*CASE STUDY ON STACKOVERFLOW TAG PREDICTOR***

Introduction:

In this casestudy we will discusses how the tags are predicted to the asked question in the stackoverflow. Though their are more advanced algorithms to predicte the tags but using a simple linear algorithms like LogisticRegression and linearSVM we will predicte the tags because these both alogrithms are computationally less expensive.

In [1]:
```
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import sqlite3
import csv
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from wordcloud import WordCloud
import re
import os
from sqlalchemy import create_engine # database connection
import datetime as dt
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem.snowball import SnowballStemmer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.multiclass import OneVsRestClassifier
from sklearn.linear_model import SGDClassifier
from sklearn import metrics
from sklearn.metrics import f1_score,precision_score,recall_score
from sklearn import svm
from sklearn.linear_model import LogisticRegression
from datetime import datetime
from sklearn.metrics import hamming_loss
```

In [2]:
```
import nltk
nltk.download("punkt")
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\USER\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Out[2]: True

STCAKOVERFLOW: TAG PREDTICTOR

1. Business Problem

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 Statement

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 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

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).

Datafield 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 '&')

2.2 Mapping RealWorld Problem toMachine 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.

2.2.2 Performence 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)

3. Explolatory Data Analysis

3.1 Data Loading and Cleaning

3.1.1 Using Pandas with SQLite to Load the data

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

```
180000 rows
360000 rows
540000 rows
720000 rows
900000 rows
1080000 rows
1260000 rows
1440000 rows
1620000 rows
1800000 rows
1980000 rows
2160000 rows
2340000 rows
2520000 rows
2700000 rows
2880000 rows
3060000 rows
3240000 rows
3420000 rows
3600000 rows
3780000 rows
3960000 rows
4140000 rows
4320000 rows
4500000 rows
4680000 rows
4860000 rows
5040000 rows
5220000 rows
5400000 rows
5580000 rows
5760000 rows
5940000 rows
6120000 rows
Time taken to run this cell : 0:11:06.348946
```

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 genarate train.db file")
```

```
Number of rows in the database :
 6034196
Time taken to count the number of rows : 0:01:27.498394
```

3.1.3 Checking for duplicates

In [0]:
```
if os.path.isfile('train.db'):
    start = datetime.now()
    con = sqlite3.connect('train.db')
    df_no_dup = pd.read_sql_query('SELECT Title, Body, Tags, COUNT(*) as cnt_dup FROM data GROUP BY Title, Body, 
Tags', con)
    con.close()
    print("Time taken to run this cell :", datetime.now() - start)
else:
    print("Please download the train.db file from drive or run the first to genarate train.db file")
```

```
Time taken to run this cell : 2:34:21.578626
```

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

Out[0]:

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

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

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

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

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

In [0]:
```
a = df_no_dup["Tags"]

b = a[0].split(" ")
#b = a.apply(lambda text: len(text.split(" ")))
```

In [0]:
```
print(len(b))
```

```
2
```

In [0]:
```
start = datetime.now()
df_no_dup["tag_count"] = df_no_dup["Tags"].apply(lambda text: len(str(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 : 0:00:06.677897
```

Out[0]:

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

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

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

In [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 [3]:
```
#This method seems more appropriate to work with this much data.
#creating the connection with database file.
if os.path.isfile('train_no_dup.db'):
    start = datetime.now()
    con = sqlite3.connect('train_no_dup.db')
    tag_data = pd.read_sql_query("""SELECT Tags FROM no_dup_train""", con)
    #Always remember to close the database
    con.close()

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

```
Time taken to run this cell : 0:02:10.340639
```

3.2 Analysis of Tag

3.2.1 Total number of unique tags

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

Out[0]:

	Tags
1	c# silverlight data-binding
2	c# silverlight data-binding columns
3	jsp jstl
4	java jdbc
5	facebook api facebook-php-sdk

In [0]:
```
tag =[]
```

In [0]:
```
tag_data_vect = tag_data['Tags'].apply(lambda text: str(text).split())
```

In [0]:
```
print((tag_data_vect))
```

```
IOPub data rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_data_rate_limit`.

Current values:
NotebookApp.iopub_data_rate_limit=1000000.0 (bytes/sec)
NotebookApp.rate_limit_window=3.0 (secs)
```

In [0]:
```
import re
REGEX = re.compile(r",\s*")
def tokenize(text):
    return [tok.strip().lower() for tok in REGEX.split(text)]
```

In [0]:
```
tag = tag_data["Tags"]
```

In [0]:
```
print(tag)
```

```
1                                   c# silverlight data-binding
2                           c# silverlight data-binding columns
3                                                      jsp jstl
4                                                     java jdbc
5                               facebook api facebook-php-sdk
6                                      javascript asp.net web
7                                                     php forms
8                              real-analysis measure-theory
9                                                hibernate hql
10                                  iphone email-integration
11                                       java servlets jboss
12                 android android-widget android-service
13                     c# .net rijndaelmanaged cryptostream
14                                        javascript listbox
15                                              sql subquery
16                                                    c feof
17                        number-theory functions inequality
18                                            exponentiation
19                                  abstract-algebra modules
20                                              group-theory
21                                                  calculus
22                           c# visual-c++ opencv emgucv emgu
23                                 fonts xetex lyx ligatures
24                             asp.net vb.net drop-down-menu
25                                parsing haskell expression
26                                            jquery append
27            javascript jquery html jquery-ui jquery-plugins
28                                         linux filesystems
29              php google google-api oauth-2.0 google-plus
30                                               asp.net-mvc
                                ...
4206285                    javascript constructor this var
4206286         css internet-explorer-6 vertical-alignment
4206287                  xetex errors texlive mactex texshop
4206288                               flash actionscript-3
4206289                                               cron
4206290                                   git merge rebase
4206291                                             c# xna
4206292                                       r data.frame
4206293   iphone sms message mfmailcomposeviewcontroll i...
```

```
4206294                                   homework haskell
4206295                               php mysql xhtml utf-8
4206296                             mysql html linux symbols
4206297                       php javascript xpath domdocument
4206298                                   logic quantifiers
4206299        php javascript mysql utf-8 encodeuricomponent
4206300                                                agda
4206301                    ruby textmate tidy textmatebundles
4206302                             textmate textmatebundles
4206303                                        html unicode
4206304                              google-chrome keyboard
4206305                        c# silverlight windows-phone-7
4206306                                             sip sdp
4206307                                          c++ opengl
4206308                                               linux
4206309                   php php-errors zend-studio php-5.2
4206310                          wordpress wordpress-plugin
4206311                                      php mysql text
4206312                   php codeigniter character-encoding
4206313                              php email outlook mime
4206314                                                html
Name: Tags, Length: 4206314, dtype: object
```

In [0]:
```
print(tag_data_vect[1])
```

```
['c#', 'silverlight', 'data-binding', 'columns']
```

In [0]:
```
tag = tag.values
```

```
c# silverlight data-binding
```

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(preprocessor=lambda x: x,tokenizer = lambda x: str(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 : 42050
```

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 tags we have : ['.a', '.app', '.asp.net-mvc', '.aspxauth', '.bash-profile', '.class-file', '.cs-fil
e', '.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-elements
#Lets now store the document term matrix in a dictionary.
freqs = tag_dtm.sum(axis=0).A1
result = dict(zip(tags, freqs))
```

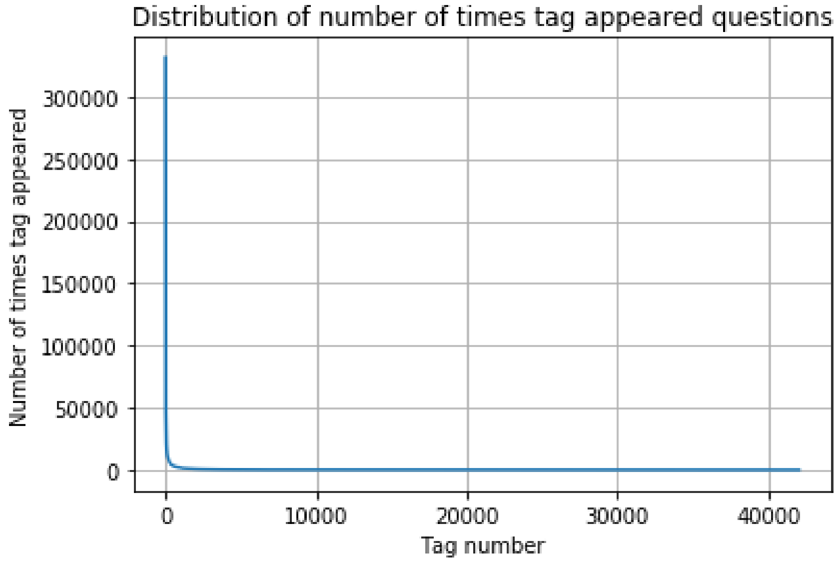
In [47]:
```
#Saving this dictionary to csv files.
if not os.path.isfile('tag_counts_dict_dtm.csv'):
    with open('tag_counts_dict_dtm.csv', 'w') as csv_file:
        writer = csv.writer(csv_file)
        for key, value in result.items():
            writer.writerow([key, value])
tag_df = pd.read_csv("tag_counts_dict_dtm.csv", names=['Tags', 'Counts'])
tag_df.head()
```

Out[47]:

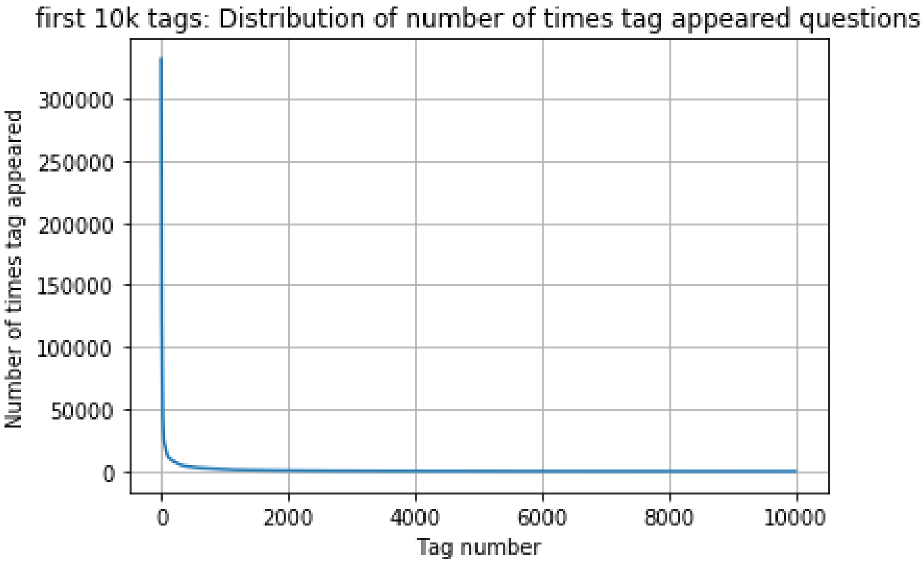
	Tags	Counts
0	.a	18
1	.app	37
2	.asp.net-mvc	1
3	.aspxauth	21
4	.bash-profile	138

In [48]:
```
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()
```



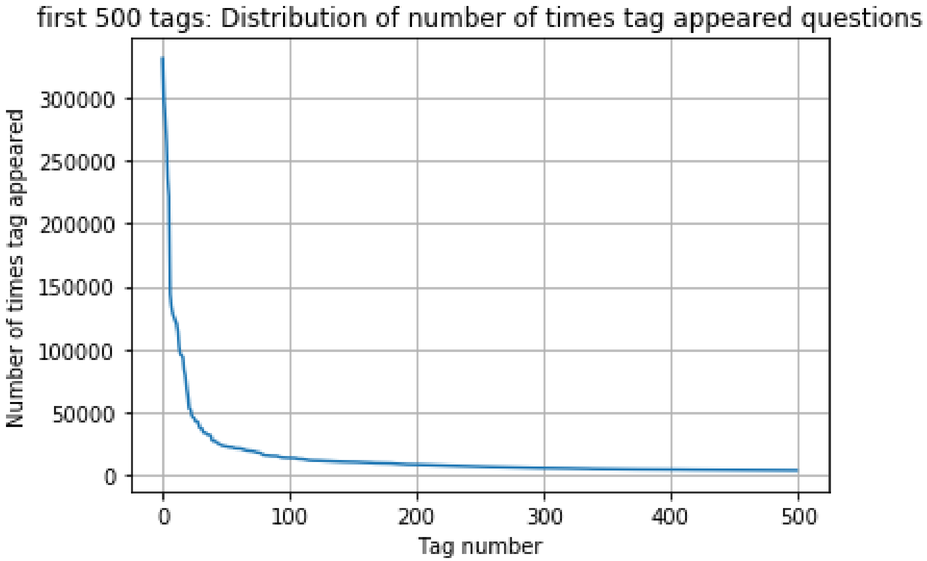
In [0]:
```
plt.plot(tag_counts[0:10000])
plt.title('first 10k tags: Distribution of number of times tag appeared questions')
plt.grid()
plt.xlabel("Tag number")
plt.ylabel("Number of times tag appeared")
plt.show()
print(len(tag_counts[0:10000:25]), tag_counts[0:10000:25])
```


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

```
400 [331505  44829  22429  17728  13364  11162  10029   9148   8054   7151
   6466   5865   5370   4983   4526   4281   4144   3929   3750   3593
   3453   3299   3123   2986   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]
```

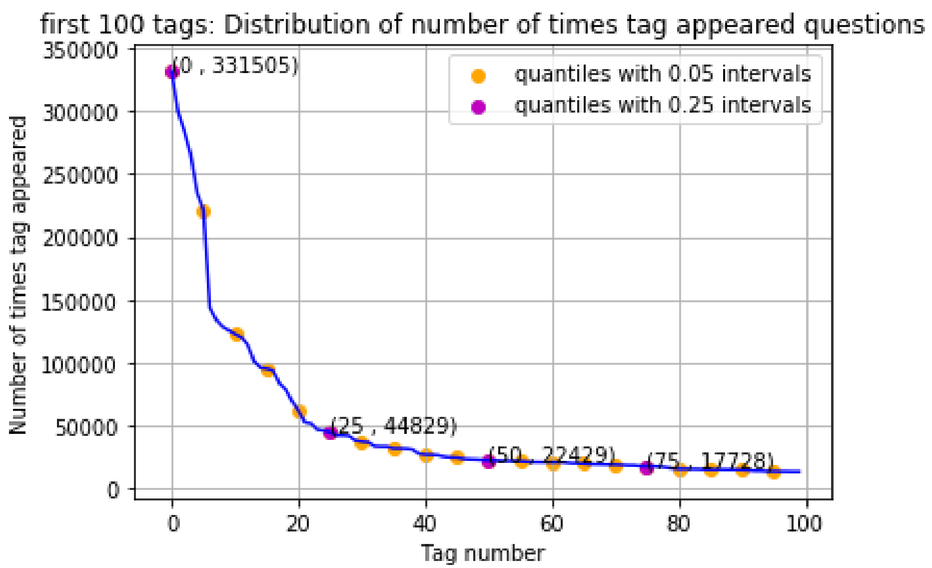
In [0]:
```
plt.plot(tag_counts[0:1000])
plt.title('first 1k tags: Distribution of number of times tag appeared questions')
plt.grid()
plt.xlabel("Tag number")
plt.ylabel("Number of times tag appeared")
plt.show()
print(len(tag_counts[0:1000:5]), tag_counts[0:1000:5])
```

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   3482]
```

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

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

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




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

In [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
```

Observation:

```
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

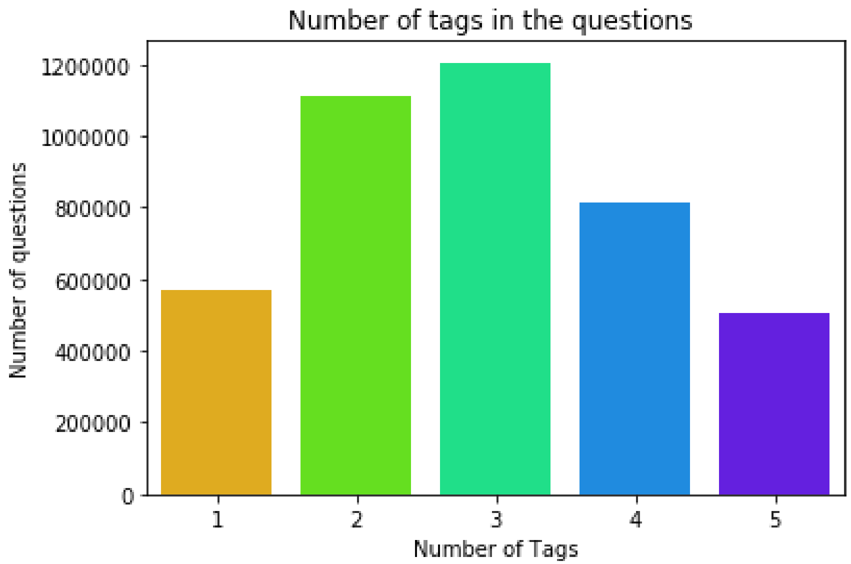
In [0]:
```
#Storing the count of tag in each question in list 'tag_count'
tag_quest_count = tag_dtm.sum(axis=1).tolist()
#Converting each value in the 'tag_quest_count' to integer.
tag_quest_count=[int(j) for i in tag_quest_count for j in i]
print ('We have total {} datapoints.'.format(len(tag_quest_count)))

print(tag_quest_count[:5])
```

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

In [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(tag_quest_count)))
```

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

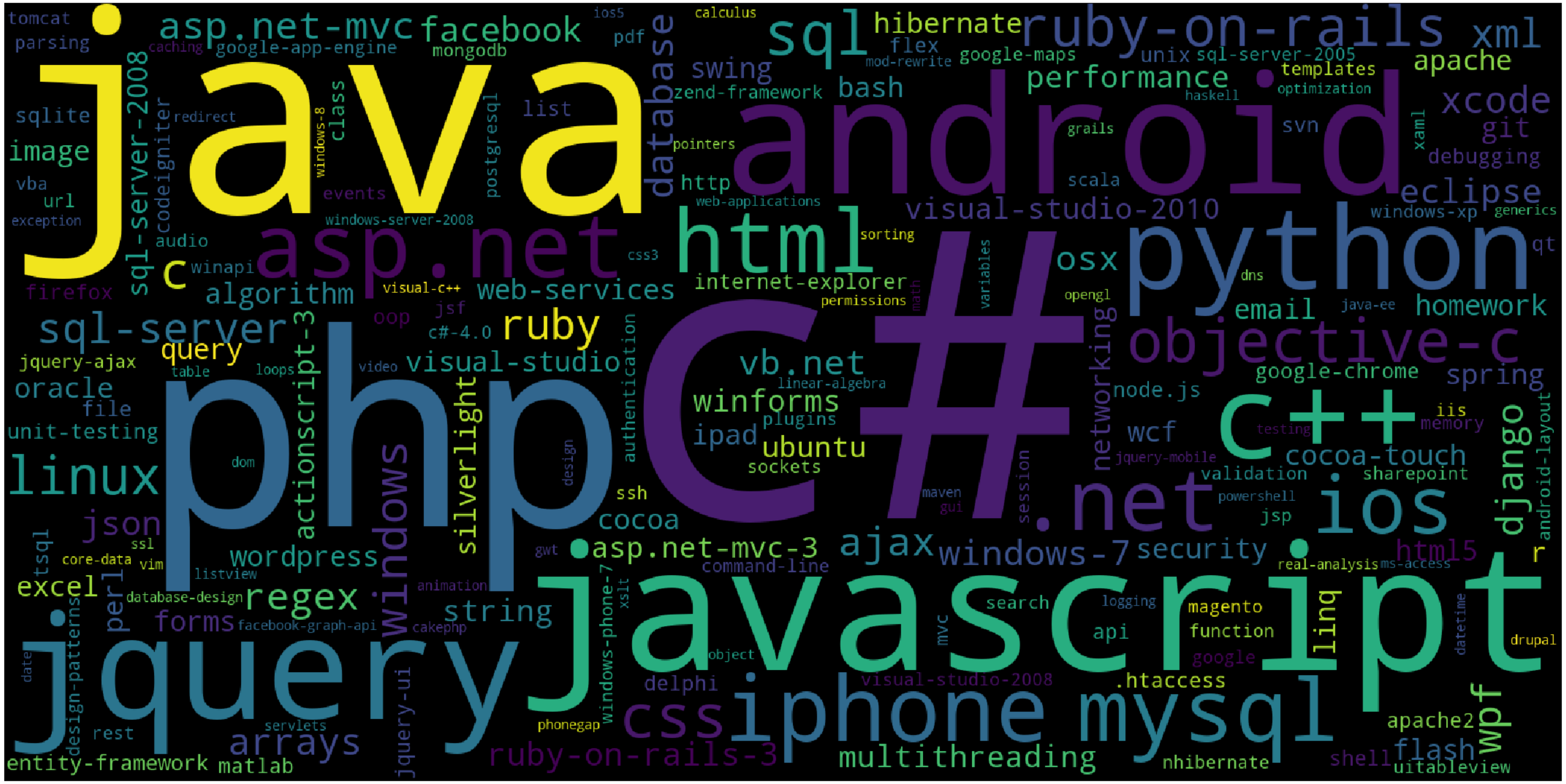
In [0]:
```
sns.countplot(tag_quest_count, palette='gist_rainbow')
plt.title("Number of tags in the questions ")
plt.xlabel("Number of Tags")
plt.ylabel("Number of questions")
plt.show()
```



Observation:

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)
```



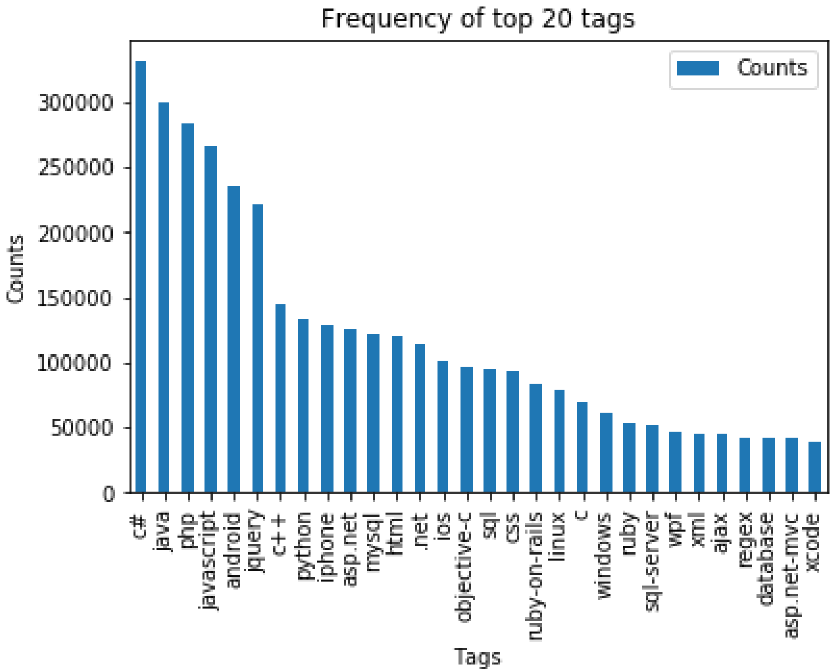
Time taken to run this cell : 0:00:22.725728

Observation:

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.

In [ ]:
```
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 Title FROM no_dup_train""", con)
    #Always remember to close the database
    con.close()

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

3.3 Title Analysis:

In [2]:
```
if os.path.isfile("train_no_dup.db"):
    con = sqlite3.connect("train_no_dup.db")
    title_data = pd.read_sql_query("SELECT Title FROM no_dup_train",con)
    con.close()
```

In [3]:
```
title_data_vect = title_data['Title'].apply(lambda text: str(text).split())
```

In [4]:
```
title_data_vect=pd.DataFrame(title_data_vect)
```

In [5]:
```
title_data_vect.shape
```

Out[5]: (4206315, 1)

In [6]:
```
len_of_title = title_data_vect["Title"].apply(lambda x: len(x))
```

In [7]:
```
len_of_title = pd.DataFrame(len_of_title)
```

In [8]:
```
len_of_title.columns = ["len_of_title"]
```

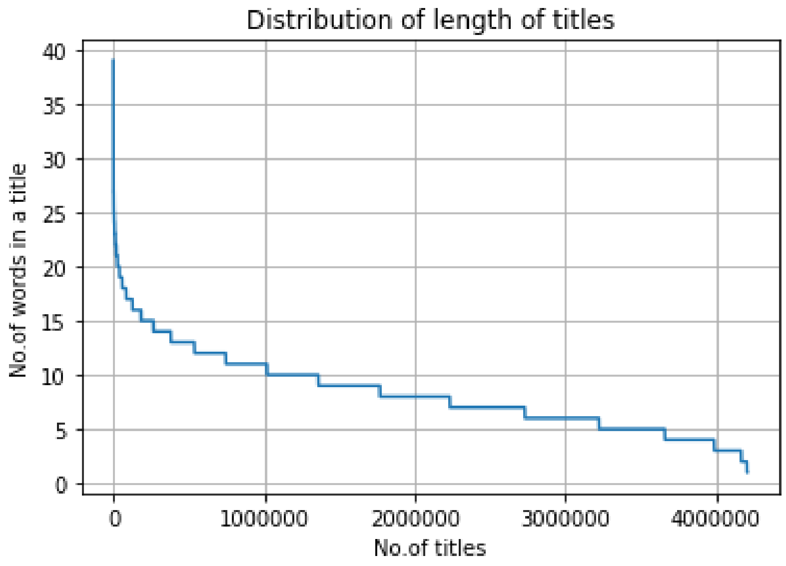
In [9]:
```
len_of_title.head()
```

Out[9]:

	len_of_title
0	11
1	5
2	5
3	2
4	6

In [10]:
```
title_df_sorted = len_of_title.sort_values(['len_of_title'], ascending=False)
title_counts = title_df_sorted['len_of_title'].values
```

In [12]:
```
#for i in range(len(sizeoftitles)):
#plt.figure(figsize=(20,10))
#plt.subplot(1,2,1)
plt.plot(title_counts)
plt.title("Distribution of length of titles")
plt.xlabel("No.of titles")
plt.ylabel("No.of words in a title")
plt.grid()
plt.show()
```



In [14]:
```
print("the average length of titles:",title_counts.mean())
```

```
the average length of titles: 8.33400850863523
```

Observation:

- From the above analysis more two and half million titles contains 5-10 words.
- On average the length of titles are 8.
- So their is much possiblity that thier are much possibility that a title may explain tags, because mostly humans tend to explain the problem in one line.

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

3.4.1 Preprocessing

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

In [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 [6]:
```
#http://www.sqlitetutorial.net/sqlite-python/create-tables/
def create_connection(db_file):
    """ create a database connection to the SQLite database
        specified by db_file
    :param db_file: database file
    :return: Connection object or None
    """
    try:
        conn = sqlite3.connect(db_file)
        return conn
    except Error as e:
        print(e)

    return None

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

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

def create_database_table(database, query):
    conn = create_connection(database)
    if conn is not None:
```

```
        create_table(conn, query)
        checkTableExists(conn)
    else:
        print("Error! cannot create the database connection.")
    conn.close()

sql_create_table = """CREATE TABLE IF NOT EXISTS QuestionsProcessed (question text NOT NULL, code text, tags tex
t, words_pre integer, words_post integer, is_code integer);"""
create_database_table("TitleMoreWeighted.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-table
start = datetime.now()
read_db = 'train_no_dup.db'
write_db = 'TitleMoreWeighted.db'
if os.path.isfile(read_db):
    conn_r = create_connection(read_db)
    if conn_r is not None:
        reader =conn_r.cursor()
        reader.execute("SELECT Title, Body, Tags From no_dup_train ORDER BY RANDOM() LIMIT 500001;")

if os.path.isfile(write_db):
    conn_w = create_connection(write_db)
    if conn_w is not None:
        tables = checkTableExists(conn_w)
        writer =conn_w.cursor()
        if tables != 0:
            writer.execute("DELETE FROM QuestionsProcessed WHERE 1")
            print("Cleared All the rows")
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 : 1:55:41.168536
```

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.DOTALL)
    question=striphtml(question.encode('utf-8'))

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

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

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

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

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

    len_post+=len(question)
```

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

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

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

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

```
number of questions completed= 100000
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: 1168
Avg. length of questions(Title+Body) after processing: 408
Percent of questions containing code: 57
Time taken to run this cell : 1:49:29.786589
```

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):
    print(".......")
    conn_r = create_connection(write_db)
    if conn_r is not None:
        reader =conn_r.cursor()
        reader.execute("SELECT question From QuestionsProcessed LIMIT 5")
        print("Questions after preprocessed")
        print('='*100)
        reader.fetchone()
        for row in reader:
            print(row)
            print('-'*100)
conn_r.commit()
conn_r.close()
```

```
.......
Questions after preprocessed
====================================================================================================
('android renderscript run gpu android renderscript run gpu android renderscript run gpu android devic renderscr
ipt execut gpu instead cpu someth yet implement anywher',)
----------------------------------------------------------------------------------------------------
('resiz uitableviewcel show imag maximum size resiz uitableviewcel show imag maximum size resiz uitableviewcel s
how imag maximum size want show imag text uitableview add imag uitableviewcel break layout overlap next header s
et size cell contain imag layout get broken found objc-cod suggest done use heightforrowatindexpath find anyth m
onotouch specif',)
----------------------------------------------------------------------------------------------------
('add radio button footer cell jqgrid add radio button footer cell jqgrid add radio button footer cell jqgrid ne
w jqgrid would like use footer section grid column selector tri use radio button certain cell user select certai
n column grid graph column select need event fire show column chosen user pleas help thank much',)
----------------------------------------------------------------------------------------------------
('filepath.walk panic filepath.walk panic filepath.walk panic tri code wiki go program languag ni put data folde
r file folder code function follow get could anyon pleas guid',)
----------------------------------------------------------------------------------------------------
```

svaing preprocessed data into database

In [41]:
```
#Taking 0.5 Million entries to a dataframe.
start = datetime.now()
write_db = 'TitleMoreWeight.db'
if os.path.isfile('TitleMoreWeight.db'):
    conn_r = create_connection('TitleMoreWeighted.db')
    if conn_r is not None:
        preprocessed_data = pd.read_sql_query("""SELECT question, Tags FROM QuestionsProcessed""", conn_r)
conn_r.commit()
conn_r.close()
print("Time taken to run this cell :", datetime.now() - start)
```

```
Time taken to run this cell : 0:00:04.012296
```

In [42]:
```
preprocessed_data.head()
```

Out[42]:

	question	tags
0	correct method updat post content plugin corre...	plugins posts content
1	android renderscript run gpu android renderscr...	android renderscript
2	resiz uitableviewcel show imag maximum size re...	iphone monotouch
3	add radio button footer cell jqgrid add radio ...	jqgrid
4	filepath.walk panic filepath.walk panic filepa...	go

In [43]:
```
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
```

In [0]:
```
import pickle
with open("preprocessed_data.pkl","wb") as f:
    pickle.dump(preprocessed_data,f)
```

In [0]:
```
!ls
```

```
gdrive  sample_data
```

In [4]:
```
%cd ./gdrive
```

```
/content/gdrive
```

In [0]:
```
import pickle
#drive.mount('/content/drive')
DATA_PATH = "My Drive/stackoverflow_tag_predicte"
infile = open(DATA_PATH+"/preprocessed_data.pkl",'rb')
preprocessed_data = pickle.load(infile)
```

Converting string Tags to multilable output variables

In [44]:
```
vectorizer = CountVectorizer(tokenizer = lambda x: x.split(), binary='true')
multilabel_y = vectorizer.fit_transform(preprocessed_data['tags'])
```

In [52]:
```
#'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:])

### 3.2.3 Number of times a tag appeared

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

Selecting 500 Tags

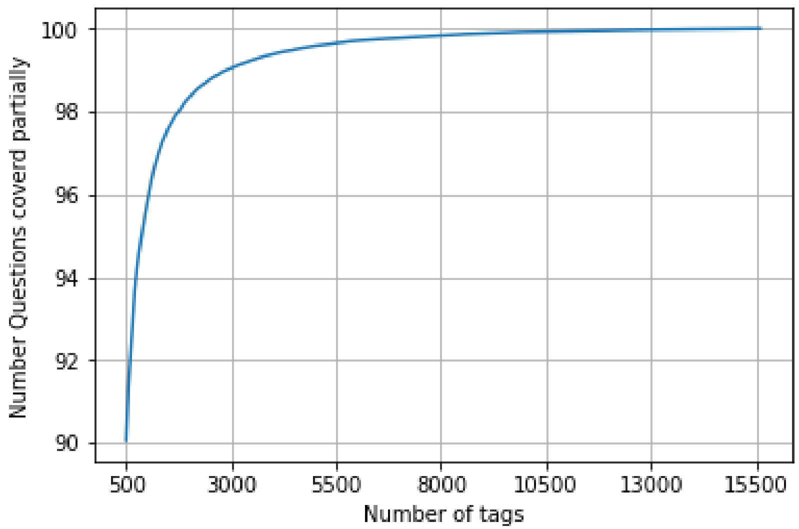
In [28]:
```
def tags_to_choose(n):
    t = multilabel_y.sum(axis=0).tolist()[0]
    sorted_tags_i = sorted(range(len(t)), key=lambda i: t[i], reverse=True)
    multilabel_yn=multilabel_y[:,sorted_tags_i[:n]]
    return multilabel_yn

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

In [31]:
```
questions_explained = []
total_tags=multilabel_y.shape[1]
total_qs=preprocessed_data.shape[0]
for i in range(500, total_tags, 100):
    questions_explained.append(np.round(((total_qs-questions_explained_fn(i))/total_qs)*100,3))
```

In [32]:
```
print(questions_explained)
```

```
[90.057, 91.341, 92.094, 92.917, 93.659, 94.159, 94.573, 94.896, 95.234, 95.521, 95.799, 96.062, 96.338, 96.569,
96.759, 96.93, 97.091, 97.244, 97.367, 97.476, 97.592, 97.686, 97.784, 97.868, 97.941, 97.999, 98.073, 98.146, 9
8.226, 98.283, 98.341, 98.403, 98.457, 98.503, 98.551, 98.59, 98.63, 98.664, 98.713, 98.746, 98.78, 98.812, 98.8
41, 98.87, 98.893, 98.935, 98.959, 98.986, 99.007, 99.03, 99.049, 99.067, 99.089, 99.112, 99.131, 99.149, 99.16
4, 99.184, 99.201, 99.223, 99.237, 99.253, 99.273, 99.286, 99.304, 99.322, 99.337, 99.35, 99.365, 99.377, 99.38
8, 99.405, 99.415, 99.426, 99.438, 99.446, 99.455, 99.463, 99.472, 99.481, 99.494, 99.505, 99.513, 99.524, 99.53
1, 99.541, 99.55, 99.559, 99.57, 99.576, 99.582, 99.591, 99.596, 99.602, 99.608, 99.615, 99.621, 99.63, 99.638,
99.644, 99.652, 99.659, 99.667, 99.673, 99.681, 99.686, 99.693, 99.699, 99.702, 99.708, 99.713, 99.716, 99.72, 9
9.724, 99.727, 99.73, 99.732, 99.736, 99.739, 99.745, 99.748, 99.751, 99.754, 99.758, 99.761, 99.764, 99.766, 9
9.769, 99.772, 99.775, 99.779, 99.782, 99.786, 99.789, 99.792, 99.794, 99.796, 99.799, 99.802, 99.804, 99.808, 9
9.812, 99.815, 99.817, 99.821, 99.824, 99.827, 99.83, 99.833, 99.836, 99.839, 99.842, 99.844, 99.847, 99.849, 9
9.851, 99.854, 99.858, 99.861, 99.864, 99.866, 99.868, 99.87, 99.871, 99.873, 99.875, 99.876, 99.878, 99.881, 9
9.883, 99.887, 99.888, 99.89, 99.891, 99.893, 99.894, 99.897, 99.899, 99.901, 99.903, 99.904, 99.906, 99.908, 9
9.908, 99.91, 99.912, 99.913, 99.914, 99.916, 99.918, 99.919, 99.92, 99.922, 99.923, 99.923, 99.924, 99.925, 99.
925, 99.926, 99.927, 99.928, 99.93, 99.93, 99.931, 99.933, 99.934, 99.936, 99.937, 99.939, 99.939, 99.941, 99.94
3, 99.944, 99.945, 99.947, 99.948, 99.949, 99.95, 99.951, 99.952, 99.953, 99.953, 99.955, 99.956, 99.957, 99.95
7, 99.959, 99.96, 99.962, 99.963, 99.964, 99.964, 99.964, 99.965, 99.965, 99.966, 99.966, 99.967, 99.968, 99.96
8, 99.968, 99.969, 99.97, 99.97, 99.971, 99.972, 99.972, 99.973, 99.973, 99.973, 99.974, 99.974, 99.975, 99.975,
99.976, 99.976, 99.977, 99.977, 99.978, 99.979, 99.979, 99.979, 99.98, 99.98, 99.98, 99.981, 99.981, 99.982, 99.
982, 99.983, 99.984, 99.985, 99.985, 99.986, 99.986, 99.986, 99.987, 99.988, 99.989, 99.99, 99.99, 99.99, 99.99,
99.991, 99.991, 99.991, 99.992, 99.993, 99.994, 99.994, 99.995, 99.995, 99.996, 99.997, 99.997, 99.997, 99.998,
99.998, 99.998, 99.998, 99.999, 99.999, 100.0]
```

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(it covers 90% of the tags)
print("with ",5500,"tags we are covering ",questions_explained[50],"% of questions")
print("with ",500,"tags we are covering ",questions_explained[0],"% of questions")
```



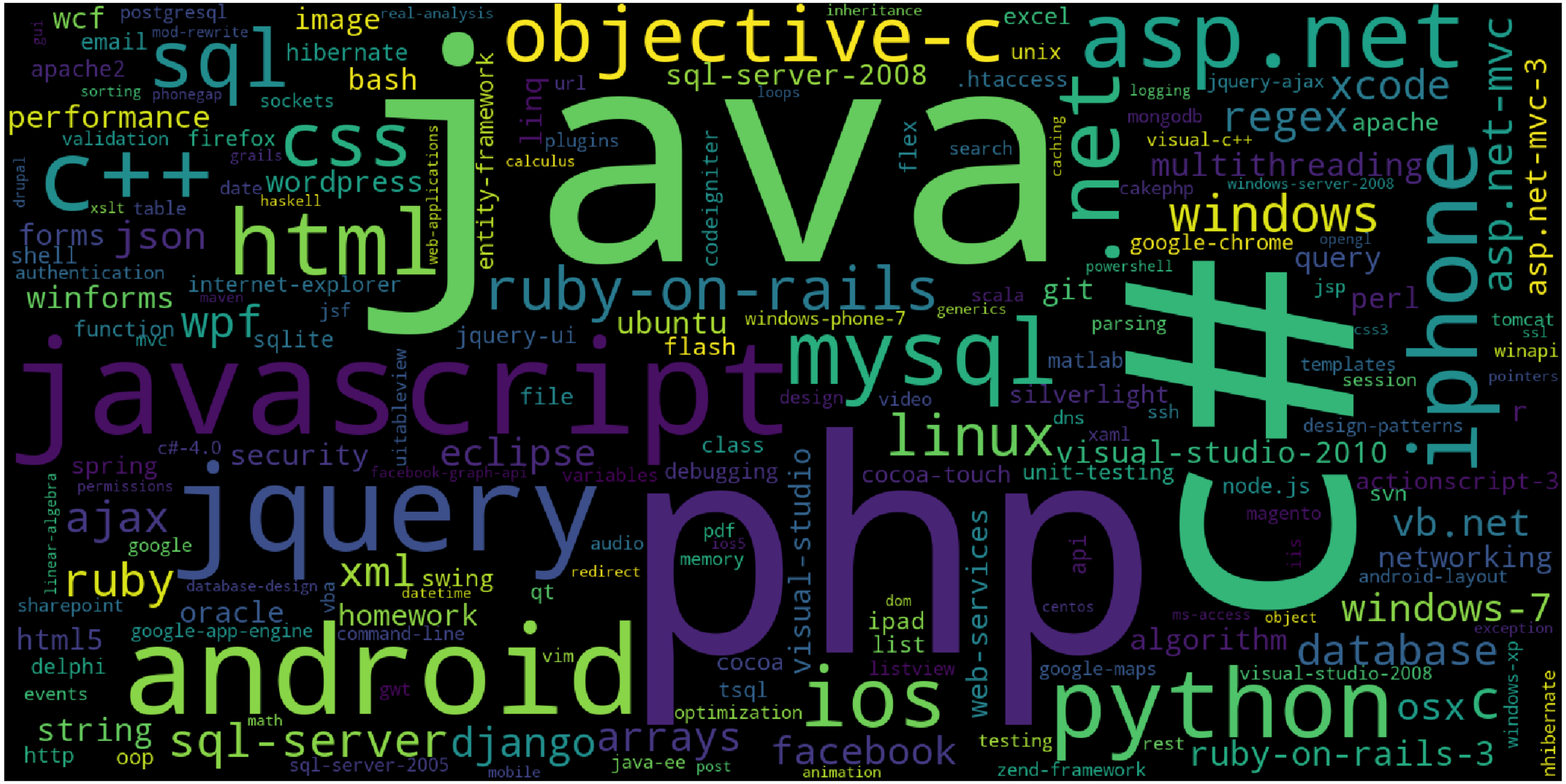

```
with  5500 tags we are covering  99.049 % of questions
with  500 tags we are covering  90.057 % of questions
```

In [10]:
```
# we will be taking 500 tags
multilabel_yx = tags_to_choose(500)
print("number of questions that are not covered :", questions_explained_fn(500),"out of ", total_qs)
```

```
number of questions that are not covered : 49715 out of  500000
```

Analysing tags on 0.5M datapoints

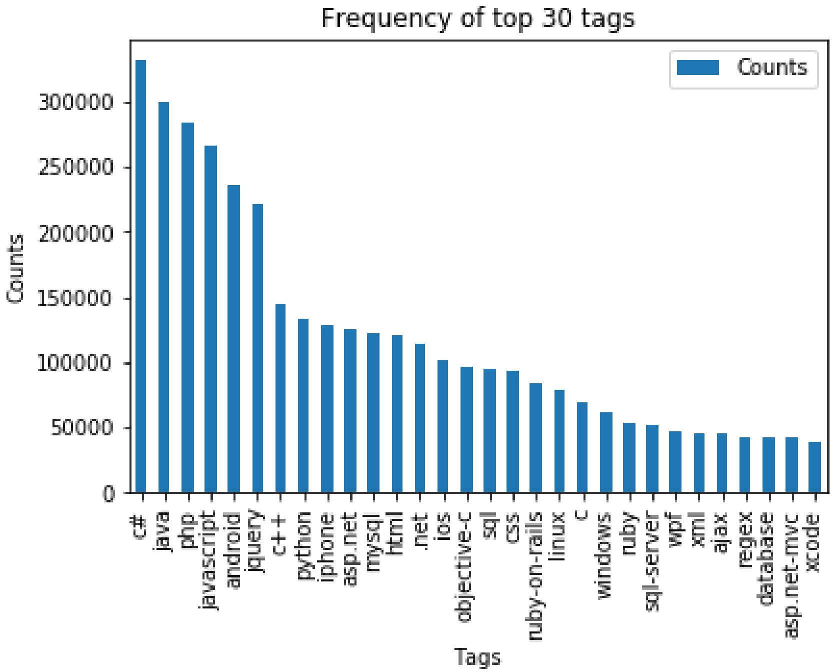
In [53]:
```
start = datetime.now()

# 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 : 0:00:09.145156

In [54]:
```
tag_df_sorted = tag_df.sort_values(['Counts'], ascending=False)
tag_counts = tag_df_sorted['Counts'].values
```

In [56]:
```
i=np.arange(30)
tag_df_sorted.head(30).plot(kind='bar')
plt.title('Frequency of top 30 tags')
plt.xticks(i, tag_df_sorted['Tags'])
plt.xlabel('Tags')
plt.ylabel('Counts')
plt.show()
```



Observation:

In the sampled 0.5M datapoints their not much change in the tags count still C#,java, php, android,etc tags are the most frequently occuring tags

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(preprocessed_data.shape[0] - 400000)

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

In [12]:
```
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)
```

3.5 Featurization

In [13]:
```
start = datetime.now()
vectorizer = CountVectorizer(min_df=0.00009, max_features=200000, \
                             tokenizer = lambda x: x.split(), ngram_range=(1,3))#Here iam taking three grams due
to insufficient memory
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 : 0:04:39.751953
```

In [14]:
```
print("Dimensions of train data X:",x_train_multilabel.shape, "Y :",y_train.shape)
print("Dimensions of test data X:",x_test_multilabel.shape,"Y:",y_test.shape)
```

```
Dimensions of train data X: (400000, 87564) Y : (400000, 500)
Dimensions of test data X: (100000, 87564) Y: (100000, 500)
```

In [0]:
```
#start = datetime.now()
clf = SGDClassifier(loss='log', penalty='l2')
tuned_parameter ={'alpha': [10**-4, 10**-2, 10**1, 10**-2, 10**-4]}
#paramGrid = ParameterGrid(tuned_parameter)
model = OneVsRestClassifier(GridSearchCV(clf,  tuned_parameter, scoring = 'f1_micro', cv=3,n_jobs=-1))
model.fit(x_train_multilabel, y_train)
#print("the time taken to run this cell:",datetime.now() - start)
```

Out[0]:
```
OneVsRestClassifier(estimator=GridSearchCV(cv=3, error_score='raise-deprecating',
       estimator=SGDClassifier(alpha=0.0001, average=False, class_weight=None,
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='log', max_iter=None,
       n_iter...re_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='f1_micro', verbose=0),
          n_jobs=None)
```

In [0]:
```
start = datetime.now()
classifier = OneVsRestClassifier(estimator=SGDClassifier(alpha=0.0001, loss='log',penalty='l2'),n_jobs=-1)
classifier.fit(x_train_multilabel, y_train)
predictions = classifier.predict (x_test_multilabel)


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


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

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

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

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

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

```
Accuracy : 0.23656
Hamming loss  0.00283316
Micro-average quality numbers
Precision: 0.6895, Recall: 0.3906, F1-measure: 0.4987
Macro-average quality numbers
Precision: 0.5484, Recall: 0.3143, F1-measure: 0.3886
              precision    recall  f1-score   support

           0       0.58      0.37      0.45      7740
           1       0.83      0.41      0.55      7039
           2       0.84      0.55      0.67      6884
           3       0.74      0.42      0.54      6261
           4       0.94      0.76      0.84      5611
           5       0.86      0.65      0.74      5273
           6       0.81      0.51      0.63      3516
           7       0.85      0.63      0.72      3121
           8       0.68      0.43      0.53      3007
           9       0.75      0.45      0.56      2992
          10       0.83      0.62      0.71      2962
          11       0.55      0.19      0.28      2768
          12       0.46      0.17      0.25      2645
          13       0.64      0.31      0.42      2402
          14       0.53      0.33      0.41      2247
          15       0.53      0.26      0.35      2183
          16       0.77      0.56      0.65      2134
          17       0.81      0.56      0.66      2037
          18       0.60      0.35      0.44      1866
          19       0.63      0.31      0.41      1698
          20       0.37      0.16      0.22      1506
          21       0.74      0.38      0.50      1244
          22       0.59      0.39      0.47      1263
          23       0.88      0.62      0.73      1178
          24       0.68      0.54      0.60      1123
          25       0.69      0.37      0.48      1060
          26       0.35      0.14      0.20      1025
          27       0.86      0.67      0.76       994
          28       0.65      0.42      0.51      1002
          29       0.63      0.26      0.37       899
          30       0.93      0.75      0.83       875
          31       0.57      0.30      0.39       862
```

32	0.57	0.21	0.31	819
33	0.61	0.40	0.48	791
34	0.85	0.34	0.49	776
35	0.80	0.58	0.67	752
36	0.79	0.57	0.67	803
37	0.79	0.58	0.67	772
38	0.35	0.18	0.24	741
39	0.47	0.22	0.30	655
40	0.64	0.45	0.53	593
41	0.37	0.11	0.17	604
42	0.70	0.33	0.44	587
43	0.71	0.40	0.51	574
44	0.71	0.31	0.44	551
45	0.64	0.34	0.45	615
46	0.25	0.05	0.09	556
47	0.33	0.06	0.10	557
48	0.89	0.69	0.78	551
49	0.43	0.15	0.22	550
50	0.78	0.35	0.48	474
51	0.70	0.53	0.60	542
52	0.37	0.17	0.24	529
53	0.51	0.14	0.21	533
54	0.79	0.43	0.56	547
55	0.69	0.37	0.49	533
56	0.30	0.06	0.10	517
57	0.67	0.15	0.24	521
58	0.53	0.24	0.33	489
59	0.47	0.23	0.31	510
60	0.79	0.56	0.65	488
61	0.90	0.77	0.83	508
62	0.77	0.51	0.61	469
63	0.94	0.68	0.79	486
64	0.80	0.32	0.46	465
65	0.70	0.42	0.53	431
66	0.74	0.45	0.56	455
67	0.75	0.50	0.60	447
68	0.75	0.30	0.43	485
69	0.47	0.23	0.31	428
70	0.80	0.52	0.63	472
71	0.83	0.49	0.62	447

72	0.80	0.26	0.39	416
73	0.57	0.35	0.43	421
74	0.72	0.57	0.64	411
75	0.19	0.04	0.07	406
76	0.89	0.65	0.75	416
77	0.49	0.35	0.41	385
78	0.20	0.03	0.05	411
79	0.43	0.16	0.23	386
80	0.65	0.41	0.50	389
81	0.72	0.38	0.50	385
82	0.41	0.20	0.27	360
83	0.77	0.51	0.61	339
84	0.78	0.53	0.63	388
85	0.52	0.25	0.34	378
86	0.57	0.30	0.39	356
87	0.88	0.53	0.66	373
88	0.84	0.64	0.72	351
89	0.96	0.55	0.70	364
90	0.92	0.59	0.72	327
91	0.62	0.22	0.33	340
92	0.77	0.49	0.60	339
93	0.62	0.16	0.25	356
94	0.72	0.40	0.52	338
95	0.49	0.23	0.32	330
96	0.88	0.70	0.78	321
97	0.31	0.05	0.09	315
98	0.57	0.18	0.28	319
99	0.88	0.55	0.67	319
100	0.70	0.59	0.64	299
101	0.95	0.70	0.81	329
102	0.27	0.06	0.10	335
103	0.51	0.23	0.32	318
104	0.93	0.71	0.81	313
105	0.87	0.59	0.70	313
106	0.38	0.09	0.15	299
107	0.48	0.30	0.37	291
108	0.37	0.18	0.25	282
109	0.69	0.43	0.53	285
110	0.79	0.55	0.65	298
111	0.61	0.43	0.51	292

112	0.61	0.21	0.32	271
113	0.52	0.43	0.47	298
114	0.80	0.45	0.58	286
115	0.39	0.20	0.27	290
116	0.62	0.19	0.29	279
117	0.35	0.15	0.21	250
118	0.65	0.37	0.47	289
119	0.97	0.70	0.82	243
120	0.95	0.80	0.87	254
121	0.66	0.54	0.59	287
122	0.41	0.12	0.19	283
123	0.56	0.27	0.36	279
124	0.47	0.24	0.31	259
125	0.61	0.25	0.36	275
126	0.29	0.02	0.04	266
127	0.43	0.19	0.26	261
128	0.85	0.60	0.70	279
129	0.95	0.71	0.82	269
130	0.30	0.08	0.13	258
131	0.57	0.43	0.49	272
132	0.67	0.43	0.53	274
133	0.45	0.12	0.19	253
134	0.80	0.62	0.70	265
135	0.29	0.06	0.10	248
136	0.71	0.40	0.52	240
137	0.61	0.57	0.59	248
138	0.21	0.01	0.02	243
139	0.41	0.10	0.16	269
140	0.65	0.22	0.33	233
141	0.42	0.15	0.23	252
142	0.88	0.73	0.80	232
143	0.27	0.13	0.18	244
144	0.63	0.35	0.45	269
145	0.52	0.26	0.35	236
146	0.94	0.74	0.82	239
147	0.22	0.04	0.07	225
148	0.25	0.07	0.10	261
149	0.34	0.26	0.29	244
150	0.43	0.08	0.14	293
151	0.49	0.19	0.28	233

152	0.61	0.51	0.56	225
153	0.52	0.12	0.20	262
154	0.73	0.56	0.63	221
155	0.30	0.29	0.30	232
156	0.38	0.16	0.23	218
157	0.73	0.50	0.60	218
158	0.98	0.72	0.83	235
159	0.52	0.36	0.43	244
160	0.93	0.70	0.80	224
161	0.49	0.26	0.34	234
162	0.43	0.18	0.26	213
163	0.81	0.57	0.67	232
164	0.71	0.32	0.44	200
165	0.42	0.38	0.40	216
166	0.68	0.53	0.60	217
167	0.50	0.16	0.25	246
168	0.73	0.45	0.56	211
169	0.42	0.33	0.37	216
170	0.51	0.33	0.40	225
171	0.90	0.71	0.80	224
172	0.67	0.18	0.28	246
173	0.50	0.33	0.39	211
174	0.66	0.39	0.49	208
175	0.76	0.49	0.59	226
176	0.42	0.08	0.13	229
177	0.49	0.22	0.30	205
178	0.50	0.15	0.23	220
179	0.28	0.05	0.09	219
180	0.44	0.20	0.27	238
181	0.67	0.61	0.64	200
182	0.96	0.69	0.81	215
183	0.29	0.09	0.14	228
184	0.50	0.25	0.33	233
185	0.67	0.56	0.61	214
186	0.27	0.09	0.14	203
187	0.91	0.63	0.74	179
188	0.45	0.37	0.41	207
189	0.76	0.39	0.51	194
190	0.50	0.15	0.23	212
191	0.78	0.54	0.64	202

192	0.17	0.02	0.04	200
193	0.97	0.67	0.79	215
194	0.46	0.16	0.24	197
195	0.55	0.23	0.33	192
196	0.95	0.56	0.70	196
197	0.42	0.26	0.32	204
198	0.38	0.32	0.35	203
199	0.29	0.14	0.19	217
200	0.70	0.39	0.50	189
201	0.80	0.59	0.68	182
202	0.75	0.40	0.52	191
203	0.59	0.15	0.24	173
204	0.17	0.02	0.04	194
205	0.89	0.69	0.78	192
206	0.08	0.01	0.01	193
207	0.29	0.12	0.17	194
208	0.20	0.10	0.13	188
209	0.71	0.45	0.55	176
210	0.68	0.44	0.53	198
211	0.38	0.08	0.13	187
212	0.68	0.57	0.62	187
213	0.18	0.03	0.05	193
214	0.54	0.31	0.40	173
215	0.61	0.30	0.40	182
216	0.97	0.67	0.79	166
217	0.39	0.16	0.23	182
218	0.41	0.13	0.20	194
219	0.97	0.77	0.86	185
220	0.28	0.17	0.21	163
221	0.97	0.77	0.86	193
222	0.76	0.47	0.58	180
223	0.53	0.30	0.39	194
224	0.72	0.41	0.53	169
225	0.58	0.33	0.42	175
226	0.26	0.06	0.10	158
227	0.86	0.47	0.61	169
228	0.49	0.16	0.24	165
229	0.58	0.50	0.54	196
230	0.54	0.43	0.48	159
231	0.56	0.36	0.44	150

232	0.80	0.48	0.60	149
233	0.21	0.03	0.06	154
234	0.19	0.06	0.09	168
235	0.68	0.52	0.59	141
236	0.40	0.29	0.34	175
237	0.72	0.34	0.46	173
238	0.59	0.13	0.21	153
239	0.84	0.57	0.68	159
240	0.13	0.02	0.03	166
241	0.25	0.19	0.22	156
242	0.78	0.51	0.62	150
243	0.34	0.22	0.27	153
244	0.86	0.64	0.73	144
245	0.73	0.34	0.46	153
246	0.60	0.28	0.38	171
247	0.64	0.38	0.47	128
248	0.44	0.08	0.13	141
249	0.47	0.18	0.26	155
250	0.69	0.56	0.62	156
251	0.50	0.23	0.31	166
252	0.61	0.27	0.38	153
253	0.57	0.41	0.47	150
254	0.41	0.30	0.35	165
255	0.68	0.53	0.59	156
256	0.55	0.48	0.51	158
257	0.79	0.53	0.63	157
258	0.38	0.03	0.06	143
259	0.39	0.19	0.25	144
260	0.21	0.09	0.12	159
261	0.49	0.39	0.44	152
262	0.14	0.02	0.03	161
263	0.64	0.44	0.52	153
264	0.68	0.29	0.41	169
265	0.24	0.07	0.11	134
266	0.65	0.37	0.47	148
267	0.53	0.27	0.36	141
268	0.35	0.17	0.23	138
269	0.57	0.21	0.31	114
270	0.64	0.41	0.50	137
271	0.11	0.01	0.01	153

272	0.82	0.51	0.63	150
273	0.65	0.31	0.41	131
274	0.75	0.47	0.58	141
275	0.84	0.73	0.79	147
276	0.00	0.00	0.00	147
277	0.76	0.64	0.69	137
278	0.86	0.52	0.65	147
279	1.00	0.70	0.82	132
280	0.52	0.38	0.44	135
281	0.12	0.01	0.02	144
282	0.14	0.01	0.02	117
283	0.32	0.06	0.10	139
284	0.62	0.37	0.46	121
285	0.39	0.22	0.28	134
286	0.58	0.27	0.37	142
287	0.66	0.32	0.43	119
288	0.59	0.43	0.50	119
289	0.48	0.11	0.18	126
290	0.94	0.56	0.70	142
291	0.37	0.12	0.18	133
292	0.78	0.42	0.55	129
293	0.64	0.33	0.43	128
294	0.31	0.17	0.22	143
295	0.18	0.02	0.04	127
296	0.32	0.17	0.22	120
297	0.37	0.22	0.27	129
298	0.56	0.22	0.31	133
299	0.55	0.29	0.38	126
300	0.21	0.11	0.15	127
301	0.18	0.06	0.09	112
302	0.71	0.47	0.57	149
303	0.83	0.66	0.73	123
304	0.23	0.04	0.07	131
305	0.45	0.33	0.38	120
306	0.42	0.25	0.31	132
307	0.88	0.61	0.72	127
308	0.29	0.21	0.24	106
309	0.34	0.21	0.26	141
310	0.57	0.29	0.39	146
311	0.55	0.47	0.51	126

```
312       0.00      0.00      0.00       127
313       0.45      0.42      0.44       107
314       0.44      0.27      0.34       118
315       0.93      0.69      0.79       150
316       0.38      0.22      0.28       130
317       0.30      0.06      0.10       119
318       0.56      0.36      0.44       131
319       0.22      0.06      0.09       126
320       0.32      0.11      0.17       116
321       0.52      0.29      0.37       117
322       0.51      0.18      0.26       136
323       0.49      0.18      0.26       135
324       0.41      0.28      0.34       109
325       0.36      0.17      0.23       119
326       0.42      0.09      0.15       119
327       0.36      0.11      0.17       110
328       0.42      0.15      0.23       110
329       0.57      0.38      0.46       125
330       0.39      0.06      0.11       110
331       0.58      0.42      0.49       116
332       0.49      0.37      0.42       105
333       0.20      0.01      0.02       108
334       0.25      0.11      0.15       125
335       0.57      0.48      0.52       122
336       0.53      0.29      0.38       106
337       0.70      0.39      0.50       118
338       0.52      0.12      0.19       119
339       0.12      0.03      0.04       116
340       0.21      0.06      0.10       113
341       0.23      0.09      0.13       119
342       0.13      0.03      0.05        94
343       0.28      0.08      0.12       103
344       0.25      0.13      0.17       119
345       0.40      0.20      0.27        95
346       0.34      0.15      0.21       109
347       0.38      0.18      0.24        96
348       0.45      0.34      0.39       113
349       0.47      0.31      0.37       116
350       0.69      0.58      0.63       121
351       0.42      0.07      0.13       107
```

```
352       0.55      0.37      0.44       111
353       0.36      0.16      0.22       110
354       0.53      0.36      0.43       107
355       0.32      0.24      0.28        95
356       0.95      0.75      0.84       105
357       0.85      0.38      0.53       104
358       0.72      0.40      0.52       114
359       0.29      0.10      0.15        98
360       0.25      0.10      0.14       113
361       0.95      0.70      0.81       111
362       0.39      0.22      0.28        92
363       0.08      0.01      0.02        96
364       0.87      0.51      0.64       102
365       0.53      0.24      0.33       108
366       0.22      0.07      0.11        95
367       0.66      0.19      0.29       111
368       0.28      0.13      0.18       116
369       0.61      0.14      0.23       121
370       0.68      0.46      0.55       113
371       0.76      0.58      0.66       118
372       0.93      0.86      0.89       107
373       0.08      0.01      0.02       103
374       0.94      0.44      0.60       110
375       0.33      0.14      0.19       110
376       0.74      0.40      0.52        99
377       0.43      0.25      0.32        96
378       0.88      0.53      0.66        96
379       0.13      0.03      0.05        92
380       0.26      0.12      0.16       103
381       0.64      0.39      0.49       112
382       0.45      0.10      0.17        87
383       0.74      0.35      0.48       100
384       0.46      0.26      0.33        92
385       0.40      0.36      0.37       107
386       0.85      0.64      0.73        83
387       0.32      0.25      0.28       104
388       0.56      0.33      0.42        99
389       0.51      0.27      0.35        89
390       0.81      0.36      0.50        98
391       0.73      0.59      0.66        79
```

392	0.25	0.04	0.07	106
393	0.42	0.18	0.25	108
394	0.43	0.34	0.38	98
395	0.18	0.17	0.18	92
396	0.59	0.21	0.31	108
397	0.29	0.13	0.18	85
398	0.33	0.17	0.22	89
399	0.83	0.63	0.72	93
400	0.42	0.05	0.09	99
401	0.33	0.17	0.22	95
402	0.29	0.10	0.15	100
403	0.72	0.61	0.66	102
404	0.76	0.41	0.53	85
405	0.38	0.15	0.21	96
406	0.56	0.37	0.45	95
407	0.60	0.47	0.53	100
408	0.33	0.14	0.20	98
409	0.46	0.06	0.11	93
410	0.00	0.00	0.00	93
411	0.48	0.25	0.33	109
412	0.41	0.28	0.33	87
413	0.27	0.21	0.24	76
414	0.13	0.05	0.07	84
415	0.27	0.09	0.13	79
416	0.21	0.08	0.12	86
417	0.11	0.03	0.04	109
418	0.30	0.20	0.24	82
419	0.29	0.13	0.18	94
420	0.21	0.04	0.06	84
421	0.75	0.55	0.63	80
422	0.21	0.06	0.09	90
423	0.16	0.11	0.13	95
424	0.44	0.08	0.14	96
425	0.68	0.47	0.56	89
426	0.81	0.25	0.39	103
427	0.33	0.25	0.29	80
428	0.28	0.17	0.21	89
429	0.30	0.09	0.14	90
430	0.85	0.19	0.31	89
431	0.15	0.10	0.12	93

432	0.54	0.38	0.45	102
433	0.55	0.32	0.40	101
434	0.92	0.64	0.75	85
435	0.35	0.08	0.12	79
436	0.41	0.25	0.31	95
437	0.90	0.75	0.82	100
438	0.73	0.39	0.51	97
439	0.16	0.07	0.10	96
440	0.88	0.47	0.61	96
441	0.87	0.39	0.54	87
442	0.28	0.18	0.22	96
443	0.75	0.48	0.58	86
444	0.36	0.09	0.14	89
445	0.43	0.06	0.11	93
446	0.54	0.44	0.48	85
447	0.39	0.24	0.30	101
448	0.00	0.00	0.00	89
449	0.37	0.27	0.31	82
450	0.50	0.26	0.34	78
451	0.34	0.13	0.19	86
452	0.57	0.31	0.40	88
453	0.98	0.69	0.81	91
454	0.20	0.02	0.04	82
455	0.55	0.18	0.27	99
456	0.50	0.28	0.36	96
457	0.67	0.29	0.41	82
458	0.70	0.52	0.60	87
459	0.58	0.29	0.39	90
460	0.26	0.07	0.11	84
461	0.96	0.51	0.67	96
462	0.83	0.53	0.65	75
463	0.85	0.13	0.23	84
464	0.31	0.21	0.25	80
465	0.46	0.19	0.27	100
466	0.40	0.10	0.15	84
467	0.60	0.29	0.39	83
468	0.82	0.60	0.69	83
469	0.98	0.57	0.72	93
470	0.88	0.52	0.65	89
471	0.62	0.50	0.55	88

```
         472       0.39      0.07      0.12        96
         473       0.74      0.54      0.62        89
         474       0.23      0.08      0.12        83
         475       0.39      0.22      0.28        90
         476       0.69      0.34      0.45        80
         477       0.23      0.07      0.11        86
         478       0.35      0.18      0.24        99
         479       0.77      0.50      0.61        74
         480       0.74      0.40      0.52        87
         481       0.00      0.00      0.00        87
         482       0.00      0.00      0.00        89
         483       0.91      0.51      0.65        99
         484       0.52      0.32      0.40        68
         485       0.33      0.26      0.29        69
         486       0.30      0.09      0.14        78
         487       0.20      0.08      0.11        76
         488       0.52      0.25      0.34        89
         489       0.25      0.21      0.23        84
         490       0.50      0.28      0.36        76
         491       0.55      0.46      0.50        85
         492       0.59      0.35      0.44        86
         493       0.61      0.16      0.26        85
         494       0.50      0.03      0.06        87
         495       0.20      0.08      0.11        89
         496       0.67      0.53      0.59        81
         497       0.92      0.49      0.64        91
         498       0.90      0.61      0.73        75
         499       0.60      0.41      0.49        75

   micro avg       0.69      0.39      0.50    180381
   macro avg       0.55      0.31      0.39    180381
weighted avg       0.65      0.39      0.48    180381
 samples avg       0.49      0.38      0.40    180381

Time taken to run this cell : 0:11:37.602788
```

In [18]:
```
#start = datetime.now()
clf_svc = SGDClassifier(loss='hinge', penalty='l2')
tuned_parameter ={'alpha': [10**-4, 10**-2, 10**1, 10**-2, 10**-4]}
#paramGrid = ParameterGrid(tuned_parameter)
model_svc = OneVsRestClassifier(GridSearchCV(clf_svc,  tuned_parameter, scoring = 'f1_micro', cv=3,n_jobs=-1))
model_svc.fit(x_train_multilabel, y_train)
#print("the time taken to run this cell:",datetime.now() - start)
```

Out[18]:
```
OneVsRestClassifier(estimator=GridSearchCV(cv=3, error_score='raise-deprecating',
       estimator=SGDClassifier(alpha=0.0001, average=False, class_weight=None,
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
       n_it...re_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='f1_micro', verbose=0),
          n_jobs=None)
```

In [19]:
```
start = datetime.now()
classifier = OneVsRestClassifier(estimator=SGDClassifier(alpha=0.0001, loss='hinge',penalty='l2'),n_jobs=-1)
classifier.fit(x_train_multilabel, y_train)
predictions = classifier.predict (x_test_multilabel)


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


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

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

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

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

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

```
Accuracy : 0.23968
Hamming loss  0.00287584
Micro-average quality numbers
Precision: 0.6677, Recall: 0.4037, F1-measure: 0.5032
Macro-average quality numbers
Precision: 0.5431, Recall: 0.3200, F1-measure: 0.3863
              precision    recall  f1-score   support

           0       0.59      0.32      0.42      7740
           1       0.76      0.47      0.58      7039
           2       0.83      0.55      0.66      6884
           3       0.66      0.47      0.55      6261
           4       0.92      0.77      0.84      5611
           5       0.80      0.68      0.73      5273
           6       0.79      0.53      0.63      3516
           7       0.85      0.63      0.73      3121
           8       0.69      0.42      0.52      3007
           9       0.65      0.50      0.57      2992
          10       0.83      0.64      0.72      2962
          11       0.48      0.26      0.34      2768
          12       0.38      0.20      0.26      2645
          13       0.53      0.39      0.45      2402
          14       0.49      0.43      0.45      2247
          15       0.50      0.28      0.36      2183
          16       0.68      0.61      0.64      2134
          17       0.75      0.58      0.65      2037
          18       0.62      0.28      0.39      1866
          19       0.54      0.32      0.40      1698
          20       0.29      0.14      0.18      1506
          21       0.70      0.47      0.56      1244
          22       0.61      0.28      0.39      1263
          23       0.80      0.68      0.73      1178
          24       0.65      0.51      0.57      1123
          25       0.62      0.49      0.55      1060
          26       0.25      0.06      0.10      1025
          27       0.81      0.70      0.75       994
          28       0.60      0.37      0.46      1002
          29       0.52      0.28      0.37       899
          30       0.89      0.78      0.83       875
          31       0.54      0.26      0.35       862
```

```
         32       0.51      0.26      0.34       819
         33       0.57      0.38      0.46       791
         34       0.63      0.36      0.46       776
         35       0.78      0.69      0.73       752
         36       0.75      0.66      0.70       803
         37       0.76      0.47      0.58       772
         38       0.42      0.11      0.18       741
         39       0.36      0.18      0.24       655
         40       0.58      0.52      0.55       593
         41       0.35      0.13      0.19       604
         42       0.69      0.35      0.46       587
         43       0.68      0.37      0.48       574
         44       0.61      0.34      0.43       551
         45       0.57      0.38      0.46       615
         46       0.25      0.11      0.15       556
         47       0.23      0.05      0.08       557
         48       0.83      0.76      0.79       551
         49       0.38      0.27      0.31       550
         50       0.70      0.38      0.50       474
         51       0.70      0.52      0.60       542
         52       0.31      0.17      0.22       529
         53       0.47      0.15      0.23       533
         54       0.70      0.50      0.58       547
         55       0.62      0.42      0.50       533
         56       0.18      0.04      0.07       517
         57       0.35      0.37      0.36       521
         58       0.46      0.23      0.31       489
         59       0.41      0.19      0.26       510
         60       0.76      0.61      0.68       488
         61       0.88      0.79      0.83       508
         62       0.75      0.55      0.63       469
         63       0.90      0.68      0.78       486
         64       0.75      0.35      0.47       465
         65       0.72      0.44      0.54       431
         66       0.73      0.48      0.58       455
         67       0.75      0.50      0.60       447
         68       0.63      0.35      0.45       485
         69       0.40      0.15      0.22       428
         70       0.79      0.57      0.66       472
         71       0.82      0.49      0.61       447
```

72	0.70	0.28	0.40	416
73	0.61	0.50	0.55	421
74	0.70	0.54	0.61	411
75	0.17	0.05	0.08	406
76	0.88	0.67	0.76	416
77	0.43	0.49	0.46	385
78	0.15	0.02	0.03	411
79	0.40	0.08	0.13	386
80	0.69	0.36	0.47	389
81	0.72	0.39	0.50	385
82	0.40	0.13	0.20	360
83	0.79	0.48	0.60	339
84	0.77	0.53	0.62	388
85	0.51	0.25	0.34	378
86	0.52	0.51	0.51	356
87	0.90	0.57	0.70	373
88	0.83	0.64	0.72	351
89	0.92	0.60	0.73	364
90	0.91	0.61	0.73	327
91	0.56	0.26	0.36	340
92	0.77	0.55	0.64	339
93	0.54	0.14	0.22	356
94	0.66	0.47	0.55	338
95	0.45	0.29	0.36	330
96	0.83	0.75	0.79	321
97	0.33	0.02	0.04	315
98	0.57	0.19	0.28	319
99	0.88	0.55	0.68	319
100	0.71	0.48	0.57	299
101	0.94	0.74	0.83	329
102	0.27	0.08	0.12	335
103	0.49	0.30	0.37	318
104	0.94	0.75	0.83	313
105	0.93	0.59	0.72	313
106	0.25	0.11	0.15	299
107	0.55	0.37	0.44	291
108	0.38	0.18	0.25	282
109	0.76	0.35	0.48	285
110	0.77	0.60	0.68	298
111	0.63	0.44	0.52	292

112	0.58	0.24	0.34	271
113	0.63	0.38	0.47	298
114	0.80	0.50	0.62	286
115	0.43	0.11	0.18	290
116	0.58	0.14	0.22	279
117	0.30	0.16	0.21	250
118	0.52	0.39	0.45	289
119	0.93	0.75	0.83	243
120	0.96	0.85	0.90	254
121	0.66	0.62	0.64	287
122	0.27	0.10	0.14	283
123	0.53	0.19	0.28	279
124	0.53	0.12	0.19	259
125	0.62	0.30	0.40	275
126	0.36	0.05	0.08	266
127	0.35	0.21	0.27	261
128	0.83	0.63	0.72	279
129	0.93	0.74	0.83	269
130	0.30	0.05	0.09	258
131	0.68	0.24	0.35	272
132	0.69	0.48	0.56	274
133	0.50	0.09	0.16	253
134	0.82	0.56	0.66	265
135	0.33	0.05	0.09	248
136	0.74	0.36	0.49	240
137	0.62	0.39	0.48	248
138	0.14	0.01	0.02	243
139	0.28	0.04	0.07	269
140	0.49	0.27	0.34	233
141	0.45	0.13	0.20	252
142	0.85	0.75	0.80	232
143	0.23	0.04	0.07	244
144	0.76	0.28	0.41	269
145	0.57	0.33	0.42	236
146	0.91	0.81	0.86	239
147	0.20	0.01	0.02	225
148	0.29	0.11	0.16	261
149	0.34	0.27	0.30	244
150	0.38	0.15	0.22	293
151	0.49	0.21	0.30	233

```
152       0.59      0.52      0.55       225
153       0.51      0.14      0.22       262
154       0.71      0.52      0.60       221
155       0.18      0.02      0.04       232
156       0.30      0.19      0.23       218
157       0.73      0.48      0.58       218
158       0.95      0.80      0.87       235
159       0.55      0.43      0.48       244
160       0.92      0.78      0.84       224
161       0.50      0.32      0.39       234
162       0.37      0.22      0.27       213
163       0.79      0.67      0.73       232
164       0.78      0.32      0.45       200
165       0.41      0.26      0.32       216
166       0.67      0.57      0.61       217
167       0.45      0.21      0.28       246
168       0.71      0.43      0.54       211
169       0.50      0.23      0.32       216
170       0.49      0.30      0.37       225
171       0.89      0.75      0.81       224
172       0.55      0.15      0.23       246
173       0.37      0.22      0.27       211
174       0.65      0.38      0.48       208
175       0.73      0.60      0.66       226
176       0.46      0.10      0.16       229
177       0.44      0.15      0.23       205
178       0.44      0.19      0.27       220
179       0.27      0.04      0.07       219
180       0.49      0.09      0.15       238
181       0.71      0.49      0.58       200
182       0.93      0.70      0.80       215
183       0.35      0.10      0.16       228
184       0.50      0.27      0.35       233
185       0.76      0.40      0.52       214
186       0.23      0.07      0.11       203
187       0.89      0.74      0.81       179
188       0.48      0.30      0.37       207
189       0.69      0.36      0.47       194
190       0.40      0.20      0.26       212
191       0.79      0.55      0.65       202
```

192	0.20	0.04	0.07	200
193	0.96	0.74	0.84	215
194	0.50	0.19	0.27	197
195	0.53	0.41	0.46	192
196	0.94	0.61	0.74	196
197	0.50	0.13	0.20	204
198	0.29	0.05	0.09	203
199	0.35	0.15	0.21	217
200	0.79	0.41	0.54	189
201	0.80	0.66	0.72	182
202	0.57	0.46	0.51	191
203	0.57	0.17	0.26	173
204	0.00	0.00	0.00	194
205	0.90	0.74	0.81	192
206	0.10	0.01	0.02	193
207	0.33	0.07	0.12	194
208	0.10	0.03	0.04	188
209	0.71	0.51	0.59	176
210	0.57	0.43	0.49	198
211	0.25	0.07	0.12	187
212	0.67	0.63	0.65	187
213	0.22	0.05	0.08	193
214	0.64	0.36	0.46	173
215	0.58	0.35	0.44	182
216	0.94	0.70	0.80	166
217	0.42	0.17	0.24	182
218	0.50	0.04	0.08	194
219	0.95	0.78	0.86	185
220	0.28	0.10	0.15	163
221	0.96	0.79	0.87	193
222	0.72	0.41	0.52	180
223	0.53	0.32	0.40	194
224	0.81	0.49	0.61	169
225	0.56	0.37	0.44	175
226	0.29	0.04	0.08	158
227	0.80	0.45	0.58	169
228	0.43	0.06	0.11	165
229	0.62	0.54	0.58	196
230	0.76	0.39	0.51	159
231	0.67	0.37	0.48	150

232	0.80	0.58	0.67	149
233	0.24	0.06	0.10	154
234	0.00	0.00	0.00	168
235	0.67	0.67	0.67	141
236	0.37	0.30	0.33	175
237	0.67	0.40	0.50	173
238	0.63	0.16	0.25	153
239	0.82	0.62	0.71	159
240	0.14	0.01	0.01	166
241	0.40	0.04	0.07	156
242	0.83	0.54	0.65	150
243	0.52	0.20	0.28	153
244	0.90	0.76	0.82	144
245	0.71	0.40	0.51	153
246	0.58	0.35	0.44	171
247	0.62	0.42	0.50	128
248	0.26	0.04	0.06	141
249	0.41	0.21	0.27	155
250	0.69	0.60	0.64	156
251	0.52	0.23	0.32	166
252	0.45	0.24	0.31	153
253	0.54	0.52	0.53	150
254	0.48	0.21	0.29	165
255	0.69	0.49	0.58	156
256	0.55	0.40	0.46	158
257	0.83	0.61	0.71	157
258	0.00	0.00	0.00	143
259	0.33	0.06	0.10	144
260	0.23	0.03	0.06	159
261	0.61	0.34	0.43	152
262	0.27	0.02	0.05	161
263	0.64	0.52	0.58	153
264	0.63	0.31	0.42	169
265	0.25	0.15	0.19	134
266	0.65	0.37	0.47	148
267	0.55	0.40	0.47	141
268	0.41	0.09	0.14	138
269	0.29	0.22	0.25	114
270	0.69	0.53	0.60	137
271	0.05	0.01	0.01	153

```
272       0.79      0.61      0.69       150
273       0.60      0.40      0.48       131
274       0.74      0.45      0.56       141
275       0.84      0.73      0.79       147
276       0.00      0.00      0.00       147
277       0.78      0.64      0.70       137
278       0.88      0.65      0.75       147
279       1.00      0.77      0.87       132
280       0.50      0.27      0.35       135
281       0.25      0.01      0.03       144
282       0.14      0.01      0.02       117
283       0.36      0.07      0.12       139
284       0.60      0.44      0.50       121
285       0.36      0.11      0.17       134
286       0.57      0.32      0.41       142
287       0.72      0.33      0.45       119
288       0.58      0.46      0.51       119
289       0.46      0.10      0.17       126
290       0.94      0.63      0.76       142
291       0.32      0.06      0.10       133
292       0.78      0.61      0.69       129
293       0.58      0.30      0.39       128
294       0.20      0.08      0.12       143
295       0.17      0.04      0.06       127
296       0.42      0.09      0.15       120
297       0.42      0.13      0.20       129
298       0.42      0.16      0.23       133
299       0.54      0.40      0.46       126
300       0.00      0.00      0.00       127
301       0.28      0.07      0.11       112
302       0.87      0.36      0.51       149
303       0.80      0.79      0.79       123
304       0.19      0.05      0.07       131
305       0.51      0.27      0.35       120
306       0.48      0.12      0.19       132
307       0.85      0.62      0.72       127
308       0.36      0.12      0.18       106
309       0.47      0.11      0.17       141
310       0.65      0.21      0.32       146
311       0.58      0.35      0.44       126
```

312	0.14	0.03	0.05	127
313	0.59	0.36	0.45	107
314	0.50	0.25	0.33	118
315	0.88	0.75	0.81	150
316	0.50	0.17	0.25	130
317	0.43	0.05	0.09	119
318	0.55	0.28	0.37	131
319	0.27	0.10	0.14	126
320	0.31	0.03	0.06	116
321	0.50	0.55	0.52	117
322	0.43	0.15	0.23	136
323	0.57	0.21	0.30	135
324	0.43	0.37	0.40	109
325	0.39	0.13	0.20	119
326	0.29	0.02	0.03	119
327	0.44	0.07	0.12	110
328	0.52	0.12	0.19	110
329	0.58	0.48	0.52	125
330	0.19	0.07	0.11	110
331	0.54	0.42	0.48	116
332	0.46	0.49	0.47	105
333	0.00	0.00	0.00	108
334	0.23	0.08	0.12	125
335	0.60	0.36	0.45	122
336	0.66	0.27	0.39	106
337	0.72	0.29	0.41	118
338	0.42	0.12	0.18	119
339	0.00	0.00	0.00	116
340	0.33	0.08	0.13	113
341	0.00	0.00	0.00	119
342	0.40	0.02	0.04	94
343	0.00	0.00	0.00	103
344	0.38	0.13	0.19	119
345	0.45	0.38	0.41	95
346	0.34	0.23	0.27	109
347	0.33	0.22	0.26	96
348	0.39	0.19	0.26	113
349	0.45	0.36	0.40	116
350	0.69	0.53	0.60	121
351	0.32	0.07	0.12	107

```
        352       0.64      0.35      0.45       111
        353       0.31      0.05      0.08       110
        354       0.57      0.43      0.49       107
        355       0.32      0.20      0.25        95
        356       0.94      0.86      0.90       105
        357       0.87      0.45      0.59       104
        358       0.71      0.36      0.48       114
        359       0.34      0.13      0.19        98
        360       0.18      0.02      0.03       113
        361       0.94      0.81      0.87       111
        362       0.41      0.13      0.20        92
        363       0.13      0.02      0.04        96
        364       0.83      0.59      0.69       102
        365       0.59      0.31      0.40       108
        366       0.22      0.04      0.07        95
        367       0.55      0.23      0.33       111
        368       0.31      0.12      0.17       116
        369       0.86      0.15      0.25       121
        370       0.84      0.51      0.64       113
        371       0.77      0.61      0.68       118
        372       0.94      0.89      0.91       107
        373       0.34      0.11      0.16       103
        374       0.92      0.50      0.65       110
        375       0.45      0.08      0.14       110
        376       0.79      0.42      0.55        99
        377       0.47      0.09      0.16        96
        378       0.89      0.58      0.70        96
        379       0.43      0.03      0.06        92
        380       0.18      0.07      0.10       103
        381       0.67      0.44      0.53       112
        382       0.39      0.08      0.13        87
        383       0.72      0.47      0.57       100
        384       0.58      0.21      0.30        92
        385       0.08      0.22      0.12       107
        386       0.92      0.67      0.78        83
        387       0.38      0.27      0.31       104
        388       0.67      0.40      0.50        99
        389       0.42      0.27      0.33        89
        390       0.90      0.48      0.63        98
        391       0.73      0.70      0.71        79
```

392	0.31	0.05	0.08	106
393	0.53	0.21	0.30	108
394	0.45	0.26	0.32	98
395	0.08	0.01	0.02	92
396	0.56	0.27	0.36	108
397	0.31	0.14	0.19	85
398	0.50	0.11	0.18	89
399	0.79	0.59	0.67	93
400	0.42	0.05	0.09	99
401	0.31	0.17	0.22	95
402	0.39	0.09	0.15	100
403	0.71	0.64	0.67	102
404	0.66	0.55	0.60	85
405	0.38	0.09	0.15	96
406	0.67	0.39	0.49	95
407	0.64	0.48	0.55	100
408	0.32	0.12	0.18	98
409	0.56	0.05	0.10	93
410	0.00	0.00	0.00	93
411	0.51	0.23	0.32	109
412	0.40	0.29	0.33	87
413	0.44	0.05	0.09	76
414	0.00	0.00	0.00	84
415	0.40	0.27	0.32	79
416	0.32	0.19	0.24	86
417	0.07	0.01	0.02	109
418	0.38	0.24	0.30	82
419	0.23	0.03	0.06	94
420	0.00	0.00	0.00	84
421	0.82	0.59	0.69	80
422	0.42	0.09	0.15	90
423	0.21	0.03	0.06	95
424	0.60	0.12	0.21	96
425	0.71	0.52	0.60	89
426	0.64	0.37	0.47	103
427	0.35	0.17	0.23	80
428	0.23	0.20	0.21	89
429	0.33	0.06	0.10	90
430	0.59	0.19	0.29	89
431	0.22	0.02	0.04	93

432	0.59	0.48	0.53	102
433	0.61	0.41	0.49	101
434	0.84	0.74	0.79	85
435	0.36	0.06	0.11	79
436	0.61	0.12	0.19	95
437	0.96	0.76	0.85	100
438	0.70	0.41	0.52	97
439	0.23	0.03	0.06	96
440	0.86	0.53	0.66	96
441	0.88	0.48	0.62	87
442	0.00	0.00	0.00	96
443	0.76	0.60	0.68	86
444	0.25	0.01	0.02	89
445	0.55	0.06	0.12	93
446	0.53	0.54	0.54	85
447	0.43	0.18	0.25	101
448	0.00	0.00	0.00	89
449	0.19	0.04	0.06	82
450	0.49	0.45	0.47	78
451	0.38	0.06	0.10	86
452	0.55	0.32	0.40	88
453	0.97	0.71	0.82	91
454	0.00	0.00	0.00	82
455	0.41	0.13	0.20	99
456	0.45	0.15	0.22	96
457	0.62	0.39	0.48	82
458	0.78	0.60	0.68	87
459	0.54	0.22	0.31	90
460	0.32	0.10	0.15	84
461	0.96	0.52	0.68	96
462	0.83	0.52	0.64	75
463	0.62	0.15	0.25	84
464	0.35	0.21	0.26	80
465	0.57	0.12	0.20	100
466	0.47	0.10	0.16	84
467	0.67	0.27	0.38	83
468	0.82	0.71	0.76	83
469	0.95	0.62	0.75	93
470	0.86	0.61	0.71	89
471	0.63	0.36	0.46	88

```
         472       0.46      0.06      0.11        96
         473       0.75      0.61      0.67        89
         474       0.27      0.04      0.06        83
         475       0.52      0.18      0.26        90
         476       0.53      0.56      0.55        80
         477       0.38      0.09      0.15        86
         478       0.38      0.14      0.21        99
         479       0.76      0.65      0.70        74
         480       0.78      0.48      0.60        87
         481       0.00      0.00      0.00        87
         482       1.00      0.01      0.02        89
         483       0.92      0.59      0.72        99
         484       0.54      0.32      0.40        68
         485       0.43      0.26      0.32        69
         486       0.29      0.06      0.11        78
         487       0.26      0.16      0.20        76
         488       0.47      0.25      0.32        89
         489       0.26      0.11      0.15        84
         490       0.53      0.42      0.47        76
         491       0.55      0.47      0.51        85
         492       0.60      0.41      0.49        86
         493       0.79      0.32      0.45        85
         494       0.29      0.09      0.14        87
         495       0.31      0.06      0.10        89
         496       0.67      0.41      0.51        81
         497       0.88      0.57      0.69        91
         498       0.89      0.72      0.79        75
         499       0.69      0.12      0.20        75

   micro avg       0.67      0.40      0.50    180381
   macro avg       0.54      0.32      0.39    180381
weighted avg       0.62      0.40      0.48    180381
 samples avg       0.49      0.39      0.41    180381

Time taken to run this cell : 0:12:00.982453
```

CONCLUSION:

Summary:

1. Load the data from train.csv to sqlite datadase.
2. Reomve the duplicate rows from the database and data is stored in train_no_dup.db.
3. check the most frequently, here their are 90% of tags are 500, So take only 500 tags.
4. sampled 0.5M data due to constrained resources.
5. Since the Stackoverflow is a website their can be html tags, So the next process is data cleaning.
6. In this process allt he html tags are removed and SnowStemming is applied
7. Title is more important because the title is itself tells what is the question about. So, giving title more weight will be benificial that is why title itself is added three times.
8. After all the above process load the data into TitleMoreWeighted.db.
9. preprocessed_data which is loaded into pickle file and uploaded into google drive.
10. checked if thier is any change in the tags. in my case their is no drastic change. 10.Finally, the data is splited into 70:30 ratio and BagOfWords Featurization is applied. 11.Then OneVsRestClassifier is used with SGDClassifier() with loss="log"and"hinge" which is basically logisticRegression and Linear_SVM and the results are compared.

In [2]:
```
!pip install PrettyTable
```

```
Collecting PrettyTable
  Using cached https://files.pythonhosted.org/packages/ef/30/4b0746848746ed5941f052479e7c23d2b56d174b82f4fd34a25
e389831f5/prettytable-0.7.2.tar.bz2
Building wheels for collected packages: PrettyTable
  Running setup.py bdist_wheel for PrettyTable: started
  Running setup.py bdist_wheel for PrettyTable: finished with status 'done'
  Stored in directory: C:\Users\USER\AppData\Local\pip\Cache\wheels\80\34\1c\3967380d9676d162cb59513bd9dc862d058
4e045a162095606
Successfully built PrettyTable
Installing collected packages: PrettyTable
Successfully installed PrettyTable-0.7.2

You are using pip version 18.0, however version 19.0.3 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.
```

In [3]:
```
from prettytable import PrettyTable
```

In [10]:
```
x= PrettyTable()

x.field_names = ["Algorithm","Accuracy","Hamming loss", "Micro_average-Precision,Recall,F1_measure","Macro_avera
ge-Precision,Recall,F1_measure","alpha"]

x.add_row(["Linear_SVM","0.23968","0.00287584","0.67, 0.40, 0.50","0.54, 0.32, 0.39","0.0001"])
x.add_row(["LogisticRegression"," 0.23656"," 0.00283316"," 0.69, 0.39, 0.50","0.55, 0.31, 0.39","0.0001"])

print(x)
```

```
+--------------------+----------+--------------+-------------------------------------------+--------------------
----------------------+--------+
|     Algorithm      | Accuracy | Hamming loss | Micro_average-Precision,Recall,F1_measure | Macro_average-Preci
sion,Recall,F1_measure | alpha  |
+--------------------+----------+--------------+-------------------------------------------+--------------------
----------------------+--------+
|     Linear_SVM     | 0.23968  |  0.00287584  |              0.67, 0.40, 0.50             |              0.54,
0.32, 0.39            | 0.0001 |
| LogisticRegression |  0.23656 |  0.00283316  |              0.69, 0.39, 0.50             |              0.55,
0.31, 0.39            | 0.0001 |
+--------------------+----------+--------------+-------------------------------------------+--------------------
----------------------+--------+
```

As we can see in the above table both the algorithms are performing very similar but performing LogisticRegression is computationally less expensive than linear_SVM.