SO_Tag_Predictor_new

November 19, 2018

In [2]:
```
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.model_selection import GridSearchCV
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
```

1 Stack Overflow: Tag Prediction

1. Business Problem

1.1 Description

Description

Stack Overflow is the largest, most trusted online community for developers to learn, share their programming knowledge, and build their careers. Stack Overflow is something which every programmer use one way or another. Each month, over 50 million developers come to Stack Overflow to learn, share their knowledge, and build their careers. It features questions and answers on a wide range of topics in computer programming. The website serves as a platform for users to ask and answer questions, and, through membership and active participation, to vote questions and answers up or down and edit questions and answers in a fashion similar to a wiki or Digg. As of April 2014 Stack Overflow has over 4,000,000 registered users, and it exceeded 10,000,000 questions in late August 2015. Based on the type of tags assigned to questions, the top eight most discussed topics on the site are: Java, JavaScript, C#, PHP, Android, jQuery, Python and HTML.

Problem Statemtent

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

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

1.2 Source / useful links

Data Source : https://www.kaggle.com/c/facebook-recruiting-iii-keyword-extraction/data Youtube : https://youtu.be/nNDqbUhtIRg Research paper : 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

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 All of the data is in 2 files: Train and Test.

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:

2.1.2 Example Data point

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 datapoint that are not mutually exclusive, such as topics that are relevant for a document. A question on Stackoverflow might be about any of C, Pointers, FileIO and/or memory-management at the same time or none of these. **Credit**: http://scikit-learn.org/stable/modules/multiclass.html

2.2.2 Performance metric

Micro-Averaged F1-Score (Mean F Score) : The F1 score can be interpreted as a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0. The relative contribution of precision and recall to the F1 score are equal. The formula for the F1 score is:

F1 = 2 * (precision * recall) / (precision + recall)

In the multi-class and multi-label case, this is the weighted average of the F1 score of each class.

'Micro f1 score': Calculate metrics globally by counting the total true positives, false negatives and false positives. This is a better metric when we have class imbalance.

'Macro f1 score': Calculate metrics for each label, and find their unweighted mean. This does not take label imbalance into account.

https://www.kaggle.com/wiki/MeanFScore http://scikit-learn.org/stable/modules/generated/sklearn.me
Hamming loss : The Hamming loss is the fraction of labels that are incorrectly predicted. 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 [3]: #Creating db file from csv
        #Learn SQL: https://www.w3schools.com/sql/default.asp
        if not os.path.isfile('train.db'):
            start = datetime.now()
            disk_engine = create_engine('sqlite:///train.db')
            start = dt.datetime.now()
            chunksize = 180000
            j = 0
            index_start = 1
            for df in pd.read_csv('Train.csv', names=['Id', 'Title', 'Body', 'Tags'], chunksize=
                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 [3]: 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 genarat
```

```
Number of rows in the database :
 6034196
Time taken to count the number of rows : 0:00:48.707495
```

3.1.3 Checking for duplicates

In [4]:
```
#Learn SQl: https://www.w3schools.com/sql/default.asp
if os.path.isfile('train.db'):
    start = datetime.now()
    con = sqlite3.connect('train.db')
    df_no_dup = pd.read_sql_query('SELECT Title, Body, Tags, COUNT(*) as cnt_dup FROM da
    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 tra
```

```
Time taken to run this cell : 0:02:31.241691
```

In [5]:
```
df_no_dup.head()
# we can observe that there are duplicates
```

Out[5]:
```
                                               Title  \
0      Implementing Boundary Value Analysis of S...
1          Dynamic Datagrid Binding in Silverlight?
2          Dynamic Datagrid Binding in Silverlight?
3     java.lang.NoClassDefFoundError: javax/serv...
4     java.sql.SQLException:[Microsoft][ODBC Dri...

                                                Body  \
0  <pre><code>#include&lt;iostream&gt;\n#include&...
1  <p>I should do binding for datagrid dynamicall...
2  <p>I should do binding for datagrid dynamicall...
3  <p>I followed the guide in <a href="http://sta...
4  <p>I use the following code</p>\n\n<pre><code>...

                                  Tags  cnt_dup
0                                c++ c        1
1          c# silverlight data-binding        1
2  c# silverlight data-binding columns        1
3                             jsp jstl        1
4                            java jdbc        2
```

In [6]:
```
print("number of duplicate questions :", num_rows['count(*)'].values[0]- df_no_dup.shape
```

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

In [7]:
```
# number of times each question appeared in our database
df_no_dup.cnt_dup.value_counts()
```

Out[7]:
```
1    2656284
2    1272336
```

```
3     277575
4         90
5         25
6          5
Name: cnt_dup, dtype: int64
```

In [8]:
```
#checking for null values
nan_rows = df_no_dup[df_no_dup.isnull().any(1)]
nan_rows
```

Out[8]:
```
                                                    Title  \
777547                                Do we really need NULL?
962680   Find all values that are not null and not in a...
1126558                                   Handle NullObjects
1256102                               How do Germans call null
2430668  Page cannot be null. Please ensure that this o...
3329908       What is the difference between NULL and "0"?
3551595         a bit of difference between null and space

                                                     Body  Tags  cnt_dup
777547   <blockquote>\n  <p><strong>Possible Duplicate:...  None        1
962680   <p>I am running into a problem which results i...  None        1
1126558  <p>I have done quite a bit of research on best...  None        1
1256102  <p>In german null means 0, so how do they call...  None        1
2430668  <p>I get this error when i remove dynamically ...  None        1
3329908  <p>What is the difference from NULL and "0"?</...  None        1
3551595  <p>I was just reading this quote</p>\n\n<block...  None        2
```

In [9]:
```
# droping the rows contain null value
df_no_dup.dropna(inplace=True)
```

In [10]:
```
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 : 0:00:03.968964
```

Out[10]:
```
                                               Title  \
0       Implementing Boundary Value Analysis of S...
1               Dynamic Datagrid Binding in Silverlight?
2               Dynamic Datagrid Binding in Silverlight?
3      java.lang.NoClassDefFoundError: javax/serv...
4      java.sql.SQLException:[Microsoft][ODBC Dri...

                                                Body  \
0  <pre><code>#include&lt;iostream&gt;\n#include&...
```

```
1  <p>I should do binding for datagrid dynamicall...
2  <p>I should do binding for datagrid dynamicall...
3  <p>I followed the guide in <a href="http://sta...
4  <p>I use the following code</p>\n\n<pre><code>...

                                 Tags  cnt_dup  tag_count
0                              c++ c        1          2
1           c# silverlight data-binding        1          3
2  c# silverlight data-binding columns        1          4
3                            jsp jstl        1          2
4                           java jdbc        2          2
```

In [11]:
```
# distribution of number of tags per question
df_no_dup.tag_count.value_counts()
```

Out[11]:
```
3    1206157
2    1111706
4     814996
1     568291
5     505158
Name: tag_count, dtype: int64
```

In [12]:
```
#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 [13]:
```
#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 genar
```

```
Time taken to run this cell : 0:00:49.989970
```

3.2 Analysis of Tags

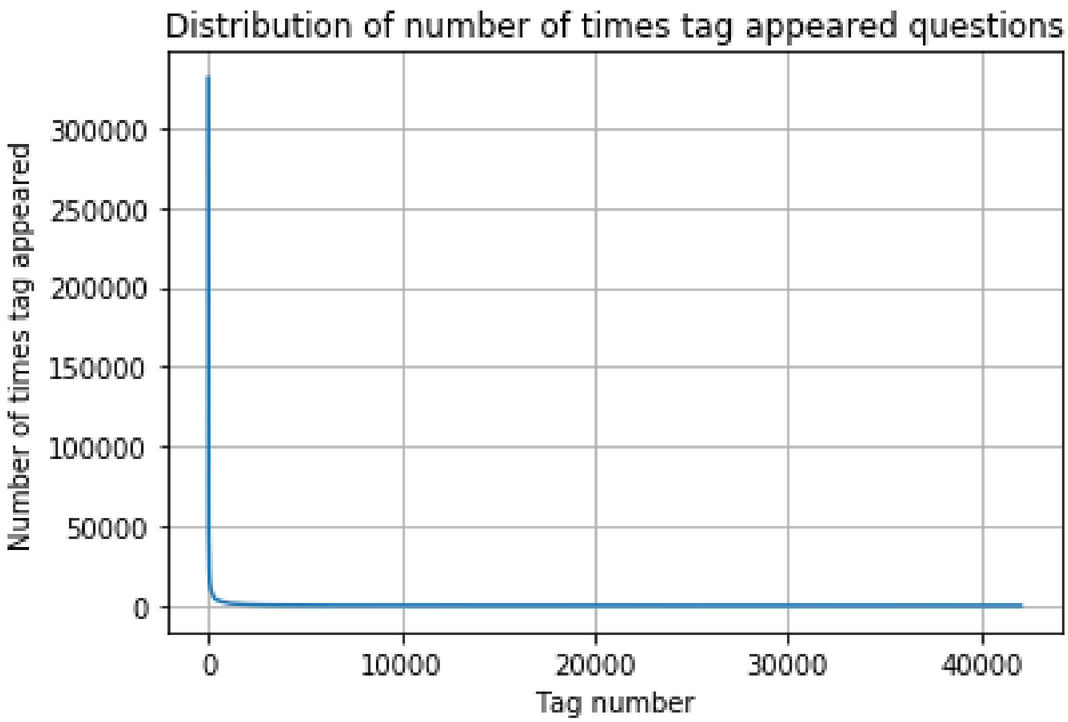
3.2.1 Total number of unique tags

```
In [14]: # 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 [15]: print("Number of data points :", tag_dtm.shape[0])
         print("Number of unique tags :", tag_dtm.shape[1])

Number of data points : 4206307
Number of unique tags : 42048

In [16]: #'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-
```

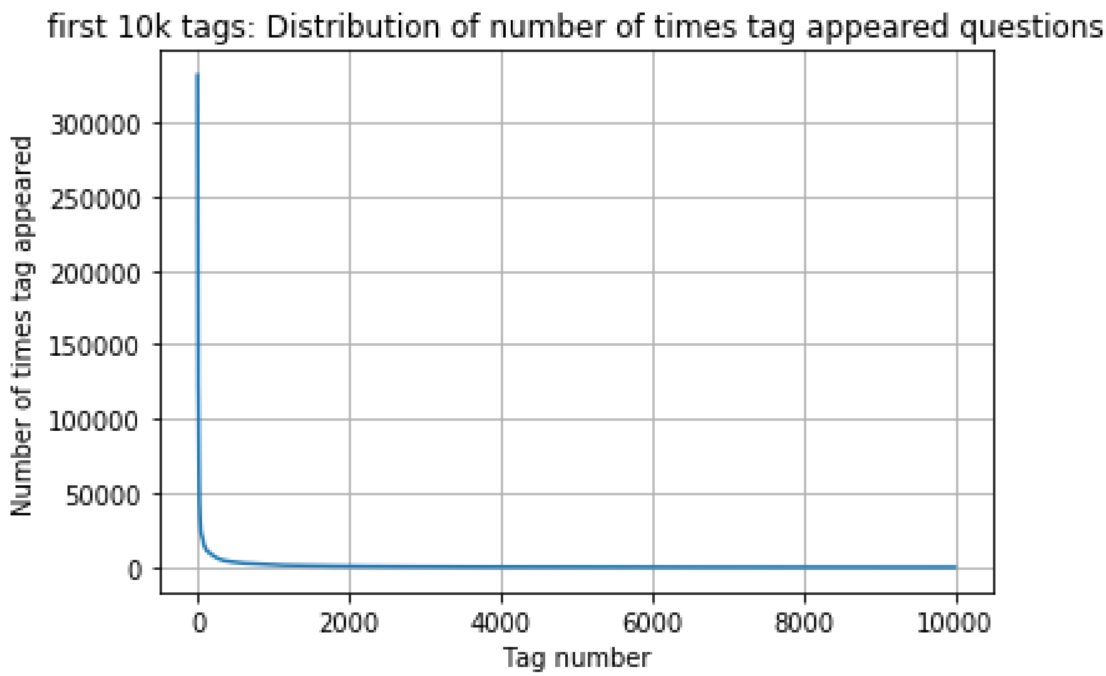
3.2.3 Number of times a tag appeared

```
In [17]: # 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 [18]: #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[18]:                  Tags  Counts
         0           jconnect      16
         1  dotnetnuke-module      90
         2         macromedia      22
         3            ibm-jsf       8
         4              rtmps       9

In [19]: tag_df_sorted = tag_df.sort_values(['Counts'], ascending=False)
         tag_counts = tag_df_sorted['Counts'].values
```

In [20]:
```
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()
```



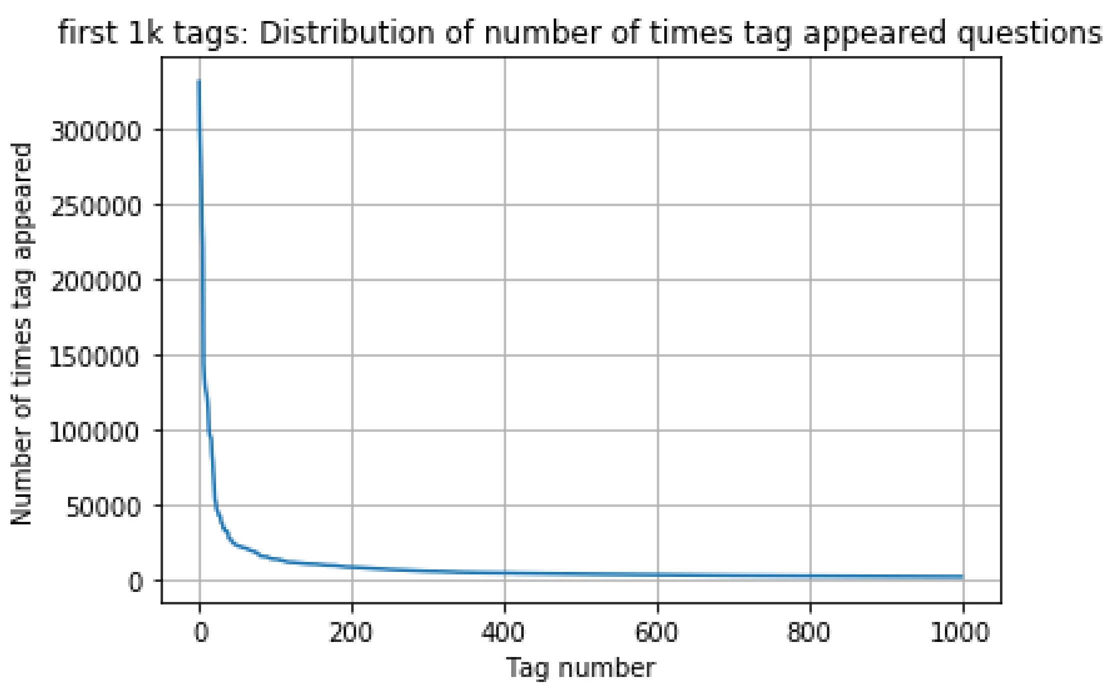
In [21]:
```
plt.plot(tag_counts[0:10000])
plt.title('first 10k tags: Distribution of number of times tag appeared questions')
plt.grid()
plt.xlabel("Tag number")
plt.ylabel("Number of times tag appeared")
plt.show()
print(len(tag_counts[0:10000:25]), tag_counts[0:10000:25])
```




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

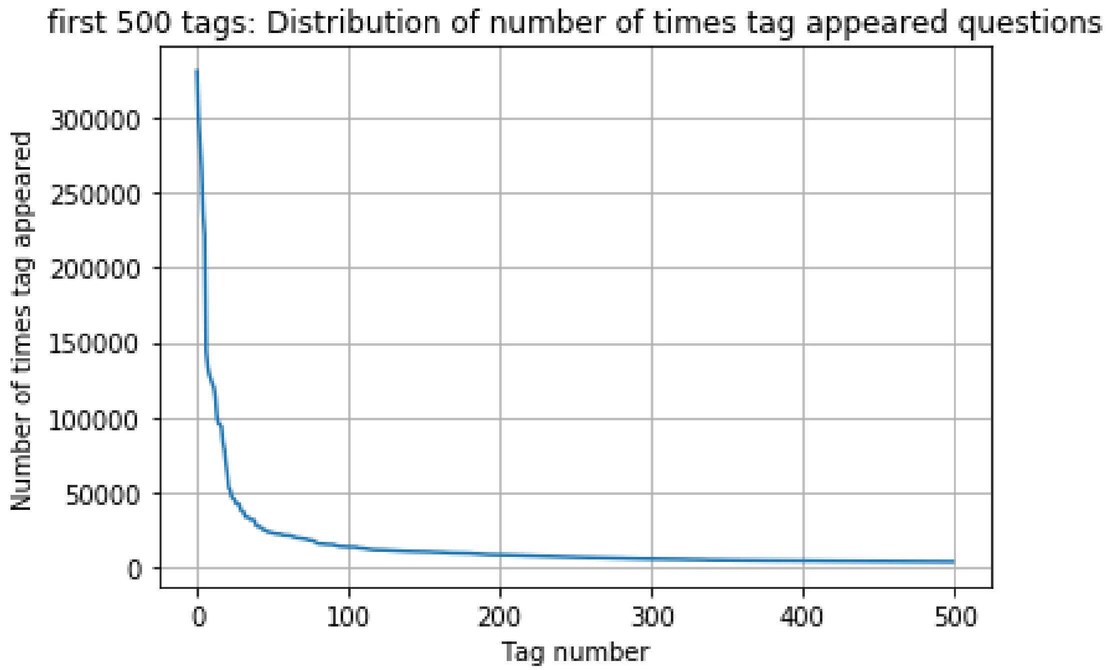
```
 154    153    152    151    150    149    149    148    147    146
 145    144    143    142    142    141    140    139    138    137
 137    136    135    134    134    133    132    131    130    130
 129    128    128    127    126    126    125    124    124    123
 123    122    122    121    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 [22]:
```
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   2986   2983   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]
```

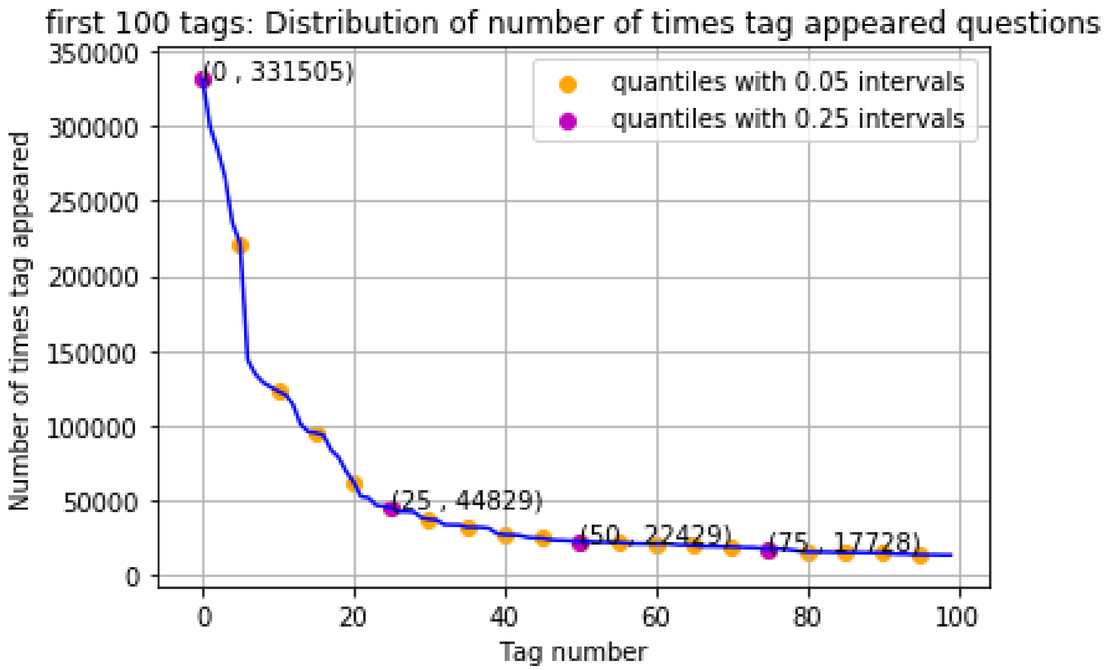
In [23]:
```
plt.plot(tag_counts[0:500])
plt.title('first 500 tags: Distribution of number of times tag appeared questions')
plt.grid()
plt.xlabel("Tag number")
plt.ylabel("Number of times tag appeared")
plt.show()
print(len(tag_counts[0:500:5]), tag_counts[0:500:5])
```




```
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 [24]:
```
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
# quantiles with 0.25 difference
plt.scatter(x=list(range(0,100,25)), y=tag_counts[0:100:25], c='m', label = "quantiles

for x,y in zip(list(range(0,100,25)), tag_counts[0:100:25]):
    plt.annotate(text="({} , {})".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 [25]:
```
# 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

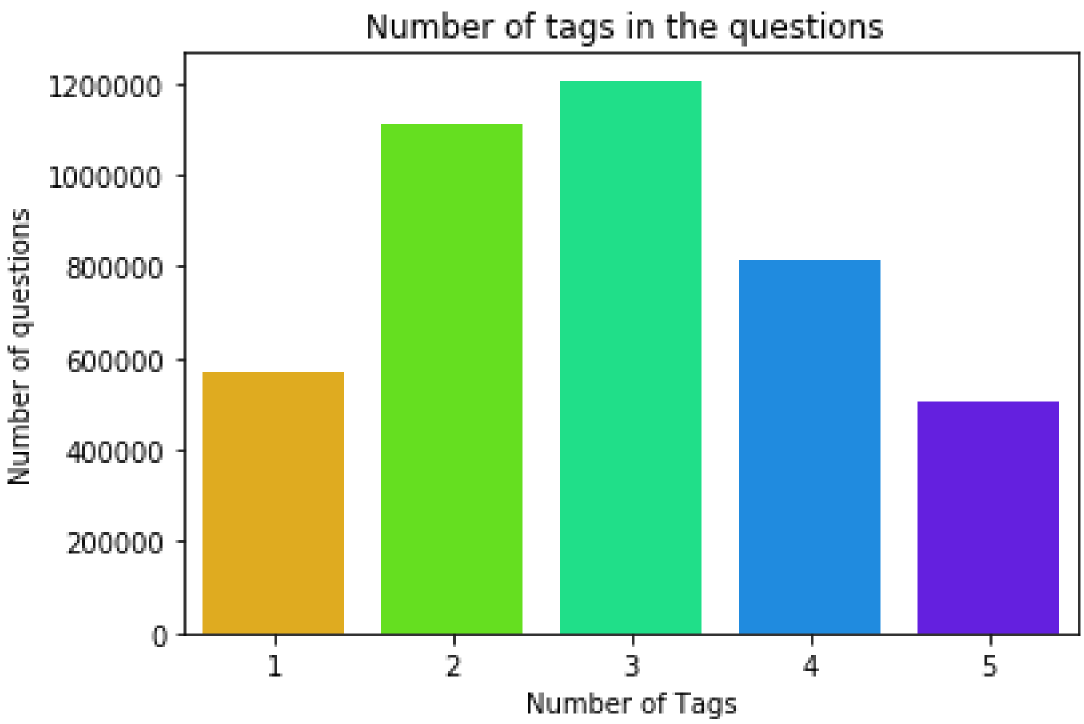
In [26]:
```
#Storing the count of tag in each question in list 'tag_count'
tag_quest_count = tag_dtm.sum(axis=1).tolist()
#Converting each value in the 'tag_quest_count' to integer.
tag_quest_count=[int(j) for i in tag_quest_count for j in i]
print ('We have total {} datapoints.'.format(len(tag_quest_count)))

print(tag_quest_count[:5])
```

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

In [27]:
```
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_ques
```

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

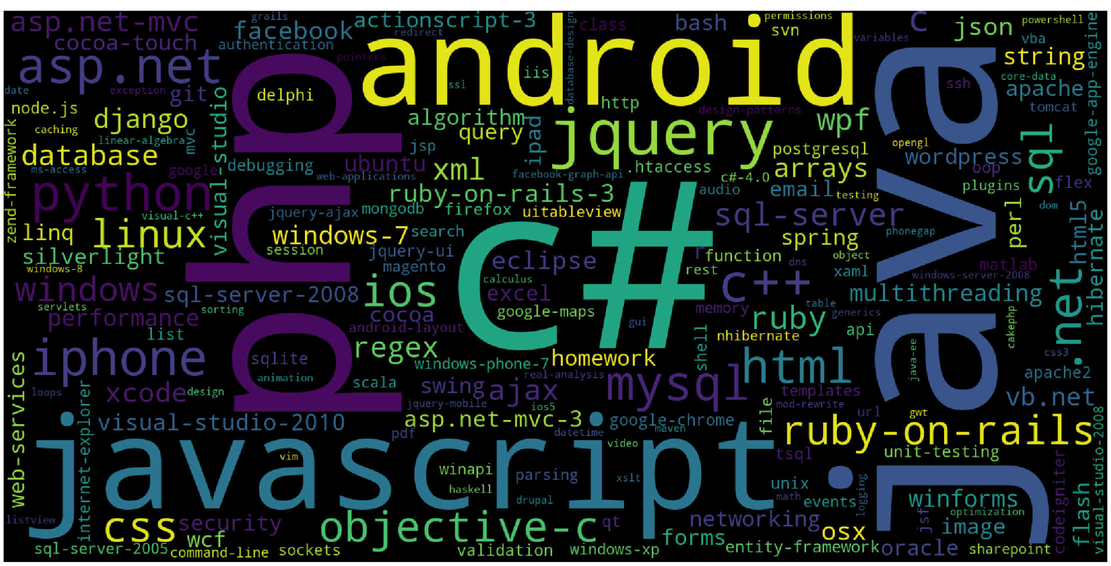
In [28]:
```
sns.countplot(tag_quest_count, palette='gist_rainbow')
plt.title("Number of tags in the questions ")
plt.xlabel("Number of Tags")
plt.ylabel("Number of questions")
plt.show()
```



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

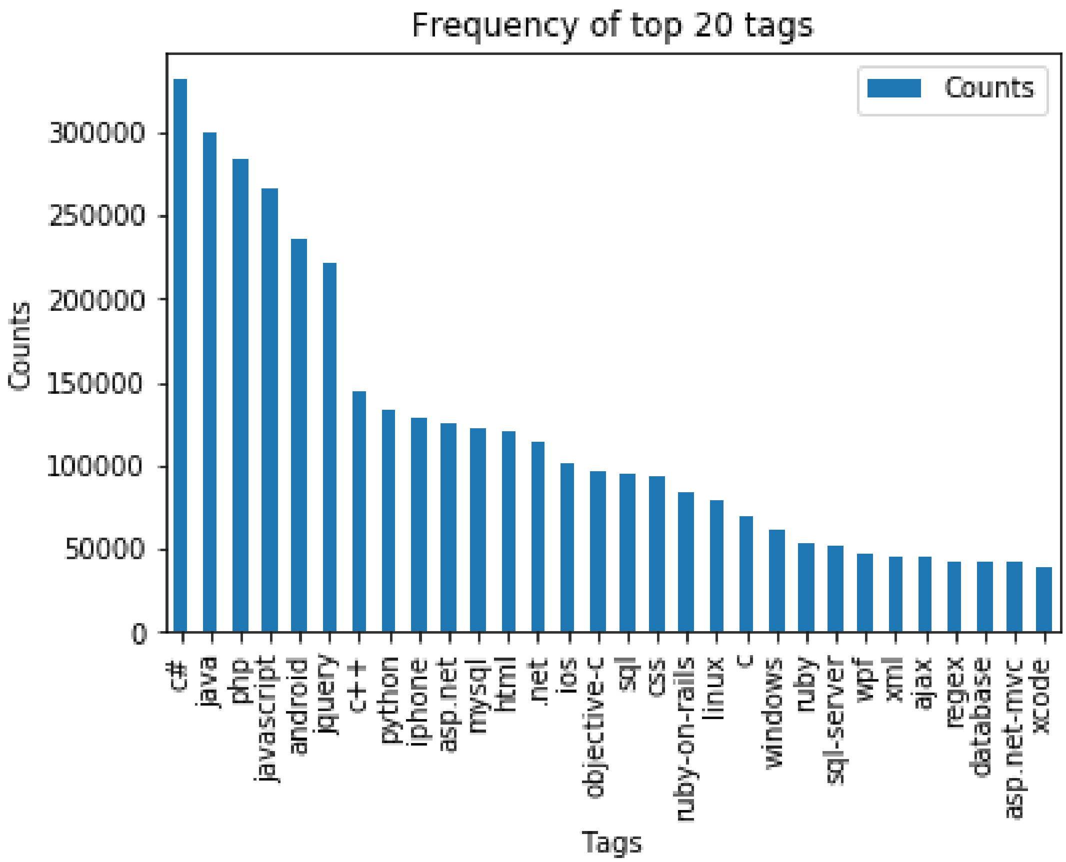
In [29]:
```python
# Ploting word cloud
start = datetime.now()

# 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 : 0:00:05.264542

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 [30]: 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

```
In [3]: def striphtml(data):
            cleanr = re.compile('<.*?>')
            cleantext = re.sub(cleanr, ' ', str(data))
            return cleantext
```

```
stop_words = set(stopwords.words('english'))
stemmer = SnowballStemmer("english")
```

In [4]:
```
#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 N
create_database_table("Processed.db", sql_create_table)
```

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

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

```
In [5]: 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 [6]: sql_create_table = """CREATE TABLE IF NOT EXISTS QuestionsProcessed (question text NOT N
        create_database_table("Titlemoreweight.db", sql_create_table)
```

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

```
In [7]: # http://www.sqlitetutorial.net/sqlite-delete/
        # https://stackoverflow.com/questions/2279706/select-random-row-from-a-sqlite-table

        read_db = 'train_no_dup.db'
        write_db = 'Titlemoreweight.db'
        train_datasize = 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 RANDOM() LI

        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
```

3.3.1 Preprocessing of questions

```
<li> Sample 0.5M data points and taking just 500 most important tags </li>
<li> Separate Code from Body </li>
<li> Remove Spcial characters from Question title and description (not in code)</li>
<li> <b> Give more weightage to title : Add title three times to the question </b> </li>

<li> Remove stop words (Except 'C') </li>
<li> Remove HTML Tags </li>
<li> Convert all the characters into small letters </li>
<li> Use SnowballStemmer to stem the words </li>
```

```
In [8]: #http://www.bernzilla.com/2008/05/13/selecting-a-random-row-from-an-sqlite-table/
        start = datetime.now()
        preprocessed_data_list=[]
        reader.fetchone()
        questions_with_code=0
        len_pre=0
        len_post=0
        questions_proccesed = 0
        for row in reader:

            is_code = 0

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

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

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

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

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

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

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

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

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

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

            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_po
            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

        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 : 0:17:07.249072
```

```
In [9]: # 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 [10]: if os.path.isfile(write_db):
             conn_r = create_connection(write_db)
             if conn_r is not None:
                 reader =conn_r.cursor()
                 reader.execute("SELECT question From QuestionsProcessed LIMIT 10")
                 print("Questions after preprocessed")
                 print('='*100)
                 reader.fetchone()
                 for row in reader:
                     print(row)
                     print('-'*100)
         conn_r.commit()
         conn_r.close()
```

```
Questions after preprocessed
====================================================================================================
('dynam datagrid bind silverlight dynam datagrid bind silverlight dynam datagrid bind silverligh
----------------------------------------------------------------------------------------------------
('java.lang.noclassdeffounderror javax servlet jsp tagext taglibraryvalid java.lang.noclassdeffo
----------------------------------------------------------------------------------------------------
('java.sql.sqlexcept microsoft odbc driver manag invalid descriptor index java.sql.sqlexcept mic
----------------------------------------------------------------------------------------------------
('better way updat feed fb php sdk better way updat feed fb php sdk better way updat feed fb php
----------------------------------------------------------------------------------------------------
('btnadd click event open two window record ad btnadd click event open two window record ad btna
----------------------------------------------------------------------------------------------------
('sql inject issu prevent correct form submiss php sql inject issu prevent correct form submiss
----------------------------------------------------------------------------------------------------
('countabl subaddit lebesgu measur countabl subaddit lebesgu measur countabl subaddit lebesgu me
----------------------------------------------------------------------------------------------------
('hql equival sql queri hql equival sql queri hql equival sql queri hql queri replac name class
----------------------------------------------------------------------------------------------------
('undefin symbol architectur i386 objc class skpsmtpmessag referenc error undefin symbol archite
----------------------------------------------------------------------------------------------------
```

__ Saving Preprocessed data to a Database __

```
In [11]: #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 QuestionsPr
         conn_r.commit()
         conn_r.close()
```

```
In [12]: preprocessed_data.head()
```

```
Out[12]:                                             question  \
         0  dynam datagrid bind silverlight dynam datagrid...
         1  dynam datagrid bind silverlight dynam datagrid...
         2  java.lang.noclassdeffounderror javax servlet j...
         3  java.sql.sqlexcept microsoft odbc driver manag...
         4  better way updat feed fb php sdk better way up...

                                         tags
         0          c# silverlight data-binding
         1  c# silverlight data-binding columns
         2                             jsp jstl
         3                            java jdbc
         4        facebook api facebook-php-sdk
```

```
In [13]: 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
```

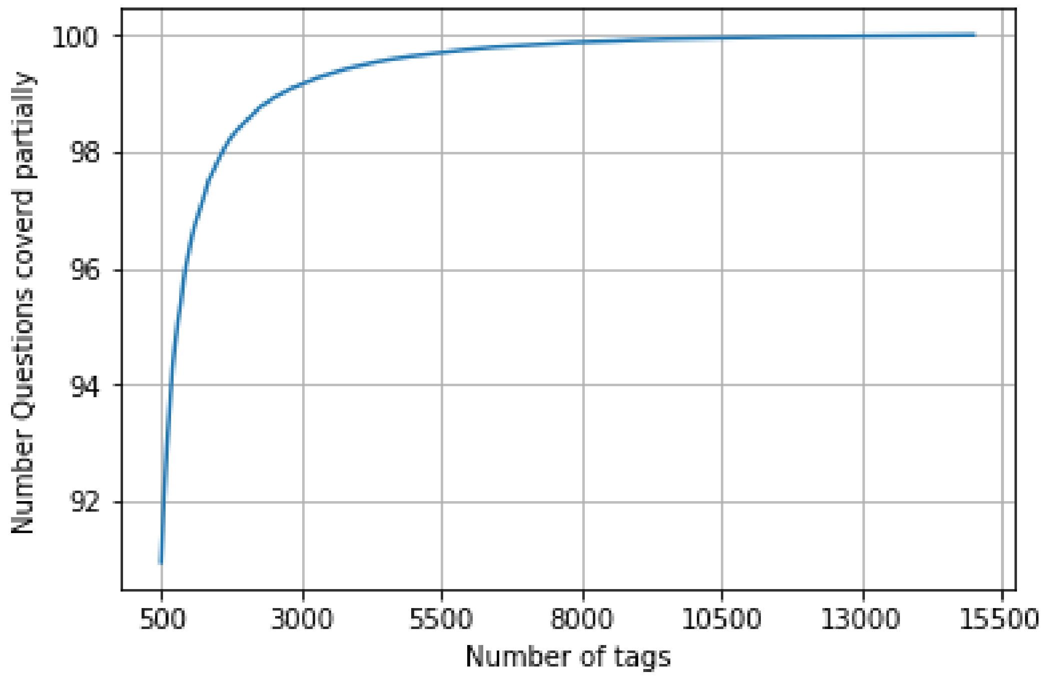
__ Converting string Tags to multilable output variables __

```
In [14]: vectorizer = CountVectorizer(tokenizer = lambda x: x.split(), binary='true')
         multilabel_y = vectorizer.fit_transform(preprocessed_data['tags'])
```

__ Selecting 500 Tags __

```
In [15]: 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)
```

```
In [16]: 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 co
         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  99.157 % of questions
with  500 tags we are covering  90.956 % of questions
```

```
In [17]: # 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
```

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

```
In [18]: 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 [19]: 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. Modeling.

4.1 Modeling using Tfidf vectorizer
4.1.1 Featurizing data with TfIdf vectorizer

```
In [20]: start = datetime.now()
         vectorizer = TfidfVectorizer(min_df=0.00009, max_features=200000, smooth_idf=True, norm
                                      tokenizer = lambda x: x.split(), sublinear_tf=False, ngram
         x_train_multilabel = vectorizer.fit_transform(x_train['question'])
         x_test_multilabel = vectorizer.transform(x_test['question'])
         print("Time taken to run this cell :", datetime.now() - start)
```

```
Time taken to run this cell : 0:05:06.846641
```

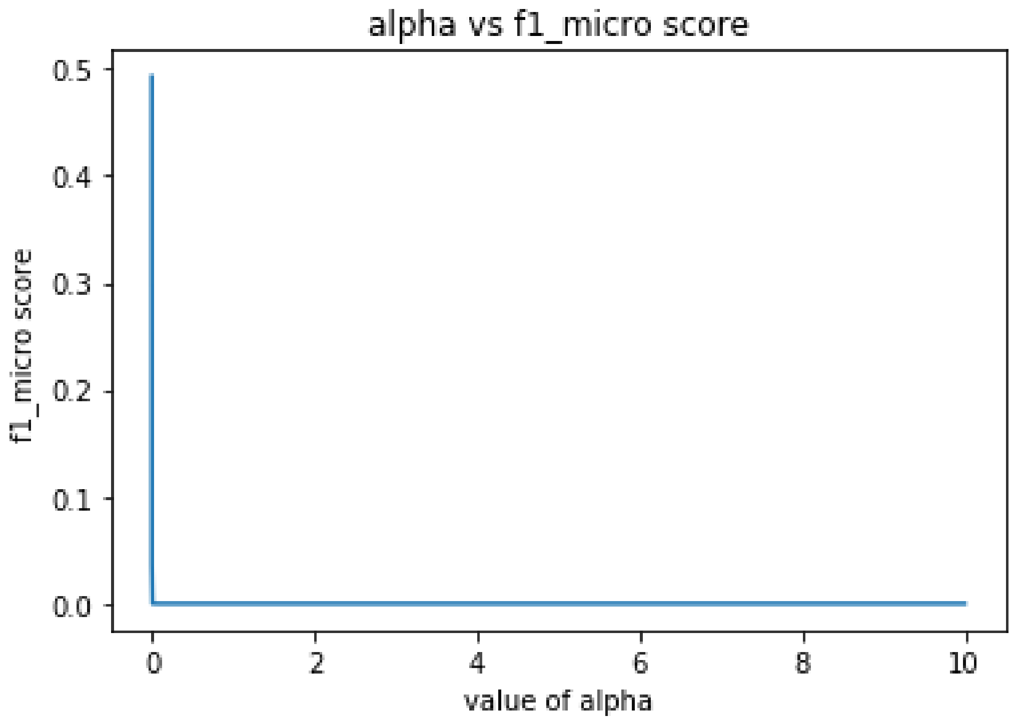
```
In [21]: print("Dimensions of train data X:",x_train_multilabel.shape, "Y :",y_train.shape)
         print("Dimensions of test data X:",x_test_multilabel.shape,"Y:",y_test.shape)
```

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

4.1.2 Logistic Regression with OneVsRestClassifier using Tfidf vectorizer
4.1.2.1 Hyperparameter tuning

```
In [25]: 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=0, scoring='
         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)
         print('------------------------------------------------------------')
         # plotting C vs f1_micro_score
         x_1=[]
         y_1=[]
         for x in gsv.grid_scores_:
             x_1.append(x[0]['estimator__alpha'])
             y_1.append(x[1])
         plt.plot(x_1,y_1)
         plt.xlabel('value of alpha')
         plt.ylabel('f1_micro score')
         plt.title('alpha vs f1_micro score')
         plt.show()
```

```
value of alpha after hyperparameter tuning :  1e-05
------------------------------------------------------------
```



4.1.2.2 Applying model using best hyperparameter

```
In [26]: start = datetime.now()
         #best_alpha = gsv.best_estimator_.get_params()['estimator__alpha']
         classifier = OneVsRestClassifier(SGDClassifier(loss='log', alpha=best_alpha, penalty='l
         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,

         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,

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

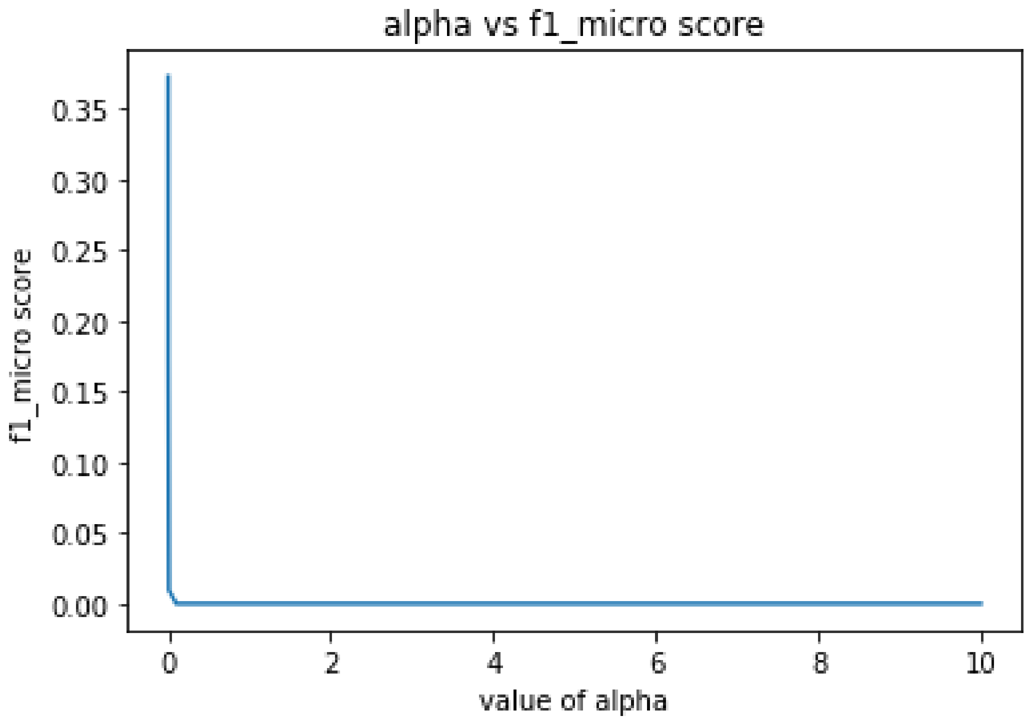
```
Accuracy : 0.23644
Hamming loss  0.00278178
Micro-average quality numbers
Precision: 0.7211, Recall: 0.3258, F1-measure: 0.4488
Macro-average quality numbers
Precision: 0.5478, Recall: 0.2573, F1-measure: 0.3340
Time taken to run this cell : 0:05:02.703501
```

4.1.3 Linear SVM with OneVsRestClassifier using Tfidf vectorizer

4.1.3.1 Hyperparameter tuning

```
In [28]: 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=0, scoring='
         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)
         print('--------------------------------------------------------------')
         # plotting C vs f1_micro_score
         x_1=[]
         y_1=[]
         for x in gsv.grid_scores_:
             x_1.append(x[0]['estimator__alpha'])
             y_1.append(x[1])
         plt.plot(x_1,y_1)
         plt.xlabel('value of alpha')
         plt.ylabel('f1_micro score')
         plt.title('alpha vs f1_micro score')
         plt.show()
```

```
value of alpha after hyperparameter tuning :  0.0001
--------------------------------------------------------------
```



4.1.3.2 Applying model using best hyperparameter

```
In [29]: start = datetime.now()
         #best_alpha = gsv.best_estimator_.get_params()['estimator__alpha']
         classifier = OneVsRestClassifier(SGDClassifier(loss='hinge', alpha=best_alpha, 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, recall,

         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,

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

```
Accuracy : 0.2105
Hamming loss  0.0029036
Micro-average quality numbers
Precision: 0.8169, Recall: 0.2123, F1-measure: 0.3370
Macro-average quality numbers
Precision: 0.2440, Recall: 0.1298, F1-measure: 0.1607
Time taken to run this cell : 0:05:01.558172
```

4.2 Modeling using Count vectorizer
4.2.1 Featurizing data with Count vectorizer

```
In [21]: start = datetime.now()
         vectorizer = CountVectorizer(min_df=0.00009, max_features=200000,  \
                                      tokenizer = lambda x: x.split(), ngram_range=(1,4))
         x_train_multilabel = vectorizer.fit_transform(x_train['question'])
         x_test_multilabel = vectorizer.transform(x_test['question'])
         print("Time taken to run this cell :", datetime.now() - start)
```

```
Time taken to run this cell : 0:11:02.103345
```

```
In [22]: 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)
```

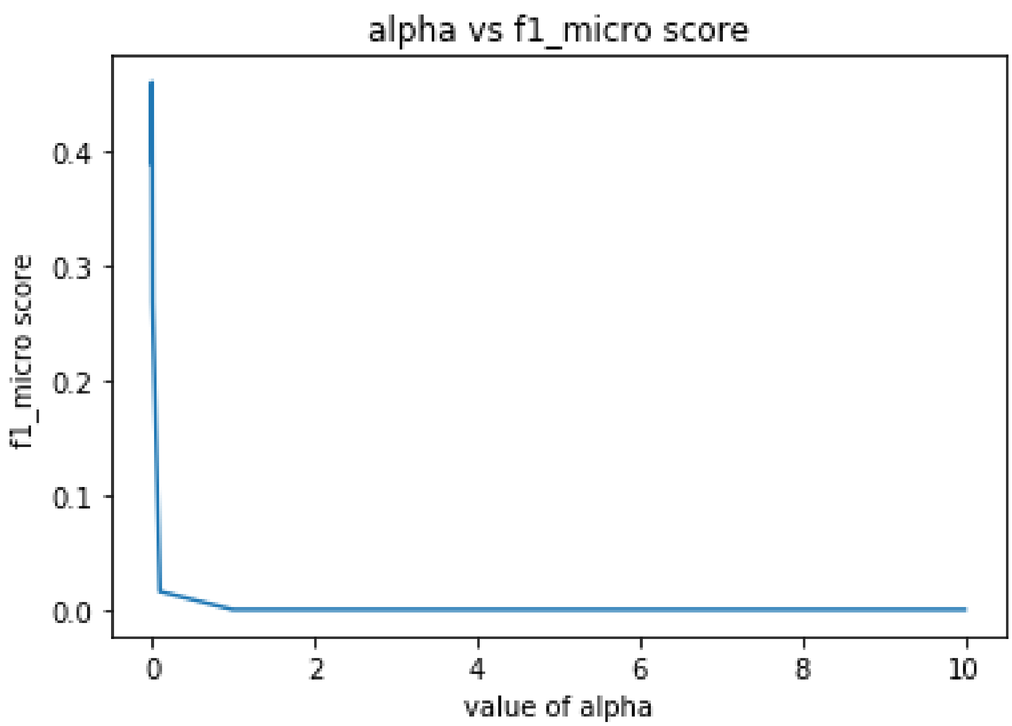
```
Dimensions of train data X: (400000, 95585) Y : (400000, 500)
Dimensions of test data X: (100000, 95585) Y: (100000, 500)
```

4.2.2 Logistic Regression with OneVsRestClassifier using count vectorizer
4.2.2.1 Hyperparameter tuning

```
In [56]: 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=0, scoring='
         gsv.fit(x_train_multilabel, y_train)

         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=0, scoring='
         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)
```

```
        print('-------------------------------------------------------------')
        # plotting C vs f1_micro_score
        x_1=[]
        y_1=[]
        for x in gsv.grid_scores_:
            x_1.append(x[0]['estimator__alpha'])
            y_1.append(x[1])
        plt.plot(x_1,y_1)
        plt.xlabel('value of alpha')
        plt.ylabel('f1_micro score')
        plt.title('alpha vs f1_micro score')
        plt.show()
```

```
value of alpha after hyperparameter tuning :  0.001
-------------------------------------------------------------
```



4.2.2.2 Applying model using best hyperparameter

```
In [57]: start = datetime.now()
         #best_alpha = gsv.best_estimator_.get_params()['estimator__alpha']
         classifier = OneVsRestClassifier(SGDClassifier(loss='log', alpha=best_alpha, penalty='l
         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,

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,

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

```
Accuracy : 0.18621
Hamming loss  0.00322218
Micro-average quality numbers
Precision: 0.5636, Recall: 0.3238, F1-measure: 0.4113
Macro-average quality numbers
Precision: 0.4073, Recall: 0.2397, F1-measure: 0.2823

Time taken to run this cell : 0:05:26.013286
```

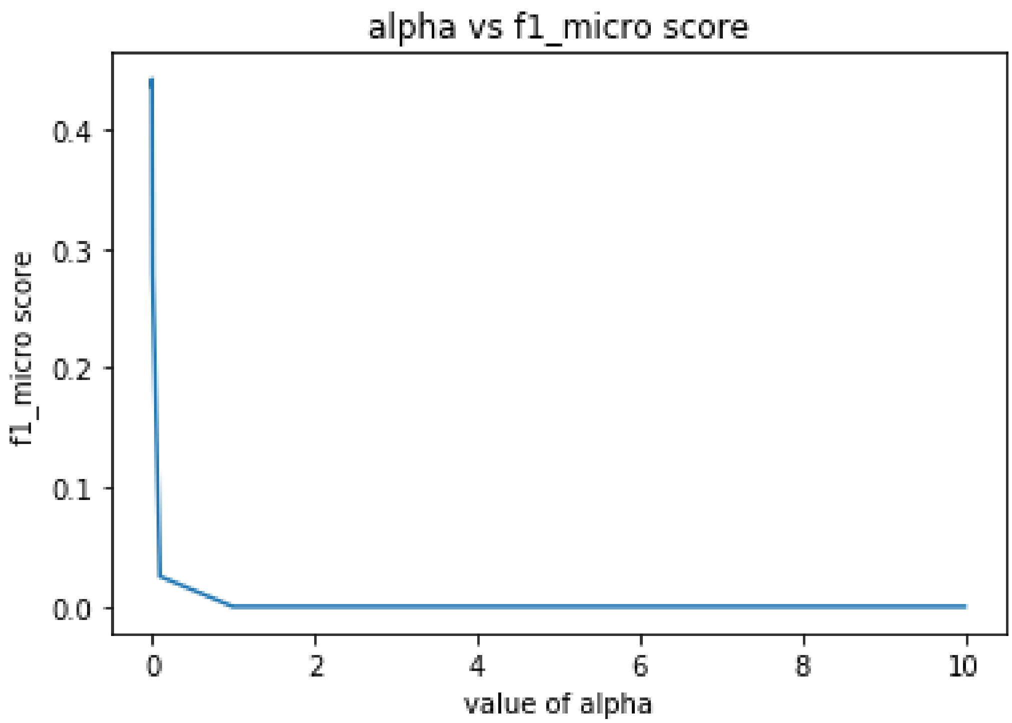
4.2.3 Linear SVM with OneVsRestClassifier

4.2.3.1 Hyperparameter tuning

```
In [25]: 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=0, scoring='
         gsv.fit(x_train_multilabel, y_train)

         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=0, scoring='
         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)
         print('--------------------------------------------------------------')
```

```
# plotting C vs f1_micro_score
x_1=[]
y_1=[]
for x in gsv.grid_scores_:
    x_1.append(x[0]['estimator__alpha'])
    y_1.append(x[1])
plt.plot(x_1,y_1)
plt.xlabel('value of alpha')
plt.ylabel('f1_micro score')
plt.title('alpha vs f1_micro score')
plt.show()
```

```
value of alpha after hyperparameter tuning :  0.001
-------------------------------------------------------------
```



4.2.3.2 Applying model using best hyperparameter

```
In [26]: start = datetime.now()
         #best_alpha = gsv.best_estimator_.get_params()['estimator__alpha']
         classifier = OneVsRestClassifier(SGDClassifier(loss='hinge', alpha=best_alpha, penalty=
         classifier.fit(x_train_multilabel, y_train)
         predictions = classifier.predict (x_test_multilabel)
```

```python
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,

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,

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

```
Accuracy : 0.17942
Hamming loss  0.00326302
Micro-average quality numbers
Precision: 0.5525, Recall: 0.3226, F1-measure: 0.4074
Macro-average quality numbers
Precision: 0.3128, Recall: 0.2396, F1-measure: 0.2549
Time taken to run this cell : 0:05:42.001801
```

1.1 Performance Table

Sr. No.	Model	Featurization	Micro f1_score	Macro f1_score	Hamming loss	Accuracy
1	Logistic Regression	Tfidf vectorizer	0.4488	0.3340	0.0027	0.2364
2	Linear SVM	Tfidf vectorizer	0.3370	0.1607	0.0029	0.2105
3	Logistic Regression	Count vectorizer	0.4113	0.2823	0.0032	0.1862
4	Linear SVM	Count vectorizer	0.4074	0.2549	0.0032	0.1794

1.2 Conclusion

- We have choosen 'f1_micro' scoring metric because of the stated business statement.

- Used bag of words upto 4 grams and Tfidf upto 3 grams.
- For logistic regression , I have used 'SGDClassifier' instead of 'LogisticRegression'. The reason is 'LogisticRegression' takes lots of time for hyperparameter tuning. Even we have not choosen any complex model like xgboost, because the dimension is very high and linear model works fairly well in high dimension and the complex model like xgboost may not work well for this much high dimension, as well as it takes lots of time for hyperparameter tuning.
- We can see in the performance table that Logistic Regression with Tfidf vectorizer works better than Linear SVM.