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

Stack Overflow: Tag Prediction

1. Business Problem

1.1 Description

Description

Stack Overflow is the largest, most trusted online community for developers to learn, share their programming knowledge, and build their careers.

Stack Overflow is something which every programmer use one way or another. Each month, over 50 million developers come to Stack Overflow to learn, share their knowledge, and build their careers. It features questions and answers on a wide range of topics in computer programming. The website serves as a platform for users to ask and answer questions, and, through membership and active participation, to vote questions and answers up or down and edit questions and answers in a fashion similar to a wiki or Digg. As of April 2014 Stack Overflow has over 4,000,000 registered

users, and it exceeded 10,000,000 questions in late August 2015. Based on the type of tags assigned to questions, the top eight most discussed topics on the site are: Java, JavaScript, C#, PHP, Android, jQuery, Python and HTML.

Problem Statemtent

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

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

1.2 Source / useful links

Data Source : https://www.kaggle.com/c/facebook-recruiting-iii-keyword-extraction/data (https://www.kaggle.com/c/facebook-recruiting-iii-keyword-extraction/data)
Youtube : https://youtu.be/nNDqbUhtIRg (https://youtu.be/nNDqbUhtIRg)
Research paper : https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/tagging-1.pdf (https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/tagging-1.pdf)
Research paper : https://dl.acm.org/citation.cfm?id=2660970&dl=ACM&coll=DL (https://dl.acm.org/citation.cfm?id=2660970&dl=ACM&coll=DL)

1.3 Real World / Business Objectives and Constraints

1. Predict as many tags as possible with high precision and recall.
2. Incorrect tags could impact customer experience on StackOverflow.
3. No strict latency constraints.

2. Machine Learning problem

2.1 Data

2.1.1 Data Overview

Refer: https://www.kaggle.com/c/facebook-recruiting-iii-keyword-extraction/data (https://www.kaggle.com/c/facebook-recruiting-iii-keyword-extraction/data)
All of the data is in 2 files: Train and Test.

Train.csv contains 4 columns: Id,Title,Body,Tags.

Test.csv contains the same columns but without the Tags, which you are to predict.

Size of Train.csv - 6.75GB

Size of Test.csv - 2GB

Number of rows in Train.csv = 6034195

The questions are randomized and contains a mix of verbose text sites as well as sites related to math and programming. The number of questions from each site may vary, and no filtering has been performed on the questions (such as closed questions).

Data Field Explaination

Dataset contains 6,034,195 rows. The columns in the table are:

Id - Unique identifier for each question

Title - The question's title

Body - The body of the question

Tags - The tags associated with the question in a space-seperated format (all lowercase, should not contain tabs '\t' or ampersands '&')

2.1.2 Example Data point

Title: Implementing Boundary Value Analysis of Software Testing in a C++ program?
Body :

```
        #include<
        iostream>\n
        #include<
        stdlib.h>\n\n
        using namespace std;\n\n
        int main()\n
        {\n
                int n,a[n],x,c,u[n],m[n],e[n][4];\n
                cout<<"Enter the number of variables";\n
cin>>n;\n\n
                cout<<"Enter the Lower, and Upper Limits of the
 variables";\n
                for(int y=1; y<n+1; y++)\n
                {\n
                   cin>>m[y];\n
                   cin>>u[y];\n
                }\n
                for(x=1; x<n+1; x++)\n
                {\n
                   a[x] = (m[x] + u[x])/2;\n
                }\n
                c=(n*4)-4;\n
                for(int a1=1; a1<n+1; a1++)\n
                {\n\n
                   e[a1][0] = m[a1];\n
                   e[a1][1] = m[a1]+1;\n
                   e[a1][2] = u[a1]-1;\n
                   e[a1][3] = u[a1];\n
                }\n
                for(int i=1; i<n+1; i++)\n
                {\n
                   for(int l=1; l<=i; l++)\n
                   {\n
                       if(l!=1)\n
                       {\n
                           cout<<a[l]<<"\\t";\n
                       }\n
                   }\n
                   for(int j=0; j<4; j++)\n
                   {\n
                       cout<<e[i][j];\n
                       for(int k=0; k<n-(i+1); k++)\n
                       {\n
                           cout<<a[k]<<"\\t";\n
                       }\n
                       cout<<"\\n";\n
                   }\n
                }    \n\n
```

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

\n\n

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

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

\n\n

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

The output is not coming,can anyone correct the code or tell me what\'s w
rong?
\n'
Tags : 'c++ c'

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

2.2.1 Type of Machine Learning Problem

2.2.1 Type of Machine Learning Problem

It is a multi-label classification problem
Multi-label Classification: Multilabel classification assigns to each sample a set of target labels. This can be thought as predicting properties of a data-point that are not mutually exclusive, such as topics that are relevant for a document. A question on Stackoverflow might be about any of C, Pointers, FileIO and/or memory-management at the same time or none of these.
Credit: http://scikit-learn.org/stable/modules/multiclass.html (http://scikit-learn.org/stable/modules/multiclass.html)

2.2.2 Performance metric

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

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

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

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

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

https://www.kaggle.com/wiki/MeanFScore (https://www.kaggle.com/wiki/MeanFScore)
http://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html (http://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html)

Hamming loss : The Hamming loss is the fraction of labels that are incorrectly predicted.
https://www.kaggle.com/wiki/HammingLoss (https://www.kaggle.com/wiki/HammingLoss)

3. Exploratory Data Analysis

3.1 Data Loading and Cleaning

3.1.1 Using Pandas with SQLite to Load the data

In [0]:
```
#we want to extract data efficeintly as the amonunt of data we have here is h

if not os.path.isfile('train.db'):
    #finding the existence of file in directory

    start = datetime.now()
    disk_engine = create_engine('sqlite://train.db')#creating the engine for

    start = dt.datetime.now()#current time after creating the engine

    chunksize = 180000 #it refers to the size that we will use for creating t

    j = 0
    index_start = 1
    for df in pd.read_csv('Train.csv',names = ['Id','Title','Body','Tags'],ch
                          encoding = 'utf-8'):
            #here we are reading for training data present in csv format to
        df.index = df.index + index_start
        j += 1
        print('{} rows'.format(j*chunksize))
        df.to_sql('data',disk_engine,if_exists = 'append')#appending the new
        index_start = df.index[-1] + 1 #index starts from the end of final ro

    print('time taken to run the cell:',datetime.now() - start)#total time ta
```

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
    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
```

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

In [0]:
```
#now we will count the number of rows in the database
if os.path.isfile('train.db'):
    start = datetime.now()
    con = sqlite3.connect('train.db')#establishig the connection
    num_rows = pd.read_sql_query('SELECT count(*) FROM data',con)#selectin al

    #now we will printand close the database
    print('Number of rows in the database are:',num_rows['count(*)'])
    con.close()
else:
    print('please download the train.db file from the google drive or run abo
```

```
please download the train.db file from the google drive or run above cell to ge
t the database
```

3.1.3 Checking for duplicates

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

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

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

Out[6]:

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

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

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

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

Out[8]:
```
1    2656284
2    1272336
3     277575
4         90
5         25
6          5
Name: cnt_dup, dtype: int64
```

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

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

Out[9]:

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

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

Out[10]:
```
3    1206157
2    1111706
4     814996
1     568298
5     505158
Name: tag_count, dtype: int64
```

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

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

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

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

In [0]:
```
con = sqlite3.connect('train_no_dup.db')#establishing connction
tag_data = pd.read_sql_query('SELECT Tags FROM no_dup_train',con)
con.close()#closing the connection

```

In [0]:
```
#dropping the unwanted column
tag_data.drop(tag_data.index[0],inplace = True)

#printting first five columns from the dataframe
tag_data.head()
```

Out[13]:

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

3.2 Analysis of Tags

3.2.1 Total number of unique tags

In [0]:
```
#we will use the countvectorizer here to study how many tags for each text an
start = datetime.now()
vectorizer = CountVectorizer(tokenizer = lambda x: x.split())#by default spli

tag_dtm = vectorizer.fit_transform(tag_data['Tags'])#fit transform will trans
print('total time taken for execution is:',datetime.now() - start)
```

```
total time taken for execution is: 0:00:16.275331
```

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

```
shapeof the data is : (4206314, 42048)
Number of data points : 4206314
Number of unique tags : 42048
```

In [0]:
```
#'get_feature_name()' gives us the vocabulary.
tags = vectorizer.get_feature_names()
#Lets look at the tags we have.
print("Some of the tags we have :", tags[:20])
print('these are some pf the tags that are beiing appended')
```

```
Some of the tags we have : ['.a', '.app', '.asp.net-mvc', '.aspxauth', '.bash-p
rofile', '.class-file', '.cs-file', '.doc', '.drv', '.ds-store', '.each', '.em
f', '.exe', '.exe.config', '.hgtags', '.htaccess', '.htpasswd', '.ico', '.lib',
'.lrc']
these are some pf the tags that are beiing appended
```

3.2.3 Number of times a tag appeared

Now we want to store the elements of matrix in a dictionary for better analysis

In [0]:
```

#Lets now store the document term matrix in a dictionary.
freqs = tag_dtm.sum(axis=0).A1 #.A1 is used to return a flattened array and a
result = dict(zip(tags, freqs))#preparing the dictionary
```

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

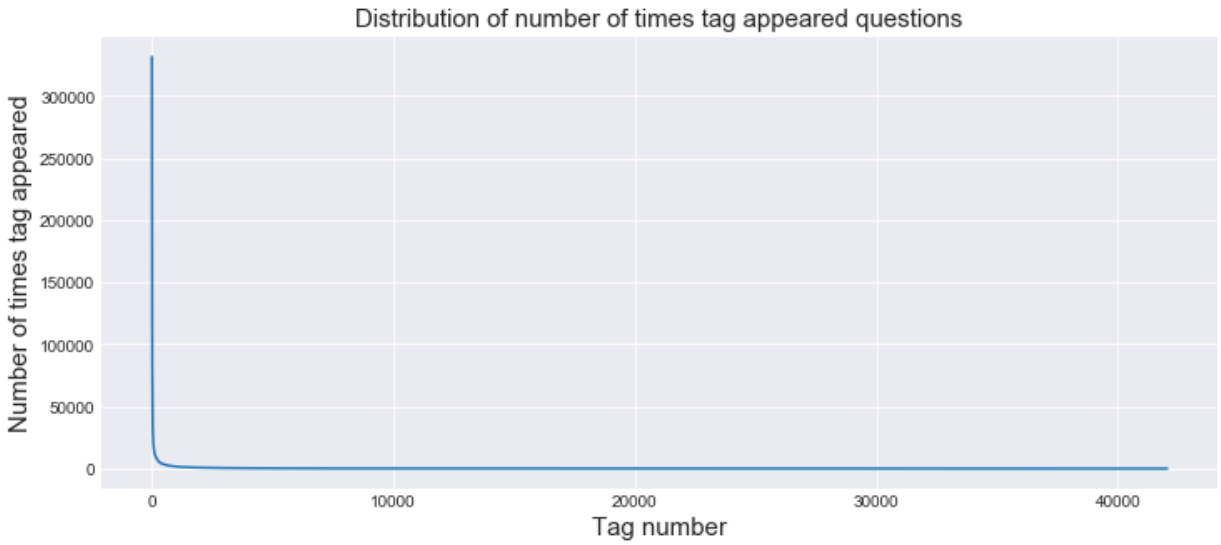
Out[25]:

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

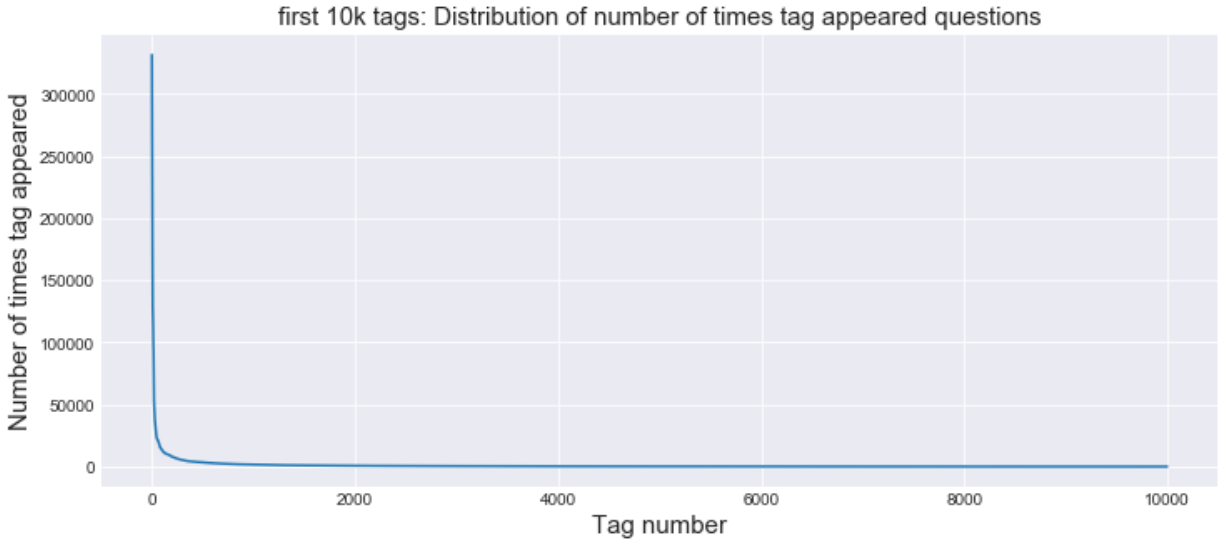
In [0]:
```
tag_df_sorted = tag_df.sort_values(['Counts'], ascending=False)#sorting in de
tag_counts = tag_df_sorted['Counts'].values #storing the values of counts in

```

Out[29]:
```
array([331505, 299414, 284103, ...,      1,      1,      1], dtype=int64)
```

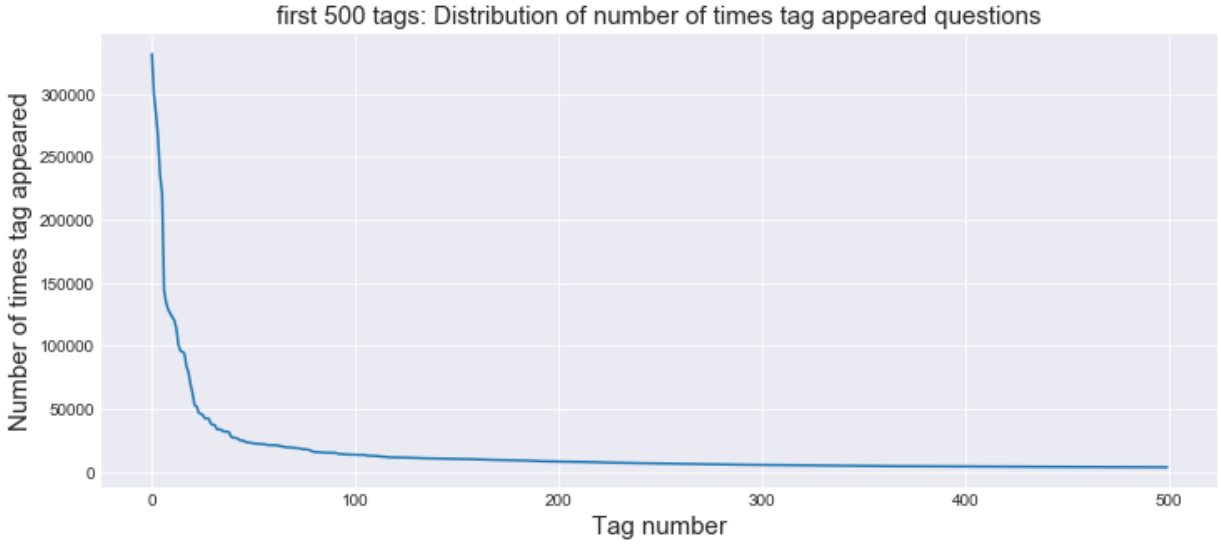
In [0]:
```
#plotting for all
sns.set_style('darkgrid')
plt.figure(figsize = (12,5))
plt.plot(tag_counts)
plt.title("Distribution of number of times tag appeared questions",size = 15)
#plt.grid()
plt.xlabel("Tag number",size = 15)
plt.ylabel("Number of times tag appeared",size = 15)
plt.show()
```



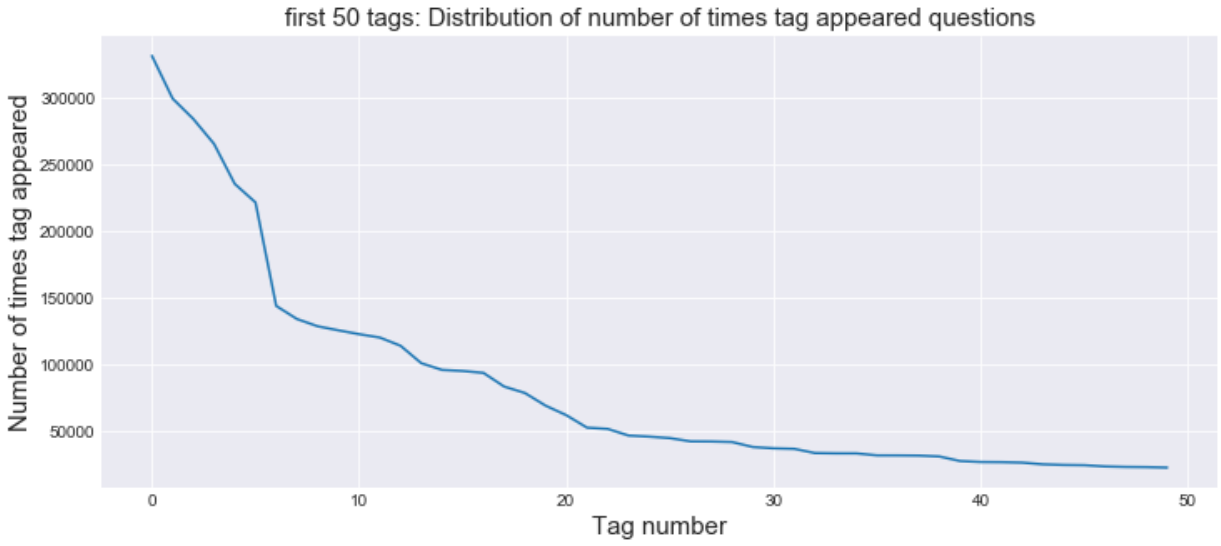
In [0]:
```
#plotting for first 10000 tags
plt.figure(figsize = (12,5))
plt.plot(tag_counts[0:10000])
plt.title('first 10k tags: Distribution of number of times tag appeared quest
#plt.grid()
plt.xlabel("Tag number",size = 15)
plt.ylabel("Number of times tag appeared",size = 15)
plt.show()
```



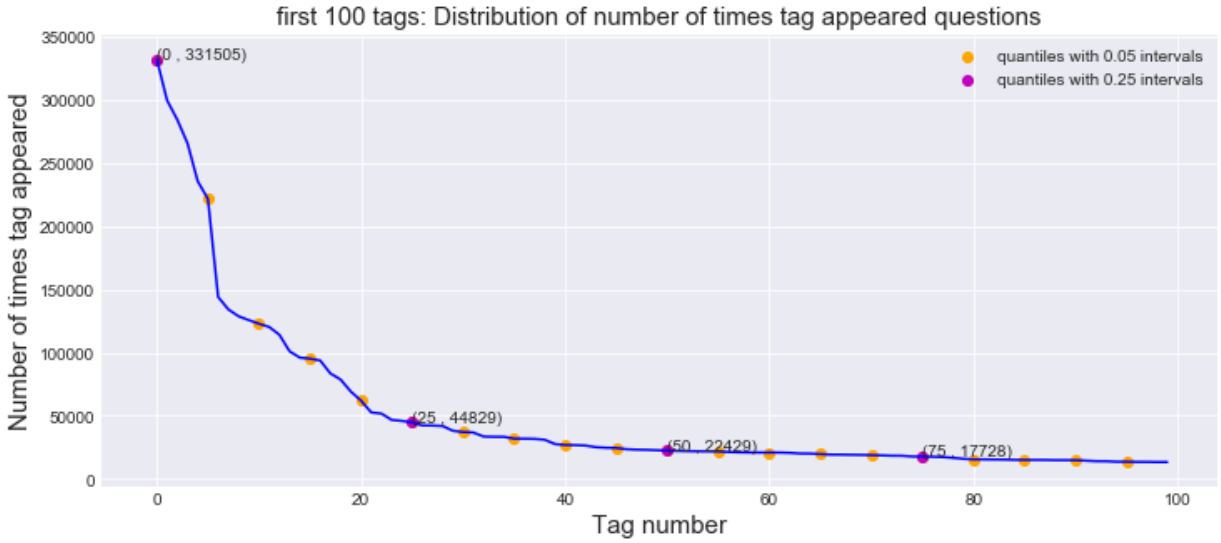
In [0]:
```
#zooming in further for 500 tags
plt.figure(figsize = (12,5))
plt.plot(tag_counts[0:500])
plt.title('first 500 tags: Distribution of number of times tag appeared quest
#plt.grid()
plt.xlabel("Tag number",size = 15)
plt.ylabel("Number of times tag appeared",size = 15)
plt.show()
```



In [0]:
```
#plotting for 50 tags
plt.figure(figsize = (12,5))
plt.plot(tag_counts[0:50])
plt.title('first 50 tags: Distribution of number of times tag appeared questi

plt.xlabel("Tag number",size = 15)
plt.ylabel("Number of times tag appeared",size = 15)
plt.show()

```



In [0]:
```
plt.figure(figsize = (12,5))
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 0.25 difference
plt.scatter(x=list(range(0,100,25)), y=tag_counts[0:100:25], c='m', label = "

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 quest
#plt.grid()
plt.xlabel("Tag number",size = 15)
plt.ylabel("Number of times tag appeared",size = 15)
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
```

In [0]:
```
tf = tag_df.sort_values(['Counts'],ascending = False)#sorting the data in des
print('Tag that occurs maximum number of times in the dataset is :',tag_df['T

```

```
Tag that occurs maximum number of times in the dataset is : c#
```

Observations:

1. There are total 153 tags which are used more than 10000 times.
2. 14 tags are used more than 100000 times.
3. Most frequent tag (i.e. c#) is used 331505 times.
4. Since some tags occur much more frequenctly than others, Micro-averaged F1-score is the appropriate metric for this probelm.

3.2.4 Tags Per Question

In [0]:
```
tag_dtm.shape# a sparse matrix
```

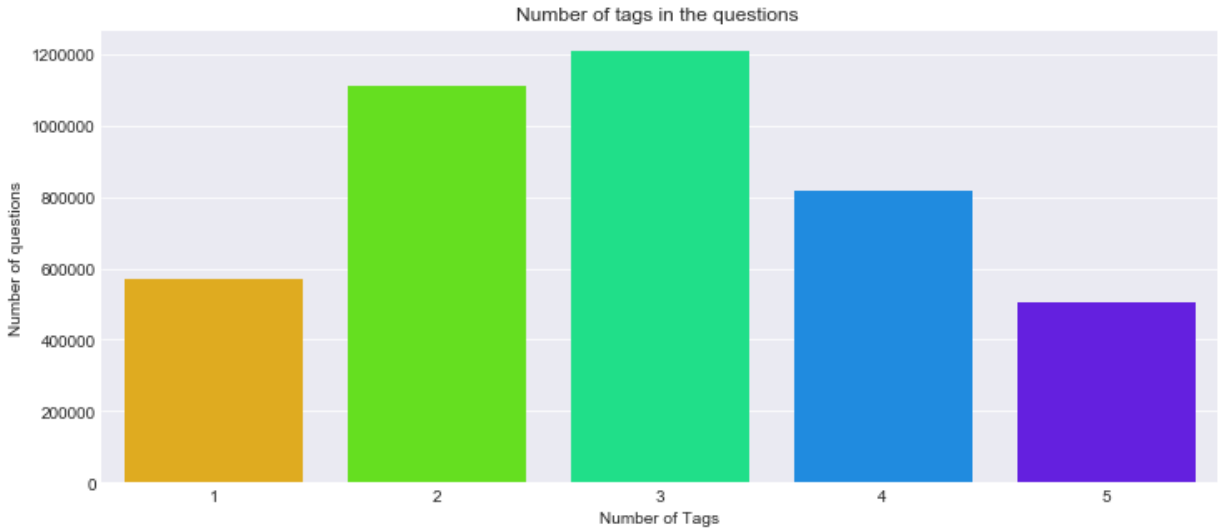
Out[79]:
```
(4206314, 42048)
```

In [0]:
```
#Storing the count of tag in each question in list 'tag_count'
tag_quest_count = tag_dtm.sum(axis=1).tolist()#sums in horizontal mannera
#Converting list of lists into single list, we will get [[3], [4], [2], [2],
tag_quest_counts = []#for counting the number of tags in each question
for i in tag_quest_count:
    for j in i:
        tag_quest_counts.append(j)
```

In [0]:
```
print( "Maximum number of tags per question: %d"%max(tag_quest_counts))
print( "Minimum number of tags per question: %d"%min(tag_quest_counts))
print( "Avg. number of tags per question: %f"% ((sum(tag_quest_counts)*1.0)/l
```

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

In [0]:
```
f = plt.figure(figsize = (12,5))
sns.countplot(tag_quest_counts, palette='gist_rainbow')
plt.title("Number of tags in the questions ")
plt.xlabel("Number of Tags")
plt.ylabel("Number of questions")
plt.show()

f.savefig('count_tags.png',bbox_inches = 'tight',dpi = 600)
```



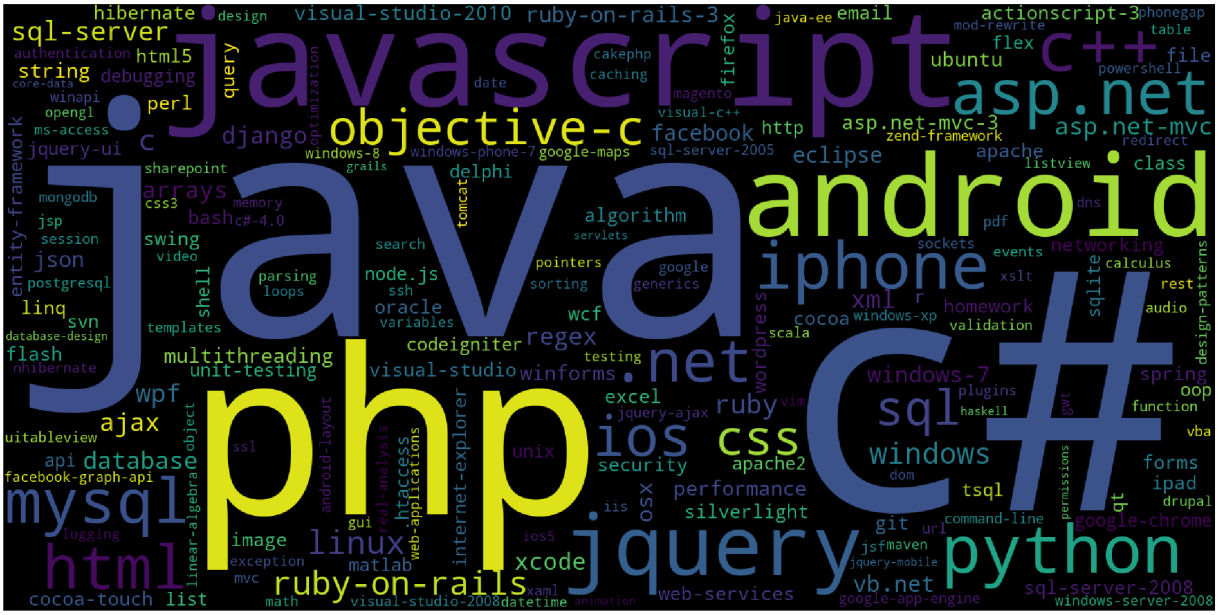
Observations:

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

3.2.5 Most Frequent Tags

In [0]:

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

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

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

fig.savefig('wordcloud.png')
```




```
Time taken to run this cell : 0:00:08.492530
```

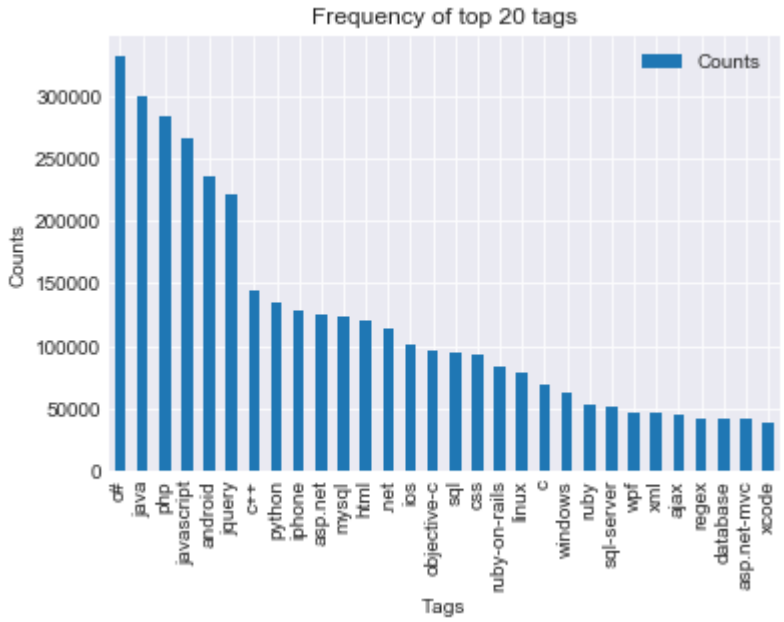
Observations:

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

3.2.6 The top 20 tags

In [0]:
```
#plotting the top 20 tags and their counts respectively
i=np.arange(30)
fig = plt.figure(figsize = (12,5))
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()
fig.savefig('top 20 counts.png')
```

```
<Figure size 864x360 with 0 Axes>
```



Observations:

1. Majority of the most frequent tags are programming language.
2. C# is the top most frequent programming language.
3. Android, IOS, Linux and windows are among the top most frequent operating systems.

3.3 Cleaning and preprocessing of Questions

3.3.1 Preprocessing

1. Sample 1M data points
2. Separate out code-snippets from Body
3. Remove Spcial characters from Question title and description (not in code)

4. Remove stop words (Except 'C')
5. Remove HTML Tags
6. Convert all the characters into small letters
7. Use SnowballStemmer to stem the words

In [0]:
```
#for eg
tag_df_sorted.head()
```

Out[99]:

	Tags	Counts
4337	c#	331505
18069	java	299414
27249	php	284103
18157	javascript	265423
1234	android	235436

In [0]:
```
con = sqlite3.connect('processed.db')
df = pd.read_sql_query('SELECT * FROM QuestionsProcessed',con)
con.close()
```

In [2]:
```
import nltk
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

Out[2]: True

In [0]:
```
def striphtml(data): #function to remove all html tags
    cleanr = re.compile('<.*?>') #helps to search a pattern without rewriting
    cleantext = re.sub(cleanr,'',str(data)) #replaces every pattern with whit
    return cleantext

#a set of all the stopwords
stop_words = set(stopwords.words('english'))
stemmer = SnowballStemmer('english') #performing the snowball stemming
```

In [4]:

```
#now we will create all the necessary tables in the database
def create_connection(db_file):
    """This function establishes a connection with the database specified by
    :param db_file : the database file
    :output: returns connection object or None
    """
    try:
        conn = sqlite3.connect(db_file)
        return conn
    except Error as e: #catches the error if any
        print(e)

    return None

#================================================================================

def create_table(conn,create_table_sql):
    """Create a table fromt the create_table_sql_statement
    :param conn,create_table_sql: establish the connection object and a creat
    :return:
    """

    try:
        c = conn.cursor() #the cursor class allows python to execute command
        c.execute(create_table_sql)#executing the statement
    except Error as e:
        print(e)

#================================================================================

def checkTableExists(dbcon):
    """This function establishes the connection to database and checks if tab
    :param dbcon: the connection to database to be established
    :return : number of tables
    """
    cursr = dbcon.cursor()
    str = "select name from sqlite_master where type = 'table'"
    tables_name = cursr.execute(str)
    print('tables in the database:')
    tables = tables_name.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 connetion.')
    conn.close()

sql_create_table = """CREATE TABLE IF NOT EXISTS QuestionProcessed(question t
,words_post integer, is_code integer);"""

```

```
create_database_table('Processed.db',sql_create_table)
```

```
tables in the database:
QuestionProcessed
```

In [ ]:

```
# http://www.sqlitetutorial.net/sqlite-delete/
# https://stackoverflow.com/questions/2279706/select-random-row-from-a-sqlite
start = datetime.now()
read_db = 'train_no_dup.db'
write_db = 'Processed.db'
if os.path.isfile(read_db):
    conn_r = create_connection(read_db)
    if conn_r is not None:
        reader =conn_r.cursor()
        reader.execute("SELECT Title, Body, Tags From no_dup_train ORDER BY R

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)
```

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

In [0]:

```
#http://www.bernzilla.com/2008/05/13/selecting-a-random-row-from-an-sqlite-ta

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

    is_code = 0

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

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

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

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

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

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

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

    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_p
    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_av
print( "Avg. length of questions(Title+Body) after processing: %d"%no_dup_avg
print ("Percent of questions containing code: %d"%((questions_with_code*100.0

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

```
number of questions completed= 100000
number of questions completed= 200000
number of questions completed= 300000
number of questions completed= 400000
```

```
number of questions completed= 500000
number of questions completed= 600000
number of questions completed= 700000
number of questions completed= 800000
number of questions completed= 900000
Avg. length of questions(Title+Body) before processing: 1169
Avg. length of questions(Title+Body) after processing: 327
Percent of questions containing code: 57
Time taken to run this cell : 0:47:05.946582
```

In [0]:

```
# dont forget to close the connections, or else you will end up with locks
conn_r.commit()
conn_w.commit()
conn_r.close()
conn_w.close()
```

In [0]:
```
if os.path.isfile(write_db):
    conn_r = create_connection(write_db)
    if conn_r is not None:
        reader =conn_r.cursor()
        reader.execute("SELECT question From QuestionsProcessed LIMIT 10")
        print("Questions after preprocessed")
        print('='*100)
        reader.fetchone()
        for row in reader:
            print(row)
            print('-'*100)
conn_r.commit()
conn_r.close()
```

```
Questions after preprocessed
====================================================================================================
('ef code first defin one mani relationship differ key troubl defin one zero
mani relationship entiti ef object model look like use fluent api object comp
osit pk defin batch id batch detail id use fluent api object composit pk defi
n batch detail id compani id map exist databas tpt basic idea submittedtransa
ct zero mani submittedsplittransact associ navig realli need one way submitte
dtransact submittedsplittransact need dbcontext class onmodelcr overrid map c
lass lazi load occur submittedtransact submittedsplittransact help would much
appreci edit taken advic made follow chang dbcontext class ad follow onmodelc
r overrid must miss someth get follow except thrown submittedtransact key bat
ch id batch detail id zero one mani submittedsplittransact key batch detail i
d compani id rather assum convent creat relationship two object configur requ
ir sinc obvious wrong',)
----------------------------------------------------------------------------------------------------
('explan new statement review section c code came accross statement block com
e accross new oper use way someon explain new call way',)
----------------------------------------------------------------------------------------------------
('error function notat function solv logic riddl iloczyni list structur list
possibl candid solut list possibl coordin matrix wan na choos one candid comp
ar possibl candid element equal wan na delet coordin call function skasuj loo
k like ni knowledg haskel cant see what wrong',)
----------------------------------------------------------------------------------------------------
('step plan move one isp anoth one work busi plan switch isp realli soon need
chang lot inform dns wan wan wifi question guy help mayb peopl plan correct c
hang current isp new one first dns know receiv new ip isp major chang need ta
ke consider exchang server owa vpn two site link wireless connect km away cit
rix server vmware exchang domain control link place import server crucial ste
p inform need know avoid downtim busi regard ndavid',)
----------------------------------------------------------------------------------------------------
('use ef migrat creat databas googl migrat tutori af first run applic creat d
atabas ef enabl migrat way creat databas migrat rune applic tri',)
----------------------------------------------------------------------------------------------------
('magento unit test problem magento site recent look way check integr magento
site given point unit test jump one method would assum would big job write wh
ole lot test check everyth site work anyon involv unit test magento advis fol
```

```
low possibl test whole site custom modul nis exampl test would amaz given sit
e heavili link databas would nbe possibl fulli test site without disturb data
bas better way automaticlli check integr magento site say integr realli mean
fault site ship payment etc work correct',)
------------------------------------------------------------------------------
----------------------
('find network devic without bonjour write mac applic need discov mac pcs iph
on ipad connect wifi network bonjour seem reason choic turn problem mani type
router mine exampl work block bonjour servic need find ip devic tri connect a
pplic specif port determin process run best approach accomplish task without
violat app store sandbox',)
------------------------------------------------------------------------------
----------------------
('send multipl row mysql databas want send user mysql databas column user ski
ll time nnow want abl add one row user differ time etc would code send databa
s nthen use help schema',)
------------------------------------------------------------------------------
----------------------
('insert data mysql php powerpoint event powerpoint present run continu way u
pdat slide present automat data mysql databas websit',)
------------------------------------------------------------------------------
----------------------
```

In [0]:
```
/content/Xtest_bow.pkl
```

In [ ]:
```
#Taking 1 Million entries to a dataframe.
start = datetime.now()
write_db = 'Stackoverflow_project/Processed.db'
if os.path.isfile(write_db):
    conn_r = create_connection(write_db)
    if conn_r is not None:
        preprocessed_data = pd.read_sql_query("""SELECT question, Tags FROM Q
conn_r.commit()
conn_r.close()
print('time taken is:',datetime.now() - start)
```

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

Out[6]:

	question	tags
0	chang cpu soni vaio pcg grx tri everywher find...	cpu motherboard sony-vaio replacement disassembly
1	display size grayscal qimag qt abl display ima...	c++ qt qt4
2	datagrid selecteditem set back null eventtocom...	mvvm silverlight-4.0
3	filter string collect base listview item resol...	c# winforms string listview collections
4	disabl home button without use type keyguard c...	android android-layout android-manifest androi...

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

```
number of data points in sample : 999999
number of dimensions : 2
```

4. Machine Learning Models

4.1 Converting tags for multilabel problems

X	y1	y2	y3	y4
x1	0	1	1	0
x1	1	0	0	0
x1	0	1	0	0

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

In [0]:
```
print('shape of the data after vectorization is:',multilabel_y.shape)
print('some of the tags we get here are:',vectorizer.get_feature_names()[:10]
```

```
shape of the data after vectorization is: (999999, 35422)
some of the tags we get here are: ['.a', '.app', '.asp.net-mvc', '.aspxauth',
'.bash-profile', '.class-file', '.cs-file', '.doc', '.ds-store', '.each']
```

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

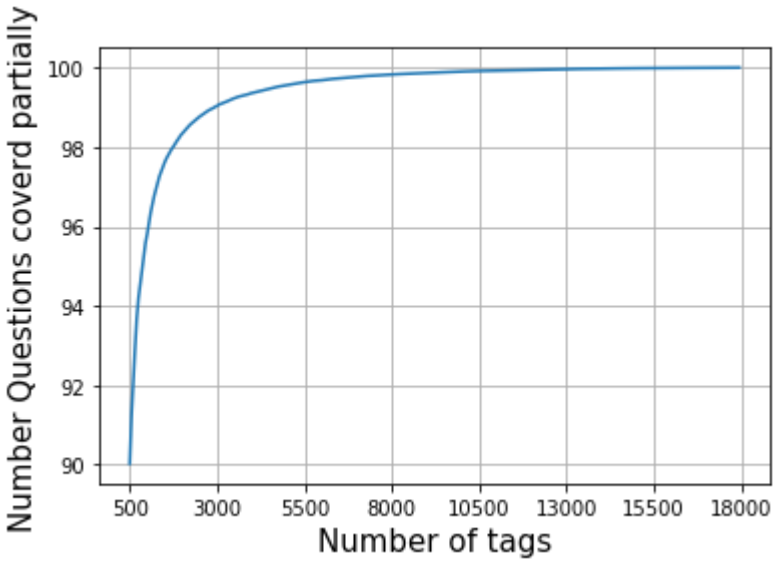
The strategy we will be using here is partial coverage ,that is instead of using all the 42k tags in the data we find its subset where we take the tags which occur most frequently and that will be able to cover most the questions appearing ,for example if t1 and t2 occur in most of the questions then they will be able to cover most of the questions here.

In [0]:
```
def tags_to_choose(n):
    """Function for giving the percentage  """
    t = multilabel_y.sum(axis=0).tolist()[0] #summing all occurence of a part
    sorted_tags_i = sorted(range(len(t)), key=lambda i: t[i], reverse=True)#s
    multilabel_yn=multilabel_y[:,sorted_tags_i[:n]]
    return multilabel_yn

def questions_explained_fn(n):
    multilabel_yn = tags_to_choose(n)# returns the questions with given tag
    x= multilabel_yn.sum(axis=1)#number of questions summed up
    return (np.count_nonzero(x==0)) #returns the count
```

In [0]:
```
questions_explained = []
total_tags=multilabel_y.shape[1]
print('total tags we have are:',total_tags)
total_qs=preprocessed_data.shape[0]
print('total number of questions we have after preprocessing is:',total_qs)
for i in range(500, total_tags, 100):
    questions_explained.append(np.round(((total_qs-questions_explained_fn(i))

print('finally total questions explained are:',len(questions_explained))
```

```
total tags we have are: 35422
total number of questions we have after preprocessing is: 999999
finally total questions explained are: 350
```

In [0]:
```
fig, ax = plt.subplots()

ax.plot(questions_explained)
xlabel = list(500+np.array(range(-50,450,50))*50)
ax.set_xticklabels(xlabel)
plt.xlabel("Number of tags",size = 15)
plt.ylabel("Number Questions coverd partially",size = 15)
plt.grid()
plt.show()
# you can choose any number of tags based on your computing power, minimun is
print("with ",5500,"tags we are covering ",questions_explained[50],"% of ques

fig.savefig('partial_coverage.png')
```




```
with  5500 tags we are covering  99.035 % of questions
```

In [0]:
```
multilabel_yx = tags_to_choose(5500)
print("number of questions that are not covered :", questions_explained_fn(55
```

```
number of questions that are not covered : 9645 out of  999999
```

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

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

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

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

for example if we encounter a datapoint at test time that has a tag which we did not have till this point of time then what we do is that we retrain the model in next itertion and then use the model

In [0]:
```
#we are splitting data here randomly as the data is not temporal in nature

total_size=preprocessed_data.shape[0]
train_size=int(0.80*total_size)# training data is 80%

x_train=preprocessed_data.head(train_size)
x_test=preprocessed_data.tail(total_size - train_size) #test data is 20%

y_train = multilabel_yx[0:train_size,:]#target variable for training data
y_test = multilabel_yx[train_size:total_size,:] #target variable for test dat
```

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

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

4.3 Featurizing data

here we have a high dimensional data because of using something like TFIDFvectorizer and linear models work very well on such a data.second things is that linear svm havs high time complexity which makes it tough to use

In [0]:
```
start = datetime.now()
vectorizer = TfidfVectorizer(min_df=0.00009, max_features=200000, smooth_idf=
                             tokenizer = lambda x: x.split(), sublinear_tf=Fa
x_train_multilabel = vectorizer.fit_transform(x_train['question'])
x_test_multilabel = vectorizer.transform(x_test['question'])
print("Time taken to run this cell :", datetime.now() - start)
```

```
Time taken to run this cell : 0:09:50.460431
```

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

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

In [0]:
```
# https://www.analyticsvidhya.com/blog/2017/08/introduction-to-multi-label-cl
#https://stats.stackexchange.com/questions/117796/scikit-multi-label-classifi
# classifier = LabelPowerset(GaussianNB())
"""
from skmultilearn.adapt import MLkNN
classifier = MLkNN(k=21)

# train
classifier.fit(x_train_multilabel, y_train)

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

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

Out[92]:
```
"\nfrom skmultilearn.adapt import MLkNN\nclassifier = MLkNN(k=21)\n\n# train\nc
lassifier.fit(x_train_multilabel, y_train)\n\n# predict\npredictions = classifi
er.predict(x_test_multilabel)\nprint(accuracy_score(y_test,predictions))\nprint
(metrics.f1_score(y_test, predictions, average = 'macro'))\nprint(metrics.f1_sc
ore(y_test, predictions, average = 'micro'))\nprint(metrics.hamming_loss(y_tes
t,predictions))\n\n"
```

4.4 Applying Logistic Regression with OneVsRest Classifier

In [0]:
```
# this will be taking so much time try not to run it, download the lr_with_eq
# This takes about 6-7 hours to run.
classifier = OneVsRestClassifier(SGDClassifier(loss='log', alpha=0.00001, pen
classifier.fit(x_train_multilabel, y_train)
predictions = classifier.predict(x_test_multilabel)

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

```

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

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

In [0]:
```
from sklearn.externals import joblib
joblib.dump(classifier, 'lr_with_equal_weight.pkl')
```

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

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

```
tables in the database:
QuestionsProcessed
```

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

read_db = 'My Drive/Stackoverflow_project/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 5000
        # for selecting random points
        #reader.execute("SELECT Title, Body, Tags From no_dup_train ORDER BY

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

```
tables in the database:
QuestionsProcessed
Cleared All the rows
```

4.5.1 Preprocessing of questions

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

In [ ]:

```
#http://www.bernzilla.com/2008/05/13/selecting-a-random-row-from-an-sqlite-ta
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
    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+" "+
#     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 t
    question=' '.join(str(stemmer.stem(j)) for j in words if j not in stop_wo

    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_p
    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_av
print( "Avg. length of questions(Title+Body) after processing: %d"%no_dup_avg
print ("Percent of questions containing code: %d"%((questions_with_code*100.0
```

```
57
58 print("Time taken to run this cell :", datetime.now() - start)
```

In [0]:
```
1 import nltk
2 nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
```

Out[4]: True

In [ ]:
```
1 # never forget to close the conections or else we will end up with database l
2 conn_r.commit()
3 conn_w.commit()
4 conn_r.close()
5 conn_w.close()
```

Sample quesitons after preprocessing of data

In [0]:
```
1  if os.path.isfile(write_db):
2      conn_r = create_connection(write_db)
3      if conn_r is not None:
4          reader =conn_r.cursor()
5          reader.execute("SELECT question From QuestionsProcessed LIMIT 10")
6          print("Questions after preprocessed")
7          print('='*100)
8          reader.fetchone()
9          for row in reader:
10             print(row)
11             print('-'*100)
12 conn_r.commit()
13 conn_r.close()
```

```
Questions after preprocessed
====================================================================================================
```

Saving Preprocessed data to a Database

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

In [2]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&response_type=code&scope=email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly (https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&response_type=code&scope=email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly)

Enter your authorization code:
··········
Mounted at /content/drive
```

In [0]:
```
preprocessed_data = pd.read_csv('drive/My Drive/Stack/preprocessed1.csv')
```

In [8]:
```
preprocessed_data.head()
```

Out[8]:

	question	tags
0	dynam datagrid bind silverlight dynam datagrid...	c# silverlight data-binding
1	dynam datagrid bind silverlight dynam datagrid...	c# silverlight data-binding columns
2	java.lang.noclassdeffounderror javax servlet j...	jsp jstl
3	java.sql.sqlexcept microsoft odbc driver manag...	java jdbc
4	better way updat feed fb php sdk better way up...	facebook api facebook-php-sdk

In [9]:
```
print("number of data points in sample :", preprocessed_data.shape[0])
print("number of dimensions :", preprocessed_data.shape[1])
```

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

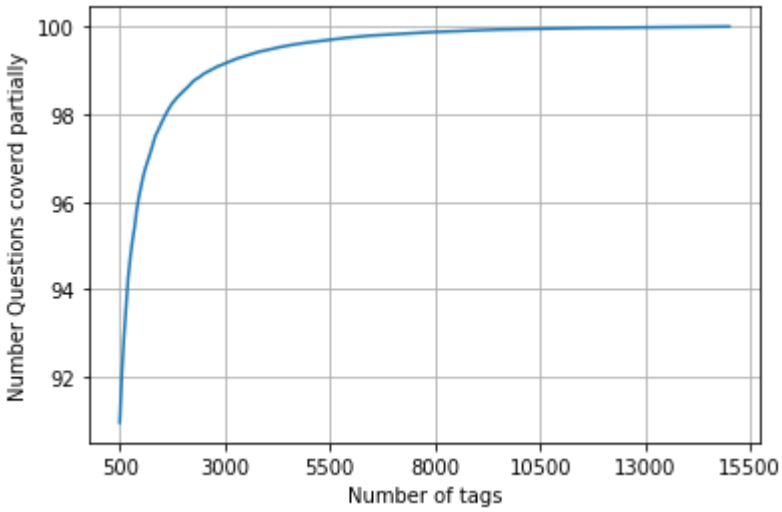
Converting string Tags to multilable output variables

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

Selecting 500 Tags

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

In [12]:
```
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
print("with ",5500,"tags we are covering ",questions_explained[50],"% of ques
print("with ",500,"tags we are covering ",questions_explained[0],"% of questi
```




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

In [13]:
```
# we will be taking 500 tags
multilabel_yx = tags_to_choose(500)
print("number of questions that are not covered :", questions_explained_fn(50
```

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

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

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

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

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

4.5.2 Featurizing data with Tfldf vectorizer

In [0]:
```
start = datetime.now()
vectorizer = TfidfVectorizer(min_df=0.00009, max_features=200000, smooth_idf=
                             tokenizer = lambda x: x.split(), sublinear_tf=Fa
x_train_multilabel = vectorizer.fit_transform(x_train['question'])
x_test_multilabel = vectorizer.transform(x_test['question'])
print("Time taken to run this cell :", datetime.now() - start)
```

```
Time taken to run this cell : 0:03:52.522389
```

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

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

4.5.3 Applying Logistic Regression with OneVsRest Classifier

In [0]:
```
start = datetime.now()
classifier = OneVsRestClassifier(SGDClassifier(loss='log', alpha=0.00001, pen
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(precisio

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(precisio

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

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

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

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

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

In [0]:
```
start = datetime.now()
classifier_2 = OneVsRestClassifier(LogisticRegression(penalty='l1'), n_jobs=-
classifier_2.fit(x_train_multilabel, y_train)
predictions_2 = classifier_2.predict(x_test_multilabel)
print("Accuracy :",metrics.accuracy_score(y_test, predictions_2))
print("Hamming loss ",metrics.hamming_loss(y_test,predictions_2))


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

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

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

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

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

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

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

5. Assignments

1. Use bag of words upto 4 grams and compute the micro f1 score with Logistic regression(OvR)
2. Perform hyperparam tuning on alpha (or lambda) for Logistic regression to improve the performance using GridSearch
3. Try OneVsRestClassifier with Linear-SVM (SGDClassifier with loss-hinge)

1.1Countvectorizer on 0.5M points with all the features

In [0]:
```
#using bag of words upto 4 grams

start = datetime.now()
vectorizer = CountVectorizer(min_df=0.00009, max_features=200000, \
                             tokenizer = lambda x: x.split(), ngram_range=(1,
vectorizer.fit(x_train['question'])#fitting for vectorization
X_train_bow = vectorizer.transform(x_train['question'])
X_test_bow = vectorizer.transform(x_test['question'])

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

```
Time taken to run this cell : 0:07:39.522230
```

In [0]:
```
#saving the training and test data for further usage
import pickle
with open('Xtrain_bow.pkl','wb') as file:
  pickle.dump(X_train_bow,file)

with open('Xtest_bow.pkl','wb') as file:
  pickle.dump(X_test_bow,file)

print('Shape of training data after vectorization is :',X_train_bow.shape,'Sh
print('Shape of test data after vectorization is :',X_test_bow.shape,'Shape o
```

In [0]:
```
print('Shape of training data after vectorization is :',X_train_bow.shape,'Sh
print('Shape of test data after vectorization is :',X_test_bow.shape,'Shape o
```

```
Shape of training data after vectorization is : (400000, 99399) Shape of multil
abel target for training data: (400000, 500)
Shape of test data after vectorization is : (100000, 99399) Shape of multilabel
target for training data: (100000, 500)
```

1.2 SGDClassifier with log loss

In [0]:
```
start = datetime.now()
classifier_2 = OneVsRestClassifier(LogisticRegression(penalty='l1'), n_jobs=-
classifier_2.fit(X_train_bow, y_train)
predictions_2 = classifier_2.predict(X_test_bow)
print("Accuracy :",metrics.accuracy_score(y_test, predictions_2))
print("Hamming loss ",metrics.hamming_loss(y_test,predictions_2))


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

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

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

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

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

```
Accuracy : 0.21289
Hamming loss  0.00313266
Micro-average quality numbers
Precision: 0.5687, Recall: 0.4093, F1-measure: 0.4760
Macro-average quality numbers
Precision: 0.4507, Recall: 0.3335, F1-measure: 0.3798
              precision    recall  f1-score   support

           0       0.90      0.74      0.81      5519
           1       0.52      0.41      0.46      8190
           2       0.63      0.47      0.54      6529
           3       0.67      0.53      0.59      3231
           4       0.66      0.49      0.56      6430
           5       0.62      0.43      0.51      2879
           6       0.74      0.56      0.64      5086
           7       0.75      0.61      0.68      4533
           8       0.34      0.18      0.24      3000
           9       0.70      0.59      0.64      2765
          10       0.43      0.29      0.35      3051
          11       0.59      0.45      0.51      3000
```

2.taking 200k data points and Hyperparameter tuning with randomized CV for LOGISTIC Regression

Computing one vs rest is fairly expensive computationally so what we are doing here is restricting to 0.2M data ppints and taking in consideration top 5000 features only

In [0]:
```
preprocessed_data = preprocessed_data.head(200000) #taking 100k datapoints
```

In [0]:
```
train_datasize = 160000 #taking the training dataset to be 80%
x_train=preprocessed_data.head(train_datasize)
x_test=preprocessed_data.tail(preprocessed_data.shape[0] - 160000)

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

In [0]:
```
del preprocessed_data
```

In [16]:
```
print('size of training data is:',x_train.shape,'Size of target label for tra
print('size of test data is:',x_test.shape,'Size of target label for test dat
```

```
size of training data is: (160000, 2) Size of target label for training data i
s: (160000, 500)
size of test data is: (40000, 2) Size of target label for test data is: (40000,
500)
```

In [17]:
```
y_train.shape
```

Out[17]: (160000, 500)

2.1 BAG of WORDS with 4 grams

In [18]:
```
start = datetime.now()
vectorizer = CountVectorizer(min_df = 0.00009,max_features = 5000,analyzer =
#taking the top 5k features in consideration here

vectorizer.fit(x_train['question'])
X_train = vectorizer.transform(x_train['question'])
X_test = vectorizer.transform(x_test['question'])

print('Time taken for vectorizing the data is:',datetime.now() - start)
```

```
Time taken for vectorizing the data is: 0:01:40.376950
```

In [20]:
```
print('After bag of words featurization shape of the data is:')
print(X_train.shape)
print(X_test.shape)
```

```
After bag of words featurization shape of the data is:
(160000, 5000)
(40000, 5000)
```

In [0]:
```
import pickle
with open('X_train.pkl','wb') as file:
  pickle.dump(X_train,file)

with open('X_test.pkl','wb') as file:
  pickle.dump(X_test,file)

with open('y_train.pkl','wb') as file:
  pickle.dump(y_train,file)

with open('y_test.pkl','wb') as file:
  pickle.dump(y_test,file)
```

2.2 Hypertuning the alpha for Logistic Regression using RandomizedSearchCV

In [0]:
```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import KFold #importing library for cross valida
import pickle
start = datetime.now()
#n_folds = 3
cv_kfold = KFold(n_splits=3).split(X_train)
alpha = [10**i for i in range(-7,7)]
params  = {"estimator__alpha":alpha}
base_estimator=OneVsRestClassifier(SGDClassifier(loss='log', penalty='l1', ra
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions
rsearch_cv.fit(X_train, y_train)
print('Total time taken for tuning the model:',datetime.now() - start)

```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 257.8min finished

Total time taken for tuning the model: 4:43:19.712564
```

In [0]:
```
#best results
print('The best estimator is:',rsearch_cv.best_estimator_)
print('Score we get:',rsearch_cv.best_score_)
print('best parameters:',rsearch_cv.best_params_)
print(rsearch_cv.scorer_)
```

```
The best estimator is: OneVsRestClassifier(estimator=SGDClassifier(alpha=0.000
1, average=False,
                                                   class_weight=None,
                                                   early_stopping=False, epsilon=0.1,
                                                   eta0=0.0, fit_intercept=True,
                                                   l1_ratio=0.15,
                                                   learning_rate='optimal', loss='lo
g',
                                                   max_iter=1000, n_iter_no_change=5,
                                                   n_jobs=None, penalty='l1',
                                                   power_t=0.5, random_state=0,
                                                   shuffle=True, tol=0.001,
                                                   validation_fraction=0.1, verbose=0,
                                                   warm_start=False),
                    n_jobs=None)
Score we get: 0.3973240028076955
best parameters: {'estimator__alpha': 0.0001}
make_scorer(f1_score, pos_label=None, average=micro)
```

In [0]:
```
lr_df = pd.DataFrame(rsearch_cv.cv_results_)#dataframe for scoring the result
lr_df
```

Out[24]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_estimator__alpha	
0	484.840469	12.192954	4.852524	0.164222	0.01	{'estimator
1	5888.638878	801.574254	4.976706	0.070345	1e-07	{'estimator
2	2048.393012	162.555626	4.847765	0.148048	0.0001	{'estimator
3	371.977394	4.488873	4.908308	0.186870	1000	{'estimator
4	5231.541365	382.304175	4.913300	0.122539	1e-06	{'estimator
5	366.726163	3.936187	4.890091	0.145349	1000000	{'estimator
6	3949.761606	287.508571	3.734705	0.908837	1e-05	{'estimator
7	365.189910	4.920660	4.708847	0.073887	100	{'estimator
8	371.991989	3.469639	4.906190	0.123916	10	{'estimator
9	357.262102	14.701879	4.722778	0.406492	100000	{'estimator

2.3 Logistic Regression with best alpha

In [0]:

```
#best classifier and prediction
start = datetime.now()
classifier = OneVsRestClassifier(estimator=SGDClassifier(alpha=0.0001, averag
                                                         class_weight=None,
                                                         early_stopping=False, epsilon=0.1
                                                         eta0=0.0, fit_intercept=True,
                                                         l1_ratio=0.15,
                                                         learning_rate='optimal', loss='lo
                                                         max_iter=1000, n_iter_no_change=5
                                                         n_jobs=None, penalty='l1',
                                                         power_t=0.5, random_state=0,
                                                         shuffle=True, tol=0.001,
                                                         validation_fraction=0.1, verbose=
                                                         warm_start=False),
                         n_jobs=None)

classifier.fit(X_train, y_train)
predictions = classifier.predict(X_test)
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(precisio

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(precisio

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

```
Accuracy : 0.26305
Hamming loss  0.00269995
Micro-average quality numbers
Precision: 0.6796, Recall: 0.6083, F1-measure: 0.6419
Macro-average quality numbers
Precision: 0.2259, Recall: 0.2022, F1-measure: 0.1899
              precision    recall  f1-score   support

           0       0.98      0.98      0.98     36915
           1       0.32      0.06      0.11       140
           2       0.31      0.17      0.22      4486
           3       0.30      0.27      0.29       121
           4       0.50      0.26      0.34       784
           5       0.24      0.22      0.23        37
           6       0.56      0.55      0.55       220
           7       0.75      0.58      0.65       486
           8       0.25      0.03      0.05        33
```

9	0.31	0.46	0.37	189
10	0.32	0.15	0.20	255

2.4 Hypertuning alpha for Linear SVM using RandomizedSearchCV

In [0]:
```
import pickle
#f = 'drive/My Drive/Stackoverflow_project'
X_train = pickle.load(open('drive/My Drive/Stack/X_train.pkl','rb'))
X_test = pickle.load(open('drive/My Drive/Stack/X_test.pkl','rb'))
y_train = pickle.load(open('drive/My Drive/Stack/y_train.pkl','rb'))
y_test = pickle.load(open('drive/My Drive/Stack/y_test.pkl','rb'))
```

In [5]:

```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import KFold #importing library for cross valida
import pickle
start = datetime.now()
#n_folds = 3
cv_kfold = KFold(n_splits=3).split(X_train)
alpha = [10**i for i in range(-7,7)]
params  = {"estimator__alpha":alpha}
base_estimator=OneVsRestClassifier(SGDClassifier(loss='hinge', penalty='l1',
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions
rsearch_cv.fit(X_train, y_train)
print('Total time taken for tuning the model:',datetime.now() - start)

#best results
print('The best estimator is:',rsearch_cv.best_estimator_)
print('Score we get:',rsearch_cv.best_score_)
print('best parameters:',rsearch_cv.best_params_)
print(rsearch_cv.scorer_)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worker
s.
[Parallel(n_jobs=1)]: Done  30 out of  30 | elapsed: 397.7min finished

Total time taken for tuning the model: 7:01:19.713908
The best estimator is: OneVsRestClassifier(estimator=SGDClassifier(alpha=0.000
1, average=False,
                                           class_weight=None,
                                           early_stopping=False, epsilon=0.1,
                                           eta0=0.0, fit_intercept=True,
                                           l1_ratio=0.15,
                                           learning_rate='optimal',
                                           loss='hinge', max_iter=1000,
                                           n_iter_no_change=5, n_jobs=None,
                                           penalty='l1', power_t=0.5,
                                           random_state=0, shuffle=True,
                                           tol=0.001, validation_fraction=0.1,
                                           verbose=0, warm_start=False),
                    n_jobs=None)
Score we get: 0.3921963386527749
best parameters: {'estimator__alpha': 0.0001}
make_scorer(f1_score, pos_label=None, average=micro)
```

In [6]:
```
lsvm_df = pd.DataFrame(rsearch_cv.cv_results_)
lsvm_df
```

Out[6]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_estimator__alpha	
0	2286.943549	36.017186	3.388520	0.011959	1e-05	{'estimator
1	157.016080	2.066268	3.310435	0.063147	100	{'estimator
2	1069.851148	53.097171	3.237592	0.090268	0.0001	{'estimator
3	181.454562	0.531341	3.202725	0.083031	10	{'estimator
4	159.340777	0.809352	3.407450	0.169823	10000	{'estimator
5	309.998456	25.909821	3.269326	0.203893	0.01	{'estimator
6	159.138113	0.810903	3.256443	0.108561	1000000	{'estimator
7	316.773936	7.720801	3.230884	0.044033	0.001	{'estimator
8	381.283375	33.687595	3.202775	0.085991	0.1	{'estimator
9	2899.618952	218.146338	3.364956	0.103348	1e-06	{'estimator

2.5 Linear SVMwith best value of alpha

In [8]:

```
#best classifier and prediction
start = datetime.now()

classifier = OneVsRestClassifier(estimator=SGDClassifier(alpha=0.0001, averag
                                                         class_weight=None,
                                                         early_stopping=False, epsilon=0.1
                                                         eta0=0.0, fit_intercept=True,
                                                         l1_ratio=0.15,
                                                         learning_rate='optimal',
                                                         loss='hinge', max_iter=1000,
                                                         n_iter_no_change=5, n_jobs=None,
                                                         penalty='l1', power_t=0.5,
                                                         random_state=0, shuffle=True,
                                                         tol=0.001, validation_fraction=0.
                                                         verbose=0, warm_start=False),
                         n_jobs=None)

classifier.fit(X_train, y_train)
predictions = classifier.predict(X_test)
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(precisio

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(precisio

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

```
Accuracy : 0.26955
Hamming loss  0.00266095
Micro-average quality numbers
Precision: 0.6899, Recall: 0.6016, F1-measure: 0.6428
Macro-average quality numbers
Precision: 0.2163, Recall: 0.1745, F1-measure: 0.1720
              precision    recall  f1-score   support

           0       0.98      0.98      0.98     36915
           1       0.36      0.07      0.12       140
           2       0.31      0.17      0.22      4486
           3       0.30      0.37      0.33       121
           4       0.51      0.25      0.33       784
           5       0.28      0.24      0.26        37
           6       0.55      0.58      0.57       220
           7       0.74      0.57      0.64       486
```

8	0.17	0.03	0.05	33
9	0.35	0.45	0.40	189
10	0.29	0.16	0.21	255
11	0.37	0.40	0.38	244
12	0.42	0.38	0.40	272
13	0.32	0.14	0.19	65
14	0.62	0.51	0.56	45
15	0.32	0.15	0.21	158
16	0.20	0.14	0.17	7
17	0.72	0.31	0.43	101
18	0.00	0.00	0.00	82
19	0.36	0.20	0.26	44
20	0.23	0.41	0.30	17
21	0.00	0.00	0.00	51
22	0.32	0.31	0.31	137
23	0.50	0.70	0.59	94
24	0.36	0.30	0.32	125
25	0.20	0.38	0.26	24
26	0.29	0.36	0.32	740
27	0.00	0.00	0.00	18
28	0.29	0.40	0.33	5
29	0.50	0.09	0.15	22
30	0.00	0.00	0.00	6
31	0.36	0.18	0.24	428
32	0.00	0.00	0.00	1
33	0.27	0.19	0.22	16
34	0.13	0.07	0.10	40
35	0.37	0.42	0.39	971
36	0.57	0.50	0.53	8
37	0.00	0.00	0.00	1
38	0.37	0.46	0.41	429
39	0.50	0.29	0.36	7
40	0.25	0.02	0.04	44
41	0.00	0.00	0.00	13
42	0.09	0.33	0.14	3
43	1.00	0.25	0.40	4
44	0.45	0.44	0.45	179
45	0.00	0.00	0.00	5
46	0.31	0.62	0.42	50
47	0.30	0.07	0.12	152
48	0.48	0.57	0.52	323
49	0.77	0.59	0.67	310
50	0.19	0.27	0.22	496
51	0.28	0.20	0.23	175
52	0.23	0.15	0.18	1654
53	0.69	0.31	0.43	35
54	0.00	0.00	0.00	9
55	0.50	0.06	0.11	17
56	0.00	0.00	0.00	104
57	0.00	0.00	0.00	88
58	0.60	0.42	0.49	146
59	0.67	0.67	0.67	12
60	0.00	0.00	0.00	4
61	0.03	0.02	0.02	54
62	0.71	0.59	0.65	37
63	0.54	0.66	0.59	29
64	0.47	0.48	0.47	1150

65	0.07	0.03	0.04	34
66	0.05	0.06	0.05	65
67	0.57	0.60	0.59	910
68	0.62	0.60	0.61	30
69	0.00	0.00	0.00	6
70	0.16	0.13	0.15	129
71	0.27	0.20	0.23	49
72	0.00	0.00	0.00	0
73	0.00	0.00	0.00	5
74	0.00	0.00	0.00	1
75	0.60	0.60	0.60	10
76	0.14	0.08	0.10	50
77	0.29	0.03	0.05	70
78	0.41	0.37	0.39	19
79	0.35	0.17	0.23	40
80	0.21	0.31	0.25	16
81	0.54	0.20	0.29	35
82	0.19	0.07	0.10	185
83	0.00	0.00	0.00	0
84	0.00	0.00	0.00	8
85	0.00	0.00	0.00	2
86	0.55	0.33	0.41	18
87	0.26	0.31	0.29	16
88	0.38	0.51	0.44	863
89	0.32	0.20	0.24	470
90	0.51	0.40	0.45	94
91	0.00	0.00	0.00	4
92	0.23	0.27	0.25	11
93	0.62	0.52	0.56	236
94	0.23	0.22	0.22	647
95	0.04	0.04	0.04	81
96	0.00	0.00	0.00	17
97	0.56	0.50	0.53	18
98	0.50	1.00	0.67	1
99	0.28	0.36	0.32	249
100	0.20	0.10	0.13	188
101	0.88	0.24	0.38	29
102	0.22	0.17	0.19	12
103	0.27	0.33	0.30	878
104	0.26	0.18	0.21	127
105	0.00	0.00	0.00	0
106	0.00	0.00	0.00	2
107	0.38	0.83	0.53	12
108	0.14	0.16	0.15	130
109	0.00	0.00	0.00	2
110	0.00	0.00	0.00	16
111	0.00	0.00	0.00	0
112	0.00	0.00	0.00	1
113	0.24	0.12	0.16	32
114	0.14	0.11	0.13	183
115	0.24	0.30	0.27	134
116	0.29	0.28	0.29	25
117	0.00	0.00	0.00	18
118	0.00	0.00	0.00	0
119	0.00	0.00	0.00	9
120	0.10	0.01	0.02	375
121	1.00	0.33	0.50	6

122	0.19	0.24	0.21	758
123	0.00	0.00	0.00	2
124	0.25	0.50	0.33	2
125	0.00	0.00	0.00	13
126	0.00	0.00	0.00	5
127	0.20	0.02	0.03	63
128	0.00	0.00	0.00	81
129	0.00	0.00	0.00	0
130	0.08	0.11	0.10	18
131	0.00	0.00	0.00	8
132	0.00	0.00	0.00	0
133	0.33	0.33	0.33	3
134	0.79	0.61	0.69	62
135	0.33	0.03	0.05	40
136	0.00	0.00	0.00	29
137	0.36	0.60	0.45	84
138	0.00	0.00	0.00	2
139	0.00	0.00	0.00	23
140	0.67	0.29	0.40	21
141	0.40	0.28	0.33	507
142	0.43	0.39	0.41	334
143	0.57	0.50	0.53	8
144	0.12	0.02	0.03	51
145	0.00	0.00	0.00	14
146	0.00	0.00	0.00	4
147	0.62	0.35	0.45	170
148	0.30	0.21	0.25	136
149	0.75	0.60	0.67	5
150	0.00	0.00	0.00	3
151	1.00	0.03	0.05	957
152	0.17	0.17	0.17	6
153	0.00	0.00	0.00	2
154	0.00	0.00	0.00	3
155	0.17	0.09	0.12	69
156	0.40	0.74	0.52	23
157	0.24	0.10	0.14	97
158	0.62	0.59	0.60	27
159	0.00	0.00	0.00	5
160	0.00	0.00	0.00	86
161	0.00	0.00	0.00	5
162	0.00	0.00	0.00	44
163	1.00	0.55	0.71	22
164	0.64	0.29	0.40	31
165	0.38	0.38	0.38	55
166	1.00	0.50	0.67	2
167	0.17	0.07	0.10	14
168	0.00	0.00	0.00	0
169	0.34	0.48	0.40	207
170	0.06	0.04	0.05	24
171	0.00	0.00	0.00	3
172	0.00	0.00	0.00	19
173	0.00	0.00	0.00	1
174	0.50	0.02	0.04	106
175	0.73	0.85	0.79	317
176	0.11	0.33	0.17	3
177	0.71	0.36	0.48	14
178	0.70	0.25	0.37	63

179	0.43	0.25	0.32	12
180	0.00	0.00	0.00	0
181	0.00	0.00	0.00	13
182	0.00	0.00	0.00	49
183	0.00	0.00	0.00	1
184	0.00	0.00	0.00	32
185	0.10	0.15	0.12	13
186	0.00	0.00	0.00	66
187	0.50	0.55	0.52	11
188	0.06	0.09	0.07	148
189	0.25	0.38	0.30	8
190	0.00	0.00	0.00	0
191	0.00	0.00	0.00	26
192	0.17	1.00	0.29	1
193	0.30	0.26	0.28	636
194	0.52	0.59	0.55	51
195	0.17	0.11	0.13	81
196	0.25	0.04	0.07	23
197	0.00	0.00	0.00	0
198	0.00	0.00	0.00	10
199	0.20	0.08	0.11	102
200	0.57	0.71	0.63	215
201	0.00	0.00	0.00	1
202	0.00	0.00	0.00	9
203	0.18	0.12	0.14	131
204	0.20	0.02	0.03	63
205	0.00	0.00	0.00	7
206	0.11	0.03	0.04	375
207	0.00	0.00	0.00	0
208	0.62	0.56	0.59	9
209	0.07	0.01	0.02	71
210	0.50	0.11	0.18	18
211	0.67	0.67	0.67	3
212	0.14	0.20	0.16	188
213	0.00	0.00	0.00	1
214	0.41	0.39	0.40	205
215	0.00	0.00	0.00	0
216	0.00	0.00	0.00	12
217	0.46	0.51	0.48	398
218	1.00	0.50	0.67	8
219	0.29	0.40	0.33	5
220	0.50	1.00	0.67	1
221	1.00	0.67	0.80	12
222	0.73	0.47	0.57	34
223	0.25	0.00	0.01	217
224	0.00	0.00	0.00	0
225	0.00	0.00	0.00	1
226	0.52	0.27	0.36	48
227	0.12	0.02	0.03	55
228	0.00	0.00	0.00	25
229	0.00	0.00	0.00	15
230	0.00	0.00	0.00	12
231	0.65	0.51	0.57	406
232	0.00	0.00	0.00	63
233	0.00	0.00	0.00	2
234	0.33	0.58	0.42	33
235	0.59	0.54	0.56	586

236	0.00	0.00	0.00	1
237	0.00	0.00	0.00	10
238	0.25	0.03	0.06	60
239	0.07	0.11	0.09	460
240	0.00	0.00	0.00	0
241	0.00	0.00	0.00	1
242	1.00	0.00	0.00	520
243	0.00	0.00	0.00	5
244	0.43	0.07	0.12	42
245	0.25	0.03	0.06	30
246	0.50	0.50	0.50	2
247	0.08	0.07	0.08	27
248	0.00	0.00	0.00	10
249	0.12	0.02	0.03	120
250	0.00	0.00	0.00	28
251	0.00	0.00	0.00	21
252	0.00	0.00	0.00	55
253	0.71	0.43	0.53	365
254	0.00	0.00	0.00	79
255	0.00	0.00	0.00	20
256	0.00	0.00	0.00	6
257	0.12	0.10	0.11	20
258	0.77	0.56	0.65	48
259	0.00	0.00	0.00	11
260	0.00	0.00	0.00	0
261	0.13	0.11	0.12	35
262	0.38	0.67	0.48	9
263	0.41	0.39	0.40	300
264	0.20	0.03	0.06	29
265	0.33	0.45	0.38	11
266	0.19	0.15	0.17	477
267	0.33	0.06	0.11	16
268	0.00	0.00	0.00	42
269	0.00	0.00	0.00	74
270	0.20	0.03	0.05	38
271	0.00	0.00	0.00	0
272	0.00	0.00	0.00	1
273	0.33	1.00	0.50	1
274	0.31	0.42	0.36	12
275	0.00	0.00	0.00	47
276	0.00	0.00	0.00	14
277	0.00	0.00	0.00	2
278	0.17	0.43	0.24	7
279	0.78	0.88	0.82	8
280	0.00	0.00	0.00	0
281	0.38	0.18	0.24	17
282	0.38	0.50	0.43	6
283	0.00	0.00	0.00	2
284	0.14	0.05	0.07	22
285	0.80	0.29	0.42	14
286	0.00	0.00	0.00	0
287	0.50	0.11	0.18	9
288	0.21	0.24	0.22	241
289	0.00	0.00	0.00	6
290	0.00	0.00	0.00	11
291	0.00	0.00	0.00	2
292	0.19	0.15	0.16	143

293	0.00	0.00	0.00	0
294	0.00	0.00	0.00	35
295	0.33	0.04	0.07	98
296	0.30	0.13	0.18	129
297	0.21	0.18	0.19	17
298	0.00	0.00	0.00	0
299	0.00	0.00	0.00	0
300	0.00	0.00	0.00	5
301	0.00	0.00	0.00	31
302	0.00	0.00	0.00	1
303	0.61	0.50	0.55	118
304	0.00	0.00	0.00	0
305	0.93	0.72	0.81	53
306	0.10	0.08	0.09	12
307	0.00	0.00	0.00	5
308	0.00	0.00	0.00	80
309	0.00	0.00	0.00	0
310	0.71	0.08	0.15	60
311	0.00	0.00	0.00	0
312	0.33	0.01	0.02	87
313	0.00	0.00	0.00	1
314	0.00	0.00	0.00	0
315	0.39	0.38	0.38	138
316	0.49	0.39	0.43	93
317	0.00	0.00	0.00	10
318	0.17	0.08	0.11	13
319	0.60	0.27	0.37	44
320	0.00	0.00	0.00	30
321	0.62	0.38	0.47	34
322	0.56	0.24	0.33	21
323	0.00	0.00	0.00	1
324	0.25	0.22	0.23	126
325	0.00	0.00	0.00	25
326	0.00	0.00	0.00	10
327	0.00	0.00	0.00	2
328	0.33	0.20	0.25	5
329	0.71	0.37	0.49	54
330	0.00	0.00	0.00	4
331	0.00	0.00	0.00	0
332	0.00	0.00	0.00	0
333	0.00	0.00	0.00	27
334	0.00	0.00	0.00	6
335	0.26	0.23	0.24	185
336	0.67	0.75	0.71	8
337	0.05	0.25	0.08	4
338	0.42	0.40	0.41	20
339	0.00	0.00	0.00	3
340	0.67	0.38	0.48	16
341	0.00	0.00	0.00	1
342	0.00	0.00	0.00	10
343	0.26	0.27	0.26	229
344	1.00	1.00	1.00	1
345	0.39	0.47	0.42	302
346	0.09	0.02	0.03	48
347	0.00	0.00	0.00	1
348	0.00	0.00	0.00	19
349	0.17	0.17	0.17	6

350	0.49	0.69	0.57	211
351	0.00	0.00	0.00	2
352	0.50	1.00	0.67	2
353	0.36	0.34	0.35	340
354	0.00	0.00	0.00	12
355	0.00	0.00	0.00	0
356	0.00	0.00	0.00	0
357	0.00	0.00	0.00	1
358	0.52	0.27	0.35	97
359	0.50	0.33	0.40	3
360	0.00	0.00	0.00	28
361	0.25	0.25	0.25	4
362	0.25	0.03	0.05	34
363	0.50	0.50	0.50	2
364	0.00	0.00	0.00	0
365	0.35	0.48	0.41	229
366	0.00	0.00	0.00	0
367	0.00	0.00	0.00	0
368	0.00	0.00	0.00	0
369	0.00	0.00	0.00	33
370	0.03	0.14	0.06	7
371	0.00	0.00	0.00	1
372	0.76	0.47	0.58	291
373	0.00	0.00	0.00	0
374	0.00	0.00	0.00	24
375	0.00	0.00	0.00	64
376	0.00	0.00	0.00	16
377	0.00	0.00	0.00	11
378	0.00	0.00	0.00	6
379	0.00	0.00	0.00	0
380	0.00	0.00	0.00	1
381	0.00	0.00	0.00	28
382	0.14	0.03	0.04	40
383	0.00	0.00	0.00	2
384	0.00	0.00	0.00	5
385	0.00	0.00	0.00	0
386	0.41	0.36	0.38	224
387	0.00	0.00	0.00	23
388	0.00	0.00	0.00	0
389	0.00	0.00	0.00	16
390	0.00	0.00	0.00	2
391	0.49	0.43	0.46	51
392	0.31	0.44	0.37	117
393	0.00	0.00	0.00	9
394	0.00	0.00	0.00	19
395	0.00	0.00	0.00	0
396	0.00	0.00	0.00	8
397	0.75	0.08	0.14	39
398	0.25	1.00	0.40	1
399	0.00	0.00	0.00	20
400	0.00	0.00	0.00	6
401	0.14	0.03	0.04	39
402	0.00	0.00	0.00	7
403	0.00	0.00	0.00	9
404	0.87	0.84	0.86	102
405	0.00	0.00	0.00	15
406	0.00	0.00	0.00	0

407	0.00	0.00	0.00	0
408	0.00	0.00	0.00	3
409	0.45	0.28	0.34	269
410	0.00	0.00	0.00	0
411	0.00	0.00	0.00	10
412	0.00	0.00	0.00	13
413	0.00	0.00	0.00	1
414	0.00	0.00	0.00	24
415	0.00	0.00	0.00	5
416	0.18	0.07	0.10	29
417	0.00	0.00	0.00	6
418	0.00	0.00	0.00	49
419	0.00	0.00	0.00	3
420	0.33	0.41	0.37	313
421	0.00	0.00	0.00	73
422	0.57	0.12	0.20	66
423	0.00	0.00	0.00	1
424	0.00	0.00	0.00	0
425	0.00	0.00	0.00	29
426	0.00	0.00	0.00	22
427	0.00	0.00	0.00	2
428	0.00	0.00	0.00	1
429	0.00	0.00	0.00	1
430	0.17	0.25	0.20	4
431	0.17	0.33	0.22	6
432	0.50	0.52	0.51	335
433	0.51	0.36	0.43	107
434	0.08	0.09	0.09	11
435	0.38	0.75	0.50	4
436	0.00	0.00	0.00	0
437	0.10	0.33	0.15	9
438	0.52	0.47	0.50	57
439	0.00	0.00	0.00	14
440	1.00	0.04	0.08	24
441	0.29	0.18	0.22	90
442	0.00	0.00	0.00	2
443	0.50	0.25	0.33	24
444	0.31	0.03	0.06	143
445	0.00	0.00	0.00	21
446	0.00	0.00	0.00	1
447	0.00	0.00	0.00	39
448	1.00	0.12	0.22	8
449	0.22	0.05	0.08	40
450	0.00	0.00	0.00	13
451	0.00	0.00	0.00	42
452	0.00	0.00	0.00	4
453	0.05	0.06	0.06	233
454	0.00	0.00	0.00	0
455	0.00	0.00	0.00	1
456	0.25	1.00	0.40	1
457	0.00	0.00	0.00	5
458	0.00	0.00	0.00	33
459	0.00	0.00	0.00	15
460	0.00	0.00	0.00	82
461	0.00	0.00	0.00	2
462	0.00	0.00	0.00	8
463	0.00	0.00	0.00	0

464	0.47	0.33	0.39	52
465	0.00	0.00	0.00	6
466	0.00	0.00	0.00	2
467	0.20	0.08	0.12	12
468	0.00	0.00	0.00	7
469	0.00	0.00	0.00	9
470	1.00	1.00	1.00	1
471	0.00	0.00	0.00	9
472	0.00	0.00	0.00	0
473	0.53	0.56	0.55	16
474	0.00	0.00	0.00	2
475	0.06	0.02	0.03	47
476	0.25	1.00	0.40	1
477	0.50	0.22	0.31	9
478	0.00	0.00	0.00	0
479	0.22	0.37	0.28	30
480	0.00	0.00	0.00	0
481	0.00	0.00	0.00	3
482	0.28	0.17	0.21	77
483	0.17	0.19	0.18	16
484	0.00	0.00	0.00	175
485	0.00	0.00	0.00	17
486	0.00	0.00	0.00	4
487	0.25	0.01	0.03	68
488	0.00	0.00	0.00	4
489	0.77	0.22	0.34	46
490	0.00	0.00	0.00	0
491	0.00	0.00	0.00	22
492	0.00	0.00	0.00	0
493	0.00	0.00	0.00	0
494	0.67	0.43	0.53	23
495	0.00	0.00	0.00	0
496	0.00	0.00	0.00	0
497	0.50	1.00	0.67	1
498	0.00	0.00	0.00	0
499	1.00	0.12	0.22	8
micro avg	0.69	0.60	0.64	79579
macro avg	0.22	0.17	0.17	79579
weighted avg	0.65	0.60	0.61	79579
samples avg	0.77	0.68	0.67	79579

```
Time taken to run this cell : 0:23:25.945543
```

4.Conclusion

In [11]:

```
#final results
from prettytable import PrettyTable

#using tfidfVectorizer with maximum features

table_1 =PrettyTable()
table_1.field_names = ["Classifier", "Model",'Micro_f1']
table_1.add_row(["OneVsRest", 'LogisticRegression',0.374270748817])

print('\t\t Tfidf Vectorizer with all features ')
print(table_1)
print('\n\n')

#table 2
table_2 = PrettyTable()
table_2.field_names = ["Classifier", "Model",'Precision','Recall','Micro_f1']
table_2.add_row(["OneVsRest", 'LogisticRegression', 0.5687,  0.4093, 0.4760])

print('Count Vectorizer on 0.5M datapoints and with all features')
print(table_2)
print('\n\n')

#table 3
table_3 = PrettyTable()
table_3.field_names = ["Classifier", "Model",'Precision','Recall','Micro_f1']
table_3.add_row(["OneVsRest", 'LogisticRegression', 0.6796,0.6083,0.6419])
table_3.add_row(["OneVsRest", 'LinearSVM',0.6899,0.6016,0.6428])


print('Count Vectorizer on 0.2M datapoints and with top 5000 features')
print(table_3)
print('\n\n')
```

```
                Tfidf Vectorizer with all features
+------------+--------------------+----------------+
| Classifier |       Model        |    Micro_f1    |
+------------+--------------------+----------------+
| OneVsRest  | LogisticRegression | 0.374270748817 |
+------------+--------------------+----------------+


Count Vectorizer on 0.5M datapoints and with all features
+------------+--------------------+-----------+--------+----------+
| Classifier |       Model        | Precision | Recall | Micro_f1 |
+------------+--------------------+-----------+--------+----------+
| OneVsRest  | LogisticRegression |   0.5687  | 0.4093 |  0.476   |
+------------+--------------------+-----------+--------+----------+


Count Vectorizer on 0.2M datapoints and with top 5000 features
```

```
+------------+--------------------+-----------+--------+----------+
| Classifier |       Model        | Precision | Recall | Micro_f1 |
+------------+--------------------+-----------+--------+----------+
| OneVsRest  | LogisticRegression |   0.6796  | 0.6083 |  0.6419  |
| OneVsRest  |     LinearSVM      |   0.6899  | 0.6016 |  0.6428  |
+------------+--------------------+-----------+--------+----------+
```

4.1The Methodology of approaching the business problem

In this case study the probelem that we wanted to solve was how to label the question posed by the user so that the right persson can be able to answer them ,so we posed it as a multilabel classification machine learning problem and the error metric we chose here were 'micro_f1 score','macro_f1 score' and hamming loss where the micro_f1 score was the most important one as the class imbalance can help us a lot in such case where some of the tags or labels are present in most of the questions while some are prsent in very few.

Next important part was how to go about approcahing the propblem,so we decided to featurize the data for text using text featurization techniques like Bag of words and TFIDF.So we went about preprocesing the data by cleaning,removing stopwords and Stemming(snowball stemmer) ,as being a multilabel classification problem which we ought to perform using OneVsRest Classifier,hence we used something like partial coverage where the 5000 top tags i.e tags which appaeared most frequent were able to cover around 99% of the questions .

What we observed was that as most of the text part was in title of the question while the summary part covered code mostly so we give 3 times more weightage to the title text and used 0.5M datapoints for it and again featurized the text data using TFIDF and Bag of words.

The reason we did not use classifier chain here as it works with dense data and the bag of words and tfidf gives sparse data matrix and multilearn is not able to convert the sparse data to dense as it gives memory error

Finally we used Logistic Regression and Linear SVM with One Vs Rest Classifier here and tuned the values of alpha in SGDClassifier with log loss and hinge loss to get the best model,on 0.2M datapoints with top 5000 features

4.2 Final results

We get the best micro f1 score on with linear svm as f1 = 0.6428 and similarly comparable with logistic regression f1 = 0.6419