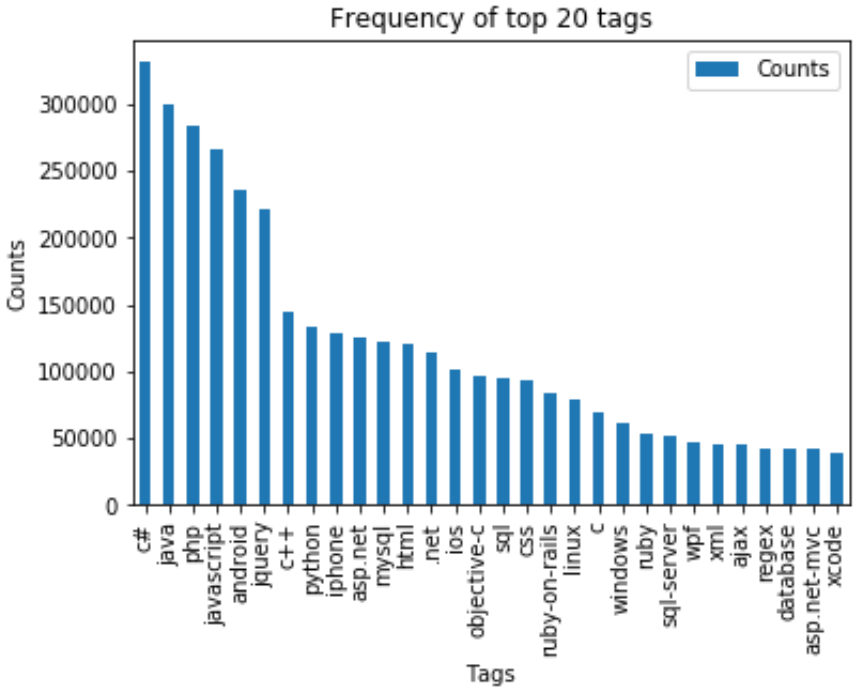

```
Time taken to run this cell : 0:00:05.470788
```

Observations:

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

3.2.6 The top 20 tags

In [0]:
```
i=np.arange(30)
tag_df_sorted.head(30).plot(kind='bar')
plt.title('Frequency of top 20 tags')
plt.xticks(i, tag_df_sorted['Tags'])
plt.xlabel('Tags')
plt.ylabel('Counts')
plt.show()
```



Observations:

1. Majority of the most frequent tags are programming language.
2. C# is the top most frequent programming language.
3. 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 [0]:
```
def striphtml(data):
    cleanr = re.compile('<.*?>')
    cleantext = re.sub(cleanr, ' ', str(data))
    return cleantext
stop_words = set(stopwords.words('english'))
stemmer = SnowballStemmer("english")
```

In [0]:
```
#http://www.sqlitetutorial.net/sqlite-python/create-tables/
def create_connection(db_file):
    """ create a database connection to the SQLite database
        specified by db_file
    :param db_file: database file
    :return: Connection object or None
    """
    try:
        conn = sqlite3.connect(db_file)
        return conn
    except Error as e:
        print(e)

    return None

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

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

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

sql_create_table = """CREATE TABLE IF NOT EXISTS QuestionsProcessed (q
create_database_table("Processed.db", sql_create_table)
```

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

In [0]:
```
# http://www.sqlitetutorial.net/sqlite-delete/
# https://stackoverflow.com/questions/2279706/select-random-row-from-a
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 ORDI

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 : 0:06:32.806567
```

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

In [0]:
```
#http://www.bernzilla.com/2008/05/13/selecting-a-random-row-from-an-sq

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.DO

    question=re.sub('<code>(.*?)</code>', '', question, flags=re.MULTI
```

```
    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 except
    question=' '.join(str(stemmer.stem(j)) for j in words if j not in s

    len_post+=len(question)
    tup = (question,code,tags,x,len(question),is_code)
    questions_proccesed += 1
    writer.execute("insert into QuestionsProcessed(question,code,tags,v
    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_
print( "Avg. length of questions(Title+Body) after processing: %d"%no_c
print ("Percent of questions containing code: %d"%((questions_with_code

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: 1169
Avg. length of questions(Title+Body) after processing: 327
Percent of questions containing code: 57
Time taken to run this cell : 0:47:05.946582
```

In [0]:
```
# dont forget to close the connections, or else you will end up with l
conn_r.commit()
conn_w.commit()
conn_r.close()
conn_w.close()
```

In [0]:
```
if os.path.isfile(write_db):
    conn_r = create_connection(write_db)
    if conn_r is not None:
        reader =conn_r.cursor()
        reader.execute("SELECT question From QuestionsProcessed LIMIT
        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
==================================================================
================================
('ef code first defin one mani relationship differ key troubl defin
one zero mani relationship entiti ef object model look like use flue
nt api object composit pk defin batch id batch detail id use fluent

api object composit pk defin batch detail id compani id map exist da
tabas tpt basic idea submittedtransact zero mani submittedsplittrans
act associ navig realli need one way submittedtransact submittedspli
ttransact need dbcontext class onmodelcr overrid map class lazi load
occur submittedtransact submittedsplittransact help would much appre
ci edit taken advic made follow chang dbcontext class ad follow onmo
delcr overrid must miss someth get follow except thrown submittedtra
nsact key batch id batch detail id zero one mani submittedsplittrans
act key batch detail id compani id rather assum convent creat relati
onship two object configur requir sinc obvious wrong',)
------------------------------------------------------------------
--------------------------------
('explan new statement review section c code came accross statement
block come accross new oper use way someon explain new call way',)
------------------------------------------------------------------
--------------------------------
('error function notat function solv logic riddl iloczyni list struc
tur list possibl candid solut list possibl coordin matrix wan na cho
os one candid compar possibl candid element equal wan na delet coord
in call function skasuj look like ni knowledg haskel cant see what w
rong',)
------------------------------------------------------------------
--------------------------------
('step plan move one isp anoth one work busi plan switch isp realli
soon need chang lot inform dns wan wan wifi question guy help mayb p
eopl plan correct chang current isp new one first dns know receiv ne
w ip isp major chang need take consider exchang server owa vpn two s
ite link wireless connect km away citrix server vmware exchang domai
n control link place import server crucial step inform need know avo
id downtim busi regard ndavid',)
------------------------------------------------------------------
--------------------------------
('use ef migrat creat databas googl migrat tutori af first run appli
c creat databas ef enabl migrat way creat databas migrat rune applic
tri',)
------------------------------------------------------------------
--------------------------------
('magento unit test problem magento site recent look way check integ
r magento site given point unit test jump one method would assum wou
ld big job write whole lot test check everyth site work anyon involv
```

```
unit test magento advis follow possibl test whole site custom modul
nis exampl test would amaz given site heavili link databas would nbe
possibl fulli test site without disturb databas better way automatic
lli check integr magento site say integr realli mean fault site ship
payment etc work correct',)
--------------------------------------------------------------------
--------------------------------
('find network devic without bonjour write mac applic need discov ma
c pcs iphon ipad connect wifi network bonjour seem reason choic turn
problem mani type router mine exampl work block bonjour servic need
find ip devic tri connect applic specif port determin process run be
st approach accomplish task without violat app store sandbox',)
--------------------------------------------------------------------

--------------------------------
('send multipl row mysql databas want send user mysql databas column
user skill time nnow want abl add one row user differ time etc would
code send databas nthen use help schema',)
--------------------------------------------------------------------
--------------------------------
('insert data mysql php powerpoint event powerpoint present run cont
inu way updat slide present automat data mysql databas websit',)
--------------------------------------------------------------------
```

In [0]:
```
#Taking 1 Million entries to a dataframe.
write_db = 'Processed.db'
if os.path.isfile(write_db):
    conn_r = create_connection(write_db)
    if conn_r is not None:
        preprocessed_data = pd.read_sql_query("""SELECT question, Tags
conn_r.commit()
conn_r.close()
```

In [0]:
```
preprocessed_data.head()
```

Out[47]:

	question	tags
0	resiz root window tkinter resiz root window re...	python tkinter
1	ef code first defin one mani relationship diff...	entity-framework-4.1
2	explan new statement review section c code cam...	c++
3	error function notat function solv logic riddl...	haskell logic
4	step plan move one isp anoth one work busi pla...	dns isp

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

```
number of data points in sample : 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

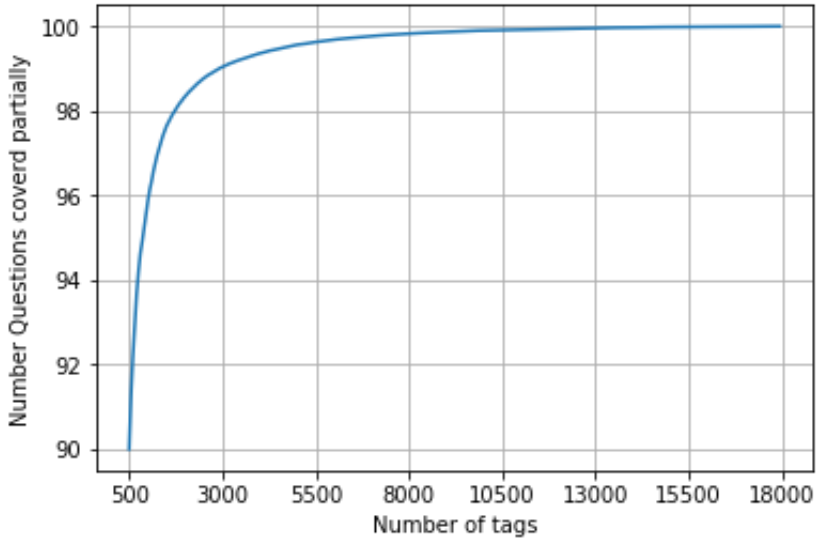
In [0]:
```
# binary='true' will give a binary vectorizer
vectorizer = CountVectorizer(tokenizer = lambda x: x.split(), binary='
multilabel_y = vectorizer.fit_transform(preprocessed_data['tags'])
```

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

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

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

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

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




```
with  5500 tags we are covering  99.04 % of questions
```

In [0]:
```
multilabel_yx = tags_to_choose(5500)
print("number of questions that are not covered :", questions_explaine
```

```
number of questions that are not covered : 9599 out of  999999
```

In [0]:
```
print("Number of tags in sample :", multilabel_y.shape[1])
print("number of tags taken :", multilabel_yx.shape[1],"(",(multilabel_
```

```
Number of tags in sample : 35422
number of tags taken : 5500 ( 15.527073570097679 %)
```

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

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

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

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

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

In [0]:
```
print("Number of data points in train data :", y_train.shape)
print("Number of data points in test data :", y_test.shape)
```

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

4.3 Featurizing data

In [0]:
```
start = datetime.now()
vectorizer = TfidfVectorizer(min_df=0.00009, max_features=200000, smoo
                             tokenizer = lambda x: x.split(), sublinea
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 : 0:09:50.460431
```

In [0]:
```
print("Dimensions of train data X:",x_train_multilabel.shape, "Y :",y_
print("Dimensions of test data X:",x_test_multilabel.shape,"Y:",y_test
```

```
Diamensions of train data X: (799999, 88244) Y : (799999, 5500)
Diamensions of test data X: (200000, 88244) Y: (200000, 5500)
```

In [0]:
```
# https://www.analyticsvidhya.com/blog/2017/08/introduction-to-multi-l
#https://stats.stackexchange.com/questions/117796/scikit-multi-label-c
# 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
#<ipython-input-170-f0e7c7f3e0be> in <module>()
#----> classifier.fit(x_train_multilabel, y_train)
```

Out[92]:
```
"\nfrom skmultilearn.adapt import MLkNN\nclassifier = MLkNN(k=21)\n\
n# train\nclassifier.fit(x_train_multilabel, y_train)\n\n# predict\n
predictions = classifier.predict(x_test_multilabel)\nprint(accuracy_
score(y_test,predictions))\nprint(metrics.f1_score(y_test, predictio
ns, average = 'macro'))\nprint(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 [0]:
```
# this will be taking so much time try not to run it, download the lr_
# This takes about 6-7 hours to run.
classifier = OneVsRestClassifier(SGDClassifier(loss='log', alpha=0.000
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
print("micro f1 scoore :",metrics.f1_score(y_test, predictions, averag
print("hamming loss :",metrics.hamming_loss(y_test,predictions))
print("Precision recall report :\n",metrics.classification_report(y_te
```

```
accuracy : 0.081965
macro f1 score : 0.0963020140154
micro f1 scoore : 0.374270748817
hamming loss : 0.00041225090909090907
Precision recall report :
              precision    recall  f1-score   support

           0       0.62      0.23      0.33     15760
           1       0.79      0.43      0.56     14039
           2       0.82      0.55      0.66     13446
           3       0.76      0.42      0.54     12730
           4       0.94      0.76      0.84     11229
           5       0.85      0.64      0.73     10561
           6       0.70      0.30      0.42      6958
           7       0.87      0.61      0.72      6309
           8       0.70      0.40      0.50      6032
           9       0.78      0.43      0.55      6020
          10       0.86      0.62      0.72      5707
          11       0.52      0.17      0.25      5723
```

In [0]:
```
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 [0]:
```
sql_create_table = """CREATE TABLE IF NOT EXISTS QuestionsProcessed (q
create_database_table("Titlemoreweight.db", sql_create_table)
```

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

In [0]:
```
# http://www.sqlitetutorial.net/sqlite-delete/
# https://stackoverflow.com/questions/2279706/select-random-row-from-a

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

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

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

4.5.1 Preprocessing of questions

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

In [0]:
```
#http://www.bernzilla.com/2008/05/13/selecting-a-random-row-from-an-sq
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.DO

    question=re.sub('<code>(.*?)</code>', '', question, flags=re.MULTI
    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)+" "+questi
#     else:
#         question=str(title)+" "+str(title)+" "+str(title)+" "+questi

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

    #Removing all single letter and and stopwords from question except
    question=' '.join(str(stemmer.stem(j)) for j in words if j not in

    len_post+=len(question)
    tup = (question,code,tags,x,len(question),is_code)
    questions_proccesed += 1
    writer.execute("insert into QuestionsProcessed(question,code,tags,
    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_
print( "Avg. length of questions(Title+Body) after processing: %d"%no_
print ("Percent of questions containing code: %d"%((questions_with_cod

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

```
number of questions completed= 100000
number of questions completed= 200000
number of questions completed= 300000
number of questions completed= 400000
number of questions completed= 500000
Avg. length of questions(Title+Body) before processing: 1239
```

```
Avg. length of questions(Title+Body) after processing: 424
Percent of questions containing code: 57
Time taken to run this cell : 0:23:12.329039
```

In [0]:
```
# never forget to close the conections or else we will end up with dat
conn_r.commit()
conn_w.commit()
conn_r.close()
conn_w.close()
```

Sample quesitons after preprocessing of data

In [0]:
```
if os.path.isfile(write_db):
    conn_r = create_connection(write_db)
    if conn_r is not None:
        reader =conn_r.cursor()
        reader.execute("SELECT question From QuestionsProcessed LIMIT
        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 dy
nam datagrid bind silverlight bind datagrid dynam code wrote code de
bug code block seem bind correct grid come column form come grid col
umn although necessari bind nthank repli advance..',)
------------------------------------------------------------------
--------------------------------
('java.lang.noclassdeffounderror javax servlet jsp tagext taglibrary
valid java.lang.noclassdeffounderror javax servlet jsp tagext taglib
raryvalid java.lang.noclassdeffounderror javax servlet jsp tagext ta
glibraryvalid follow guid link instal jstl got follow error tri laun
ch jsp page java.lang.noclassdeffounderror javax servlet jsp tagext
taglibraryvalid taglib declar instal jstl 1.1 tomcat webapp tri proj
ect work also tri version 1.2 jstl still messag caus solv',)
------------------------------------------------------------------
--------------------------------
('java.sql.sqlexcept microsoft odbc driver manag invalid descriptor
index java.sql.sqlexcept microsoft odbc driver manag invalid descrip
tor index java.sql.sqlexcept microsoft odbc driver manag invalid des
criptor 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 confused.I find post feed api method like correct second way u
se curl someth like way better',)
--------------------------------------------------------------------
--------------------------------
('btnadd click event open two window record ad btnadd click event op
en two window record ad btnadd click event open two window record ad
open window search.aspx use code hav add button search.aspx nwhen in
sert record btnadd click event open anoth window nafter insert recor
d close window',)
--------------------------------------------------------------------
--------------------------------
('sql inject issu prevent correct form submiss php sql inject issu p
revent 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 html use templat file forgiv okay entir php s
cript get execut see data post none forum field post problem use som
eth titl field none data get post current use print post see submit
noth work flawless statement though also mention script work flawles
s local machin use host come across problem state list input test me
ss',)
--------------------------------------------------------------------
--------------------------------
('countabl subaddit lebesgu measur countabl subaddit lebesgu measur
countabl subaddit lebesgu measur let lbrace rbrace sequenc set sigma
-algebra mathcal want show left bigcup right leq sum left right coun
tabl addit measur defin set sigma algebra mathcal think use monoton
properti somewher proof start appreci littl help nthank ad han answe
r make follow addit construct given han answer clear bigcup bigcup c
ap emptyset neq left bigcup right left bigcup right sum left right a
lso construct subset monoton left right leq left right final would s
um 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 refer
enc error undefin symbol architectur i386 objc class skpsmtpmessag r
eferenc error import framework send email applic background import f
ramework i.e skpsmtpmessag somebodi suggest get error collect2 ld re
turn exit status import framework correct sorc taken framework follo
w mfmailcomposeviewcontrol question lock field updat answer drag dro
p folder project click copi nthat',)
--------------------------------------------------------------------
--------------------------------
```

Saving Preprocessed data to a Database

In [0]:
```
#Taking 0.5 Million entries to a dataframe.
write_db = 'Titlemoreweight.db'
if os.path.isfile(write_db):
    conn_r = create_connection(write_db)
    if conn_r is not None:
        preprocessed_data = pd.read_sql_query("""SELECT question, Tags
conn_r.commit()
conn_r.close()
```

In [0]:
```
preprocessed_data.head()
```

Out[100]:

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

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

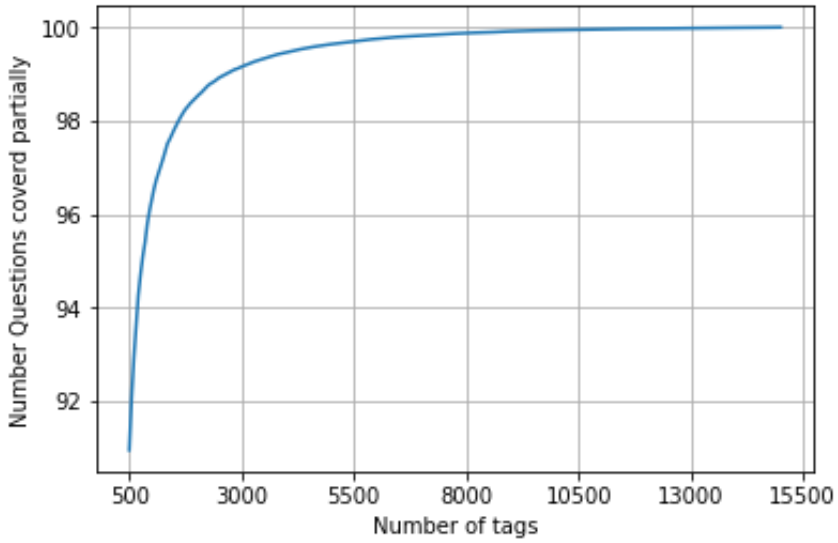
Converting string Tags to multilable output variables

In [0]:
```
vectorizer = CountVectorizer(tokenizer = lambda x: x.split(), binary='
multilabel_y = vectorizer.fit_transform(preprocessed_data['tags'])
```

Selecting 500 Tags

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

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




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

In [0]:
```
# we will be taking 500 tags
multilabel_yx = tags_to_choose(500)
print("number of questions that are not covered :", questions_explaine
```

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

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

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

In [0]:
```
print("Number of data points in train data :", y_train.shape)
print("Number of data points in test data :", y_test.shape)
```

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

4.5.2 Featurizing data with Tfldf vectorizer

In [0]:
```
start = datetime.now()
vectorizer = TfidfVectorizer(min_df=0.00009, max_features=200000, smoo
                             tokenizer = lambda x: x.split(), sublinea
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 : 0:03:52.522389
```

In [0]:
```
print("Dimensions of train data X:",x_train_multilabel.shape, "Y :",y_
print("Dimensions of test data X:",x_test_multilabel.shape,"Y:",y_test
```

```
Diamensions of train data X: (400000, 94927) Y : (400000, 500)
Diamensions of test data X: (100000, 94927) Y: (100000, 500)
```

4.5.3 Applying Logistic Regression with OneVsRest Classifier

In [0]:
```
start = datetime.now()
classifier = OneVsRestClassifier(SGDClassifier(loss='log', alpha=0.000(
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(p

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(p

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

```
Accuracy : 0.23623
Hamming loss  0.00278088
Micro-average quality numbers
Precision: 0.7216, Recall: 0.3256, F1-measure: 0.4488
Macro-average quality numbers
Precision: 0.5473, Recall: 0.2572, F1-measure: 0.3339
              precision    recall  f1-score   support

           0       0.94      0.64      0.76      5519
           1       0.69      0.26      0.38      8190
           2       0.81      0.37      0.51      6529
           3       0.81      0.43      0.56      3231
           4       0.81      0.40      0.54      6430
           5       0.82      0.33      0.47      2879
           6       0.87      0.50      0.63      5086
           7       0.87      0.54      0.67      4533
           8       0.60      0.13      0.22      3000
           9       0.81      0.53      0.64      2765
          10       0.59      0.17      0.26      3051
```

In [0]:
```
joblib.dump(classifier, 'lr_with_more_title_weight.pkl')
```

Out[113]:
```
['lr_with_more_title_weight.pkl']
```

In [0]:
```
start = datetime.now()
classifier_2 = OneVsRestClassifier(LogisticRegression(penalty='l1'), n
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(p

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(p

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

```
Accuracy : 0.25108
Hamming loss  0.00270302
Micro-average quality numbers
Precision: 0.7172, Recall: 0.3672, F1-measure: 0.4858
Macro-average quality numbers
Precision: 0.5570, Recall: 0.2950, F1-measure: 0.3710
              precision    recall  f1-score   support

           0       0.94      0.72      0.82      5519
           1       0.70      0.34      0.45      8190
           2       0.80      0.42      0.55      6529
           3       0.82      0.49      0.61      3231
           4       0.80      0.44      0.57      6430
           5       0.82      0.38      0.52      2879
           6       0.86      0.53      0.66      5086
           7       0.87      0.58      0.70      4533
           8       0.60      0.13      0.22      3000
           9       0.82      0.57      0.67      2765
          10       0.60      0.20      0.30      3051
```

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)

Loading files

In [14]:

```
#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 (q
create_database_table("Processed.db", sql_create_table)
```

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

In [0]:
```
#Taking 1 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
conn_r.commit()
conn_r.close()
```

Due to memory error I took 25K points

In [16]:
```
# Sampling data beacause of memory error we are getting while featuriz
preprocessed_data = preprocessed_data.iloc[:250000,:]
print(preprocessed_data.shape)
preprocessed_data.head()
```

```
(250000, 2)
```

Out[16]:

	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 [17]:
```
print("number of data points in sample :", preprocessed_data.shape[0])
print("number of dimensions :", preprocessed_data.shape[1])
```

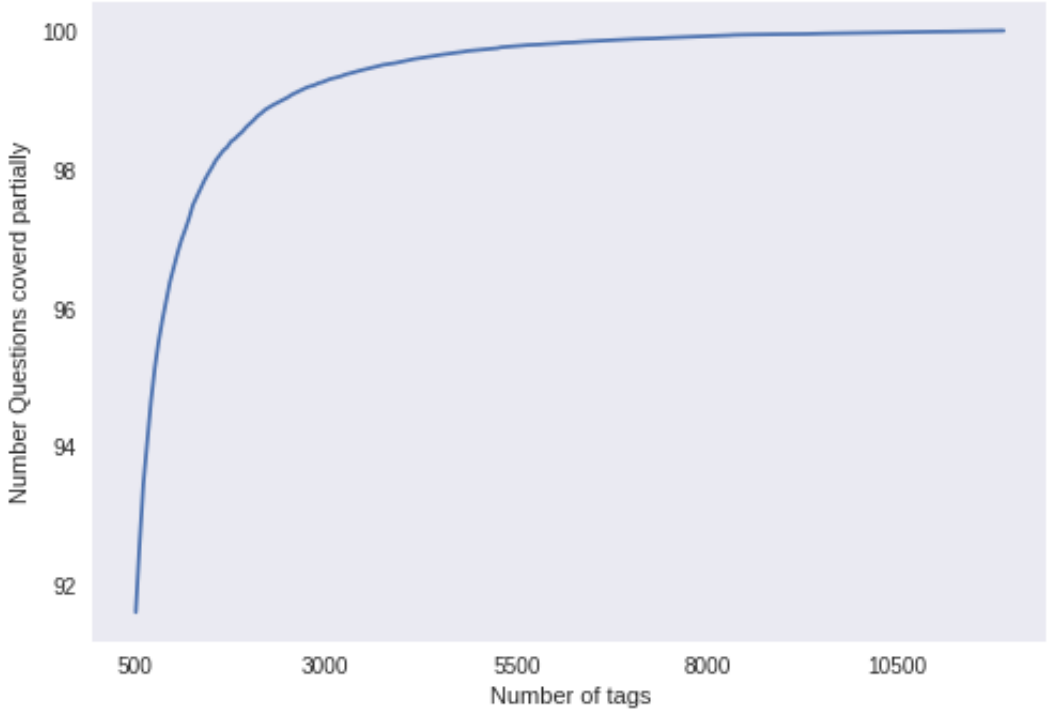
```
number of data points in sample : 250000
number of dimensions : 2
```

In [0]:
```
# binary='true' will give a binary vectorizer
count_vectorizer = CountVectorizer(tokenizer = lambda x: x.split(), bi
multilabel_y = count_vectorizer.fit_transform(preprocessed_data['tags'
```

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

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

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

In [22]:
```
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, min
print("with ",5500,"tags we are covering ",questions_explained[50],"% 
print("with ",500,"tags we are covering ",questions_explained[0],"% of
```




```
with  5500 tags we are covering  99.28 % of questions
with  500 tags we are covering  91.621 % of questions
```

In [23]:
```
# we will be taking 500 tags
multilabel_yx = tags_to_choose(500)
print("number of questions that are not covered :", questions_explaine
```

```
number of questions that are not covered : 20948 out of  250000
```

EDA on preprossed_data

In [25]:
```
print("Number of data points :", multilabel_y.shape[0])
print("Number of tags that are unique :", multilabel_y.shape[1])
```

```
Number of data points : 250000
Number of tags that are unique : 23391
```

In [26]:
```
#'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', '.aspxauth', '.bash-profile', '.cl
ass-file', '.cs-file', '.doc', '.ds-store', '.each', '.emf', '.exe']
```

In [0]:
```
freqs = multilabel_y.sum(axis=0).A1
result = dict(zip(tags, freqs))
```

In [28]:
```
#Storing the count of tag in each question in list 'tag_count'
tag_quest_count = multilabel_y.sum(axis=1).tolist()
#Converting each value in the 'tag_quest_count' to integer.
tag_quest_count=[int(j) for i in tag_quest_count for j in i]
print ('We have total {} datapoints.'.format(len(tag_quest_count)))

print(tag_quest_count[:5])
```

```
We have total 250000 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)*
```

```
Maximum number of tags per question: 5
Minimum number of tags per question: 1
Avg. number of tags per question: 2.921108
```

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

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

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

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

In [32]:
```
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 : (200000, 500)
Number of data points in test data : (50000, 500)
```

Featurizing data

In [33]:
```
start = datetime.now()
vectorizer = CountVectorizer(min_df=0.00009, max_features=25000,tokeni
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 : 0:04:20.735734
```

In [34]:
```
print("Dimensions of train data X:",x_train_multilabel.shape, "Y :",y_
print("Dimensions of test data X:",x_test_multilabel.shape,"Y:",y_test
```

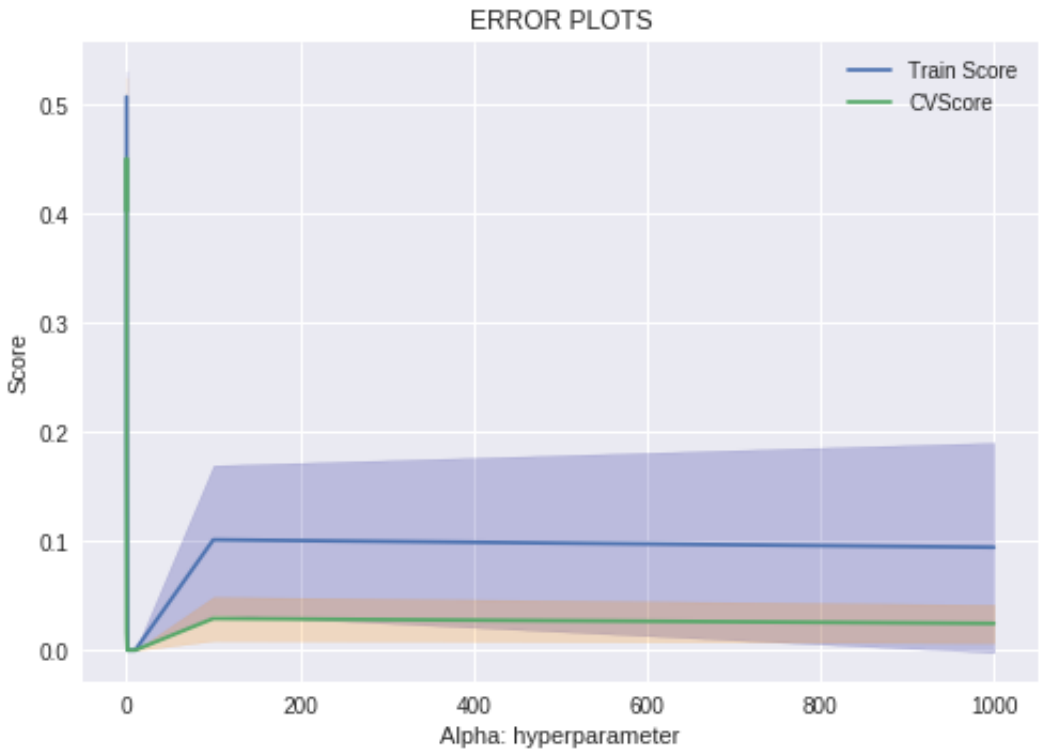
```
Dimensions of train data X: (200000, 25000) Y : (200000, 500)
Dimensions of test data X: (50000, 25000) Y: (50000, 500)
```

Applying Logistic Regression with OneVsRest Classifier

In [36]:
```
from sklearn.model_selection import GridSearchCV

param={'estimator__alpha': [0.0001, 0.001, 0.01, 0.1, 1, 10,100,1000]}
classifier = OneVsRestClassifier(SGDClassifier(loss='log', penalty='l1
gsv = GridSearchCV(estimator = classifier, param_grid=param, cv=3, verl
gsv.fit(x_train_multilabel, y_train)


best_alpha = gsv.best_estimator_.get_params()['estimator__alpha']
print('value of alpha after hyperparameter tuning : ',best_alpha)
```

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

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent w
orkers.
[Parallel(n_jobs=-1)]: Done  24 out of  24 | elapsed: 123.8min finis
hed

value of alpha after hyperparameter tuning :  0.001
```

In [37]:
```
alpha = [0.0001, 0.001, 0.01, 0.1, 1, 10,100,1000]
train_score= gsv.cv_results_['mean_train_score']
train_score_std= gsv.cv_results_['std_train_score']
cv_score = gsv.cv_results_['mean_test_score']
cv_score_std= gsv.cv_results_['std_test_score']

plt.plot(alpha, train_score, label='Train Score')
# this code is copied from here: https://stackoverflow.com/a/48803361/
plt.gca().fill_between(alpha,train_score - train_score_std,train_score

plt.plot(alpha, cv_score, label='CVScore')
# this code is copied from here: https://stackoverflow.com/a/48803361/
plt.gca().fill_between(alpha,cv_score - cv_score_std,cv_score + cv_sco
plt.legend()
plt.xlabel("Alpha: hyperparameter")
plt.ylabel("Score")
plt.title("ERROR PLOTS")
plt.show()
```



In [38]:
```
start = datetime.now()
#best_alpha = gsv.best_estimator_.get_params()['estimator__alpha']
classifier = OneVsRestClassifier(SGDClassifier(loss='log', alpha=best_
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(p

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(p

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

```
Accuracy : 0.15452
Hamming loss  0.00359352
Micro-average quality numbers
Precision: 0.4841, Recall: 0.3381, F1-measure: 0.3981
Macro-average quality numbers
Precision: 0.3465, Recall: 0.2516, F1-measure: 0.2694
Time taken to run this cell : 0:07:53.546939
```

In [39]:
```
print (metrics.classification_report(y_test, predictions))
```

	precision	recall	f1-score	support
0	0.64	0.66	0.65	2220
1	0.43	0.20	0.27	3473
2	0.75	0.31	0.44	3976
3	0.71	0.65	0.68	2437
4	0.69	0.39	0.50	2054
5	0.60	0.60	0.60	2580
6	0.72	0.61	0.66	1475
7	0.62	0.18	0.28	1493
8	0.52	0.50	0.51	957
9	0.57	0.38	0.46	1781
10	0.69	0.44	0.54	1568
11	0.54	0.32	0.40	1477
12	0.41	0.40	0.40	306

13	0.43	0.25	0.32	916
14	0.47	0.12	0.19	1161
15	0.52	0.60	0.55	811
16	0.66	0.61	0.63	1200
17	0.54	0.60	0.57	686
18	0.76	0.60	0.67	236
19	0.62	0.63	0.63	680
20	0.39	0.46	0.42	2233
21	0.48	0.39	0.43	2785
22	0.46	0.57	0.51	409
23	0.38	0.26	0.31	533
24	0.58	0.23	0.33	860
25	0.24	0.41	0.30	391
26	0.32	0.20	0.25	889
27	0.50	0.52	0.51	1280
28	0.27	0.08	0.12	739
29	0.57	0.46	0.51	337
30	0.07	0.06	0.06	54
31	0.63	0.47	0.54	325
32	0.44	0.29	0.35	274
33	0.41	0.25	0.31	398
34	0.58	0.33	0.42	305
35	0.45	0.50	0.47	221
36	0.52	0.42	0.46	486
37	0.68	0.16	0.26	583
38	0.44	0.31	0.36	340
39	0.57	0.42	0.49	80
40	0.74	0.52	0.61	243
41	0.85	0.52	0.64	420
42	0.68	0.39	0.50	685
43	0.54	0.37	0.44	262
44	0.59	0.64	0.61	283
45	0.47	0.34	0.39	301
46	0.38	0.22	0.28	507
47	0.25	0.62	0.36	85
48	0.21	0.11	0.15	280
49	0.24	0.28	0.26	160
50	0.33	0.17	0.23	419
51	0.42	0.25	0.32	446
52	0.34	0.05	0.09	242
53	0.25	0.15	0.19	188
54	0.42	0.40	0.41	178
55	0.25	0.31	0.28	214
56	0.12	0.16	0.14	56
57	0.22	0.28	0.24	221
58	0.74	0.80	0.77	282
59	0.19	0.18	0.18	33
60	0.77	0.43	0.56	134
61	0.25	0.30	0.28	211
62	0.44	0.11	0.18	159
63	0.56	0.24	0.34	372
64	0.09	0.02	0.03	53
65	0.14	0.07	0.10	41

66	0.25	0.46	0.32	13
67	0.38	0.24	0.29	253
68	0.40	0.16	0.23	232
69	0.19	0.20	0.19	202
70	0.48	0.52	0.50	125
71	0.79	0.54	0.64	217
72	0.44	0.35	0.39	229
73	0.52	0.83	0.64	161
74	0.80	0.31	0.45	225
75	0.24	0.14	0.17	118
76	0.55	0.27	0.37	175
77	0.98	0.41	0.57	687
78	0.24	0.38	0.30	76
79	0.95	0.88	0.91	698
80	0.05	0.02	0.03	299
81	0.11	0.14	0.12	65
82	0.31	0.11	0.16	133
83	0.40	0.71	0.51	42
84	0.37	0.51	0.43	126
85	0.00	0.00	0.00	202
86	0.11	0.02	0.04	48
87	0.68	0.20	0.31	285
88	0.41	0.31	0.35	49
89	0.16	0.11	0.13	217
90	0.27	0.23	0.25	47
91	0.17	0.14	0.16	49
92	0.64	0.50	0.56	76
93	0.06	0.14	0.09	36
94	0.61	0.39	0.48	213
95	0.29	0.10	0.14	135
96	0.37	0.23	0.29	200
97	0.07	0.05	0.06	118
98	0.13	0.22	0.17	59
99	0.73	0.22	0.34	121
100	0.64	0.26	0.37	425
101	0.21	0.04	0.07	159
102	0.46	0.31	0.37	114
103	0.59	0.59	0.59	74
104	0.28	0.29	0.28	126
105	0.50	0.06	0.10	106
106	0.71	0.21	0.33	206
107	0.49	0.40	0.44	90
108	0.28	0.30	0.29	54
109	0.24	0.18	0.21	161
110	0.22	0.17	0.19	163
111	0.00	0.00	0.00	204
112	0.03	0.04	0.04	142
113	0.26	0.47	0.33	15
114	0.80	0.57	0.66	187
115	0.25	0.01	0.02	166
116	0.52	0.32	0.40	170
117	0.13	0.08	0.10	105
118	0.12	0.08	0.10	37

119	0.16	0.15	0.16	97
120	0.78	0.38	0.51	152
121	0.44	0.51	0.47	178
122	0.20	0.14	0.16	130
123	0.51	0.27	0.35	89
124	0.12	0.22	0.16	132
125	0.34	0.49	0.40	98
126	0.06	0.08	0.07	49
127	0.52	0.11	0.18	108
128	0.52	0.16	0.25	361
129	0.44	0.11	0.17	156
130	0.87	0.29	0.44	160
131	0.28	0.11	0.16	118
132	0.71	0.36	0.48	14
133	0.45	0.42	0.44	177
134	0.26	0.51	0.35	109
135	0.62	0.57	0.59	251
136	0.19	0.12	0.15	25
137	0.47	0.08	0.14	101
138	0.28	0.11	0.16	214
139	0.88	0.70	0.78	134
140	0.38	0.30	0.33	124
141	0.30	0.26	0.28	204
142	0.35	0.20	0.25	107
143	0.29	0.46	0.36	192
144	0.35	0.37	0.36	89
145	0.14	0.33	0.19	21
146	0.42	0.29	0.34	56
147	0.00	0.00	0.00	126
148	0.33	0.54	0.41	100
149	0.42	0.30	0.35	97
150	0.77	0.59	0.67	282
151	0.85	0.67	0.75	70
152	0.41	0.57	0.48	138
153	0.95	0.51	0.66	152
154	0.21	0.14	0.17	97
155	0.56	0.39	0.46	51
156	0.55	0.31	0.40	106
157	0.29	0.14	0.18	117
158	0.00	0.00	0.00	118
159	0.69	0.50	0.58	119
160	0.19	0.18	0.18	150
161	0.29	0.10	0.14	94
162	0.21	0.14	0.17	103
163	0.17	0.36	0.23	86
164	0.75	0.42	0.54	78
165	0.61	0.48	0.53	96
166	0.07	0.06	0.06	109
167	0.24	0.22	0.23	111
168	0.13	0.32	0.18	88
169	0.14	0.06	0.09	77
170	0.42	0.30	0.35	53
171	0.27	0.86	0.41	14

172	0.07	0.12	0.09	50
173	0.35	0.42	0.38	122
174	0.00	0.00	0.00	110
175	0.75	0.43	0.55	7
176	0.74	0.49	0.59	180
177	1.00	0.25	0.40	16
178	0.20	0.02	0.04	45
179	0.34	0.35	0.34	69
180	0.36	0.05	0.08	87
181	0.27	0.30	0.28	226
182	0.08	0.14	0.10	59
183	0.23	0.40	0.29	110
184	0.60	0.51	0.55	257
185	0.35	0.21	0.26	113
186	0.29	0.40	0.34	47
187	0.00	0.00	0.00	145
188	0.33	0.15	0.21	193
189	0.04	0.01	0.01	116
190	0.73	0.12	0.20	135
191	0.42	0.38	0.40	13
192	0.14	0.05	0.08	111
193	0.61	0.62	0.62	152
194	0.20	0.11	0.14	75
195	0.89	0.89	0.89	9
196	0.00	0.00	0.00	142
197	0.31	0.26	0.29	19
198	0.15	0.11	0.13	45
199	0.00	0.00	0.00	77
200	0.67	0.19	0.30	83
201	0.00	0.00	0.00	113
202	0.80	0.65	0.72	37
203	0.94	0.55	0.70	116
204	0.19	0.12	0.15	114
205	0.22	0.04	0.06	189
206	0.71	0.66	0.68	85
207	0.15	0.10	0.12	67
208	0.75	0.42	0.54	93
209	0.29	0.11	0.16	149
210	0.70	0.25	0.36	114
211	0.00	0.00	0.00	407
212	0.00	0.00	0.00	5
213	0.26	0.35	0.30	109
214	0.70	0.73	0.71	97
215	0.35	0.50	0.41	14
216	0.50	0.41	0.45	174
217	0.58	0.30	0.39	47
218	0.57	0.15	0.24	114
219	0.61	0.21	0.31	96
220	0.24	0.17	0.20	71
221	0.00	0.00	0.00	114
222	0.12	0.40	0.19	20
223	0.14	0.19	0.16	81
224	0.78	0.21	0.33	33

225	0.23	0.05	0.08	59
226	0.67	0.27	0.39	66
227	0.36	0.23	0.28	44
228	0.47	0.43	0.45	68
229	0.06	0.03	0.04	98
230	0.76	0.54	0.63	130
231	0.91	0.49	0.64	102
232	0.78	0.73	0.75	44
233	0.60	0.21	0.32	14
234	0.00	0.00	0.00	67
235	0.00	0.00	0.00	84
236	0.46	0.50	0.48	52
237	0.81	0.74	0.77	99
238	0.17	0.50	0.25	8
239	0.22	0.20	0.21	137
240	0.00	0.00	0.00	63
241	0.00	0.00	0.00	61
242	0.15	0.18	0.16	50
243	0.29	0.42	0.34	71
244	0.47	0.51	0.48	95
245	0.12	0.05	0.07	57
246	0.14	0.29	0.18	28
247	0.06	0.40	0.11	20
248	0.54	0.24	0.33	83
249	0.00	0.00	0.00	117
250	0.20	0.12	0.15	96
251	0.53	0.34	0.42	70
252	0.61	0.58	0.60	60
253	0.10	0.20	0.13	65
254	0.46	0.10	0.16	62
255	0.61	0.58	0.59	74
256	0.11	0.11	0.11	27
257	0.00	0.00	0.00	90
258	0.40	0.60	0.48	20
259	0.00	0.00	0.00	98
260	0.65	0.40	0.50	89
261	0.00	0.00	0.00	279
262	0.97	0.76	0.85	84
263	0.00	0.00	0.00	13
264	0.14	0.33	0.20	48
265	0.46	0.39	0.42	113
266	0.23	0.14	0.18	105
267	0.17	0.09	0.12	78
268	0.42	0.38	0.40	47
269	0.74	0.14	0.24	141
270	0.00	0.00	0.00	83
271	0.70	0.58	0.64	74
272	0.13	0.11	0.12	38
273	0.72	0.52	0.60	60
274	0.20	0.14	0.16	81
275	0.29	0.04	0.07	54
276	0.33	0.22	0.26	37
277	0.30	0.11	0.16	90

278	0.10	0.15	0.12	26
279	0.76	0.47	0.58	99
280	0.57	0.04	0.07	114
281	0.30	0.25	0.27	61
282	0.52	0.45	0.48	78
283	0.17	0.07	0.09	46
284	0.47	0.17	0.25	84
285	0.65	0.33	0.44	67
286	0.83	0.02	0.03	292
287	0.86	0.76	0.80	321
288	0.43	0.48	0.46	97
289	0.00	0.00	0.00	85
290	0.57	0.65	0.61	43
291	0.00	0.00	0.00	108
292	0.17	0.05	0.07	127
293	0.23	0.04	0.07	79
294	0.39	0.37	0.38	160
295	0.79	0.40	0.53	57
296	0.24	0.08	0.12	52
297	0.47	0.38	0.42	64
298	0.03	0.20	0.06	60
299	0.40	0.44	0.42	9
300	0.46	0.49	0.48	51
301	0.31	0.07	0.11	58
302	0.12	0.04	0.06	52
303	0.40	0.36	0.38	81
304	0.00	0.00	0.00	68
305	0.00	0.00	0.00	53
306	0.25	0.18	0.21	45
307	0.00	0.00	0.00	116
308	0.24	0.17	0.20	47
309	0.39	0.13	0.19	70
310	0.18	0.16	0.17	37
311	0.86	0.85	0.85	254
312	0.00	0.00	0.00	101
313	0.15	0.07	0.10	107
314	0.11	0.25	0.15	4
315	0.00	0.00	0.00	9
316	0.85	0.34	0.48	68
317	0.00	0.00	0.00	38
318	0.64	0.11	0.19	125
319	0.19	0.27	0.22	48
320	0.00	0.00	0.00	28
321	0.00	0.00	0.00	128
322	0.00	0.00	0.00	42
323	0.25	0.43	0.32	7
324	0.06	0.06	0.06	71
325	0.55	0.60	0.57	10
326	0.80	0.42	0.55	76
327	0.09	0.10	0.10	29
328	0.60	0.33	0.43	36
329	0.93	0.59	0.72	63
330	0.23	0.05	0.08	102

331	0.67	0.51	0.57	95
332	0.84	0.26	0.40	80
333	0.45	0.26	0.33	38
334	0.55	0.20	0.29	30
335	0.60	0.42	0.49	36
336	0.38	0.36	0.37	25
337	0.00	0.00	0.00	16
338	0.00	0.00	0.00	228
339	0.27	0.23	0.25	62
340	0.00	0.00	0.00	156
341	0.58	0.20	0.30	55
342	0.88	0.52	0.65	85
343	0.00	0.00	0.00	72
344	0.00	0.00	0.00	22
345	0.90	0.43	0.58	195
346	0.88	0.40	0.55	75
347	0.24	0.11	0.15	46
348	0.05	0.07	0.06	68
349	0.19	0.19	0.19	16
350	0.37	0.40	0.38	60
351	0.13	0.10	0.12	67
352	0.95	0.45	0.61	47
353	0.83	0.69	0.75	42
354	0.00	0.00	0.00	31
355	0.09	0.07	0.08	41
356	0.26	0.23	0.25	39
357	0.97	0.34	0.50	85
358	0.79	0.78	0.79	83
359	0.50	0.19	0.27	27
360	0.37	0.35	0.36	113
361	0.03	0.09	0.04	11
362	0.03	0.02	0.02	54
363	0.37	0.19	0.26	98
364	0.79	0.50	0.61	22
365	0.62	0.29	0.39	35
366	0.00	0.00	0.00	10
367	0.00	0.00	0.00	189
368	0.04	0.08	0.05	97
369	0.00	0.00	0.00	45
370	0.95	0.28	0.43	72
371	0.00	0.00	0.00	15
372	0.18	0.43	0.25	21
373	0.26	0.28	0.27	32
374	0.22	0.28	0.24	65
375	0.78	0.52	0.62	27
376	0.50	0.33	0.40	3
377	0.27	0.09	0.14	95
378	0.00	0.00	0.00	62
379	0.74	0.58	0.65	53
380	0.28	0.25	0.26	53
381	0.00	0.00	0.00	93
382	0.29	0.18	0.22	11
383	0.00	0.00	0.00	82

384	0.63	0.46	0.53	52
385	0.09	0.06	0.07	17
386	0.17	0.69	0.27	13
387	0.20	0.38	0.26	8
388	0.00	0.00	0.00	106
389	0.84	0.54	0.66	68
390	0.20	0.50	0.29	4
391	0.27	0.04	0.07	92
392	0.00	0.00	0.00	34
393	0.11	0.14	0.13	35
394	0.18	0.20	0.19	15
395	0.20	0.07	0.10	60
396	0.36	0.29	0.32	28
397	0.00	0.00	0.00	38
398	0.27	0.08	0.12	37
399	0.00	0.00	0.00	59
400	0.50	0.35	0.41	57
401	0.11	0.09	0.10	45
402	0.00	0.00	0.00	131
403	0.03	0.13	0.05	47
404	0.05	0.04	0.04	26
405	0.00	0.00	0.00	105
406	0.29	0.03	0.06	60
407	0.21	0.14	0.17	43
408	0.00	0.00	0.00	62
409	0.31	0.46	0.37	50
410	0.60	0.38	0.46	8
411	0.06	0.23	0.10	30
412	0.00	0.00	0.00	59
413	0.27	0.15	0.19	40
414	0.03	0.06	0.04	53
415	0.46	0.44	0.45	52
416	0.62	0.17	0.27	47
417	0.60	0.25	0.35	12
418	0.10	0.27	0.15	62
419	0.14	0.12	0.13	24
420	0.04	0.02	0.03	53
421	0.05	0.03	0.03	40
422	0.07	0.02	0.03	56
423	0.00	0.00	0.00	63
424	0.12	0.22	0.15	27
425	0.08	0.07	0.07	14
426	0.07	0.02	0.04	41
427	0.31	0.29	0.30	48
428	0.19	0.65	0.29	37
429	0.57	0.25	0.35	52
430	0.45	0.20	0.28	50
431	0.07	0.05	0.06	55
432	0.00	0.00	0.00	49
433	0.10	0.04	0.06	46
434	0.13	0.67	0.22	3
435	0.27	0.12	0.17	121
436	0.00	0.00	0.00	68

437	0.89	0.47	0.62	70
438	0.00	0.00	0.00	34
439	0.00	0.00	0.00	62
440	0.05	0.13	0.07	45
441	0.00	0.00	0.00	3
442	0.00	0.00	0.00	5
443	0.00	0.00	0.00	19
444	0.09	0.05	0.07	38
445	1.00	1.00	1.00	1
446	0.00	0.00	0.00	60
447	0.20	0.25	0.22	20
448	0.25	0.03	0.05	38
449	0.00	0.00	0.00	15
450	0.01	0.05	0.02	22
451	0.48	0.49	0.48	43
452	0.00	0.00	0.00	110
453	0.29	0.10	0.14	21
454	0.04	0.21	0.07	29
455	0.67	0.41	0.51	49
456	0.00	0.00	0.00	3
457	0.89	0.62	0.73	52
458	0.03	0.02	0.03	41
459	0.00	0.00	0.00	42
460	0.12	0.09	0.10	32
461	0.83	0.23	0.36	22
462	0.07	0.17	0.10	6
463	0.12	0.20	0.15	5
464	0.01	0.03	0.01	29
465	0.05	0.06	0.05	16
466	0.27	0.07	0.11	44
467	0.94	0.25	0.39	65
468	0.17	0.13	0.15	38
469	0.29	0.04	0.06	55
470	0.30	0.16	0.20	58
471	0.50	0.38	0.43	8
472	0.00	0.00	0.00	64
473	0.50	0.21	0.30	14
474	0.00	0.00	0.00	34
475	0.88	0.61	0.72	36
476	0.19	0.16	0.17	19
477	0.51	0.36	0.42	50
478	0.51	0.35	0.42	63
479	0.27	0.25	0.26	24
480	0.00	0.00	0.00	54
481	0.15	0.49	0.23	68
482	0.40	0.29	0.33	14
483	0.00	0.00	0.00	31
484	0.86	0.13	0.23	45
485	0.00	0.00	0.00	49
486	0.25	0.04	0.07	25
487	0.73	0.22	0.34	50
488	0.80	0.36	0.50	11
489	0.45	0.21	0.29	67

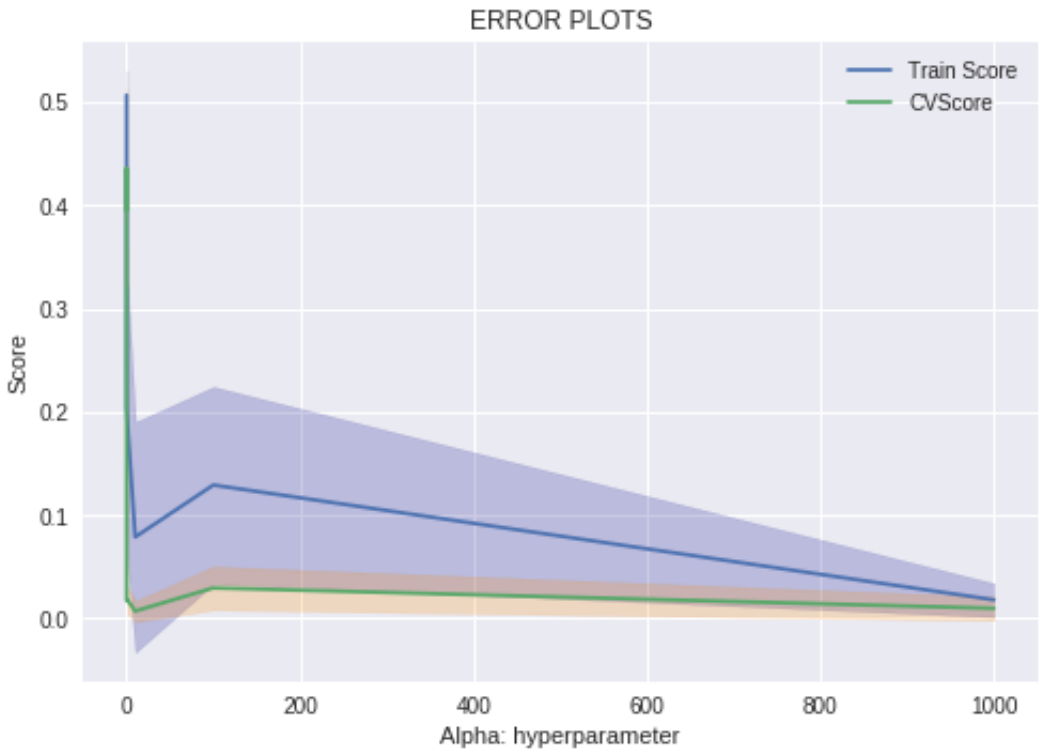
490	0.23	0.12	0.16	42
491	0.13	0.17	0.15	42
492	0.00	0.00	0.00	19
493	0.04	0.10	0.06	78
494	0.45	0.28	0.34	18
495	0.06	0.06	0.06	31
496	0.00	0.00	0.00	161
497	0.00	0.00	0.00	35
498	0.00	0.00	0.00	34
499	0.67	0.40	0.50	10
micro avg	0.48	0.34	0.40	87885
macro avg	0.35	0.25	0.27	87885
weighted avg	0.48	0.34	0.38	87885
samples avg	0.36	0.31	0.31	87885

Applying Linear SVM with OneVsRestClassifier

In [40]:
```
param={'estimator__alpha': [0.0001, 0.001, 0.01, 0.1, 1, 10,100,1000]}
classifier = OneVsRestClassifier(SGDClassifier(loss='hinge', penalty='
gsv = GridSearchCV(estimator = classifier, param_grid=param, cv=3, verb
gsv.fit(x_train_multilabel, y_train)


best_alpha = gsv.best_estimator_.get_params()['estimator__alpha']
print('value of alpha after hyperparameter tuning : ',best_alpha)
```

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

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent w
orkers.
[Parallel(n_jobs=-1)]: Done  24 out of  24 | elapsed: 106.9min finis
hed

value of alpha after hyperparameter tuning :  0.001
```

In [41]:
```
alpha = [0.0001, 0.001, 0.01, 0.1, 1, 10,100,1000]
train_score= gsv.cv_results_['mean_train_score']
train_score_std= gsv.cv_results_['std_train_score']
cv_score = gsv.cv_results_['mean_test_score']
cv_score_std= gsv.cv_results_['std_test_score']

plt.plot(alpha, train_score, label='Train Score')
# this code is copied from here: https://stackoverflow.com/a/48803361/
plt.gca().fill_between(alpha,train_score - train_score_std,train_score

plt.plot(alpha, cv_score, label='CVScore')
# this code is copied from here: https://stackoverflow.com/a/48803361/
plt.gca().fill_between(alpha,cv_score - cv_score_std,cv_score + cv_sco
plt.legend()
plt.xlabel("Alpha: hyperparameter")
plt.ylabel("Score")
plt.title("ERROR PLOTS")
plt.show()
```



In [42]:
```
start = datetime.now()
#best_alpha = gsv.best_estimator_.get_params()['estimator__alpha']
classifier = OneVsRestClassifier(SGDClassifier(loss='hinge', alpha=bes
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(p

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(p
print (metrics.classification_report(y_test, predictions))

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

```
Accuracy : 0.1461
Hamming loss  0.00372256
Micro-average quality numbers
Precision: 0.4590, Recall: 0.3302, F1-measure: 0.3841
Macro-average quality numbers
Precision: 0.2819, Recall: 0.2355, F1-measure: 0.2319
              precision    recall  f1-score   support

           0       0.68      0.62      0.65      2220
           1       0.41      0.12      0.19      3473
           2       0.69      0.35      0.46      3976
           3       0.77      0.66      0.71      2437
           4       0.52      0.49      0.50      2054
           5       0.65      0.57      0.61      2580
           6       0.73      0.56      0.63      1475
           7       0.43      0.23      0.30      1493
           8       0.34      0.57      0.42       957
           9       0.68      0.35      0.46      1781
          10       0.58      0.43      0.50      1568
          11       0.43      0.29      0.35      1477
          12       0.42      0.46      0.44       306
          13       0.43      0.14      0.22       916
          14       0.47      0.15      0.23      1161
          15       0.53      0.55      0.54       811
          16       0.70      0.61      0.65      1200
          17       0.63      0.62      0.62       686
          18       0.28      0.62      0.39       236
          19       0.61      0.61      0.61       680
          20       0.38      0.71      0.49      2233
          21       0.42      0.34      0.38      2785
          22       0.50      0.58      0.54       409
          23       0.58      0.20      0.30       533
          24       0.36      0.28      0.31       860
          25       0.25      0.42      0.31       391
          26       0.49      0.12      0.19       889
          27       0.53      0.36      0.43      1280
```

28	0.22	0.17	0.19	739
29	0.55	0.43	0.48	337
30	0.05	0.04	0.04	54
31	0.43	0.57	0.49	325
32	0.22	0.32	0.27	274
33	0.00	0.00	0.00	398
34	0.73	0.31	0.44	305
35	0.43	0.41	0.42	221
36	0.53	0.44	0.48	486
37	0.52	0.31	0.39	583
38	0.63	0.26	0.37	340
39	0.45	0.36	0.40	80
40	0.50	0.53	0.52	243
41	0.76	0.56	0.65	420
42	0.24	0.74	0.36	685
43	0.39	0.53	0.45	262
44	0.45	0.66	0.54	283
45	0.31	0.37	0.34	301
46	0.37	0.24	0.29	507
47	0.49	0.62	0.55	85
48	0.24	0.07	0.11	280
49	0.11	0.03	0.04	160
50	0.00	0.00	0.00	419
51	0.34	0.30	0.32	446
52	0.08	0.02	0.03	242
53	0.00	0.00	0.00	188
54	0.40	0.42	0.41	178
55	0.55	0.36	0.44	214
56	0.12	0.23	0.16	56
57	0.74	0.12	0.20	221
58	0.26	0.51	0.35	282
59	0.00	0.00	0.00	33
60	0.59	0.58	0.58	134
61	0.39	0.24	0.29	211
62	0.62	0.09	0.16	159
63	0.42	0.38	0.40	372
64	0.06	0.15	0.09	53
65	0.00	0.00	0.00	41
66	0.57	0.62	0.59	13
67	0.00	0.00	0.00	253
68	0.00	0.00	0.00	232
69	0.17	0.18	0.18	202
70	0.33	0.58	0.42	125
71	0.70	0.59	0.64	217
72	0.35	0.36	0.35	229
73	0.70	0.78	0.74	161
74	0.65	0.36	0.46	225
75	0.05	0.19	0.08	118
76	0.00	0.00	0.00	175
77	0.94	0.40	0.56	687
78	0.60	0.39	0.48	76
79	0.93	0.88	0.90	698
80	0.00	0.00	0.00	299

81	0.00	0.00	0.00	65
82	1.00	0.02	0.04	133
83	0.37	0.71	0.49	42
84	0.14	0.42	0.21	126
85	0.00	0.00	0.00	202
86	0.00	0.00	0.00	48
87	0.68	0.19	0.29	285
88	0.35	0.35	0.35	49
89	0.00	0.00	0.00	217
90	1.00	0.02	0.04	47
91	0.00	0.00	0.00	49
92	0.60	0.66	0.62	76
93	0.00	0.00	0.00	36
94	0.48	0.56	0.52	213
95	0.00	0.00	0.00	135
96	0.26	0.15	0.19	200
97	0.18	0.03	0.06	118
98	0.50	0.25	0.34	59
99	0.69	0.34	0.46	121
100	0.71	0.13	0.22	425
101	0.62	0.06	0.11	159
102	0.25	0.43	0.31	114
103	0.68	0.59	0.63	74
104	0.26	0.33	0.29	126
105	0.00	0.00	0.00	106
106	0.83	0.12	0.21	206
107	0.41	0.29	0.34	90
108	0.45	0.31	0.37	54
109	0.00	0.00	0.00	161
110	0.48	0.07	0.12	163
111	0.02	0.00	0.01	204
112	0.00	0.00	0.00	142
113	0.32	0.60	0.42	15
114	0.80	0.60	0.69	187
115	0.00	0.00	0.00	166
116	0.00	0.00	0.00	170
117	0.00	0.00	0.00	105
118	0.00	0.00	0.00	37
119	0.08	0.02	0.03	97
120	0.59	0.47	0.52	152
121	0.37	0.65	0.47	178
122	0.00	0.00	0.00	130
123	0.48	0.44	0.46	89
124	0.04	0.02	0.02	132
125	0.35	0.48	0.41	98
126	0.06	0.04	0.05	49
127	0.00	0.00	0.00	108
128	0.51	0.21	0.30	361
129	0.56	0.06	0.11	156
130	0.70	0.39	0.50	160
131	0.09	0.14	0.11	118
132	0.78	0.50	0.61	14
133	0.31	0.43	0.36	177

134	0.25	0.38	0.30	109
135	0.58	0.53	0.55	251
136	0.20	0.20	0.20	25
137	0.57	0.12	0.20	101
138	0.09	0.05	0.07	214
139	0.73	0.78	0.75	134
140	0.34	0.38	0.36	124
141	0.36	0.19	0.25	204
142	0.22	0.32	0.26	107
143	0.55	0.42	0.48	192
144	0.23	0.27	0.25	89
145	0.24	0.38	0.29	21
146	0.37	0.45	0.40	56
147	0.00	0.00	0.00	126
148	0.50	0.33	0.40	100
149	0.22	0.05	0.08	97
150	0.60	0.61	0.60	282
151	0.84	0.73	0.78	70
152	0.49	0.60	0.54	138
153	0.93	0.55	0.69	152
154	0.10	0.19	0.13	97
155	0.42	0.39	0.40	51
156	0.66	0.20	0.30	106
157	0.26	0.09	0.14	117
158	1.00	0.10	0.18	118
159	0.45	0.64	0.53	119
160	0.00	0.00	0.00	150
161	0.00	0.00	0.00	94
162	0.21	0.24	0.22	103
163	0.56	0.22	0.32	86
164	0.82	0.42	0.56	78
165	0.26	0.62	0.37	96
166	0.50	0.01	0.02	109
167	0.18	0.21	0.19	111
168	0.00	0.00	0.00	88
169	0.00	0.00	0.00	77
170	0.54	0.13	0.21	53
171	0.20	0.86	0.32	14
172	0.00	0.00	0.00	50
173	0.21	0.53	0.30	122
174	0.00	0.00	0.00	110
175	0.50	0.57	0.53	7
176	0.62	0.53	0.57	180
177	0.57	0.25	0.35	16
178	0.33	0.09	0.14	45
179	0.23	0.30	0.26	69
180	0.00	0.00	0.00	87
181	0.22	0.41	0.28	226
182	0.00	0.00	0.00	59
183	0.24	0.39	0.30	110
184	0.60	0.67	0.63	257
185	0.26	0.49	0.34	113
186	0.17	0.34	0.23	47

187	0.00	0.00	0.00	145
188	0.00	0.00	0.00	193
189	0.00	0.00	0.00	116
190	0.75	0.27	0.39	135
191	0.17	0.08	0.11	13
192	0.00	0.00	0.00	111
193	0.60	0.54	0.57	152
194	0.11	0.16	0.13	75
195	0.80	0.89	0.84	9
196	0.00	0.00	0.00	142
197	0.00	0.00	0.00	19
198	0.17	0.02	0.04	45
199	0.00	0.00	0.00	77
200	0.63	0.27	0.37	83
201	0.00	0.00	0.00	113
202	0.61	0.73	0.67	37
203	0.83	0.62	0.71	116
204	0.00	0.00	0.00	114
205	0.00	0.00	0.00	189
206	0.68	0.49	0.57	85
207	0.09	0.13	0.11	67
208	0.60	0.57	0.59	93
209	0.26	0.21	0.23	149
210	0.52	0.28	0.36	114
211	0.00	0.00	0.00	407
212	1.00	0.40	0.57	5
213	0.38	0.48	0.42	109
214	0.66	0.62	0.64	97
215	0.19	0.57	0.28	14
216	0.43	0.20	0.27	174
217	0.28	0.40	0.33	47
218	0.53	0.17	0.25	114
219	0.50	0.23	0.31	96
220	0.00	0.00	0.00	71
221	0.00	0.00	0.00	114
222	0.04	0.15	0.07	20
223	0.10	0.16	0.12	81
224	0.69	0.27	0.39	33
225	0.00	0.00	0.00	59
226	0.71	0.38	0.50	66
227	0.55	0.27	0.36	44
228	0.39	0.54	0.45	68
229	0.00	0.00	0.00	98
230	0.80	0.47	0.59	130
231	0.81	0.58	0.67	102
232	0.75	0.68	0.71	44
233	0.24	0.43	0.31	14
234	0.38	0.21	0.27	67
235	0.00	0.00	0.00	84
236	0.52	0.54	0.53	52
237	0.69	0.78	0.73	99
238	0.20	0.25	0.22	8
239	0.00	0.00	0.00	137

240	0.00	0.00	0.00	63
241	0.00	0.00	0.00	61
242	0.00	0.00	0.00	50
243	0.31	0.31	0.31	71
244	0.39	0.51	0.44	95
245	0.00	0.00	0.00	57
246	0.20	0.29	0.23	28
247	0.00	0.00	0.00	20
248	0.34	0.28	0.30	83
249	0.00	0.00	0.00	117
250	0.00	0.00	0.00	96
251	0.33	0.46	0.38	70
252	0.61	0.55	0.58	60
253	0.00	0.00	0.00	65
254	0.41	0.18	0.25	62
255	0.47	0.54	0.50	74
256	0.09	0.11	0.10	27
257	0.00	0.00	0.00	90
258	0.41	0.60	0.49	20
259	0.00	0.00	0.00	98
260	0.65	0.27	0.38	89
261	0.00	0.00	0.00	279
262	0.96	0.85	0.90	84
263	0.00	0.00	0.00	13
264	0.30	0.15	0.20	48
265	0.23	0.30	0.26	113
266	0.18	0.28	0.22	105
267	0.10	0.04	0.06	78
268	0.20	0.26	0.23	47
269	0.61	0.08	0.14	141
270	0.00	0.00	0.00	83
271	0.31	0.55	0.40	74
272	0.33	0.16	0.21	38
273	0.76	0.52	0.61	60
274	0.15	0.21	0.17	81
275	0.37	0.24	0.29	54
276	0.40	0.16	0.23	37
277	0.00	0.00	0.00	90
278	0.06	0.23	0.10	26
279	0.58	0.56	0.57	99
280	0.12	0.18	0.14	114
281	0.24	0.28	0.26	61
282	0.43	0.36	0.39	78
283	0.07	0.04	0.05	46
284	0.26	0.29	0.27	84
285	0.46	0.57	0.51	67
286	0.47	0.17	0.25	292
287	0.84	0.47	0.60	321
288	0.31	0.57	0.40	97
289	0.00	0.00	0.00	85
290	0.58	0.60	0.59	43
291	0.00	0.00	0.00	108
292	0.33	0.01	0.02	127

293	0.00	0.00	0.00	79
294	0.30	0.45	0.36	160
295	0.44	0.49	0.47	57
296	0.38	0.10	0.15	52
297	0.31	0.27	0.29	64
298	0.67	0.13	0.22	60
299	0.12	0.33	0.18	9
300	0.32	0.49	0.39	51
301	0.00	0.00	0.00	58
302	0.00	0.00	0.00	52
303	0.16	0.41	0.23	81
304	0.00	0.00	0.00	68
305	0.00	0.00	0.00	53
306	0.21	0.31	0.25	45
307	0.00	0.00	0.00	116
308	0.23	0.38	0.28	47
309	0.42	0.40	0.41	70
310	0.14	0.27	0.18	37
311	0.00	0.00	0.00	254
312	0.01	0.01	0.01	101
313	0.00	0.00	0.00	107
314	0.00	0.00	0.00	4
315	0.29	0.22	0.25	9
316	0.80	0.51	0.62	68
317	0.00	0.00	0.00	38
318	0.42	0.26	0.32	125
319	0.14	0.48	0.22	48
320	0.00	0.00	0.00	28
321	0.00	0.00	0.00	128
322	0.00	0.00	0.00	42
323	0.42	0.71	0.53	7
324	0.00	0.00	0.00	71
325	0.50	0.40	0.44	10
326	0.88	0.37	0.52	76
327	0.00	0.00	0.00	29
328	0.19	0.50	0.27	36
329	0.78	0.60	0.68	63
330	0.26	0.16	0.20	102
331	0.54	0.40	0.46	95
332	0.67	0.30	0.41	80
333	0.31	0.34	0.33	38
334	0.42	0.17	0.24	30
335	0.24	0.33	0.28	36
336	0.22	0.40	0.28	25
337	0.00	0.00	0.00	16
338	0.00	0.00	0.00	228
339	0.00	0.00	0.00	62
340	0.00	0.00	0.00	156
341	0.00	0.00	0.00	55
342	0.77	0.51	0.61	85
343	0.00	0.00	0.00	72
344	0.33	0.14	0.19	22
345	0.00	0.00	0.00	195

346	0.72	0.45	0.56	75
347	0.00	0.00	0.00	46
348	0.00	0.00	0.00	68
349	0.00	0.00	0.00	16
350	0.20	0.15	0.17	60
351	0.00	0.00	0.00	67
352	0.89	0.51	0.65	47
353	0.65	0.86	0.74	42
354	0.00	0.00	0.00	31
355	0.00	0.00	0.00	41
356	0.26	0.41	0.32	39
357	0.00	0.00	0.00	85
358	0.72	0.77	0.74	83
359	0.31	0.41	0.35	27
360	0.32	0.26	0.28	113
361	0.02	0.09	0.03	11
362	0.00	0.00	0.00	54
363	0.13	0.02	0.04	98
364	0.46	0.50	0.48	22
365	0.58	0.20	0.30	35
366	0.00	0.00	0.00	10
367	0.00	0.00	0.00	189
368	0.00	0.00	0.00	97
369	0.14	0.02	0.04	45
370	0.47	0.43	0.45	72
371	0.00	0.00	0.00	15
372	0.19	0.38	0.25	21
373	0.00	0.00	0.00	32
374	0.00	0.00	0.00	65
375	0.75	0.67	0.71	27
376	0.50	0.33	0.40	3
377	0.00	0.00	0.00	95
378	0.92	0.19	0.32	62
379	0.62	0.72	0.67	53
380	0.21	0.49	0.30	53
381	0.00	0.00	0.00	93
382	0.44	0.36	0.40	11
383	0.04	0.04	0.04	82
384	0.60	0.52	0.56	52
385	0.06	0.18	0.09	17
386	0.13	0.46	0.20	13
387	0.06	0.38	0.10	8
388	0.04	0.06	0.04	106
389	0.77	0.69	0.73	68
390	0.40	0.50	0.44	4
391	0.00	0.00	0.00	92
392	0.00	0.00	0.00	34
393	0.00	0.00	0.00	35
394	0.22	0.27	0.24	15
395	0.00	0.00	0.00	60
396	0.04	0.18	0.06	28
397	0.00	0.00	0.00	38
398	1.00	0.03	0.05	37

399	0.00	0.00	0.00	59
400	0.67	0.11	0.18	57
401	0.02	0.04	0.03	45
402	0.00	0.00	0.00	131
403	0.44	0.43	0.43	47
404	0.00	0.00	0.00	26
405	0.00	0.00	0.00	105
406	0.00	0.00	0.00	60
407	0.00	0.00	0.00	43
408	0.00	0.00	0.00	62
409	0.29	0.38	0.33	50
410	0.38	0.38	0.38	8
411	0.18	0.23	0.20	30
412	0.00	0.00	0.00	59
413	0.00	0.00	0.00	40
414	0.00	0.00	0.00	53
415	0.23	0.46	0.31	52
416	0.00	0.00	0.00	47
417	0.15	0.33	0.21	12
418	0.41	0.29	0.34	62
419	0.27	0.29	0.28	24
420	0.00	0.00	0.00	53
421	0.00	0.00	0.00	40
422	0.00	0.00	0.00	56
423	0.05	0.06	0.05	63
424	0.00	0.00	0.00	27
425	0.00	0.00	0.00	14
426	0.00	0.00	0.00	41
427	0.34	0.21	0.26	48
428	0.52	0.46	0.49	37
429	0.35	0.50	0.41	52
430	0.28	0.32	0.30	50
431	0.00	0.00	0.00	55
432	0.00	0.00	0.00	49
433	0.00	0.00	0.00	46
434	0.07	0.33	0.12	3
435	0.00	0.00	0.00	121
436	0.00	0.00	0.00	68
437	0.68	0.39	0.49	70
438	0.00	0.00	0.00	34
439	0.00	0.00	0.00	62
440	0.00	0.00	0.00	45
441	0.00	0.00	0.00	3
442	0.00	0.00	0.00	5
443	0.36	0.26	0.30	19
444	0.00	0.00	0.00	38
445	0.10	1.00	0.18	1
446	0.00	0.00	0.00	60
447	0.00	0.00	0.00	20
448	0.00	0.00	0.00	38
449	0.00	0.00	0.00	15
450	0.00	0.00	0.00	22
451	0.46	0.37	0.41	43

452	0.00	0.00	0.00	110
453	0.00	0.00	0.00	21
454	0.00	0.00	0.00	29
455	0.44	0.55	0.49	49
456	0.00	0.00	0.00	3
457	0.82	0.71	0.76	52
458	0.00	0.00	0.00	41
459	0.00	0.00	0.00	42
460	0.00	0.00	0.00	32
461	0.75	0.27	0.40	22
462	0.00	0.00	0.00	6
463	1.00	0.20	0.33	5
464	0.00	0.00	0.00	29
465	0.00	0.00	0.00	16
466	0.00	0.00	0.00	44
467	0.00	0.00	0.00	65
468	0.07	0.11	0.08	38
469	0.00	0.00	0.00	55
470	0.00	0.00	0.00	58
471	0.29	0.62	0.40	8
472	0.00	0.00	0.00	64
473	0.25	0.14	0.18	14
474	0.00	0.00	0.00	34
475	0.85	0.31	0.45	36
476	0.00	0.00	0.00	19
477	0.00	0.00	0.00	50
478	0.00	0.00	0.00	63
479	0.00	0.00	0.00	24
480	0.00	0.00	0.00	54
481	0.42	0.37	0.39	68
482	1.00	0.29	0.44	14
483	0.00	0.00	0.00	31
484	0.00	0.00	0.00	45
485	0.00	0.00	0.00	49
486	0.00	0.00	0.00	25
487	0.59	0.38	0.46	50
488	0.71	0.45	0.56	11
489	0.39	0.42	0.40	67
490	0.00	0.00	0.00	42
491	0.00	0.00	0.00	42
492	0.67	0.53	0.59	19
493	0.61	0.18	0.28	78
494	0.37	0.56	0.44	18
495	0.00	0.00	0.00	31
496	0.00	0.00	0.00	161
497	0.00	0.00	0.00	35
498	0.00	0.00	0.00	34
499	0.22	0.20	0.21	10
micro avg	0.46	0.33	0.38	87885
macro avg	0.28	0.24	0.23	87885
weighted avg	0.43	0.33	0.35	87885
samples avg	0.35	0.31	0.30	87885

```
Time taken to run this cell : 0:06:29.179095
```

Precedure Followed

1. Countvectorizing of data with 25000 features and ngram from 1 to 4
2. Hyperparameter tuning is done for both th algorithms
3. The best parameter found for both the algorithm is used to train the model
4. For Logistic Regression, the best Micro F1 Score : 0.3981
5. For Linear SVM, the best Micro F1 Score : 0.3841
6. Linear SVM performs slightly well when compared with Logistic Regression.

In [2]:
```
from prettytable import PrettyTable
x = PrettyTable()

x.field_names = ["FEATURIZATION", "MODEL", "HAMMING_LOSS","MICRO_f1_SC

x.add_row(["BOW(4 gram)", "Logistic Regression", 0.00359352, 0.3981])
x.add_row(["", "Linear SVM", 0.00372256, 0.3841])

print('\t\t\tPerformance Table')
print(x)
```

```
                        Performance Table
+---------------+---------------------+-------------+-------------
--+
| FEATURIZATION |        MODEL        | HAMMING_LOSS | MICRO_f1_SCOR
E |
+---------------+---------------------+-------------+-------------
--+
|  BOW(4 gram)  | Logistic Regression |  0.00359352  |     0.3981
|
|               |      Linear SVM     |  0.00372256  |     0.3841
|
+---------------+---------------------+-------------+-------------
--+
```