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
```

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

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



\n\n

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



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


\n\n

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

The output is not coming,can anyone correct the code or tell me what\'s w
rong?
\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', 'Tags'], chu
        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(*)'].values[0])
    con.close()
    print("Time taken to count the number of rows :", datetime.now() - start)
else:
    print("Please download the train.db file from drive or run the above cell to
```

```
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(*) as cnt_dup
    con.close()
    print("Time taken to run this cell :", datetime.now() - start)
else:
    print("Please download the train.db file from drive or run the first to genar
```

```
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]- df_no_du
```

```
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.split(" ")))
# adding a new feature number of tags per question
print("Time taken to run this cell :", datetime.now() - start)
df_no_dup.head()
```

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

Out[9]:

	Title	Body	Tags	cnt_dup	tag_
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""", con)
    #Always remember to close the database
    con.close()

    # Let's now drop unwanted column.
    tag_data.drop(tag_data.index[0], inplace=True)
    #Printing first 5 columns from our data frame
    tag_data.head()
    print("Time taken to run this cell :", datetime.now() - start)
else:
    print("Please download the train.db file from drive or run the above cells to
```

```
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 strings.
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', '.aspxauth', '.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-matrix-elemen
#Lets now store the document term matrix in a dictionary.
freqs = tag_dtm.sum(axis=0).A1
result = dict(zip(tags, freqs))
```

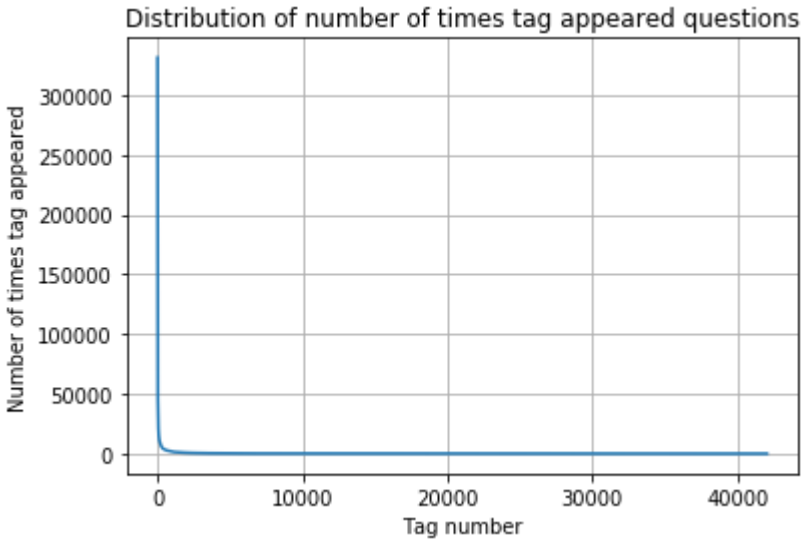
In [0]:
```
#Saving this dictionary to csv files.
if not os.path.isfile('tag_counts_dict_dtm.csv'):
    with open('tag_counts_dict_dtm.csv', 'w') as csv_file:
        writer = csv.writer(csv_file)
        for key, value in result.items():
            writer.writerow([key, value])
tag_df = pd.read_csv("tag_counts_dict_dtm.csv",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

Out[17]:

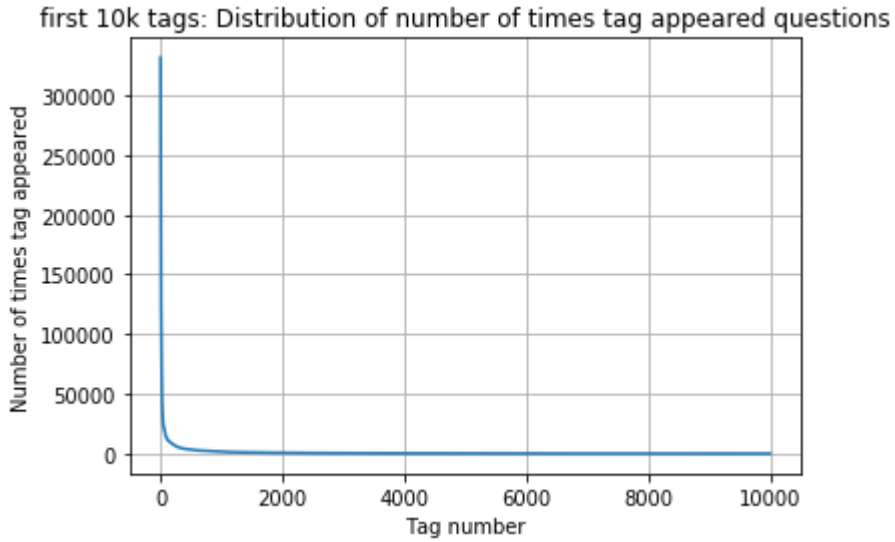
	Tags	Counts
0	.a	18
1	.app	37
2	.asp.net-mvc	1
3	.aspxauth	21
4	.bash-profile	138

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()
```



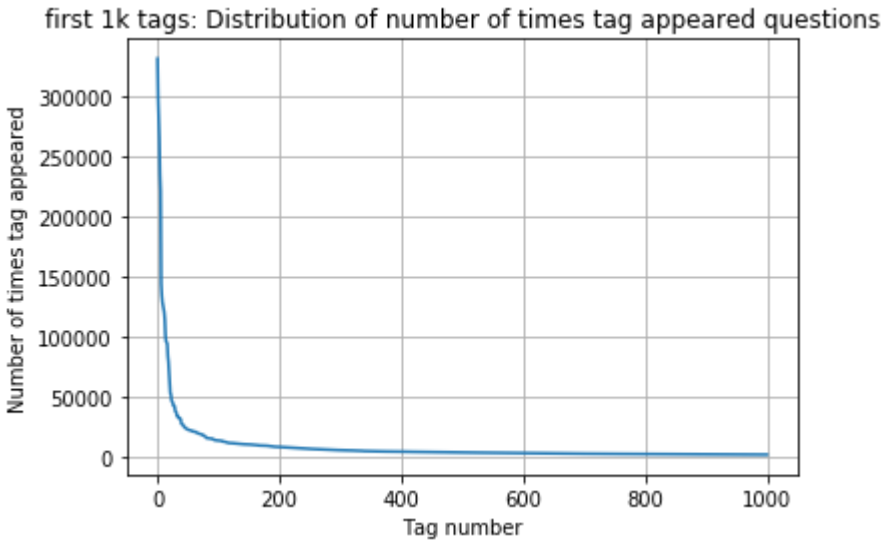
In [0]:
```
plt.plot(tag_counts[0:10000])
plt.title('first 10k tags: Distribution of number of times tag appeared questions
plt.grid()
plt.xlabel("Tag number")
plt.ylabel("Number of times tag appeared")
plt.show()
print(len(tag_counts[0:10000:25]), tag_counts[0:10000:25])
```




```
400 [331505  44829  22429  17728  13364  11162  10029   9148   8054   7151
   6466   5865   5370   4983   4526   4281   4144   3929   3750   3593
   3453   3299   3123   2989   2891   2738   2647   2527   2431   2331
   2259   2186   2097   2020   1959   1900   1828   1770   1723   1673
   1631   1574   1532   1479   1448   1406   1365   1328   1300   1266
   1245   1222   1197   1181   1158   1139   1121   1101   1076   1056
   1038   1023   1006    983    966    952    938    926    911    891
    882    869    856    841    830    816    804    789    779    770
    752    743    733    725    712    702    688    678    671    658
    650    643    634    627    616    607    598    589    583    577
    568    559    552    545    540    533    526    518    512    506
    500    495    490    485    480    477    469    465    457    450
    447    442    437    432    426    422    418    413    408    403
    398    393    388    385    381    378    374    370    367    365
    361    357    354    350    347    344    342    339    336    332
    330    326    323    319    315    312    309    307    304    301
    299    296    293    291    289    286    284    281    278    276
    275    272    270    268    265    262    260    258    256    254
    252    250    249    247    245    243    241    239    238    236
    234    233    232    230    228    226    224    222    220    219
    217    215    214    212    210    209    207    205    204    203
    201    200    199    198    196    194    193    192    191    189
    188    186    185    183    182    181    180    179    178    177
    175    174    172    171    170    169    168    167    166    165
    164    162    161    160    159    158    157    156    156    155
    154    153    152    151    150    149    149    148    147    146
    145    144    143    142    142    141    140    139    138    137
    137    136    135    134    134    133    132    131    130    130
    129    128    128    127    126    126    125    124    124    123
    123    122    122    121    120    120    119    118    118    117
    117    116    116    115    115    114    113    113    112    111
```

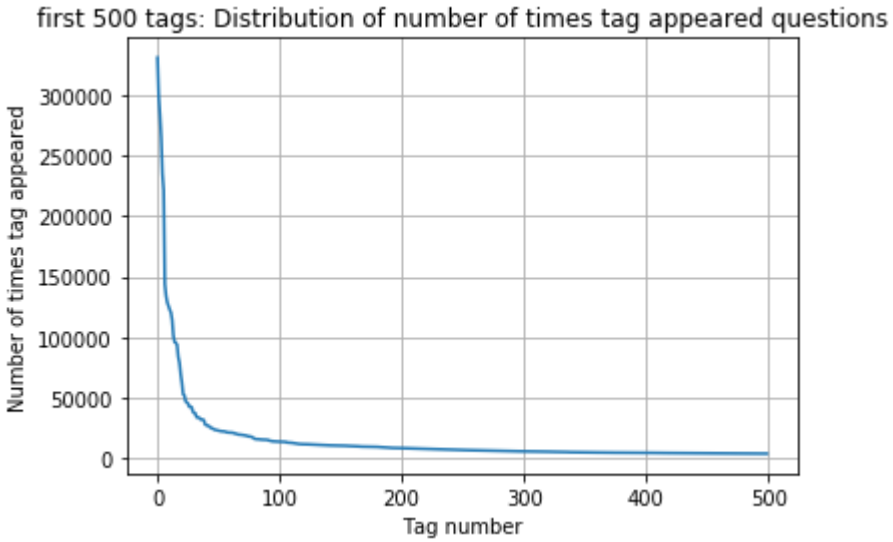
```
 111   110   109   109   108   108   107   106   106   106
 105   105   104   104   103   103   102   102   101   101
 100   100    99    99    98    98    97    97    96    96
  95    95    94    94    93    93    93    92    92    91
  91    90    90    89    89    88    88    87    87    86
  86    86    85    85    84    84    83    83    83    82
  82    82    81    81    80    80    80    79    79    78
  78    78    78    77    77    76    76    76    75    75
  75    74    74    74    73    73    73    73    72    72]
```

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



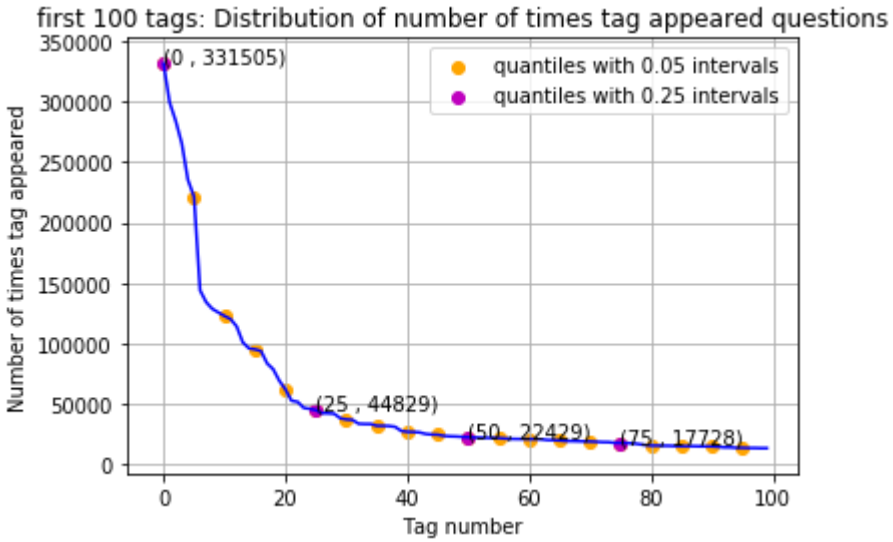

```
200 [331505 221533 122769  95160  62023  44829  37170  31897  26925  24537
  22429  21820  20957  19758  18905  17728  15533  15097  14884  13703
  13364  13157  12407  11658  11228  11162  10863  10600  10350  10224
  10029   9884   9719   9411   9252   9148   9040   8617   8361   8163
   8054   7867   7702   7564   7274   7151   7052   6847   6656   6553
   6466   6291   6183   6093   5971   5865   5760   5577   5490   5411
   5370   5283   5207   5107   5066   4983   4891   4785   4658   4549
   4526   4487   4429   4335   4310   4281   4239   4228   4195   4159
   4144   4088   4050   4002   3957   3929   3874   3849   3818   3797
   3750   3703   3685   3658   3615   3593   3564   3521   3505   3483
   3453   3427   3396   3363   3326   3299   3272   3232   3196   3168
   3123   3094   3073   3050   3012   2989   2984   2953   2934   2903
   2891   2844   2819   2784   2754   2738   2726   2708   2681   2669
   2647   2621   2604   2594   2556   2527   2510   2482   2460   2444
   2431   2409   2395   2380   2363   2331   2312   2297   2290   2281
   2259   2246   2222   2211   2198   2186   2162   2142   2132   2107
   2097   2078   2057   2045   2036   2020   2011   1994   1971   1965
   1959   1952   1940   1932   1912   1900   1879   1865   1855   1841
   1828   1821   1813   1801   1782   1770   1760   1747   1741   1734
   1723   1707   1697   1688   1683   1673   1665   1656   1646   1639]
```

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




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

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

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

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




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

In [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_10k)))
# Store tags greater than 100K in one list
lst_tags_gt_100k = tag_df[tag_df.Counts>100000].Tags
#Print the length of the list.
print ('{} Tags are used more than 100000 times'.format(len(lst_tags_gt_100k)))
```

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

Observations:

1. There are total 153 tags which are used more than 10000 times.
2. 14 tags are used more than 100000 times.
3. Most frequent tag (i.e. c#) is used 331505 times.

4. Since some tags occur much more frequenctly than others, Micro-averaged F1-score is the appropriate metric for this probelm.

3.2.4 Tags Per Question

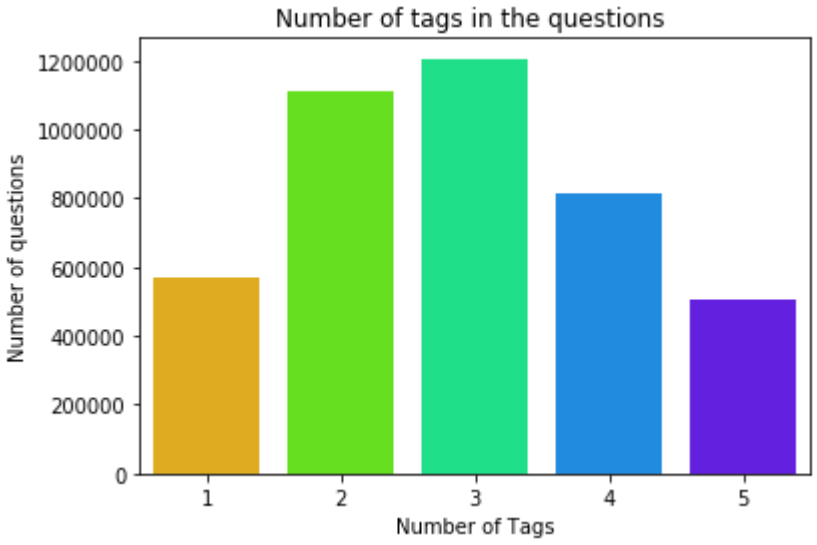
In [0]:
```
#Storing the count of tag in each question in list 'tag_count'
tag_quest_count = tag_dtm.sum(axis=1).tolist()
#Converting list of lists into single list, we will get [[3], [4], [2], [2], [3]]
tag_quest_count=[int(j) for i in tag_quest_count for j in i]
print ('We have total {} datapoints.'.format(len(tag_quest_count)))

print(tag_quest_count[:5])
```

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

In [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)*1.0)/len(ta
```

```
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()
```



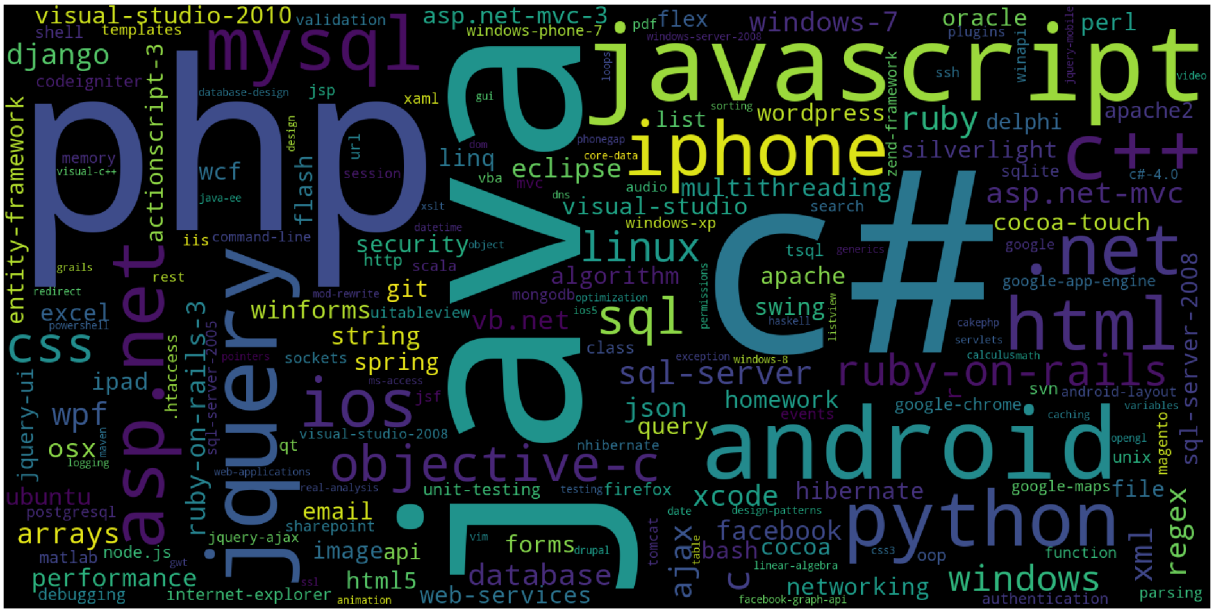
Observations:

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)
```

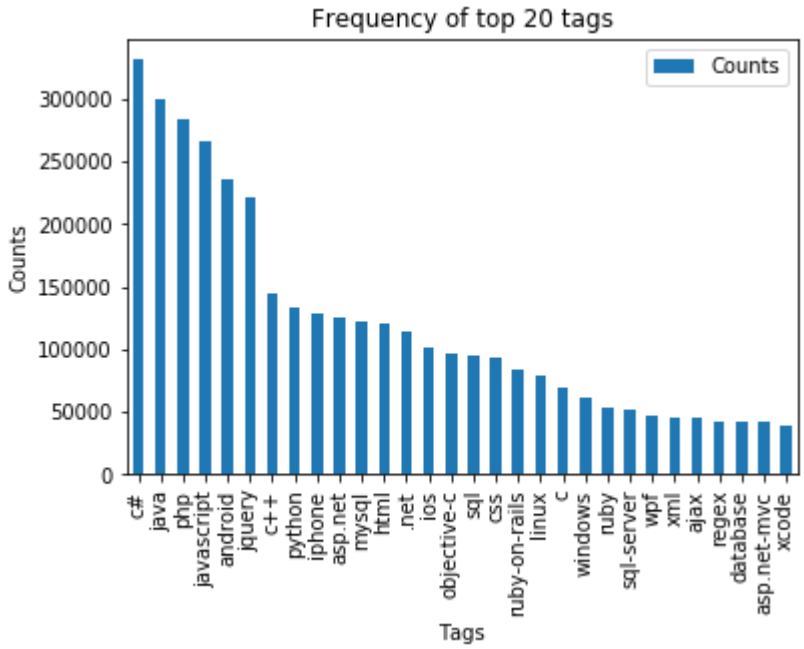



```
Time taken to run this cell : 0:00:05.470788
```

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 (question tex
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-sqlite-tab
start = datetime.now()
read_db = 'train_no_dup.db'
write_db = 'Processed.db'
if os.path.isfile(read_db):
    conn_r = create_connection(read_db)
    if conn_r is not None:
        reader =conn_r.cursor()
        reader.execute("SELECT Title, Body, Tags From no_dup_train ORDER BY RANDOM

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-sqlite-table/

start = datetime.now()
preprocessed_data_list=[]
reader.fetchone()
questions_with_code=0
len_pre=0
len_post=0
questions_proccesed = 0
for row in reader:

    is_code = 0

    title, question, tags = row[0], row[1], row[2]

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

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

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

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

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

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

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

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

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

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 locks
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 10")
        print("Questions after preprocessed")
        print('='*100)
        reader.fetchone()
        for row in reader:
            print(row)
            print('-'*100)
conn_r.commit()
conn_r.close()
```

```
Questions after preprocessed
====================================================================================================
('ef code first defin one mani relationship differ key troubl defin one zero ma
ni relationship entiti ef object model look like use fluent api object composit
pk defin batch id batch detail id use fluent api object composit pk defin batch
detail id compani id map exist databas tpt basic idea submittedtransact zero ma
ni submittedsplittransact associ navig realli need one way submittedtransact su
bmittedsplittransact need dbcontext class onmodelcr overrid map class lazi load
occur submittedtransact submittedsplittransact help would much appreci edit tak
en advic made follow chang dbcontext class ad follow onmodelcr overrid must mis
s someth get follow except thrown submittedtransact key batch id batch detail i
d zero one mani submittedsplittransact key batch detail id compani id rather as
sum convent creat relationship 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 structur list po
ssibl candid solut list possibl coordin matrix wan na choos one candid compar p
ossibl candid element equal wan na delet coordin call function skasuj look like
ni knowledg haskel cant see what wrong',)
----------------------------------------------------------------------------------------------------
('step plan move one isp anoth one work busi plan switch isp realli soon need c
hang lot inform dns wan wan wifi question guy help mayb peopl plan correct chan
g current isp new one first dns know receiv new ip isp major chang need take co
nsider exchang server owa vpn two site link wireless connect km away citrix ser
ver vmware exchang domain control link place import server crucial step inform
need know avoid downtim busi regard ndavid',)
----------------------------------------------------------------------------------------------------
('use ef migrat creat databas googl migrat tutori af first run applic creat dat
abas ef enabl migrat way creat databas migrat rune applic tri',)
----------------------------------------------------------------------------------------------------
('magento unit test problem magento site recent look way check integr magento s
ite given point unit test jump one method would assum would big job write whole
lot test check everyth site work anyon involv unit test magento advis follow po
ssibl test whole site custom modul nis exampl test would amaz given site heavil
```

```
i link databas would nbe possibl fulli test site without disturb databas better
way automaticlli check integr magento site say integr realli mean fault site sh
ip payment etc work correct',)
--------------------------------------------------------------------------------
--------------------
('find network devic without bonjour write mac applic need discov mac pcs iphon
ipad connect wifi network bonjour seem reason choic turn problem mani type rout
er mine exampl work block bonjour servic need find ip devic tri connect applic
specif port determin process run best approach accomplish task without violat a
pp 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 nth
en use help schema',)
--------------------------------------------------------------------------------
--------------------
('insert data mysql php powerpoint event powerpoint present run continu way upd
at 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 FROM Quest
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

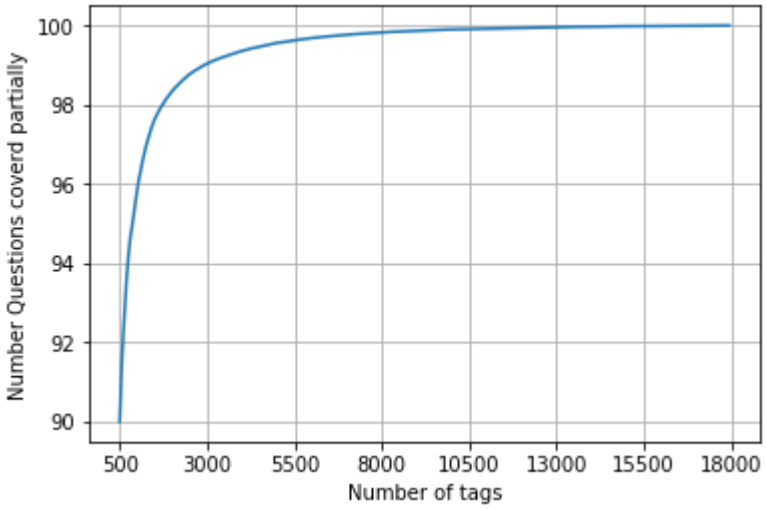
In [0]:
```
# binary='true' will give a binary vectorizer
vectorizer = CountVectorizer(tokenizer = lambda x: x.split(), binary='true')
multilabel_y = vectorizer.fit_transform(preprocessed_data['tags'])
```

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

In [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=True)
    multilabel_yn=multilabel_y[:,sorted_tags_i[:n]]
    return multilabel_yn

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

In [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_fn(i))/tot
```

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, minimun is 50(
print("with ",5500,"tags we are covering ",questions_explained[50],"% of question
```




```
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_explained_fn(5500),
```

```
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_yx.shape[1
```

```
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, smooth_idf=True
                             tokenizer = lambda x: x.split(), sublinear_tf=False,
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_train.shape
print("Dimensions of test data X:",x_test_multilabel.shape,"Y:",y_test.shape)
```

```
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-label-classi
#https://stats.stackexchange.com/questions/117796/scikit-multi-label-classificati
# classifier = LabelPowerset(GaussianNB())
"""
from skmultilearn.adapt import MLkNN
classifier = MLkNN(k=21)

# train
classifier.fit(x_train_multilabel, y_train)

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

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

Out[92]:
```
"\nfrom skmultilearn.adapt import MLkNN\nclassifier = MLkNN(k=21)\n\n# train\nc
lassifier.fit(x_train_multilabel, y_train)\n\n# predict\npredictions = classifi
er.predict(x_test_multilabel)\nprint(accuracy_score(y_test,predictions))\nprint
(metrics.f1_score(y_test, predictions, average = 'macro'))\nprint(metrics.f1_sc
ore(y_test, predictions, average = 'micro'))\nprint(metrics.hamming_loss(y_tes
t,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_with_equal_
# This takes about 6-7 hours to run.
classifier = OneVsRestClassifier(SGDClassifier(loss='log', alpha=0.00001, penalty
classifier.fit(x_train_multilabel, y_train)
predictions = classifier.predict(x_test_multilabel)

print("accuracy :",metrics.accuracy_score(y_test,predictions))
print("macro f1 score :",metrics.f1_score(y_test, predictions, average = 'macro')
print("micro f1 scoore :",metrics.f1_score(y_test, predictions, average = 'micro'
print("hamming loss :",metrics.hamming_loss(y_test,predictions))
print("Precision recall report :\n",metrics.classification_report(y_test, predict
```

```
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 (question tex
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-sqlite-tab

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

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-sqlite-table/
start = datetime.now()
preprocessed_data_list=[]
reader.fetchone()
questions_with_code=0
len_pre=0
len_post=0
questions_proccesed = 0
for row in reader:

    is_code = 0

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

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

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

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

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

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

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

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

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

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

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

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

print( "Avg. length of questions(Title+Body) before processing: %d"%no_dup_avg_le
print( "Avg. length of questions(Title+Body) after processing: %d"%no_dup_avg_len
print ("Percent of questions containing code: %d"%((questions_with_code*100.0)/qu
```

```
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 database locks
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 10")
        print("Questions after preprocessed")
        print('='*100)
        reader.fetchone()
        for row in reader:
            print(row)
            print('-'*100)
conn_r.commit()
conn_r.close()
```

```
Questions after preprocessed
====================================================================================================
('dynam datagrid bind silverlight dynam datagrid bind silverlight dynam datagri
d bind silverlight bind datagrid dynam code wrote code debug code block seem bi
nd correct grid come column form come grid column although necessari bind nthan
k repli advance..',)
----------------------------------------------------------------------------------------------------
('java.lang.noclassdeffounderror javax servlet jsp tagext taglibraryvalid java.
lang.noclassdeffounderror javax servlet jsp tagext taglibraryvalid java.lang.no
classdeffounderror javax servlet jsp tagext taglibraryvalid follow guid link in
stal jstl got follow error tri launch jsp page java.lang.noclassdeffounderror j
avax servlet jsp tagext taglibraryvalid taglib declar instal jstl 1.1 tomcat we
bapp tri project 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 descriptor index java.sql.sql
except microsoft odbc driver manag invalid descriptor index use follow code dis
play 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 use curl someth like way better',)
----------------------------------------------------------------------------------------------------
('btnadd click event open two window record ad btnadd click event open two wind
ow record ad btnadd click event open two window record ad open window search.as
px use code hav add button search.aspx nwhen insert record btnadd click event o
pen anoth window nafter insert record close window',)
----------------------------------------------------------------------------------------------------
('sql inject issu prevent correct form submiss php sql inject issu prevent corr
ect form submiss php sql inject issu prevent correct form submiss php check eve
ryth think make sure input field safe type sql inject good news safe bad news o
ne tag mess form submiss place even touch life figur exact html use templat fil
e forgiv okay entir php script get execut see data post none forum field post p
roblem use someth titl field none data get post current use print post see subm
it noth work flawless statement though also mention script work flawless local
machin use host come across problem state list input test mess',)
```

```
--------------------------------------------------------------------------------
--------------------
('countabl subaddit lebesgu measur countabl subaddit lebesgu measur countabl su
baddit lebesgu measur let lbrace rbrace sequenc set sigma -algebra mathcal want
show left bigcup right leq sum left right countabl addit measur defin set sigma
algebra mathcal think use monoton properti somewher proof start appreci littl h
elp nthank ad han answer make follow addit construct given han answer clear big
cup bigcup cap emptyset neq left bigcup right left bigcup right sum left right
also construct subset monoton left right leq left right final would sum leq sum
result follow',)
--------------------------------------------------------------------------------
--------------------
('hql equival sql queri hql equival sql queri hql equival sql queri hql queri r
eplac name class properti name error occur hql error',)
--------------------------------------------------------------------------------
--------------------
('undefin symbol architectur i386 objc class skpsmtpmessag referenc error undef
in symbol architectur i386 objc class skpsmtpmessag referenc error undefin symb
ol architectur i386 objc class skpsmtpmessag referenc error import framework se
nd email applic background import framework i.e skpsmtpmessag somebodi suggest
get error collect2 ld return exit status import framework correct sorc taken fr
amework follow mfmailcomposeviewcontrol question lock field updat answer drag d
rop 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 FROM Quest
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
```

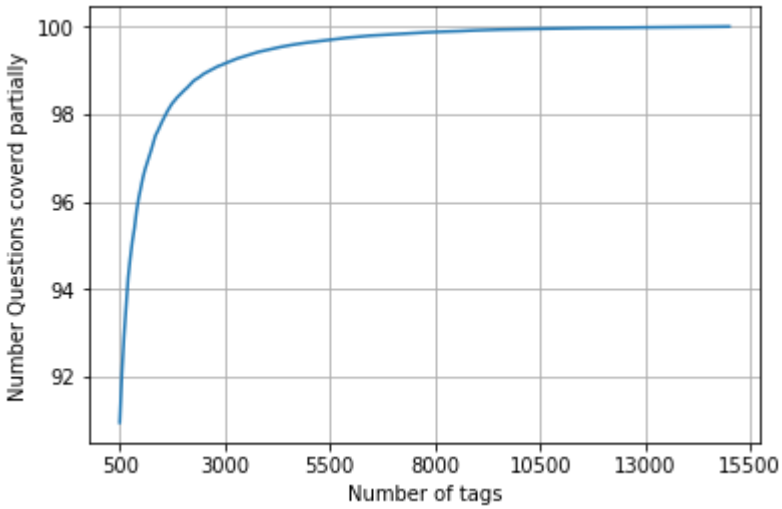
Converting string Tags to multilable output variables

In [0]:
```
vectorizer = CountVectorizer(tokenizer = lambda x: x.split(), binary='true')
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_fn(i))/tot
```

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, minimun is 500
print("with ",5500,"tags we are covering ",questions_explained[50],"% of question
print("with ",500,"tags we are covering ",questions_explained[0],"% of questions"
```




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

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

```
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 Tfidf vectorizer

In [0]:
```
start = datetime.now()
vectorizer = TfidfVectorizer(min_df=0.00009, max_features=200000, smooth_idf=True
                             tokenizer = lambda x: x.split(), sublinear_tf=False,
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_train.shape
print("Dimensions of test data X:",x_test_multilabel.shape,"Y:",y_test.shape)
```

```
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=0.00001, penalty
classifier.fit(x_train_multilabel, y_train)
predictions = classifier.predict (x_test_multilabel)


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


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

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

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

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

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_jobs=-1)
classifier_2.fit(x_train_multilabel, y_train)
predictions_2 = classifier_2.predict(x_test_multilabel)
print("Accuracy :",metrics.accuracy_score(y_test, predictions_2))
print("Hamming loss ",metrics.hamming_loss(y_test,predictions_2))


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

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

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

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

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)

In [1]:
```
!pip install kaggle
from google.colab import files
files.upload()
```

```
Requirement already satisfied: kaggle in /usr/local/lib/python3.6/dist-packages
(1.5.3)
Requirement already satisfied: urllib3<1.25,>=1.21.1 in /usr/local/lib/python3.
6/dist-packages (from kaggle) (1.22)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.6/dist-packa
ges (from kaggle) (1.11.0)
Requirement already satisfied: certifi in /usr/local/lib/python3.6/dist-package
s (from kaggle) (2019.3.9)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.6/dist
-packages (from kaggle) (2.5.3)
Requirement already satisfied: requests in /usr/local/lib/python3.6/dist-packag
es (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/python3.6/dist-
packages (from kaggle) (3.0.0)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /usr/local/lib/python3.
6/dist-packages (from requests->kaggle) (3.0.4)
Requirement already satisfied: idna<2.7,>=2.5 in /usr/local/lib/python3.6/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 file chosen

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[1]:
```
{'kaggle.json': b'{"username":"pankajkarki","key":"563115ce1ea9892ab835dfbe5b8a
cba1"}'}
```

In [2]:
```
!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
 99% 473M/478M [00:06<00:00, 97.1MB/s]
100% 478M/478M [00:06<00:00, 80.8MB/s]
kaggle.json  sample_data  stackoverflow.zip
```

Loading files

In [3]:
```
!unzip stackoverflow.zip
```

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

In [3]:

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

    return None

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

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

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

sql_create_table = """CREATE TABLE IF NOT EXISTS QuestionsProcessed (question tex
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 FROM Quest
conn_r.commit()
conn_r.close()
```

Due to memory error I took 25K points

In [29]:
```
# Sampling data beacause of memory error we are getting while featurizing 4 gram.
preprocessed_data = preprocessed_data.iloc[:250000,:]
print(preprocessed_data.shape)
preprocessed_data.head()
```

```
(250000, 2)
```

Out[29]:

	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 [30]:
```
print("number of data points in sample :", preprocessed_data.shape[0])
print("number of dimensions :", preprocessed_data.shape[1])
```

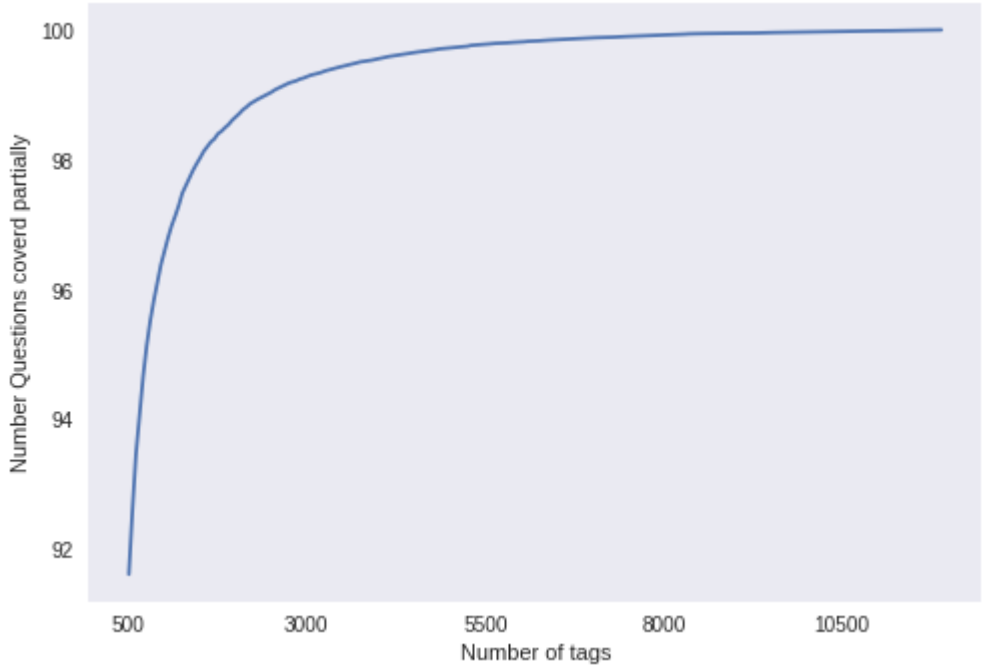
```
number of data points in sample : 250000
number of dimensions : 2
```

In [0]:
```
# binary='true' will give a binary vectorizer
vectorizer = CountVectorizer(tokenizer = lambda x: x.split(), binary='true')
multilabel_y = 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=True)
    multilabel_yn=multilabel_y[:,sorted_tags_i[:n]]
    return multilabel_yn

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

In [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_fn(i))/tot
```

In [34]:
```
fig, ax = plt.subplots()
ax.plot(questions_explained)
xlabel = list(500+np.array(range(-50,450,50))*50)
ax.set_xticklabels(xlabel)
plt.xlabel("Number of tags")
plt.ylabel("Number Questions coverd partially")
plt.grid()
plt.show()
# you can choose any number of tags based on your computing power, minimun is 500
print("with ",5500,"tags we are covering ",questions_explained[50],"% of question
print("with ",500,"tags we are covering ",questions_explained[0],"% of questions"
```




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

In [35]:
```
# we will be taking 500 tags
multilabel_yx = tags_to_choose(500)
print("number of questions that are not covered :", questions_explained_fn(500),"
```

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

EDA on preprossed_data

In [47]:
```
print("Number of data points :", multilabel_y.shape[0])
print("Number of unique tags :", multilabel_y.shape[1])
```

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

In [38]:
```
#'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', '.class-file',
'.cs-file', '.doc', '.ds-store', '.each', '.emf', '.exe']
```

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

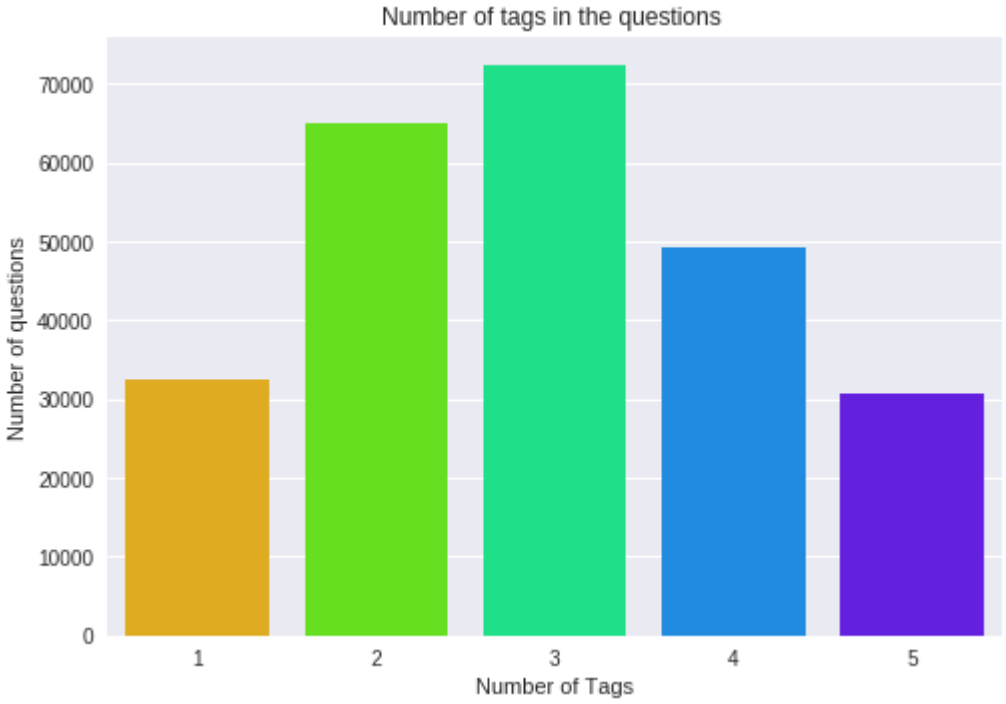
In [41]:
```
#Storing the count of tag in each question in list 'tag_count'
tag_quest_count = multilabel_y.sum(axis=1).tolist()
#Converting each value in the 'tag_quest_count' to integer.
tag_quest_count=[int(j) for i in tag_quest_count for j in i]
print ('We have total {} datapoints.'.format(len(tag_quest_count)))

print(tag_quest_count[:5])
```

```
We have total 250000 datapoints.
[3, 4, 2, 2, 3]
```

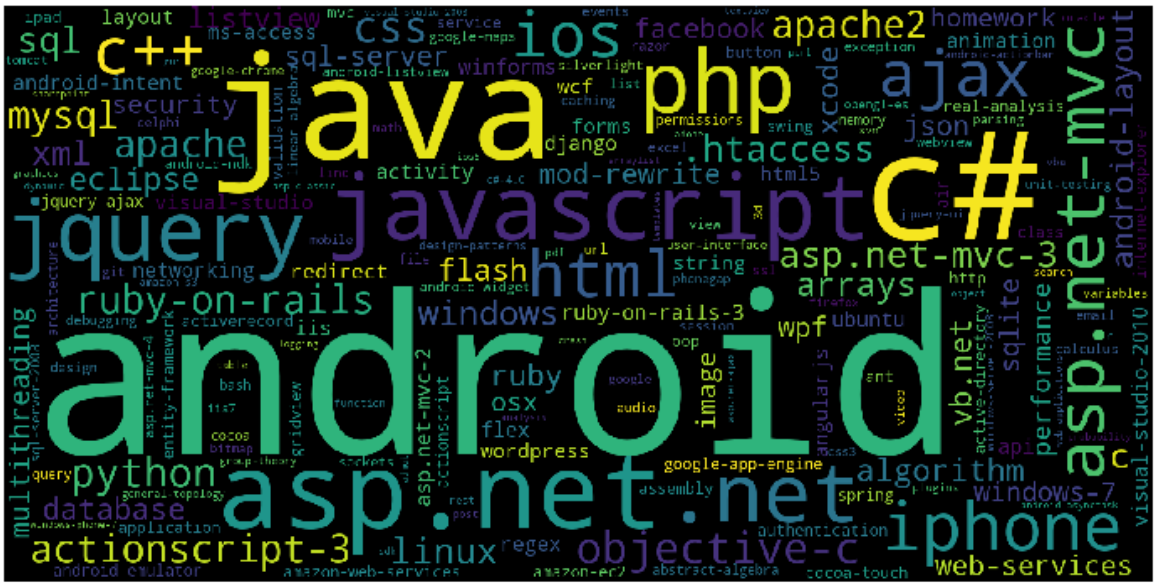
In [42]:
```
print( "Maximum number of tags per question: %d"%max(tag_quest_count))
print( "Minimum number of tags per question: %d"%min(tag_quest_count))
print( "Avg. number of tags per question: %f"% ((sum(tag_quest_count)*1.0)/len(ta
```

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

In [43]:
```
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()
```



In [45]:
```
# 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=(10,20))
plt.imshow(wordcloud)
plt.axis('off')
plt.tight_layout(pad=0)
fig.savefig("tag.png")
plt.show()
print("Time taken to run this cell :", datetime.now() - start)
```




```
Time taken to run this cell : 0:00:04.342871
```

Observations

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

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 [14]:
```
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 [15]:
```
start = datetime.now()
vectorizer = CountVectorizer(min_df=0.00009, max_features=25000,tokenizer = lambd
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 : 0:04:24.264990
```

In [16]:
```
print("Dimensions of train data X:",x_train_multilabel.shape, "Y :",y_train.shape
print("Dimensions of test data X:",x_test_multilabel.shape,"Y:",y_test.shape)
```

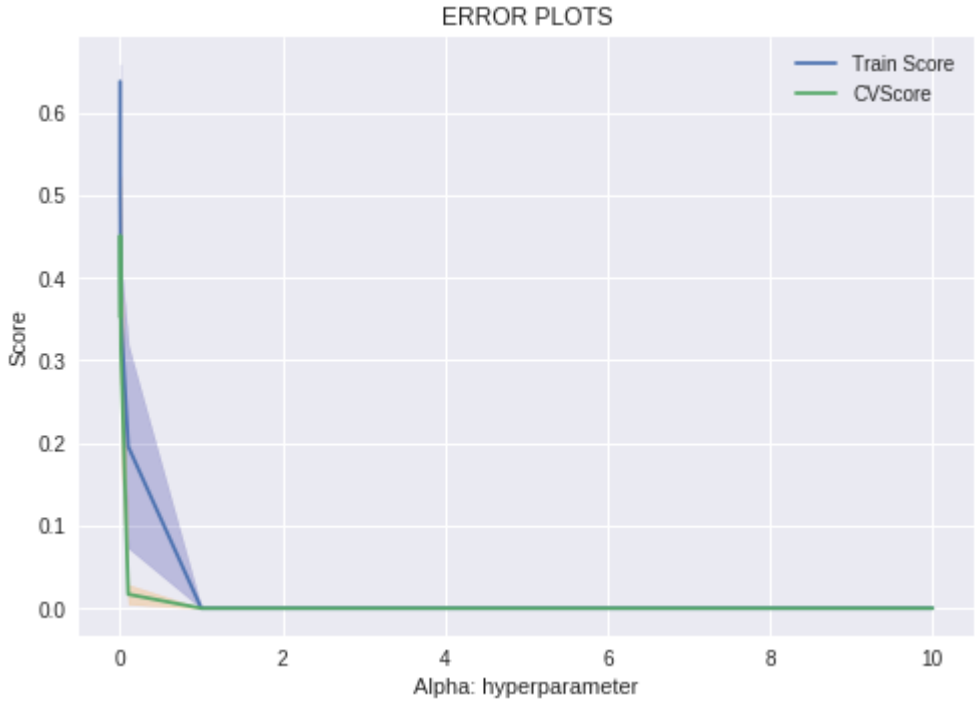
```
Dimensions of train data X: (200000, 25000) Y : (200000, 500)
Dimensions of test data X: (50000, 25000) Y: (50000, 500)
```

Applying Logistic Regression with OneVsRest Classifier

In [17]:
```
from sklearn.model_selection import GridSearchCV

param={'estimator__alpha': [10**-5, 10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1]
classifier = OneVsRestClassifier(SGDClassifier(loss='log', penalty='l1'))
gsv = GridSearchCV(estimator = classifier, param_grid=param, cv=3, verbose=1, sco
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 7 candidates, totalling 21 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  21 out of  21 | elapsed: 98.2min finished

value of alpha after hyperparameter tuning :  0.001
```

In [18]:
```
alpha = [10**-5, 10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1]
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/48803361/4084039
plt.gca().fill_between(alpha,train_score - train_score_std,train_score + train_sc

plt.plot(alpha, cv_score, label='CVScore')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(alpha,cv_score - cv_score_std,cv_score + cv_score_std,alph
plt.legend()
plt.xlabel("Alpha: hyperparameter")
plt.ylabel("Score")
plt.title("ERROR PLOTS")
plt.show()
```



In [19]:
```
start = datetime.now()
#best_alpha = gsv.best_estimator_.get_params()['estimator__alpha']
classifier = OneVsRestClassifier(SGDClassifier(loss='log', alpha=best_alpha, pena
classifier.fit(x_train_multilabel, y_train)
predictions = classifier.predict (x_test_multilabel)


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


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

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

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

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

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

```
Accuracy : 0.15508
Hamming loss  0.00359556
Micro-average quality numbers
Precision: 0.4835, Recall: 0.3345, F1-measure: 0.3954
Macro-average quality numbers
Precision: 0.3523, Recall: 0.2476, F1-measure: 0.2699
Time taken to run this cell : 0:07:30.145413
```

In [21]:
```
print (metrics.classification_report(y_test, predictions))
```

	precision	recall	f1-score	support
0	0.58	0.62	0.60	2220
1	0.45	0.16	0.24	3473
2	0.67	0.35	0.46	3976
3	0.70	0.68	0.69	2437
4	0.63	0.45	0.52	2054
5	0.69	0.56	0.62	2580
6	0.74	0.56	0.64	1475
7	0.34	0.27	0.30	1493
8	0.56	0.55	0.56	957
9	0.65	0.38	0.48	1781
10	0.66	0.47	0.55	1568
11	0.42	0.38	0.40	1477
12	0.49	0.46	0.47	306
13	0.38	0.24	0.30	916
14	0.62	0.18	0.28	1161
15	0.56	0.56	0.56	811
16	0.71	0.59	0.64	1200
17	0.73	0.57	0.64	686
18	0.80	0.58	0.67	236
19	0.76	0.55	0.64	680
20	0.39	0.47	0.43	2233
21	0.52	0.35	0.42	2785
22	0.60	0.47	0.53	409
23	0.40	0.33	0.36	533
24	0.34	0.33	0.34	860
25	0.52	0.29	0.37	391
26	0.44	0.22	0.29	889
27	0.46	0.49	0.48	1280
28	0.27	0.09	0.14	739
29	0.54	0.44	0.48	337
30	0.06	0.04	0.05	54
31	0.66	0.48	0.56	325
32	0.11	0.30	0.16	274
33	0.35	0.24	0.28	398
34	0.27	0.25	0.26	305
35	0.57	0.40	0.47	221
36	0.73	0.34	0.46	486
37	0.63	0.19	0.29	583
38	0.37	0.34	0.35	340
39	0.48	0.42	0.45	80
40	0.44	0.59	0.50	243
41	0.72	0.61	0.66	420
42	0.71	0.37	0.49	685
43	0.43	0.42	0.42	262
44	0.73	0.49	0.59	283
45	0.34	0.40	0.37	301
46	0.46	0.20	0.27	507
47	0.50	0.54	0.52	85
48	0.17	0.19	0.18	280
49	0.22	0.19	0.20	160
50	0.37	0.14	0.20	419
51	0.58	0.22	0.32	446

52	0.00	0.00	0.00	242
53	0.19	0.10	0.13	188
54	0.44	0.44	0.44	178
55	0.43	0.25	0.32	214
56	0.55	0.21	0.31	56
57	0.34	0.19	0.24	221
58	0.76	0.81	0.78	282
59	0.07	0.18	0.10	33
60	0.51	0.54	0.53	134
61	0.16	0.26	0.20	211
62	0.49	0.11	0.18	159
63	0.55	0.27	0.36	372
64	0.10	0.28	0.15	53
65	0.17	0.10	0.12	41
66	0.62	0.62	0.62	13
67	0.32	0.26	0.28	253
68	0.35	0.12	0.18	232
69	0.36	0.16	0.23	202
70	0.54	0.42	0.47	125
71	0.79	0.57	0.66	217
72	0.37	0.44	0.40	229
73	0.74	0.82	0.78	161
74	0.77	0.32	0.45	225
75	0.24	0.12	0.16	118
76	0.55	0.29	0.38	175
77	0.98	0.37	0.54	687
78	0.53	0.41	0.46	76
79	0.94	0.87	0.91	698
80	0.08	0.06	0.07	299
81	0.13	0.14	0.14	65
82	0.00	0.00	0.00	133
83	0.36	0.74	0.48	42
84	0.49	0.38	0.43	126
85	0.00	0.00	0.00	202
86	0.22	0.08	0.12	48
87	0.65	0.19	0.30	285
88	0.23	0.37	0.28	49
89	0.31	0.08	0.12	217
90	0.26	0.23	0.25	47
91	0.13	0.12	0.13	49
92	0.63	0.58	0.60	76
93	0.05	0.33	0.09	36
94	0.71	0.55	0.62	213
95	0.38	0.04	0.08	135
96	0.40	0.21	0.28	200
97	0.08	0.08	0.08	118
98	0.26	0.29	0.27	59
99	0.75	0.15	0.25	121
100	0.61	0.36	0.45	425
101	0.10	0.01	0.01	159
102	0.48	0.34	0.40	114
103	0.65	0.61	0.63	74
104	0.09	0.42	0.15	126
105	0.30	0.03	0.05	106
106	0.22	0.05	0.08	206
107	0.50	0.41	0.45	90
108	0.24	0.22	0.23	54

109	0.25	0.23	0.24	161
110	0.09	0.25	0.13	163
111	0.50	0.00	0.01	204
112	0.07	0.03	0.04	142
113	0.19	0.33	0.24	15
114	0.78	0.62	0.69	187
115	0.05	0.01	0.02	166
116	0.47	0.29	0.36	170
117	0.13	0.13	0.13	105
118	0.08	0.03	0.04	37
119	0.12	0.13	0.13	97
120	0.64	0.51	0.56	152
121	0.43	0.54	0.48	178
122	0.14	0.14	0.14	130
123	0.59	0.33	0.42	89
124	0.19	0.14	0.16	132
125	0.54	0.38	0.44	98
126	0.05	0.02	0.03	49
127	0.48	0.15	0.23	108
128	0.52	0.16	0.24	361
129	0.41	0.08	0.14	156
130	0.68	0.42	0.52	160
131	0.27	0.11	0.16	118
132	0.67	0.57	0.62	14
133	0.52	0.43	0.47	177
134	0.45	0.42	0.43	109
135	0.55	0.65	0.60	251
136	0.09	0.08	0.09	25
137	0.54	0.07	0.12	101
138	0.26	0.10	0.15	214
139	0.82	0.69	0.75	134
140	0.34	0.26	0.29	124
141	0.50	0.23	0.31	204
142	0.36	0.22	0.28	107
143	0.58	0.36	0.45	192
144	0.35	0.38	0.37	89
145	0.36	0.24	0.29	21
146	0.49	0.38	0.42	56
147	0.00	0.00	0.00	126
148	0.30	0.53	0.39	100
149	0.33	0.26	0.29	97
150	0.76	0.63	0.69	282
151	0.88	0.64	0.74	70
152	0.55	0.59	0.57	138
153	0.96	0.46	0.62	152
154	0.13	0.11	0.12	97
155	0.44	0.37	0.40	51
156	0.57	0.25	0.35	106
157	0.26	0.13	0.17	117
158	0.00	0.00	0.00	118
159	0.77	0.55	0.64	119
160	0.15	0.19	0.17	150
161	0.31	0.09	0.13	94
162	0.26	0.12	0.16	103
163	0.77	0.23	0.36	86
164	0.75	0.38	0.51	78
165	0.65	0.53	0.59	96

166	0.12	0.01	0.02	109
167	0.24	0.20	0.22	111
168	0.19	0.25	0.22	88
169	0.07	0.03	0.04	77
170	0.39	0.23	0.29	53
171	0.46	0.79	0.58	14
172	0.09	0.16	0.11	50
173	0.59	0.36	0.45	122
174	0.00	0.00	0.00	110
175	0.27	0.57	0.36	7
176	0.69	0.57	0.62	180
177	0.80	0.25	0.38	16
178	0.33	0.02	0.04	45
179	0.34	0.39	0.36	69
180	0.27	0.07	0.11	87
181	0.23	0.38	0.29	226
182	0.08	0.07	0.07	59
183	0.45	0.26	0.33	110
184	0.60	0.51	0.55	257
185	0.30	0.20	0.24	113
186	0.47	0.40	0.44	47
187	0.00	0.00	0.00	145
188	0.38	0.18	0.25	193
189	0.29	0.03	0.06	116
190	0.83	0.18	0.29	135
191	0.50	0.31	0.38	13
192	0.11	0.05	0.07	111
193	0.70	0.55	0.62	152
194	0.25	0.12	0.16	75
195	0.89	0.89	0.89	9
196	0.00	0.00	0.00	142
197	0.31	0.21	0.25	19
198	0.12	0.09	0.10	45
199	0.00	0.00	0.00	77
200	0.16	0.18	0.17	83
201	0.00	0.00	0.00	113
202	0.47	0.81	0.59	37
203	0.94	0.63	0.75	116
204	0.17	0.11	0.14	114
205	0.08	0.10	0.09	189
206	0.84	0.45	0.58	85
207	0.12	0.09	0.10	67
208	0.78	0.42	0.55	93
209	0.33	0.08	0.13	149
210	0.68	0.25	0.36	114
211	0.00	0.00	0.00	407
212	0.00	0.00	0.00	5
213	0.69	0.30	0.42	109
214	0.62	0.82	0.71	97
215	0.19	0.36	0.25	14
216	0.53	0.34	0.41	174
217	0.43	0.19	0.26	47
218	0.44	0.24	0.31	114
219	0.55	0.28	0.37	96
220	0.21	0.13	0.16	71
221	0.57	0.07	0.12	114
222	0.06	0.20	0.10	20

223	0.06	0.02	0.03	81
224	0.70	0.21	0.33	33
225	0.33	0.05	0.09	59
226	0.66	0.32	0.43	66
227	0.53	0.39	0.45	44
228	0.51	0.49	0.50	68
229	0.09	0.06	0.07	98
230	0.94	0.38	0.54	130
231	0.89	0.46	0.61	102
232	0.81	0.68	0.74	44
233	0.36	0.29	0.32	14
234	0.00	0.00	0.00	67
235	0.25	0.01	0.02	84
236	0.54	0.27	0.36	52
237	0.88	0.60	0.71	99
238	0.30	0.38	0.33	8
239	0.30	0.22	0.25	137
240	0.07	0.02	0.03	63
241	0.00	0.00	0.00	61
242	0.22	0.18	0.20	50
243	0.17	0.39	0.24	71
244	0.61	0.54	0.57	95
245	0.16	0.09	0.11	57
246	0.19	0.25	0.22	28
247	0.19	0.35	0.25	20
248	0.55	0.25	0.35	83
249	0.00	0.00	0.00	117
250	0.19	0.16	0.17	96
251	0.54	0.30	0.39	70
252	0.51	0.63	0.57	60
253	0.15	0.20	0.17	65
254	0.50	0.02	0.03	62
255	0.61	0.58	0.60	74
256	0.17	0.19	0.18	27
257	0.00	0.00	0.00	90
258	0.62	0.50	0.56	20
259	0.02	0.04	0.03	98
260	0.79	0.30	0.44	89
261	0.00	0.00	0.00	279
262	0.97	0.79	0.87	84
263	0.00	0.00	0.00	13
264	0.20	0.38	0.26	48
265	0.28	0.42	0.33	113
266	0.23	0.18	0.20	105
267	0.11	0.05	0.07	78
268	0.39	0.38	0.39	47
269	0.47	0.40	0.43	141
270	0.17	0.01	0.02	83
271	0.68	0.41	0.51	74
272	0.17	0.08	0.11	38
273	0.79	0.50	0.61	60
274	0.20	0.12	0.15	81
275	0.38	0.11	0.17	54
276	0.50	0.22	0.30	37
277	0.24	0.09	0.13	90
278	0.13	0.19	0.16	26
279	0.34	0.55	0.42	99

280	1.00	0.01	0.02	114
281	0.26	0.20	0.22	61
282	0.68	0.35	0.46	78
283	0.13	0.04	0.07	46
284	0.54	0.15	0.24	84
285	0.69	0.27	0.39	67
286	0.00	0.00	0.00	292
287	0.00	0.00	0.00	321
288	0.59	0.35	0.44	97
289	0.00	0.00	0.00	85
290	0.58	0.67	0.62	43
291	0.00	0.00	0.00	108
292	0.22	0.07	0.11	127
293	0.18	0.03	0.04	79
294	0.30	0.67	0.42	160
295	0.51	0.53	0.52	57
296	0.12	0.04	0.06	52
297	0.20	0.53	0.29	64
298	0.26	0.15	0.19	60
299	0.33	0.44	0.38	9
300	0.43	0.49	0.46	51
301	0.27	0.07	0.11	58
302	0.17	0.02	0.03	52
303	0.49	0.38	0.43	81
304	0.00	0.00	0.00	68
305	0.00	0.00	0.00	53
306	0.35	0.24	0.29	45
307	0.33	0.01	0.02	116
308	0.29	0.26	0.27	47
309	0.45	0.13	0.20	70
310	0.20	0.22	0.21	37
311	0.00	0.00	0.00	254
312	0.33	0.02	0.04	101
313	0.12	0.19	0.15	107
314	0.00	0.00	0.00	4
315	0.08	0.11	0.09	9
316	0.93	0.37	0.53	68
317	0.09	0.13	0.11	38
318	0.70	0.11	0.19	125
319	0.19	0.17	0.18	48
320	0.00	0.00	0.00	28
321	0.17	0.14	0.16	128
322	0.00	0.00	0.00	42
323	0.36	0.71	0.48	7
324	0.00	0.00	0.00	71
325	0.67	0.40	0.50	10
326	0.85	0.38	0.53	76
327	0.03	0.03	0.03	29
328	0.27	0.44	0.33	36
329	0.85	0.62	0.72	63
330	0.31	0.08	0.12	102
331	0.69	0.45	0.55	95
332	0.76	0.24	0.36	80
333	0.38	0.39	0.38	38
334	0.50	0.17	0.25	30
335	0.57	0.36	0.44	36
336	0.29	0.24	0.26	25

337	0.00	0.00	0.00	16
338	0.00	0.00	0.00	228
339	0.22	0.10	0.13	62
340	0.00	0.00	0.00	156
341	0.59	0.18	0.28	55
342	0.92	0.40	0.56	85
343	0.00	0.00	0.00	72
344	0.00	0.00	0.00	22
345	0.92	0.58	0.71	195
346	0.83	0.47	0.60	75
347	0.78	0.15	0.25	46
348	0.14	0.07	0.10	68
349	0.19	0.19	0.19	16
350	0.34	0.35	0.34	60
351	0.05	0.01	0.02	67
352	0.96	0.49	0.65	47
353	0.82	0.33	0.47	42
354	0.00	0.00	0.00	31
355	0.12	0.10	0.11	41
356	0.35	0.31	0.33	39
357	0.95	0.41	0.57	85
358	0.80	0.78	0.79	83
359	0.39	0.26	0.31	27
360	0.34	0.27	0.30	113
361	0.04	0.18	0.07	11
362	0.00	0.00	0.00	54
363	0.08	0.03	0.04	98
364	0.43	0.55	0.48	22
365	0.69	0.31	0.43	35
366	0.07	0.10	0.08	10
367	0.00	0.00	0.00	189
368	0.13	0.04	0.06	97
369	0.00	0.00	0.00	45
370	0.89	0.24	0.37	72
371	0.00	0.00	0.00	15
372	0.32	0.33	0.33	21
373	0.28	0.28	0.28	32
374	0.20	0.26	0.23	65
375	0.62	0.74	0.68	27
376	0.00	0.00	0.00	3
377	0.24	0.08	0.12	95
378	1.00	0.29	0.45	62
379	0.75	0.74	0.74	53
380	0.31	0.30	0.31	53
381	0.00	0.00	0.00	93
382	0.10	0.09	0.10	11
383	0.00	0.00	0.00	82
384	0.61	0.38	0.47	52
385	0.17	0.12	0.14	17
386	0.21	0.54	0.30	13
387	0.20	0.38	0.26	8
388	0.00	0.00	0.00	106
389	0.92	0.49	0.63	68
390	0.67	0.50	0.57	4
391	0.28	0.08	0.12	92
392	0.00	0.00	0.00	34
393	0.06	0.03	0.04	35

394	0.21	0.20	0.21	15
395	0.15	0.03	0.05	60
396	0.19	0.18	0.19	28
397	0.00	0.00	0.00	38
398	0.00	0.00	0.00	37
399	0.00	0.00	0.00	59
400	0.64	0.25	0.35	57
401	0.09	0.24	0.13	45
402	0.00	0.00	0.00	131
403	0.41	0.23	0.30	47
404	0.05	0.04	0.04	26
405	0.02	0.02	0.02	105
406	0.50	0.02	0.03	60
407	0.19	0.14	0.16	43
408	0.00	0.00	0.00	62
409	0.40	0.42	0.41	50
410	1.00	0.38	0.55	8
411	0.20	0.20	0.20	30
412	0.00	0.00	0.00	59
413	0.60	0.07	0.13	40
414	0.00	0.00	0.00	53
415	0.51	0.48	0.50	52
416	0.00	0.00	0.00	47
417	0.60	0.25	0.35	12
418	0.46	0.34	0.39	62
419	0.18	0.12	0.15	24
420	0.20	0.06	0.09	53
421	0.06	0.03	0.04	40
422	0.03	0.12	0.05	56
423	0.00	0.00	0.00	63
424	0.14	0.30	0.19	27
425	0.14	0.07	0.10	14
426	0.06	0.02	0.03	41
427	0.23	0.35	0.28	48
428	0.73	0.43	0.54	37
429	0.44	0.23	0.30	52
430	0.45	0.26	0.33	50
431	0.18	0.18	0.18	55
432	0.00	0.00	0.00	49
433	0.15	0.07	0.09	46
434	0.60	1.00	0.75	3
435	0.00	0.00	0.00	121
436	0.00	0.00	0.00	68
437	0.86	0.46	0.60	70
438	0.04	0.06	0.05	34
439	0.00	0.00	0.00	62
440	0.20	0.07	0.10	45
441	0.00	0.00	0.00	3
442	0.00	0.00	0.00	5
443	0.07	0.05	0.06	19
444	0.14	0.08	0.10	38
445	1.00	1.00	1.00	1
446	0.12	0.02	0.03	60
447	0.17	0.20	0.19	20
448	0.08	0.13	0.10	38
449	0.00	0.00	0.00	15
450	0.00	0.00	0.00	22

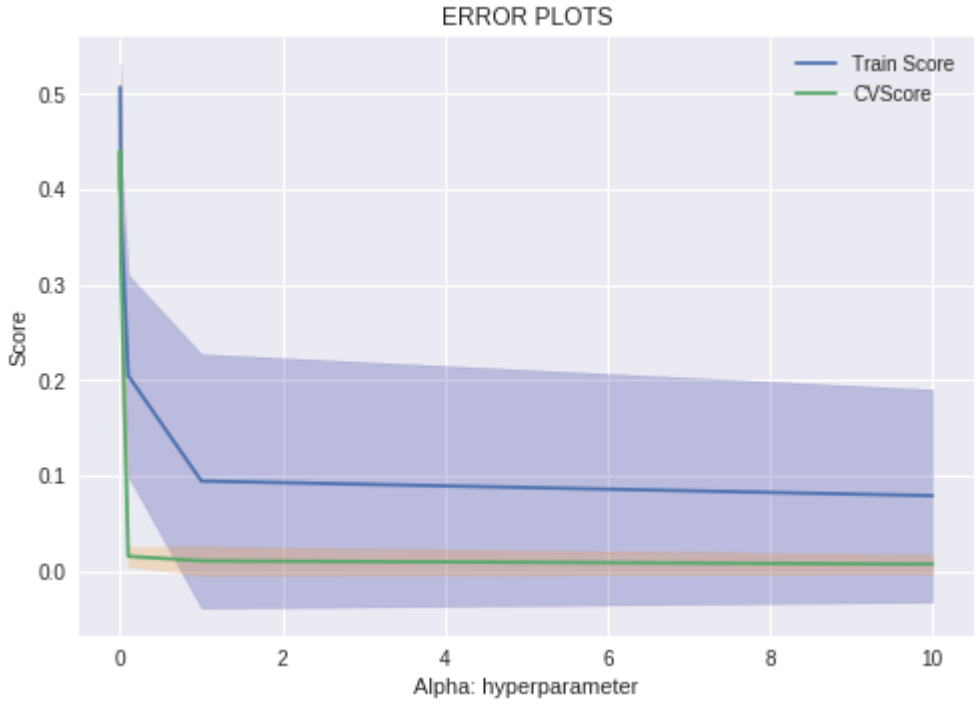
451	0.53	0.47	0.49	43
452	0.00	0.00	0.00	110
453	0.00	0.00	0.00	21
454	0.16	0.21	0.18	29
455	0.54	0.39	0.45	49
456	0.00	0.00	0.00	3
457	0.90	0.83	0.86	52
458	0.19	0.29	0.23	41
459	0.00	0.00	0.00	42
460	0.12	0.19	0.15	32
461	0.86	0.27	0.41	22
462	0.06	0.17	0.08	6
463	0.00	0.00	0.00	5
464	0.00	0.00	0.00	29
465	0.05	0.06	0.05	16
466	0.07	0.02	0.03	44
467	0.00	0.00	0.00	65
468	0.17	0.08	0.11	38
469	0.06	0.07	0.06	55
470	0.17	0.07	0.10	58
471	0.31	0.50	0.38	8
472	0.00	0.00	0.00	64
473	0.03	0.07	0.04	14
474	0.00	0.00	0.00	34
475	0.88	0.42	0.57	36
476	0.00	0.00	0.00	19
477	0.47	0.32	0.38	50
478	0.48	0.24	0.32	63
479	0.25	0.17	0.20	24
480	0.54	0.13	0.21	54
481	0.56	0.28	0.37	68
482	0.50	0.36	0.42	14
483	0.43	0.10	0.16	31
484	0.24	0.27	0.25	45
485	0.00	0.00	0.00	49
486	0.00	0.00	0.00	25
487	0.00	0.00	0.00	50
488	0.75	0.55	0.63	11
489	0.44	0.16	0.24	67
490	0.26	0.17	0.20	42
491	0.35	0.36	0.35	42
492	0.86	0.32	0.46	19
493	0.64	0.53	0.58	78
494	0.40	0.22	0.29	18
495	0.10	0.10	0.10	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	0.20	0.31	10
micro avg	0.48	0.33	0.40	87885
macro avg	0.35	0.25	0.27	87885
weighted avg	0.48	0.33	0.38	87885
samples avg	0.36	0.31	0.30	87885

Applying Linear SVM with OneVsRestClassifier

In [22]:
```
param={'estimator__alpha': [10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1]}
classifier = OneVsRestClassifier(SGDClassifier(loss='hinge', penalty='l1'))
gsv = GridSearchCV(estimator = classifier, param_grid=param, cv=3, verbose=1, sco
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 6 candidates, totalling 18 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  18 out of  18 | elapsed: 67.4min finished

value of alpha after hyperparameter tuning :  0.001
```

In [25]:
```
alpha = [10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1]
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/48803361/4084039
plt.gca().fill_between(alpha,train_score - train_score_std,train_score + train_sc

plt.plot(alpha, cv_score, label='CVScore')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(alpha,cv_score - cv_score_std,cv_score + cv_score_std,alph
plt.legend()
plt.xlabel("Alpha: hyperparameter")
plt.ylabel("Score")
plt.title("ERROR PLOTS")
plt.show()
```



In [23]:
```
start = datetime.now()
#best_alpha = gsv.best_estimator_.get_params()['estimator__alpha']
classifier = OneVsRestClassifier(SGDClassifier(loss='hinge', alpha=best_alpha, pe
classifier.fit(x_train_multilabel, y_train)
predictions = classifier.predict (x_test_multilabel)


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


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

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

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

print("Macro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, r
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 : 0.14932
Hamming loss  0.00363736
Micro-average quality numbers
Precision: 0.4740, Recall: 0.3167, F1-measure: 0.3797
Macro-average quality numbers
Precision: 0.2728, Recall: 0.2382, F1-measure: 0.2334
              precision    recall  f1-score   support

           0       0.62      0.65      0.63      2220
           1       0.37      0.11      0.17      3473
           2       0.64      0.34      0.44      3976
           3       0.74      0.65      0.69      2437
           4       0.61      0.43      0.51      2054
           5       0.68      0.51      0.58      2580
           6       0.70      0.57      0.63      1475
           7       0.53      0.20      0.29      1493
           8       0.58      0.53      0.55       957
           9       0.65      0.32      0.43      1781
          10       0.56      0.47      0.51      1568
          11       0.51      0.32      0.39      1477
          12       0.36      0.47      0.41       306
          13       0.07      0.00      0.01       916
          14       0.42      0.17      0.24      1161
          15       0.65      0.55      0.59       811
          16       0.67      0.60      0.63      1200
          17       0.73      0.61      0.66       686
```

18	0.81	0.60	0.69	236
19	0.73	0.46	0.56	680
20	0.41	0.48	0.44	2233
21	0.44	0.49	0.47	2785
22	0.54	0.60	0.57	409
23	0.31	0.29	0.30	533
24	0.39	0.30	0.34	860
25	0.33	0.41	0.36	391
26	0.16	0.04	0.06	889
27	0.48	0.41	0.44	1280
28	0.02	0.00	0.00	739
29	0.57	0.44	0.49	337
30	0.05	0.02	0.03	54
31	0.43	0.57	0.49	325
32	0.28	0.35	0.31	274
33	0.00	0.00	0.00	398
34	0.70	0.32	0.44	305
35	0.50	0.38	0.43	221
36	0.39	0.45	0.42	486
37	0.33	0.36	0.35	583
38	0.43	0.30	0.35	340
39	0.59	0.44	0.50	80
40	0.53	0.51	0.52	243
41	0.79	0.58	0.67	420
42	0.84	0.30	0.44	685
43	0.37	0.52	0.43	262
44	0.55	0.70	0.61	283
45	0.48	0.27	0.34	301
46	0.45	0.12	0.19	507
47	0.21	0.56	0.31	85
48	0.00	0.00	0.00	280
49	0.00	0.00	0.00	160
50	0.00	0.00	0.00	419
51	0.34	0.19	0.24	446
52	0.00	0.00	0.00	242
53	0.07	0.01	0.01	188
54	0.41	0.33	0.36	178
55	0.47	0.39	0.42	214
56	0.19	0.29	0.23	56
57	0.42	0.20	0.27	221
58	0.71	0.77	0.74	282
59	0.25	0.09	0.13	33
60	0.56	0.48	0.52	134
61	0.19	0.28	0.23	211
62	0.45	0.09	0.16	159
63	0.42	0.33	0.37	372
64	0.02	0.15	0.03	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	0.22	0.26	0.24	202
70	0.30	0.54	0.39	125
71	0.30	0.62	0.41	217
72	0.36	0.34	0.35	229
73	0.68	0.84	0.75	161
74	0.71	0.35	0.47	225

75	0.00	0.00	0.00	118
76	0.48	0.09	0.15	175
77	0.57	0.32	0.41	687
78	0.13	0.45	0.20	76
79	0.93	0.84	0.89	698
80	0.00	0.00	0.00	299
81	0.00	0.00	0.00	65
82	0.00	0.00	0.00	133
83	0.33	0.71	0.45	42
84	0.36	0.39	0.38	126
85	0.00	0.00	0.00	202
86	0.03	0.04	0.03	48
87	0.51	0.25	0.34	285
88	0.12	0.31	0.17	49
89	0.00	0.00	0.00	217
90	0.17	0.30	0.22	47
91	0.38	0.12	0.18	49
92	0.53	0.62	0.57	76
93	0.00	0.00	0.00	36
94	0.60	0.56	0.58	213
95	0.07	0.10	0.08	135
96	0.28	0.18	0.22	200
97	0.00	0.00	0.00	118
98	0.52	0.27	0.36	59
99	0.67	0.33	0.44	121
100	0.00	0.00	0.00	425
101	0.00	0.00	0.00	159
102	0.36	0.31	0.33	114
103	0.60	0.66	0.63	74
104	0.23	0.26	0.24	126
105	0.39	0.15	0.22	106
106	0.00	0.00	0.00	206
107	0.37	0.38	0.37	90
108	0.48	0.37	0.42	54
109	0.00	0.00	0.00	161
110	0.00	0.00	0.00	163
111	0.00	0.00	0.00	204
112	0.00	0.00	0.00	142
113	0.18	0.53	0.27	15
114	0.36	0.51	0.42	187
115	0.00	0.00	0.00	166
116	0.33	0.29	0.31	170
117	0.00	0.00	0.00	105
118	0.00	0.00	0.00	37
119	0.12	0.18	0.14	97
120	0.57	0.41	0.48	152
121	0.29	0.54	0.38	178
122	0.00	0.00	0.00	130
123	0.56	0.31	0.40	89
124	0.09	0.16	0.11	132
125	0.43	0.53	0.47	98
126	0.00	0.00	0.00	49
127	0.58	0.13	0.21	108
128	0.27	0.26	0.26	361
129	0.39	0.14	0.21	156
130	0.66	0.35	0.46	160
131	0.47	0.06	0.11	118

132	0.47	0.50	0.48	14
133	0.38	0.44	0.41	177
134	0.34	0.36	0.35	109
135	0.58	0.63	0.60	251
136	0.00	0.00	0.00	25
137	0.61	0.19	0.29	101
138	0.08	0.14	0.10	214
139	0.70	0.80	0.75	134
140	0.31	0.35	0.33	124
141	0.72	0.28	0.41	204
142	0.29	0.22	0.25	107
143	0.61	0.44	0.51	192
144	0.00	0.00	0.00	89
145	0.22	0.48	0.30	21
146	0.16	0.48	0.24	56
147	0.00	0.00	0.00	126
148	0.26	0.38	0.31	100
149	0.27	0.35	0.31	97
150	0.64	0.71	0.67	282
151	0.83	0.74	0.78	70
152	0.68	0.51	0.59	138
153	0.93	0.54	0.68	152
154	0.00	0.00	0.00	97
155	0.31	0.27	0.29	51
156	0.83	0.14	0.24	106
157	0.00	0.00	0.00	117
158	0.00	0.00	0.00	118
159	0.52	0.64	0.57	119
160	0.00	0.00	0.00	150
161	0.00	0.00	0.00	94
162	0.00	0.00	0.00	103
163	0.58	0.33	0.42	86
164	0.86	0.46	0.60	78
165	0.38	0.58	0.46	96
166	0.00	0.00	0.00	109
167	0.28	0.17	0.21	111
168	0.00	0.00	0.00	88
169	0.05	0.01	0.02	77
170	0.88	0.13	0.23	53
171	0.31	0.64	0.42	14
172	0.00	0.00	0.00	50
173	0.27	0.41	0.33	122
174	0.00	0.00	0.00	110
175	0.67	0.57	0.62	7
176	0.64	0.68	0.66	180
177	0.41	0.56	0.47	16
178	0.27	0.07	0.11	45
179	0.23	0.26	0.24	69
180	0.18	0.10	0.13	87
181	0.00	0.00	0.00	226
182	0.00	0.00	0.00	59
183	0.33	0.30	0.32	110
184	0.61	0.62	0.61	257
185	0.31	0.22	0.26	113
186	0.23	0.51	0.31	47
187	0.00	0.00	0.00	145
188	0.00	0.00	0.00	193

189	0.39	0.14	0.20	116
190	0.77	0.30	0.44	135
191	0.25	0.15	0.19	13
192	0.00	0.00	0.00	111
193	0.66	0.57	0.61	152
194	0.12	0.24	0.16	75
195	0.69	1.00	0.82	9
196	0.00	0.00	0.00	142
197	0.31	0.26	0.29	19
198	0.00	0.00	0.00	45
199	0.00	0.00	0.00	77
200	0.53	0.20	0.30	83
201	0.00	0.00	0.00	113
202	0.64	0.76	0.69	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	0.66	0.34	0.45	85
207	0.10	0.19	0.13	67
208	0.59	0.38	0.46	93
209	0.00	0.00	0.00	149
210	0.52	0.39	0.45	114
211	0.00	0.00	0.00	407
212	0.10	0.20	0.13	5
213	0.43	0.50	0.47	109
214	0.70	0.48	0.57	97
215	0.31	0.36	0.33	14
216	0.61	0.23	0.33	174
217	0.55	0.38	0.45	47
218	0.45	0.38	0.41	114
219	0.56	0.16	0.24	96
220	0.30	0.10	0.15	71
221	0.00	0.00	0.00	114
222	0.03	0.10	0.04	20
223	0.17	0.12	0.14	81
224	0.53	0.24	0.33	33
225	0.14	0.08	0.11	59
226	0.35	0.47	0.40	66
227	0.40	0.18	0.25	44
228	0.20	0.54	0.30	68
229	0.08	0.07	0.08	98
230	0.88	0.40	0.55	130
231	0.59	0.55	0.57	102
232	0.75	0.55	0.63	44
233	0.19	0.43	0.27	14
234	0.00	0.00	0.00	67
235	0.00	0.00	0.00	84
236	0.45	0.40	0.42	52
237	0.80	0.56	0.65	99
238	0.20	0.38	0.26	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	0.00	0.00	0.00	71
244	0.51	0.48	0.50	95
245	0.00	0.00	0.00	57

246	0.18	0.32	0.23	28
247	0.09	0.25	0.13	20
248	0.00	0.00	0.00	83
249	0.00	0.00	0.00	117
250	0.00	0.00	0.00	96
251	0.36	0.37	0.36	70
252	0.56	0.48	0.52	60
253	0.08	0.15	0.11	65
254	0.56	0.08	0.14	62
255	0.45	0.74	0.56	74
256	0.06	0.07	0.06	27
257	0.00	0.00	0.00	90
258	0.29	0.75	0.42	20
259	0.00	0.00	0.00	98
260	0.76	0.38	0.51	89
261	0.00	0.00	0.00	279
262	0.83	0.87	0.85	84
263	0.00	0.00	0.00	13
264	0.25	0.33	0.28	48
265	0.22	0.35	0.27	113
266	0.12	0.15	0.14	105
267	0.05	0.08	0.06	78
268	0.19	0.36	0.25	47
269	0.75	0.17	0.28	141
270	0.00	0.00	0.00	83
271	0.57	0.65	0.61	74
272	0.14	0.05	0.08	38
273	0.52	0.48	0.50	60
274	0.00	0.00	0.00	81
275	0.22	0.11	0.15	54
276	0.36	0.14	0.20	37
277	0.00	0.00	0.00	90
278	0.00	0.00	0.00	26
279	0.79	0.38	0.52	99
280	0.03	0.02	0.02	114
281	0.17	0.23	0.20	61
282	0.38	0.63	0.47	78
283	0.00	0.00	0.00	46
284	0.22	0.24	0.23	84
285	0.42	0.33	0.37	67
286	0.00	0.00	0.00	292
287	0.00	0.00	0.00	321
288	0.36	0.38	0.37	97
289	0.00	0.00	0.00	85
290	0.51	0.74	0.60	43
291	0.00	0.00	0.00	108
292	0.00	0.00	0.00	127
293	0.00	0.00	0.00	79
294	0.37	0.33	0.34	160
295	0.54	0.61	0.57	57
296	0.25	0.06	0.09	52
297	0.42	0.16	0.23	64
298	0.16	0.20	0.18	60
299	0.15	0.44	0.22	9
300	0.20	0.71	0.31	51
301	0.27	0.12	0.17	58
302	0.00	0.00	0.00	52

303	0.25	0.44	0.32	81
304	0.00	0.00	0.00	68
305	0.19	0.06	0.09	53
306	0.29	0.29	0.29	45
307	0.00	0.00	0.00	116
308	0.10	0.30	0.15	47
309	0.44	0.33	0.38	70
310	0.00	0.00	0.00	37
311	0.00	0.00	0.00	254
312	0.00	0.00	0.00	101
313	0.00	0.00	0.00	107
314	0.20	0.25	0.22	4
315	0.67	0.22	0.33	9
316	0.84	0.53	0.65	68
317	0.21	0.18	0.19	38
318	0.38	0.20	0.26	125
319	0.10	0.17	0.13	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	0.38	0.43	0.40	7
324	0.00	0.00	0.00	71
325	0.75	0.60	0.67	10
326	0.74	0.42	0.54	76
327	0.00	0.00	0.00	29
328	0.72	0.64	0.68	36
329	0.60	0.40	0.48	63
330	0.00	0.00	0.00	102
331	0.49	0.67	0.57	95
332	0.59	0.29	0.39	80
333	0.33	0.47	0.39	38
334	0.64	0.23	0.34	30
335	0.29	0.28	0.28	36
336	0.00	0.00	0.00	25
337	0.50	0.06	0.11	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	0.81	0.55	0.66	85
343	0.00	0.00	0.00	72
344	0.00	0.00	0.00	22
345	0.89	0.87	0.88	195
346	0.84	0.56	0.67	75
347	0.00	0.00	0.00	46
348	0.00	0.00	0.00	68
349	0.17	0.06	0.09	16
350	0.27	0.27	0.27	60
351	0.00	0.00	0.00	67
352	0.82	0.57	0.68	47
353	0.68	0.71	0.70	42
354	0.00	0.00	0.00	31
355	0.05	0.07	0.06	41
356	0.34	0.54	0.42	39
357	0.81	0.34	0.48	85
358	0.64	0.84	0.73	83
359	0.31	0.44	0.36	27

360	0.36	0.65	0.46	113
361	0.00	0.00	0.00	11
362	0.00	0.00	0.00	54
363	0.00	0.00	0.00	98
364	0.60	0.68	0.64	22
365	0.55	0.51	0.53	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	0.00	0.00	0.00	45
370	0.49	0.31	0.38	72
371	0.00	0.00	0.00	15
372	0.18	0.67	0.28	21
373	0.09	0.34	0.15	32
374	0.16	0.20	0.18	65
375	0.77	0.63	0.69	27
376	0.25	0.33	0.29	3
377	0.00	0.00	0.00	95
378	0.94	0.24	0.38	62
379	0.51	0.68	0.58	53
380	0.33	0.57	0.42	53
381	0.00	0.00	0.00	93
382	0.43	0.27	0.33	11
383	0.00	0.00	0.00	82
384	0.38	0.71	0.50	52
385	0.00	0.00	0.00	17
386	0.17	0.23	0.19	13
387	0.07	0.50	0.12	8
388	0.25	0.01	0.02	106
389	0.67	0.29	0.41	68
390	0.10	0.50	0.16	4
391	0.00	0.00	0.00	92
392	0.00	0.00	0.00	34
393	0.05	0.06	0.06	35
394	0.10	0.20	0.13	15
395	0.00	0.00	0.00	60
396	0.00	0.00	0.00	28
397	0.00	0.00	0.00	38
398	0.64	0.19	0.29	37
399	0.22	0.07	0.10	59
400	0.00	0.00	0.00	57
401	0.50	0.02	0.04	45
402	0.33	0.01	0.01	131
403	0.28	0.26	0.27	47
404	0.00	0.00	0.00	26
405	0.00	0.00	0.00	105
406	1.00	0.02	0.03	60
407	0.23	0.16	0.19	43
408	0.00	0.00	0.00	62
409	0.28	0.20	0.23	50
410	0.40	0.50	0.44	8
411	0.11	0.13	0.12	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	0.32	0.48	0.38	52
416	0.00	0.00	0.00	47

417	0.80	0.33	0.47	12
418	0.33	0.35	0.34	62
419	0.10	0.12	0.11	24
420	0.00	0.00	0.00	53
421	0.03	0.03	0.03	40
422	0.00	0.00	0.00	56
423	0.00	0.00	0.00	63
424	0.06	0.19	0.09	27
425	0.00	0.00	0.00	14
426	0.00	0.00	0.00	41
427	0.00	0.00	0.00	48
428	0.64	0.57	0.60	37
429	0.62	0.25	0.36	52
430	0.25	0.28	0.27	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	0.40	0.67	0.50	3
435	0.00	0.00	0.00	121
436	0.00	0.00	0.00	68
437	0.84	0.54	0.66	70
438	0.00	0.00	0.00	34
439	0.03	0.11	0.05	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	0.00	0.00	0.00	19
444	0.00	0.00	0.00	38
445	0.00	0.00	0.00	1
446	0.00	0.00	0.00	60
447	0.11	0.40	0.18	20
448	0.00	0.00	0.00	38
449	0.08	0.07	0.07	15
450	0.00	0.00	0.00	22
451	0.65	0.60	0.63	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	0.69	0.45	0.54	49
456	0.17	0.33	0.22	3
457	0.76	0.81	0.79	52
458	0.00	0.00	0.00	41
459	0.00	0.00	0.00	42
460	0.12	0.09	0.10	32
461	0.48	0.50	0.49	22
462	0.00	0.00	0.00	6
463	1.00	0.40	0.57	5
464	0.00	0.00	0.00	29
465	0.11	0.12	0.12	16
466	0.00	0.00	0.00	44
467	0.63	0.18	0.29	65
468	0.06	0.18	0.09	38
469	0.00	0.00	0.00	55
470	0.00	0.00	0.00	58
471	0.50	0.50	0.50	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       0.77      0.64      0.70        36
              476       0.00      0.00      0.00        19
              477       0.42      0.36      0.39        50
              478       0.00      0.00      0.00        63
              479       0.05      0.17      0.08        24
              480       0.24      0.43      0.30        54
              481       0.42      0.71      0.53        68
              482       0.05      0.57      0.09        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       0.29      0.08      0.12        25
              487       0.00      0.00      0.00        50
              488       1.00      0.45      0.62        11
              489       0.31      0.28      0.29        67
              490       0.00      0.00      0.00        42
              491       0.22      0.55      0.32        42
              492       0.55      0.32      0.40        19
              493       0.38      0.42      0.40        78
              494       0.00      0.00      0.00        18
              495       0.08      0.26      0.12        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       0.23      0.30      0.26        10

   micro avg       0.47      0.32      0.38     87885
   macro avg       0.27      0.24      0.23     87885
weighted avg       0.41      0.32      0.34     87885
 samples avg       0.35      0.30      0.29     87885

Time taken to run this cell : 0:05:50.596905
```

Precedure Followed

1. Applied CountVectorizer with 25000 features and ngram_range (1,4)
2. Tuned hyperparameters alpha for logistic regression and linear svm model using Gridsearch method.
3. Trained Logistic Regression with SGDClassifier and Liner SVM model with the best hyperparameter found.
4. I choose 25k data points for the assignment, due to my low system configuration.
5. Logistic Regression models gives the best F1-measure: 0.3954 among both the models.
6. Because the dimension is very high and linear model works fairly well and the complex model like random forest ,xgboost may not work well for this high dimensional, and it will be computationally more expensive than linear models.

In [48]:
```
from prettytable import PrettyTable
x = PrettyTable()

x.field_names = ["FEATURIZATION", "MODEL", "HAMMING_LOSS","MICRO_f1_SCORE"]

x.add_row(["BOW(4 gram)", "Logistic Regression", 0.00363736, 0.3797])
x.add_row(["", "Linear SVM", 0.00359556, 0.3954])



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

```
                        Performance Table
+---------------+---------------------+--------------+----------------+
| FEATURIZATION |        MODEL        | HAMMING_LOSS | MICRO_f1_SCORE |
+---------------+---------------------+--------------+----------------+
|  BOW(4 gram)  | Logistic Regression |  0.00363736  |     0.3797     |
|               |      Linear SVM     |  0.00359556  |     0.3954     |
+---------------+---------------------+--------------+----------------+
```

In [0]:
```

```