SO_Tag_Predictor

May 5, 2019

In [1]:
```
import warnings
warnings.filterwarnings("ignore")

import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud

import re
import os
import csv

from sqlalchemy import create_engine # database connection
import sqlite3

import datetime as dt
from datetime import datetime

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.naive_bayes import GaussianNB
from sklearn.linear_model import SGDClassifier
from sklearn.linear_model import LogisticRegression
from sklearn import svm
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.multiclass import OneVsRestClassifier

from sklearn import metrics
```

```
from sklearn.metrics import f1_score,precision_score,recall_score
from sklearn.externals import joblib
from sklearn.model_selection import GridSearchCV,RandomizedSearchCV

from prettytable import PrettyTable
```

1 Stack Overflow: Tag Prediction

1. Business Problem

1.1 Description

Description

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

Problem Statemtent

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

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

1.2 Source / useful links

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

1.3 Real World / Business Objectives and Constraints

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

2. Machine Learning problem

2.1 Data

2.1.1 Data Overview

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

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

Data Field Explaination

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

2.1.2 Example Data point

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

2.2.1 Type of Machine Learning Problem

It is a multi-label classification problem Multi-label Classification: Multilabel classification assigns to each sample a set of target labels. This can be thought as predicting properties of a 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

2.2.2 Performance metric

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

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

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

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

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

https://www.kaggle.com/wiki/MeanFScore http://scikit-learn.org/stable/modules/generated/sklearn.me
Hamming loss : The Hamming loss is the fraction of labels that are incorrectly predicted. https://www.kaggle.com/wiki/HammingLoss

3. Exploratory Data Analysis

3.1 Data Loading and Cleaning

3.1.1 Using Pandas with SQLite to Load the data

In [2]:
```
#Creating db file from csv
#Learn SQL: https://www.w3schools.com/sql/default.asp
if not os.path.isfile('train.db'):
    start = datetime.now()
    disk_engine = create_engine('sqlite:///train.db')
    start = dt.datetime.now()
    chunksize = 180000
    j = 0
    index_start = 1
    for df in pd.read_csv('Train.csv',
                          names=['Id', 'Title', 'Body', 'Tags'],
                          chunksize=chunksize,
                          iterator=True,
                          encoding='utf-8'):
        df.index += index_start
        j+=1
        print('{} rows'.format(j*chunksize))
        df.to_sql('data', disk_engine, if_exists='append')
        index_start = df.index[-1] + 1
    print("Time taken to run this cell :", datetime.now() - start)
```

3.1.2 Counting the number of rows

```
In [3]: if os.path.isfile('train.db'):
            start = datetime.now()
            con = sqlite3.connect('train.db')
            num_rows = pd.read_sql_query("""SELECT count(*) FROM data""", con)
            #Always remember to close the database
            print("Number of rows in the database :","\n",num_rows['count(*)'].values[0])
            con.close()
            print("Time taken to count the number of rows :", datetime.now() - start)
        else:
            print("Please download the train.db file from drive or run the above cell to genara
```

```
Number of rows in the database :
 6034196
Time taken to count the number of rows : 0:00:59.788828
```

3.1.3 Checking for duplicates

```
In [4]: #Learn SQl: https://www.w3schools.com/sql/default.asp
        if os.path.isfile('train.db'):
            start = datetime.now()
            con = sqlite3.connect('train.db')
            df_no_dup = pd.read_sql_query('SELECT Title, Body, Tags, COUNT(*) as cnt_dup FROM d
            con.close()
            print("Time taken to run this cell :", datetime.now() - start)
        else:
            print("Please download the train.db file from drive or run the first to genarate th
```

```
Time taken to run this cell : 0:01:52.934592
```

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

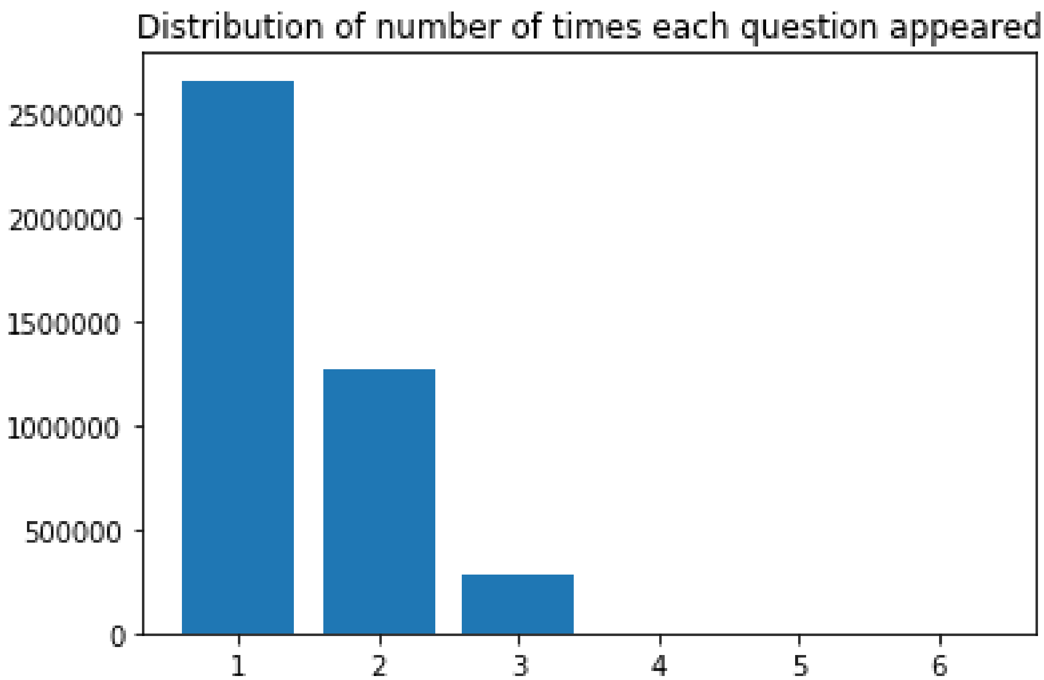
```
Out[5]:                                                Title  \
        0       Implementing Boundary Value Analysis of S...
        1           Dynamic Datagrid Binding in Silverlight?
        2           Dynamic Datagrid Binding in Silverlight?
        3      java.lang.NoClassDefFoundError: javax/serv...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

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

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

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

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

```
In [7]: # number of times each question appeared in our database
        plt.bar(x=df_no_dup.cnt_dup.value_counts().index,height=df_no_dup.cnt_dup.value_counts
        plt.title("Distribution of number of times each question appeared")
        plt.show()
```




```
In [8]: # sample cases that has no tags linked to them
        df_no_dup[df_no_dup["Tags"].isnull()]
```

```
Out[8]:                                               Title  \
        777547                          Do we really need NULL?
        962680   Find all values that are not null and not in a...
        1126558                                 Handle NullObjects
```

```
1256102                            How do Germans call null
2430668  Page cannot be null. Please ensure that this o...
3329908       What is the difference between NULL and "0"?
3551595         a bit of difference between null and space

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

In [9]:
```python
df_no_dup['Tags'] = df_no_dup.Tags.apply(lambda x: x if not pd.isnull(x) else '')
```

In [10]:
```python
df_no_dup[df_no_dup["Tags"].isnull()]
```

Out[10]:
```
Empty DataFrame
Columns: [Title, Body, Tags, cnt_dup]
Index: []
```

In [11]:
```python
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 : 0:00:02.371912
```

Out[11]:
```
                                                Title  \
0       Implementing Boundary Value Analysis of S...
1           Dynamic Datagrid Binding in Silverlight?
2           Dynamic Datagrid Binding in Silverlight?
3      java.lang.NoClassDefFoundError: javax/serv...
4      java.sql.SQLException:[Microsoft][ODBC Dri...

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

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

```
3                              jsp jstl        1           2
4                             java jdbc        2           2
```

```
In [12]: #Creating a new database with no duplicates
         if not os.path.isfile('train_no_dup.db'):
             disk_dup = create_engine("sqlite:///train_no_dup.db")
             no_dup = pd.DataFrame(df_no_dup, columns=['Title', 'Body', 'Tags'])
             no_dup.to_sql('no_dup_train',disk_dup)
```

```
In [13]: #This method seems more appropriate to work with this much data.
         #creating the connection with database file.
         if os.path.isfile('train_no_dup.db'):
             start = datetime.now()
             con = sqlite3.connect('train_no_dup.db')
             tag_data = pd.read_sql_query("""SELECT Tags FROM no_dup_train""", con)
             #Always remember to close the database
             con.close()

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

```
Time taken to run this cell : 0:00:44.251206
```

3.2 Analysis of Tags

3.2.1 Total number of unique tags

```
In [14]: # Importing & Initializing the "CountVectorizer" object, which
         #is scikit-learn's bag of words tool.

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

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

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

In [16]:
```python
#'get_feature_name()' gives us the vocabulary.
tags = vectorizer.get_feature_names()
#Lets look at the tags we have.
print("Some of the tags we have :", tags[:10])
```

```
Some of the tags we have : ['.a', '.app', '.asp.net-mvc', '.aspxauth', '.bash-profile', '.clas
```

3.2.3 Number of times a tag appeared

In [17]:
```python
# https://stackoverflow.com/questions/15115765/how-to-access-sparse-matrix-elements
#Lets now store the document term matrix in a dictionary.
freqs = tag_dtm.sum(axis=0).A1
result = dict(zip(tags, freqs))
```

In [18]:
```python
#Saving this dictionary to csv files.
if not os.path.isfile('tag_counts_dict_dtm.csv'):
    with open('tag_counts_dict_dtm.csv', 'w') as csv_file:
        writer = csv.writer(csv_file)
        for key, value in result.items():
            writer.writerow([key, value])
tag_df = pd.read_csv("tag_counts_dict_dtm.csv", names=['Tags', 'Counts'])
tag_df.head()
```

Out[18]:

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

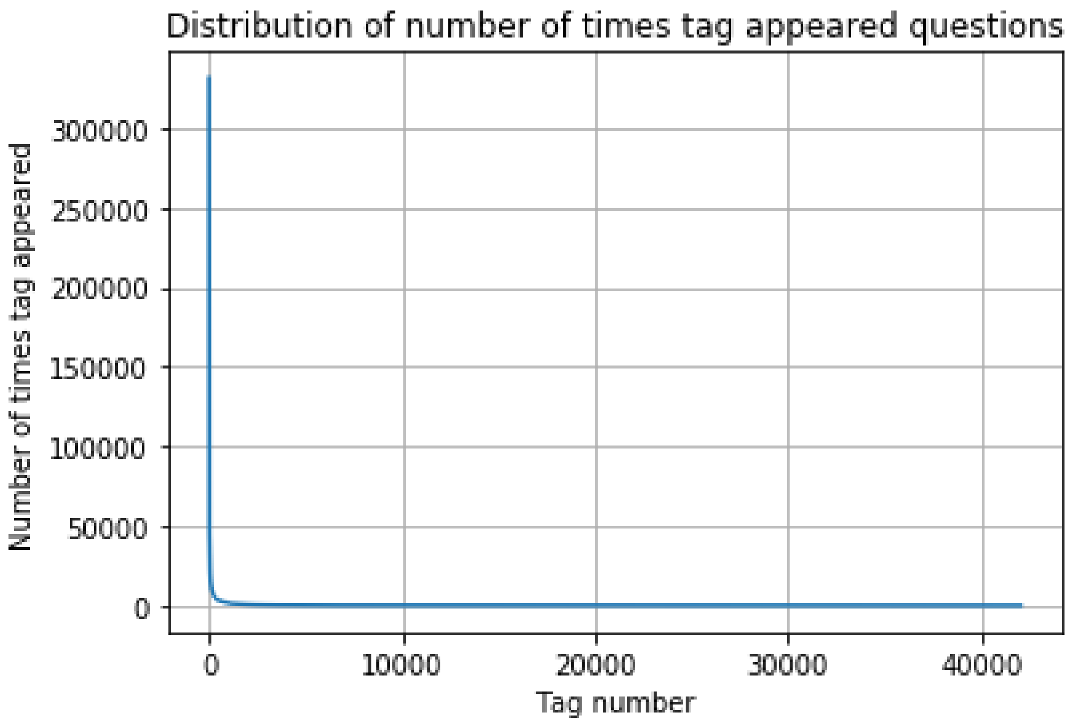
In [19]:
```python
# top 10 used tags (these words should all be fairly large in the wordcloud)
tag_df.sort_values(by=['Counts'],ascending=False).head(n=10)
```

Out[19]:

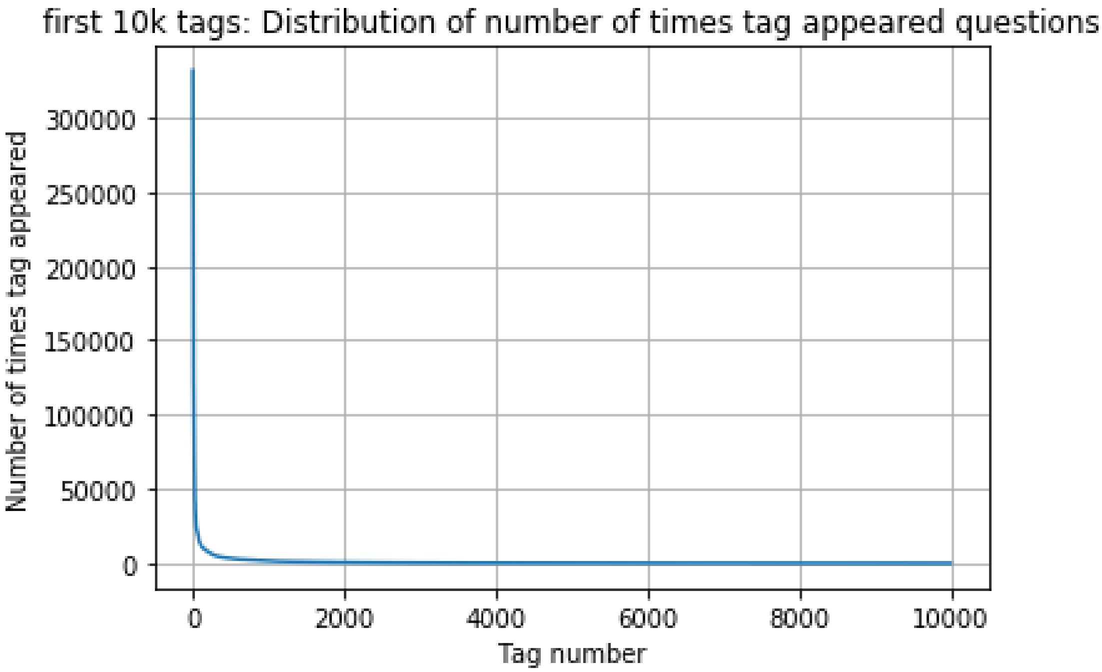
	Tags	Counts
4337	c#	331505
18069	java	299414
27249	php	284103
18157	javascript	265423
1234	android	235436
18608	jquery	221533
4346	c++	143935
29101	python	134137
17643	iphone	128681
2215	asp.net	125651

In [20]:
```python
tag_df_sorted = tag_df.sort_values(by=['Counts'], ascending=False)
tag_counts = tag_df_sorted['Counts'].values
```

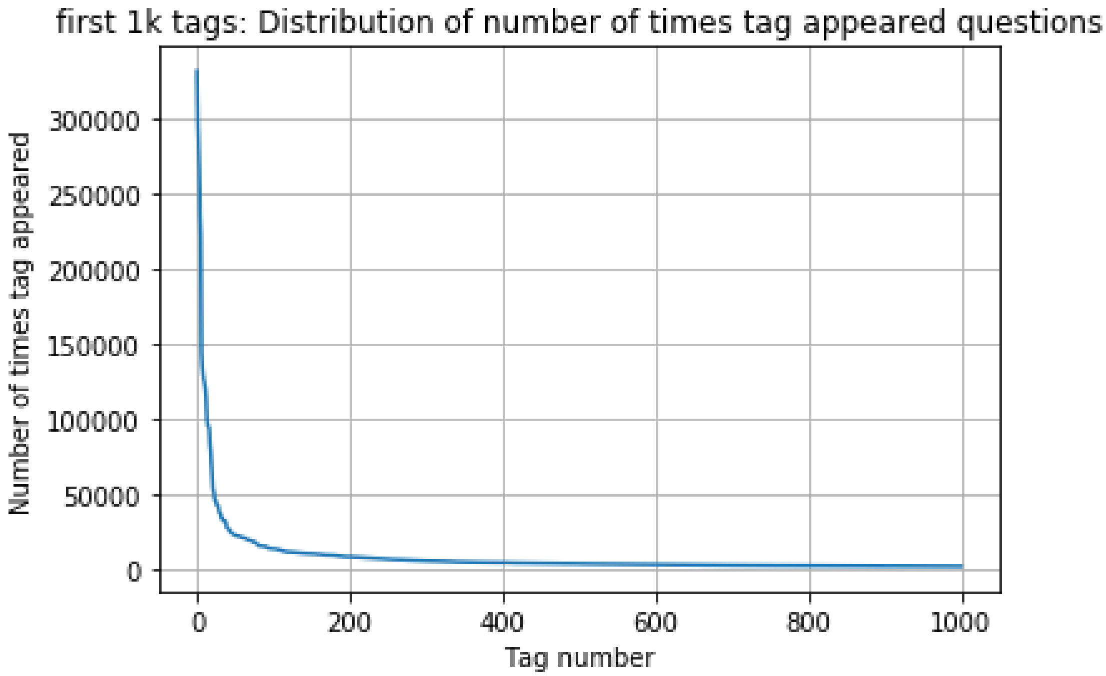
In [21]:
```
plt.plot(tag_counts)
plt.title("Distribution of number of times tag appeared questions")
plt.grid()
plt.xlabel("Tag number")
plt.ylabel("Number of times tag appeared")
plt.show()
```



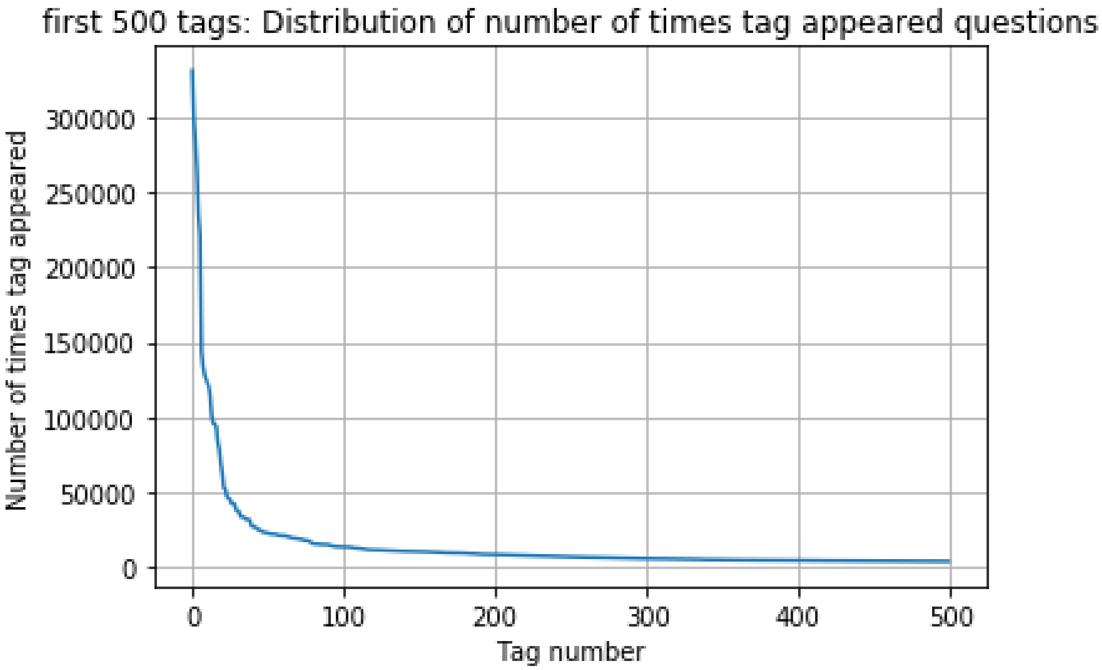
In [22]:
```
plt.plot(tag_counts[0:10000])
plt.title('first 10k tags: Distribution of number of times tag appeared questions')
plt.grid()
plt.xlabel("Tag number")
plt.ylabel("Number of times tag appeared")
plt.show()
```



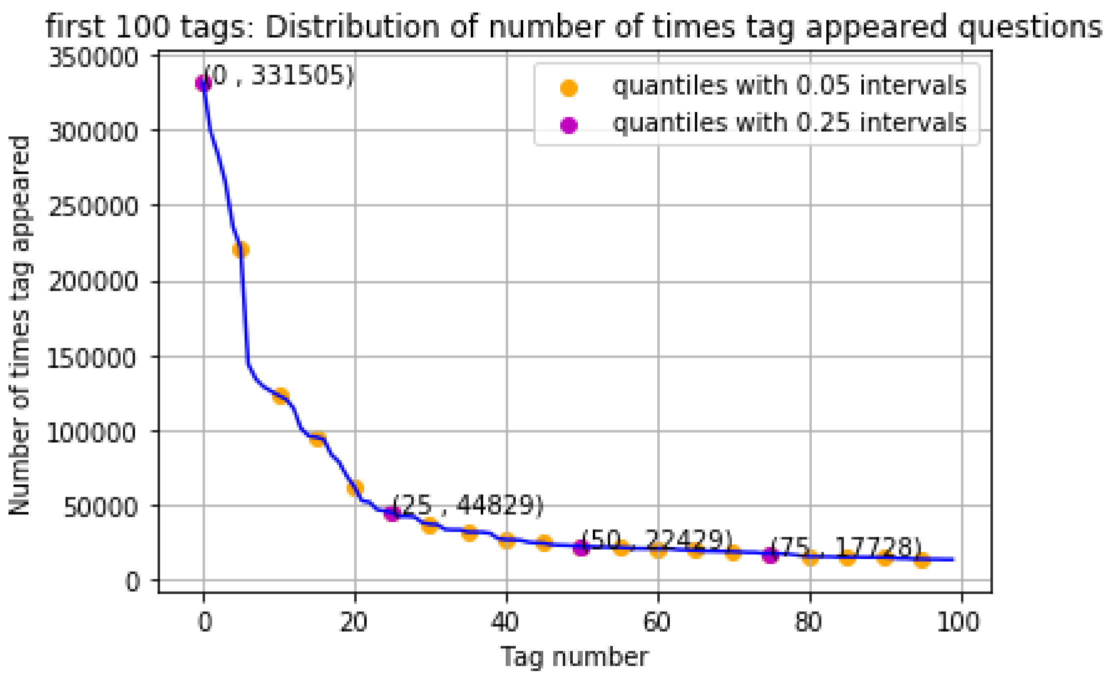
In [23]:
```
plt.plot(tag_counts[0:1000])
plt.title('first 1k tags: Distribution of number of times tag appeared questions')
plt.grid()
plt.xlabel("Tag number")
plt.ylabel("Number of times tag appeared")
plt.show()
```



In [24]:
```
plt.plot(tag_counts[0:500])
plt.title('first 500 tags: Distribution of number of times tag appeared questions')
plt.grid()
plt.xlabel("Tag number")
plt.ylabel("Number of times tag appeared")
plt.show()
```




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

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

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




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

In [26]:
```
# Store tags greater than 10K in one list
lst_tags_gt_10k = tag_df[tag_df.Counts>10000].Tags
#Print the length of the list
print ('{} Tags are used more than 10000 times'.format(len(lst_tags_gt_10k)))
# Store tags greater than 100K in one list
lst_tags_gt_100k = tag_df[tag_df.Counts>100000].Tags
#Print the length of the list.
print ('{} Tags are used more than 100000 times'.format(len(lst_tags_gt_100k)))
```

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

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

3.2.4 Tags Per Question

In [27]:
```
#Storing the count of tag in each question in list 'tag_count'
tag_quest_count = tag_dtm.sum(axis=1).tolist()
#Converting each value in the 'tag_quest_count' to integer.
```

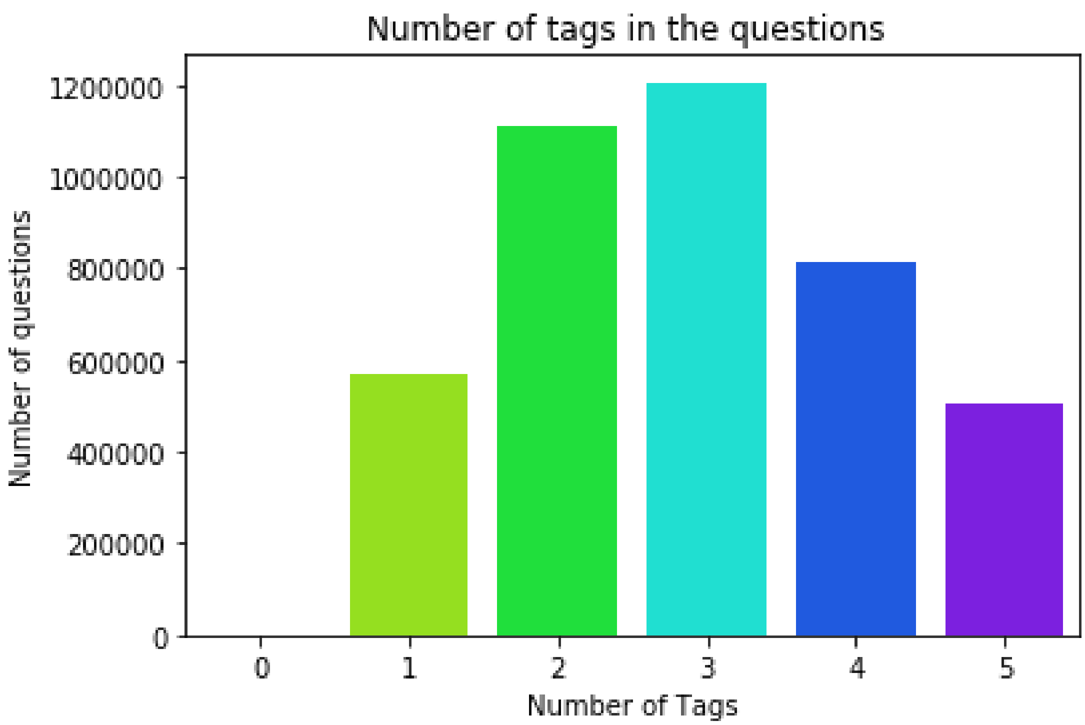
```
tag_quest_count=[int(j) for i in tag_quest_count for j in i]
print ('We have total {} datapoints.'.format(len(tag_quest_count)))

print(tag_quest_count[:5])
```

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

```
In [28]: print( "Maximum number of tags per question: %d"%max(tag_quest_count))
         print( "Minimum number of tags per question: %d"%min(tag_quest_count))
         print( "Avg. number of tags per question: %f"% ((sum(tag_quest_count)*1.0)/len(tag_que
```

```
Maximum number of tags per question: 5
Minimum number of tags per question: 0
Avg. number of tags per question: 2.899438
```

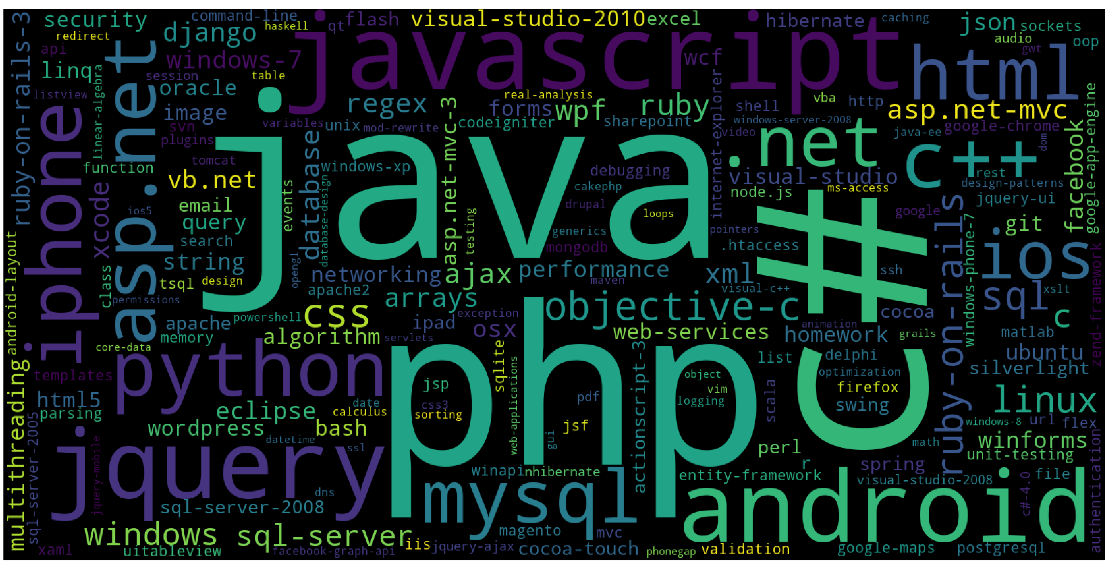
```
In [29]: sns.countplot(tag_quest_count, palette='gist_rainbow')
         plt.title("Number of tags in the questions ")
         plt.xlabel("Number of Tags")
         plt.ylabel("Number of questions")
         plt.show()
```



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

3.2.5 Most Frequent Tags

In [30]:
```python
# Ploting word cloud
start = datetime.now()

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

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



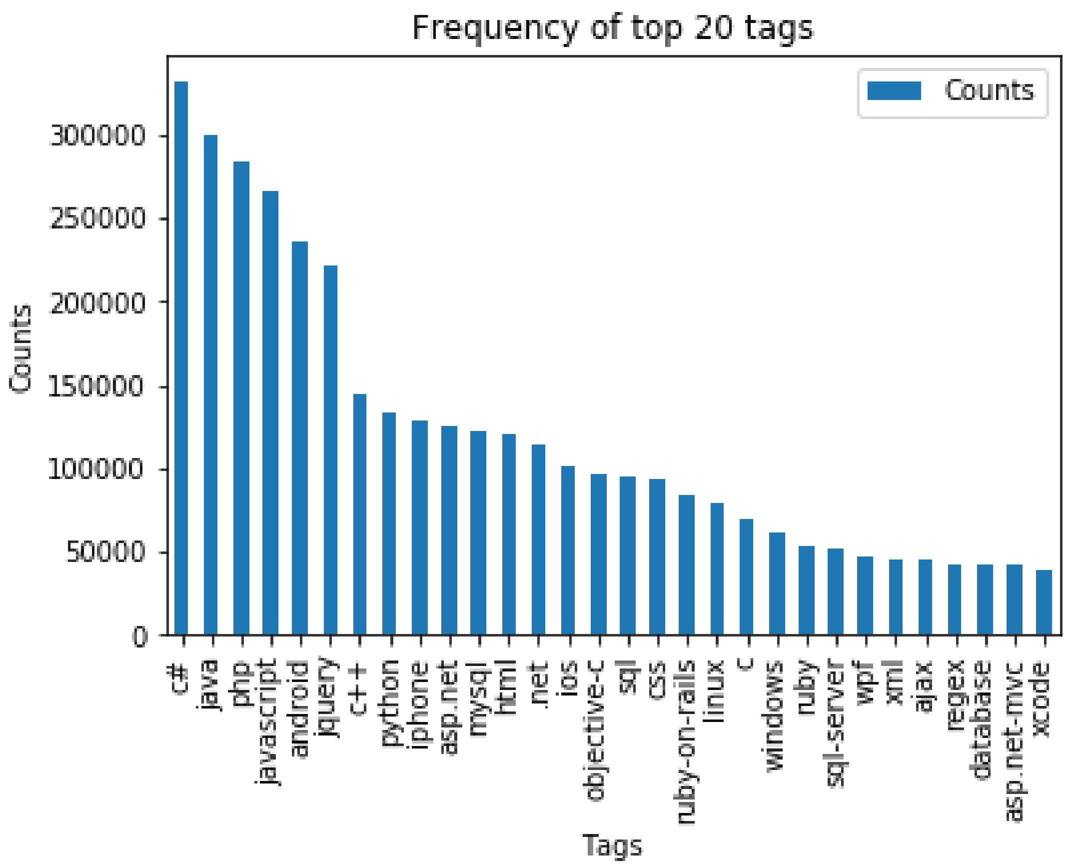

```
Time taken to run this cell : 0:00:03.328469
```

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 [31]: plt.figure(figsize=(10,10))
         i=np.arange(30)
         tag_df_sorted.head(30).plot(kind='bar')
         plt.title('Frequency of top 20 tags')
         plt.xticks(i, tag_df_sorted['Tags'])
         plt.xlabel('Tags')
         plt.ylabel('Counts')
         plt.show()
```

```
<Figure size 720x720 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

Sample 1M data points

Separate out code-snippets from Body

Remove Spcial characters from Question title and description (not in code)

Remove stop words (Except 'C')

Remove HTML Tags
Convert all the characters into small letters
Use SnowballStemmer to stem the words

In [32]:
```
# het the list of stop words
stop = list()
with open('english','r') as stopwords:
    stopwords = stopwords.readlines()
for word in stopwords:
    stop.append(word[:-1])
stop;
```

In [33]:
```
def striphtml(data):
    cleanr = re.compile('<.*?>')
    cleantext = re.sub(cleanr, ' ', str(data))
    return cleantext
stop_words = set(stop)
stemmer = SnowballStemmer("english")
```

In [34]:
```
#http://www.sqlitetutorial.net/sqlite-python/create-tables/
def create_connection(db_file):
    """ create a database connection to the SQLite database
        specified by db_file
    :param db_file: database file
    :return: Connection object or None
    """
    try:
        conn = sqlite3.connect(db_file)
        return conn
    except Error as e:
        print(e)

    return None

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

def checkTableExists(dbcon):
    cursr = dbcon.cursor()
```

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

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

sql_create_table = """CREATE TABLE IF NOT EXISTS QuestionsProcessed (question text NO
create_database_table("Processed.db", sql_create_table)
```

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

In [35]:
```
# http://www.sqlitetutorial.net/sqlite-delete/
# https://stackoverflow.com/questions/2279706/select-random-row-from-a-sqlite-table
start = datetime.now()
read_db = 'train_no_dup.db'
write_db = 'Processed.db'
if os.path.isfile(read_db):
    conn_r = create_connection(read_db)
    if conn_r is not None:
        reader =conn_r.cursor()
        reader.execute("SELECT Title, Body, Tags From no_dup_train ORDER BY RANDOM() 

if os.path.isfile(write_db):
    conn_w = create_connection(write_db)
    if conn_w is not None:
        tables = checkTableExists(conn_w)
        writer =conn_w.cursor()
        if tables != 0:
            writer.execute("DELETE FROM QuestionsProcessed WHERE 1")
            print("Cleared All the rows")
print("Time taken to run this cell :", datetime.now() - start)
```

```
Tables in the databse:
QuestionsProcessed
Cleared All the rows
Time taken to run this cell : 0:04:39.229391
```

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

In [36]:
```
#http://www.bernzilla.com/2008/05/13/selecting-a-random-row-from-an-sqlite-table/

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

    is_code = 0

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

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

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

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

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

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

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

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

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

print( "Avg. length of questions(Title+Body) before processing: %d"%no_dup_avg_len_pr
```

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

    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
Avg. length of questions(Title+Body) before processing: 1170
Avg. length of questions(Title+Body) after processing: 327
Percent of questions containing code: 57
Time taken to run this cell : 0:11:10.173215
```

```
In [37]: # 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 [38]: 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
=============================================================================================
('izpack access use screen reader curious way set izpack instal work screen reader access featu
---------------------------------------------------------------------------------------------
('want pdf pictur smaller one page possibl duplic crop includ pdf document creat file use appea
---------------------------------------------------------------------------------------------
('html video fullscreen stream hls bit problem stream hls use video tag origin use flowplay fal
---------------------------------------------------------------------------------------------
('view base nstableview view control sure thing right problem view base nstableview use bind a
---------------------------------------------------------------------------------------------
('start learn sap work ms develop work provid bridg product ms technolog sap use ms space seem
---------------------------------------------------------------------------------------------
('anyon use googl analyt googl avoid count owner websit visitor want count visitor everi time t
---------------------------------------------------------------------------------------------
```

```
('programm jdbc driver mock tri test legaci java applic without abil factor code moment need u
------------------------------------------------------------------------------------------------
('combin differ jqueri function tri combin two jqueri function without success tri follow hash
------------------------------------------------------------------------------------------------
('linq select contit condit pseudo code select tabl date lt today visibl fals linq',)
------------------------------------------------------------------------------------------------
```

In [39]:
```python
#Taking 100 000k entries to a dataframe.
write_db = 'Processed.db'
if os.path.isfile(write_db):
    conn_r = create_connection(write_db)
    if conn_r is not None:
        preprocessed_data = pd.read_sql_query("""SELECT question, Tags FROM Questions
conn_r.commit()
conn_r.close()
```

In [40]:
```python
preprocessed_data.head()
```

Out[40]:
```
                                            question  \
0  quicktim qtsession open fail packag use osx ja...
1  izpack access use screen reader curious way se...
2  want pdf pictur smaller one page possibl dupli...
3  html video fullscreen stream hls bit problem s...
4  view base nstableview view control sure thing ...

                                       tags
0                        java osx quicktime
1             java swing accessibility izpack
2                   pdf margins pstricks crop
3  android html5 video-streaming html5-video
4            cocoa nstableview cocoa-bindings
```

In [41]:
```python
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 : 499999
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 [42]: # 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 [43]: multilabel_y.getrow(3).toarray().shape
```

```
Out[43]: (1, 30681)
```

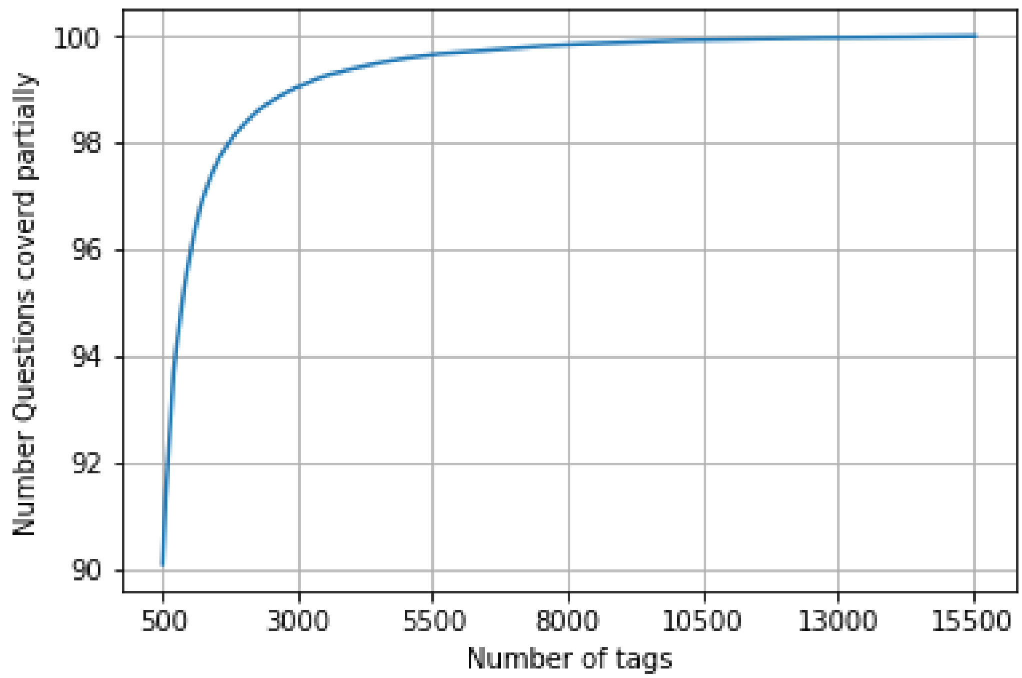
__ We will sample the number of tags instead considering all of them (due to limitation of computing power) __

```
In [44]: def tags_to_choose(n):
             t = multilabel_y.sum(axis=0).tolist()[0]
             sorted_tags_i = sorted(range(len(t)), key=lambda i: t[i], reverse=True)
             multilabel_yn=multilabel_y[:,sorted_tags_i[:n]]
             return multilabel_yn

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

```
In [45]: questions_explained = []
         total_tags=multilabel_y.shape[1]
         total_qs=preprocessed_data.shape[0]
         for i in range(500, total_tags, 100):
             questions_explained.append(np.round(((total_qs-questions_explained_fn(i))/total_q
```

```
In [46]: fig, ax = plt.subplots()
         ax.plot(questions_explained)
         xlabel = list(500+np.array(range(-50,450,50))*50)
         ax.set_xticklabels(xlabel)
         plt.xlabel("Number of tags")
         plt.ylabel("Number Questions coverd partially")
```

```
plt.grid()
plt.show()
# you can choose any number of tags based on your computing power, minimun is 50(it c
print("with ",5500,"tags we are covering ",questions_explained[50],"% of questions")
```




```
with  5500 tags we are covering  99.044 % of questions
```

```
In [47]: multilabel_yx = tags_to_choose(500)
         print("number of questions that are not covered :", questions_explained_fn(500),"out o
```

```
number of questions that are not covered : 49537 out of  499999
```

```
In [48]: print("Number of tags in sample :", multilabel_y.shape[1])
         print("number of tags taken :", multilabel_yx.shape[1],"(",(multilabel_yx.shape[1]/mul
```

```
Number of tags in sample : 30681
number of tags taken : 500 ( 1.6296730875786318 %)
```

__ We consider top 15% tags which covers 99% of the questions __
4.2 Split the data into test and train (80:20)

```
In [49]: total_size=preprocessed_data.shape[0]
         train_size=int(0.80*total_size)

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

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

In [50]: 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 : (399999, 500)
Number of data points in test data : (100000, 500)
```

4.3 Featurizing data

```
In [51]: start = datetime.now()
         vectorizer = TfidfVectorizer(min_df=0.00009, max_features=200000, smooth_idf=True, no
                                      tokenizer = lambda x: x.split(), sublinear_tf=False, ngra
         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 : 0:03:09.250528
```

```
In [52]: print("Dimensions of train data X:",x_train_multilabel.shape, "Y :",y_train.shape)
         print("Dimensions of test data X:",x_test_multilabel.shape,"Y:",y_test.shape)
```

```
Dimensions of train data X: (399999, 89590) Y : (399999, 500)
Dimensions of test data X: (100000, 89590) Y: (100000, 500)
```

```
In [53]: #https://www.analyticsvidhya.com/blog/2017/08/introduction-to-multi-label-classificat
         #https://stats.stackexchange.com/questions/117796/scikit-multi-label-classification
         #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[53]: "\nfrom skmultilearn.adapt import MLkNN\nclassifier = MLkNN(k=21)\n\n# train\nclassifi

4.4 Applying Logistic Regression with OneVsRest Classifier

In [54]:
```
# this will be taking so much time try not to run it, download the lr_with_equal_weig
# This takes about 6-7 hours to run.
start = datetime.now()

if os.path.isfile('lr_with_equal_weight.pkl') == True:
    classifier = joblib.load('lr_with_equal_weight.pkl')
if os.path.isfile('lr_with_equal_weight.pkl') == False:
    classifier = OneVsRestClassifier(SGDClassifier(loss='log', alpha=0.00001, penalty=
    classifier.fit(x_train_multilabel, y_train)
    joblib.dump(classifier, 'lr_with_equal_weight.pkl')

predictions = classifier.predict(x_test_multilabel)

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

```
Time taken to run this cell : 0:08:29.355413
```

In [55]:
```
print("accuracy :",metrics.accuracy_score(y_test,predictions))
print("macro f1 score :",metrics.f1_score(y_test, predictions, average = 'macro'))
print("micro f1 score :",metrics.f1_score(y_test, predictions, average = 'micro'))
print("hamming loss :",metrics.hamming_loss(y_test,predictions))
print("Precision recall report :\n",metrics.classification_report(y_test, predictions)
```

```
accuracy : 0.22175
macro f1 score : 0.31439562056331466
micro f1 score : 0.45236447781848704
hamming loss : 0.00289674
Precision recall report :
              precision    recall  f1-score   support

          0       0.62      0.23      0.34      7969
          1       0.78      0.44      0.56      7085
          2       0.82      0.54      0.65      6705
```

3	0.75	0.43	0.55	6286
4	0.94	0.76	0.84	5506
5	0.86	0.66	0.74	5222
6	0.71	0.32	0.44	3399
7	0.88	0.61	0.72	3200
8	0.68	0.38	0.49	3061
9	0.79	0.44	0.57	2905
10	0.83	0.61	0.70	2866
11	0.52	0.17	0.26	2796
12	0.54	0.10	0.17	2753
13	0.61	0.26	0.36	2420
14	0.62	0.21	0.31	2297
15	0.54	0.26	0.35	2273
16	0.78	0.52	0.63	2243
17	0.78	0.54	0.64	1915
18	0.66	0.27	0.38	1878
19	0.63	0.18	0.28	1630
20	0.34	0.07	0.11	1444
21	0.76	0.37	0.50	1237
22	0.55	0.26	0.36	1243
23	0.87	0.62	0.73	1198
24	0.70	0.44	0.54	1107
25	0.66	0.39	0.49	1052
26	0.88	0.63	0.74	1031
27	0.40	0.08	0.13	1026
28	0.62	0.34	0.44	1012
29	0.57	0.16	0.25	905
30	0.91	0.75	0.82	849
31	0.56	0.27	0.36	866
32	0.49	0.16	0.24	784
33	0.75	0.31	0.44	819
34	0.62	0.24	0.35	775
35	0.77	0.53	0.63	746
36	0.75	0.66	0.70	729
37	0.33	0.13	0.18	711
38	0.73	0.57	0.64	710
39	0.40	0.12	0.18	694
40	0.62	0.40	0.49	640
41	0.40	0.11	0.17	649
42	0.72	0.27	0.39	658
43	0.65	0.32	0.43	582
44	0.39	0.12	0.18	620
45	0.38	0.10	0.16	597
46	0.23	0.05	0.08	568
47	0.43	0.13	0.20	572
48	0.53	0.15	0.23	575
49	0.44	0.08	0.14	514
50	0.39	0.11	0.17	567

51	0.91	0.71	0.80	534
52	0.52	0.03	0.05	529
53	0.78	0.43	0.56	546
54	0.66	0.39	0.49	525
55	0.50	0.14	0.22	535
56	0.22	0.03	0.05	531
57	0.78	0.44	0.56	518
58	0.64	0.37	0.47	507
59	0.43	0.15	0.23	491
60	0.89	0.75	0.82	489
61	0.77	0.45	0.57	530
62	0.91	0.57	0.70	498
63	0.27	0.04	0.07	522
64	0.71	0.27	0.39	466
65	0.83	0.22	0.35	496
66	0.57	0.17	0.26	463
67	0.39	0.11	0.17	495
68	0.79	0.50	0.61	481
69	0.29	0.00	0.01	428
70	0.74	0.19	0.30	422
71	0.82	0.47	0.60	457
72	0.80	0.35	0.49	450
73	0.58	0.31	0.40	426
74	0.58	0.38	0.46	419
75	0.87	0.64	0.73	396
76	0.48	0.27	0.35	415
77	0.69	0.44	0.53	409
78	0.25	0.02	0.04	420
79	0.52	0.29	0.37	339
80	0.70	0.33	0.45	379
81	0.36	0.12	0.18	365
82	0.82	0.45	0.58	349
83	0.93	0.55	0.69	400
84	0.33	0.09	0.14	329
85	0.70	0.42	0.52	354
86	0.95	0.51	0.67	348
87	0.52	0.25	0.34	357
88	0.77	0.44	0.56	349
89	0.82	0.59	0.69	334
90	0.67	0.48	0.56	352
91	0.63	0.17	0.27	361
92	0.72	0.41	0.52	355
93	0.55	0.09	0.15	299
94	0.47	0.05	0.10	300
95	0.68	0.44	0.53	316
96	0.93	0.55	0.69	319
97	0.94	0.69	0.79	335
98	0.54	0.08	0.14	318

99	0.21	0.03	0.05	304
100	0.41	0.21	0.28	291
101	0.51	0.18	0.27	311
102	0.88	0.72	0.79	296
103	0.85	0.51	0.64	318
104	0.90	0.62	0.73	304
105	0.16	0.01	0.02	318
106	0.61	0.14	0.23	302
107	0.15	0.01	0.02	311
108	0.66	0.32	0.43	321
109	0.32	0.10	0.15	282
110	0.51	0.22	0.31	302
111	0.64	0.29	0.40	275
112	0.61	0.37	0.46	279
113	0.35	0.12	0.18	277
114	0.71	0.42	0.53	266
115	0.94	0.72	0.82	288
116	0.80	0.61	0.69	281
117	0.59	0.08	0.15	274
118	0.55	0.21	0.30	291
119	0.49	0.19	0.27	263
120	0.92	0.63	0.75	248
121	0.69	0.40	0.51	268
122	0.51	0.22	0.31	284
123	0.55	0.23	0.32	263
124	0.37	0.06	0.10	278
125	0.62	0.37	0.46	259
126	0.68	0.36	0.47	253
127	0.94	0.66	0.78	265
128	0.40	0.13	0.19	272
129	0.19	0.04	0.07	277
130	0.21	0.06	0.09	252
131	0.42	0.19	0.26	252
132	0.82	0.51	0.63	269
133	0.70	0.39	0.50	262
134	0.73	0.46	0.56	251
135	0.34	0.04	0.08	255
136	0.53	0.25	0.34	238
137	0.54	0.35	0.43	257
138	0.54	0.30	0.39	266
139	0.22	0.03	0.05	238
140	0.36	0.11	0.17	245
141	0.27	0.06	0.10	239
142	0.89	0.65	0.75	251
143	0.00	0.00	0.00	259
144	0.51	0.28	0.36	240
145	0.59	0.23	0.33	245
146	0.27	0.04	0.07	259

147	0.14	0.04	0.07	234
148	0.91	0.67	0.77	230
149	0.49	0.29	0.36	251
150	0.33	0.10	0.16	246
151	0.57	0.11	0.19	232
152	0.45	0.14	0.21	243
153	0.35	0.12	0.17	251
154	0.38	0.05	0.08	243
155	0.61	0.16	0.26	240
156	0.55	0.10	0.16	231
157	0.39	0.05	0.08	238
158	0.82	0.43	0.57	234
159	0.41	0.22	0.29	241
160	0.50	0.08	0.14	235
161	0.51	0.28	0.36	236
162	0.92	0.70	0.79	224
163	0.40	0.16	0.23	218
164	0.45	0.04	0.07	235
165	0.24	0.04	0.07	232
166	0.48	0.14	0.22	208
167	0.71	0.31	0.43	217
168	0.20	0.03	0.05	207
169	0.43	0.21	0.28	212
170	0.68	0.43	0.53	233
171	0.35	0.08	0.13	213
172	0.96	0.68	0.79	222
173	0.88	0.62	0.73	210
174	0.56	0.15	0.24	220
175	0.74	0.46	0.57	213
176	0.38	0.09	0.15	217
177	0.73	0.37	0.49	218
178	0.39	0.11	0.17	200
179	0.89	0.65	0.75	214
180	0.31	0.05	0.08	196
181	0.79	0.27	0.40	195
182	0.49	0.15	0.23	194
183	0.93	0.64	0.76	212
184	0.26	0.03	0.06	194
185	0.97	0.72	0.83	203
186	0.65	0.38	0.48	196
187	0.79	0.48	0.59	208
188	0.14	0.01	0.02	198
189	0.12	0.03	0.04	189
190	0.81	0.44	0.57	196
191	0.39	0.07	0.12	206
192	0.70	0.30	0.42	189
193	0.23	0.03	0.06	215
194	0.80	0.55	0.65	201

195	0.38	0.10	0.16	212
196	0.19	0.02	0.04	183
197	0.92	0.54	0.68	217
198	0.68	0.24	0.35	192
199	0.97	0.51	0.67	202
200	0.62	0.04	0.08	198
201	0.81	0.42	0.55	178
202	0.73	0.27	0.39	193
203	0.25	0.04	0.07	175
204	0.76	0.41	0.53	178
205	0.39	0.16	0.23	200
206	0.78	0.38	0.51	178
207	0.18	0.03	0.06	179
208	0.45	0.13	0.20	190
209	0.26	0.08	0.12	182
210	0.62	0.19	0.29	162
211	0.52	0.32	0.39	177
212	0.30	0.02	0.03	189
213	0.43	0.14	0.21	189
214	0.00	0.00	0.00	189
215	0.68	0.28	0.39	162
216	0.74	0.39	0.51	173
217	0.43	0.10	0.16	199
218	0.93	0.80	0.86	168
219	0.23	0.04	0.06	186
220	0.76	0.42	0.54	170
221	0.33	0.03	0.06	184
222	0.50	0.30	0.38	175
223	0.33	0.07	0.12	157
224	0.67	0.43	0.52	174
225	0.28	0.10	0.14	175
226	0.55	0.24	0.34	161
227	0.53	0.14	0.22	185
228	0.93	0.73	0.82	175
229	0.95	0.70	0.81	178
230	0.59	0.32	0.41	157
231	0.17	0.02	0.03	175
232	0.35	0.04	0.08	158
233	0.52	0.32	0.40	157
234	0.68	0.38	0.49	182
235	0.42	0.14	0.21	158
236	0.27	0.07	0.12	150
237	0.09	0.02	0.03	151
238	0.84	0.43	0.57	171
239	0.43	0.20	0.27	148
240	0.74	0.41	0.52	158
241	0.50	0.11	0.18	157
242	0.63	0.38	0.47	152

243	0.32	0.04	0.06	166
244	0.34	0.13	0.18	159
245	0.29	0.11	0.16	152
246	0.42	0.05	0.09	162
247	0.60	0.34	0.43	150
248	0.84	0.55	0.66	158
249	0.81	0.56	0.66	144
250	0.59	0.26	0.36	156
251	0.78	0.51	0.61	184
252	0.78	0.22	0.34	163
253	0.68	0.09	0.16	160
254	0.56	0.28	0.37	143
255	0.36	0.03	0.05	145
256	0.60	0.25	0.35	146
257	0.18	0.01	0.02	164
258	0.61	0.19	0.29	157
259	0.80	0.47	0.59	158
260	0.64	0.26	0.36	145
261	0.26	0.07	0.11	133
262	0.57	0.40	0.47	138
263	0.78	0.33	0.47	154
264	0.50	0.01	0.02	160
265	0.39	0.11	0.18	150
266	0.42	0.06	0.10	145
267	0.00	0.00	0.00	144
268	0.52	0.10	0.17	138
269	0.00	0.00	0.00	141
270	0.66	0.44	0.53	139
271	0.20	0.01	0.03	146
272	0.64	0.24	0.35	146
273	0.17	0.04	0.06	127
274	0.95	0.69	0.80	157
275	0.83	0.51	0.64	134
276	0.66	0.34	0.45	145
277	0.00	0.00	0.00	143
278	0.31	0.10	0.15	138
279	0.39	0.16	0.22	121
280	0.10	0.01	0.01	136
281	0.57	0.25	0.35	136
282	0.78	0.43	0.55	162
283	0.12	0.01	0.03	136
284	0.87	0.64	0.74	130
285	0.83	0.58	0.68	140
286	0.53	0.28	0.37	128
287	0.00	0.00	0.00	142
288	0.57	0.24	0.34	121
289	0.82	0.55	0.66	145
290	0.36	0.12	0.17	139

291	0.84	0.64	0.72	136
292	0.14	0.01	0.01	129
293	0.28	0.09	0.13	129
294	0.34	0.09	0.14	141
295	0.55	0.05	0.09	128
296	0.00	0.00	0.00	136
297	0.68	0.24	0.35	135
298	0.44	0.14	0.22	133
299	0.64	0.33	0.44	114
300	0.27	0.02	0.04	137
301	0.48	0.12	0.19	138
302	0.71	0.45	0.55	124
303	0.50	0.12	0.19	116
304	0.53	0.22	0.31	126
305	0.23	0.06	0.10	117
306	0.43	0.05	0.09	117
307	0.15	0.02	0.03	110
308	0.59	0.14	0.22	123
309	0.63	0.20	0.31	108
310	0.00	0.00	0.00	116
311	0.59	0.26	0.36	140
312	0.49	0.16	0.24	119
313	0.58	0.21	0.31	125
314	0.51	0.24	0.33	94
315	0.34	0.16	0.21	121
316	0.27	0.02	0.04	137
317	0.32	0.09	0.14	116
318	0.78	0.49	0.60	127
319	0.71	0.30	0.42	129
320	0.49	0.29	0.36	121
321	0.00	0.00	0.00	129
322	0.25	0.03	0.05	102
323	0.93	0.69	0.79	118
324	0.24	0.04	0.07	128
325	0.78	0.39	0.52	110
326	0.00	0.00	0.00	122
327	0.31	0.06	0.10	136
328	0.43	0.14	0.21	114
329	0.67	0.02	0.04	94
330	0.29	0.06	0.11	109
331	0.44	0.06	0.11	114
332	0.60	0.12	0.20	100
333	0.35	0.06	0.10	105
334	0.27	0.05	0.08	120
335	0.73	0.34	0.46	94
336	0.44	0.13	0.21	112
337	0.14	0.01	0.02	104
338	0.54	0.21	0.30	100

339	0.77	0.42	0.54	112
340	0.44	0.22	0.30	117
341	0.58	0.32	0.41	94
342	0.54	0.12	0.19	113
343	0.33	0.06	0.10	86
344	0.57	0.04	0.07	110
345	0.47	0.16	0.24	113
346	0.11	0.02	0.03	117
347	0.00	0.00	0.00	107
348	0.00	0.00	0.00	122
349	0.50	0.30	0.37	105
350	0.46	0.23	0.31	107
351	0.45	0.21	0.29	105
352	0.68	0.33	0.44	104
353	0.38	0.16	0.22	114
354	0.94	0.61	0.74	122
355	0.61	0.33	0.43	100
356	0.58	0.23	0.33	108
357	0.28	0.05	0.09	91
358	0.81	0.41	0.55	102
359	0.27	0.09	0.14	96
360	0.67	0.27	0.38	104
361	0.45	0.23	0.30	105
362	0.14	0.03	0.05	97
363	0.22	0.02	0.04	102
364	0.25	0.03	0.05	101
365	0.40	0.02	0.03	118
366	0.57	0.25	0.35	108
367	0.47	0.19	0.27	120
368	1.00	0.03	0.05	108
369	0.78	0.43	0.56	99
370	0.10	0.01	0.02	96
371	0.10	0.01	0.02	114
372	0.79	0.55	0.65	100
373	1.00	0.44	0.61	108
374	0.36	0.08	0.14	108
375	0.96	0.43	0.59	100
376	0.45	0.05	0.09	106
377	0.77	0.42	0.54	98
378	0.39	0.10	0.16	91
379	0.25	0.05	0.08	118
380	0.20	0.03	0.05	109
381	0.59	0.24	0.34	114
382	0.20	0.05	0.08	102
383	0.95	0.49	0.64	117
384	0.59	0.09	0.15	115
385	0.68	0.16	0.26	93
386	0.41	0.10	0.16	111

387	0.57	0.04	0.08	93
388	0.32	0.17	0.22	99
389	0.29	0.12	0.17	101
390	0.00	0.00	0.00	107
391	0.39	0.14	0.21	90
392	0.53	0.24	0.33	97
393	0.73	0.43	0.54	84
394	0.91	0.63	0.74	115
395	0.24	0.07	0.10	75
396	0.75	0.35	0.48	108
397	0.17	0.05	0.07	110
398	0.50	0.21	0.30	99
399	0.47	0.08	0.13	89
400	0.55	0.06	0.10	105
401	0.00	0.00	0.00	85
402	0.94	0.60	0.73	114
403	0.46	0.14	0.21	95
404	0.52	0.16	0.24	108
405	0.65	0.38	0.48	94
406	0.10	0.01	0.02	104
407	0.58	0.14	0.23	106
408	0.65	0.14	0.23	106
409	0.00	0.00	0.00	88
410	0.52	0.28	0.37	102
411	0.00	0.00	0.00	93
412	0.77	0.41	0.54	99
413	0.12	0.03	0.05	88
414	0.75	0.48	0.59	96
415	0.85	0.55	0.67	80
416	0.62	0.28	0.39	93
417	0.57	0.13	0.21	91
418	0.46	0.16	0.24	97
419	0.34	0.12	0.18	101
420	0.00	0.00	0.00	92
421	0.98	0.55	0.71	92
422	0.50	0.01	0.02	80
423	0.17	0.01	0.02	100
424	0.50	0.08	0.14	97
425	0.95	0.55	0.69	106
426	0.63	0.13	0.22	90
427	0.50	0.04	0.08	91
428	0.33	0.04	0.07	101
429	0.71	0.14	0.23	108
430	0.28	0.09	0.13	94
431	0.48	0.18	0.27	87
432	0.09	0.01	0.02	99
433	0.63	0.36	0.46	99
434	0.73	0.41	0.53	87

435	0.30	0.07	0.11	87
436	0.64	0.10	0.17	93
437	0.00	0.00	0.00	76
438	0.80	0.45	0.58	97
439	0.94	0.38	0.54	81
440	0.33	0.11	0.16	103
441	0.40	0.10	0.16	101
442	0.80	0.41	0.54	91
443	0.49	0.31	0.38	89
444	0.44	0.06	0.11	110
445	0.70	0.42	0.52	91
446	0.25	0.04	0.07	95
447	0.92	0.57	0.70	79
448	0.71	0.42	0.53	92
449	0.44	0.13	0.20	95
450	0.42	0.18	0.25	90
451	0.25	0.04	0.07	94
452	0.27	0.09	0.14	98
453	0.34	0.16	0.22	68
454	0.66	0.50	0.57	86
455	0.98	0.51	0.67	98
456	0.71	0.06	0.11	81
457	0.46	0.13	0.20	93
458	0.68	0.28	0.40	96
459	0.35	0.11	0.17	80
460	0.32	0.10	0.15	79
461	0.20	0.03	0.05	77
462	0.00	0.00	0.00	86
463	0.22	0.08	0.11	78
464	0.43	0.09	0.15	97
465	0.00	0.00	0.00	72
466	0.00	0.00	0.00	84
467	0.42	0.09	0.14	93
468	0.00	0.00	0.00	93
469	0.72	0.32	0.44	81
470	0.60	0.17	0.26	90
471	0.53	0.20	0.29	98
472	0.95	0.55	0.70	76
473	0.36	0.11	0.17	82
474	0.10	0.01	0.03	67
475	0.78	0.42	0.54	77
476	0.20	0.01	0.02	94
477	0.49	0.21	0.29	82
478	0.38	0.03	0.06	88
479	0.80	0.35	0.48	92
480	0.50	0.22	0.30	74
481	0.38	0.15	0.22	93
482	0.00	0.00	0.00	103

```
        483       0.73      0.47      0.58        80
        484       0.48      0.24      0.32        89
        485       0.87      0.39      0.54        99
        486       0.91      0.67      0.77        79
        487       0.75      0.04      0.07        80
        488       0.61      0.32      0.42        69
        489       0.12      0.01      0.02        76
        490       0.47      0.09      0.15        76
        491       0.25      0.10      0.14        72
        492       0.00      0.00      0.00        77
        493       0.59      0.22      0.31        79
        494       0.00      0.00      0.00        82
        495       0.58      0.28      0.38        78
        496       0.85      0.53      0.65        83
        497       0.65      0.13      0.22        83
        498       0.87      0.68      0.76        87
        499       0.62      0.28      0.39        89

avg / total       0.63      0.33      0.42    180312
```

4.5 Modeling with less data points (100k data points) and more weight to title and 500 tags only.

In [56]:
```
sql_create_table = """CREATE TABLE IF NOT EXISTS QuestionsProcessed (question text NO
create_database_table("Titlemoreweight.db", sql_create_table)
```

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

In [57]:
```
# http://www.sqlitetutorial.net/sqlite-delete/
# https://stackoverflow.com/questions/2279706/select-random-row-from-a-sqlite-table

read_db = 'train_no_dup.db'
write_db = 'Titlemoreweight.db'
train_datasize = 400000
if os.path.isfile(read_db):
    conn_r = create_connection(read_db)
    if conn_r is not None:
        reader =conn_r.cursor()
        # for selecting first 0.5M rows
        reader.execute("SELECT Title, Body, Tags From no_dup_train LIMIT 100000;")
        # for selecting random points
        #reader.execute("SELECT Title, Body, Tags From no_dup_train ORDER BY RANDOM()

if os.path.isfile(write_db):
    conn_w = create_connection(write_db)
```

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

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

4.5.1 Preprocessing of questions
Separate Code from Body
Remove Spcial characters from Question title and description (not in code)
Give more weightage to title : Add title three times to the question
Remove stop words (Except 'C')
Remove HTML Tags
Convert all the characters into small letters
Use SnowballStemmer to stem the words

```
In [58]: #http://www.bernzilla.com/2008/05/13/selecting-a-random-row-from-an-sqlite-table/
         start = datetime.now()
         preprocessed_data_list=[]
         reader.fetchone()
         questions_with_code=0
         len_pre=0
         len_post=0
         questions_proccesed = 0
         for row in reader:

             is_code = 0

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

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

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

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

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

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

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

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

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

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

            len_post+=len(question)
            tup = (question,code,tags,x,len(question),is_code)
            questions_proccesed += 1
            writer.execute("insert into QuestionsProcessed(question,code,tags,words_pre,words_
            if (questions_proccesed%10000==0):
                print("number of questions completed=",questions_proccesed)

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

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

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

```
number of questions completed= 10000
number of questions completed= 20000
number of questions completed= 30000
number of questions completed= 40000
number of questions completed= 50000
number of questions completed= 60000
number of questions completed= 70000
number of questions completed= 80000
number of questions completed= 90000
Avg. length of questions(Title+Body) before processing: 1232
Avg. length of questions(Title+Body) after processing: 441
Percent of questions containing code: 57
Time taken to run this cell : 0:03:22.420709
```

In [59]:
```
# never forget to close the conections or else we will end up with database locks
conn_r.commit()
```

```
    conn_w.commit()
    conn_r.close()
    conn_w.close()
```

__ Sample quesitons after preprocessing of data __

```
In [60]: 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)
                 for row in reader:
                     print(row)
                     print('-'*100)
         conn_r.commit()
         conn_r.close()
```

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

__ Saving Preprocessed data to a Database __

```
In [61]: #Taking 100k entries to a dataframe.
         write_db = 'Titlemoreweight.db'
         if os.path.isfile(write_db):
             conn_r = create_connection(write_db)
```

```
            if conn_r is not None:
                preprocessed_data = pd.read_sql_query("""SELECT question, Tags FROM Questions
        conn_r.commit()
        conn_r.close()
```

In [62]:
```
preprocessed_data.head()
```

Out[62]:
```
                                            question  \
0  dynam datagrid bind silverlight dynam datagrid...
1  dynam datagrid bind silverlight dynam datagrid...
2  java.lang.noclassdeffounderror javax servlet j...
3  java.sql.sqlexcept microsoft odbc driver manag...
4  better way updat feed fb php sdk better way up...

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

In [63]:
```
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 : 99999
number of dimensions : 2
```

__ Converting string Tags to multilable output variables __

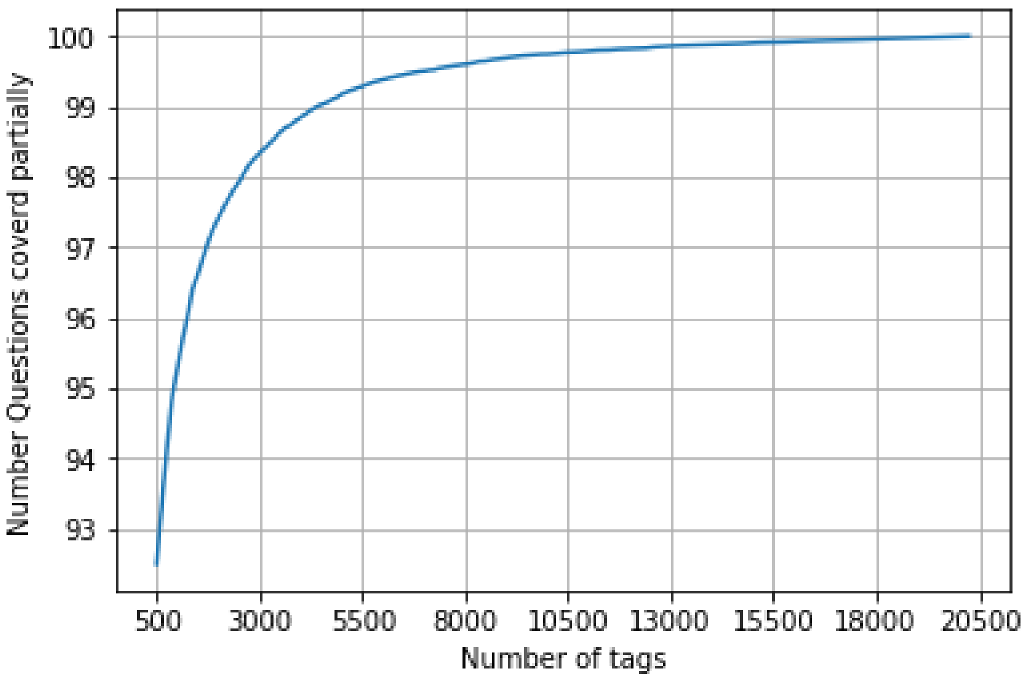
In [64]:
```
vectorizer = CountVectorizer(tokenizer = lambda x: x.split(), binary='true')
multilabel_y = vectorizer.fit_transform(preprocessed_data['tags'])
labels = vectorizer.get_feature_names()
```

__ Selecting 500 Tags __

In [65]:
```
questions_explained = []
total_tags=multilabel_y.shape[1]
total_qs=preprocessed_data.shape[0]
for i in range(500, total_tags, 100):
    questions_explained.append(np.round(((total_qs-questions_explained_fn(i))/total_q
```

In [66]:
```
fig, ax = plt.subplots()
ax.plot(questions_explained)
xlabel = list(500+np.array(range(-50,450,50))*50)
ax.set_xticklabels(xlabel)
plt.xlabel("Number of tags")
plt.ylabel("Number Questions coverd partially")
plt.grid()
```

```
plt.show()
# you can choose any number of tags based on your computing power, minimun is 500(it 
print("with ",5500,"tags we are covering ",questions_explained[50],"% of questions")
print("with ",500,"tags we are covering ",questions_explained[0],"% of questions")
```




```
with  5500 tags we are covering  99.481 % of questions
with  500 tags we are covering  92.5 % of questions
```

In [67]:
```
# we will be taking 500 tags
multilabel_yx = tags_to_choose(500)
print("number of questions that are not covered :", questions_explained_fn(500),"out 
```

```
number of questions that are not covered : 7500 out of  99999
```

In [68]:
```
total_size=preprocessed_data.shape[0]
train_size=int(0.80*total_size)

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

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

```
In [69]: # 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 [70]: 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 : (79999, 500)
Number of data points in test data : (20000, 500)
```

4.5.2 Featurizing data with TfIdf vectorizer

```
In [71]: start = datetime.now()
         vectorizer = TfidfVectorizer(min_df=0.00009, max_features=200000, smooth_idf=True, no
                                      tokenizer = lambda x: x.split(), sublinear_tf=False, ngra
         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 : 0:00:38.910869
```

```
In [72]: print("Dimensions of train data X:",x_train_multilabel.shape, "Y :",y_train.shape)
         print("Dimensions of test data X:",x_test_multilabel.shape,"Y:",y_test.shape)
```

```
Dimensions of train data X: (79999, 100247) Y : (79999, 500)
Dimensions of test data X: (20000, 100247) Y: (20000, 500)
```

4.5.3 Applying Logistic Regression with OneVsRest Classifier

```
In [73]: start = datetime.now()

         if os.path.isfile('lr_with_more_title_weight_SGD.pkl') == True:
             classifier = joblib.load('lr_with_more_title_weight_SGD.pkl')

         if os.path.isfile('lr_with_more_title_weight_SGD.pkl') == False:
             classifier = OneVsRestClassifier(SGDClassifier(loss='log', alpha=0.00001, penalty=
             classifier.fit(x_train_multilabel, y_train)
             joblib.dump(classifier, 'lr_with_more_title_weight_SGD.pkl')

         predictions = classifier.predict(x_test_multilabel)

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

```
Time taken to run this cell : 0:01:56.785369
```

```
In [74]: print("Accuracy :",metrics.accuracy_score(y_test, predictions))
         print("Hamming loss ",metrics.hamming_loss(y_test,predictions))

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

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

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

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

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

```
Accuracy : 0.18595
Hamming loss  0.0030815
Micro-average quality numbers
Precision: 0.7114, Recall: 0.2989, F1-measure: 0.4209
Macro-average quality numbers
Precision: 0.5237, Recall: 0.2357, F1-measure: 0.3042
             precision    recall  f1-score   support

          0       0.84      0.34      0.49       820
          1       0.71      0.19      0.30      1931
          2       0.64      0.12      0.21       544
          3       0.65      0.22      0.33       222
          4       0.82      0.46      0.59      1311
          5       0.89      0.48      0.62      1014
          6       0.78      0.37      0.50      1374
          7       0.90      0.57      0.70       702
          8       0.92      0.57      0.70      1424
          9       0.73      0.40      0.52      1037
         10       0.80      0.49      0.61       797
         11       0.74      0.36      0.48       156
         12       0.73      0.31      0.43        36
         13       0.77      0.39      0.52       610
         14       0.47      0.19      0.27       405
         15       0.86      0.17      0.29       144
         16       0.64      0.25      0.36       425
         17       0.66      0.23      0.34       485
         18       0.83      0.55      0.66       269
         19       0.90      0.55      0.68       518
         20       0.56      0.19      0.29       529
```

21	0.84	0.57	0.68	294
22	0.85	0.40	0.54	520
23	0.66	0.30	0.41	246
24	0.67	0.28	0.39	312
25	0.53	0.25	0.34	314
26	0.70	0.23	0.35	190
27	0.34	0.07	0.12	342
28	0.54	0.15	0.23	96
29	0.50	0.09	0.16	32
30	0.56	0.26	0.35	747
31	0.83	0.36	0.50	14
32	0.67	0.58	0.62	166
33	0.61	0.36	0.45	171
34	0.73	0.25	0.37	256
35	0.83	0.56	0.67	199
36	0.33	0.02	0.03	60
37	0.36	0.18	0.24	203
38	0.69	0.46	0.55	201
39	0.47	0.20	0.28	208
40	0.60	0.23	0.33	13
41	0.63	0.11	0.19	154
42	0.43	0.30	0.36	69
43	0.34	0.05	0.08	426
44	0.55	0.30	0.39	77
45	0.56	0.29	0.38	223
46	0.77	0.21	0.33	144
47	0.90	0.41	0.57	245
48	0.58	0.16	0.26	91
49	0.75	0.29	0.41	157
50	0.90	0.68	0.78	132
51	0.91	0.71	0.79	41
52	0.63	0.42	0.50	124
53	0.25	0.16	0.19	96
54	0.30	0.09	0.14	128
55	0.71	0.37	0.49	46
56	0.69	0.58	0.63	151
57	1.00	0.01	0.02	80
58	0.32	0.12	0.18	65
59	0.56	0.15	0.24	182
60	0.95	0.65	0.77	148
61	0.29	0.03	0.06	196
62	0.43	0.21	0.28	58
63	0.75	0.21	0.33	43
64	0.75	0.28	0.41	197
65	0.67	0.43	0.52	82
66	0.68	0.30	0.42	50
67	0.68	0.48	0.56	105
68	0.25	0.08	0.12	98

69	0.26	0.04	0.07	238
70	0.67	0.06	0.11	35
71	0.63	0.31	0.42	54
72	0.22	0.08	0.12	25
73	0.38	0.21	0.27	29
74	0.50	0.03	0.06	29
75	0.53	0.20	0.29	40
76	0.82	0.55	0.66	105
77	0.52	0.39	0.45	28
78	0.27	0.03	0.06	202
79	0.53	0.43	0.48	37
80	0.75	0.20	0.32	15
81	0.41	0.29	0.34	52
82	0.38	0.18	0.24	50
83	0.25	0.02	0.03	56
84	0.71	0.54	0.61	54
85	0.48	0.38	0.43	34
86	0.45	0.17	0.24	30
87	0.59	0.34	0.43	29
88	0.90	0.75	0.82	24
89	0.89	0.78	0.83	116
90	0.17	0.05	0.07	66
91	0.47	0.13	0.21	68
92	0.86	0.28	0.43	67
93	0.47	0.25	0.33	28
94	0.67	0.24	0.35	17
95	0.89	0.47	0.62	51
96	0.77	0.38	0.51	53
97	0.00	0.00	0.00	61
98	0.00	0.00	0.00	79
99	0.75	0.33	0.46	18
100	0.00	0.00	0.00	11
101	0.70	0.47	0.56	207
102	0.00	0.00	0.00	6
103	0.50	0.03	0.06	30
104	0.67	0.04	0.07	54
105	0.81	0.44	0.57	39
106	0.38	0.11	0.18	70
107	1.00	0.14	0.25	14
108	0.65	0.17	0.27	66
109	0.53	0.18	0.27	50
110	0.79	0.13	0.22	87
111	0.43	0.47	0.45	51
112	0.00	0.00	0.00	291
113	0.97	0.76	0.85	49
114	0.00	0.00	0.00	110
115	0.00	0.00	0.00	28
116	0.00	0.00	0.00	5

117	0.31	0.07	0.12	56
118	0.79	0.44	0.56	125
119	0.88	0.34	0.49	44
120	0.89	0.19	0.31	42
121	0.63	0.22	0.32	55
122	0.80	0.49	0.61	68
123	0.00	0.00	0.00	82
124	0.00	0.00	0.00	0
125	1.00	0.71	0.83	7
126	0.22	0.11	0.15	18
127	0.55	0.19	0.29	31
128	0.86	0.46	0.60	13
129	0.71	0.54	0.61	50
130	0.10	0.01	0.02	91
131	0.72	0.60	0.66	35
132	0.11	0.04	0.06	26
133	0.00	0.00	0.00	32
134	0.72	0.37	0.49	35
135	0.91	0.54	0.68	37
136	0.00	0.00	0.00	55
137	0.26	0.24	0.25	41
138	0.40	0.13	0.20	15
139	0.39	0.11	0.17	99
140	0.93	0.50	0.65	86
141	0.68	0.25	0.36	53
142	0.20	0.06	0.09	36
143	0.59	0.45	0.51	66
144	0.71	0.39	0.51	64
145	0.25	0.04	0.07	25
146	0.00	0.00	0.00	125
147	0.29	0.27	0.28	15
148	0.79	0.48	0.60	48
149	0.38	0.25	0.30	65
150	0.50	0.09	0.15	11
151	0.30	0.20	0.24	15
152	0.36	0.15	0.22	52
153	0.55	0.33	0.41	18
154	0.75	0.19	0.30	16
155	0.80	0.20	0.32	20
156	0.54	0.11	0.18	121
157	0.52	0.29	0.37	107
158	0.50	0.13	0.21	15
159	0.79	0.48	0.60	105
160	0.50	0.25	0.33	69
161	0.67	0.21	0.32	56
162	0.00	0.00	0.00	47
163	0.38	0.02	0.05	121
164	0.48	0.26	0.34	42

165	0.00	0.00	0.00	229
166	0.85	0.11	0.20	98
167	0.60	0.18	0.28	33
168	0.69	0.25	0.37	44
169	0.62	0.47	0.53	45
170	0.88	0.41	0.56	51
171	0.00	0.00	0.00	18
172	0.66	0.48	0.55	48
173	0.25	0.17	0.20	12
174	0.28	0.08	0.12	62
175	0.72	0.59	0.65	44
176	0.95	0.70	0.81	30
177	0.61	0.37	0.46	30
178	0.00	0.00	0.00	0
179	1.00	1.00	1.00	1
180	0.61	0.28	0.38	40
181	0.29	0.05	0.08	44
182	0.00	0.00	0.00	2
183	0.61	0.37	0.46	75
184	1.00	0.25	0.40	4
185	0.77	0.27	0.40	64
186	0.50	0.25	0.33	12
187	1.00	0.60	0.75	55
188	0.83	0.62	0.71	64
189	0.43	0.10	0.17	96
190	0.00	0.00	0.00	22
191	0.89	0.22	0.36	76
192	0.63	0.38	0.47	45
193	0.71	0.36	0.48	14
194	0.76	0.50	0.60	50
195	1.00	0.15	0.26	20
196	0.88	0.60	0.71	35
197	0.66	0.22	0.33	94
198	1.00	0.07	0.13	14
199	0.00	0.00	0.00	25
200	1.00	0.07	0.14	54
201	0.50	0.14	0.21	22
202	0.71	0.12	0.20	43
203	0.00	0.00	0.00	43
204	0.97	0.52	0.67	62
205	0.00	0.00	0.00	3
206	0.29	0.05	0.08	43
207	0.50	0.14	0.22	7
208	0.33	0.12	0.18	8
209	0.23	0.07	0.11	42
210	0.40	0.40	0.40	10
211	0.50	0.17	0.26	40
212	0.78	0.30	0.44	23

213	0.00	0.00	0.00	6
214	0.69	0.38	0.49	47
215	0.50	0.10	0.16	62
216	0.66	0.27	0.39	77
217	0.60	0.14	0.22	22
218	0.00	0.00	0.00	3
219	0.00	0.00	0.00	28
220	0.80	0.05	0.09	81
221	0.50	0.06	0.11	31
222	1.00	0.03	0.06	34
223	1.00	0.37	0.54	60
224	0.50	0.10	0.17	10
225	0.83	0.50	0.62	10
226	0.75	0.63	0.69	92
227	0.71	0.38	0.50	13
228	0.50	0.08	0.13	13
229	0.89	0.74	0.81	43
230	0.45	0.14	0.22	35
231	0.00	0.00	0.00	4
232	0.40	0.20	0.27	20
233	0.60	0.14	0.23	145
234	0.91	0.53	0.67	55
235	0.00	0.00	0.00	2
236	0.44	0.19	0.26	37
237	0.82	0.47	0.60	90
238	0.67	0.03	0.07	58
239	1.00	0.25	0.40	20
240	0.94	0.51	0.66	61
241	0.82	0.64	0.72	42
242	0.56	0.77	0.65	30
243	0.87	0.50	0.63	66
244	0.56	0.21	0.31	42
245	0.00	0.00	0.00	31
246	1.00	0.33	0.50	6
247	0.75	0.17	0.27	18
248	0.82	0.53	0.64	51
249	0.80	0.47	0.59	17
250	0.53	0.36	0.43	22
251	0.78	0.35	0.48	52
252	0.50	0.03	0.06	29
253	0.08	0.04	0.05	28
254	0.00	0.00	0.00	10
255	0.25	0.20	0.22	5
256	0.17	0.33	0.22	3
257	0.80	0.20	0.31	41
258	0.62	0.17	0.26	30
259	1.00	0.33	0.50	3
260	0.00	0.00	0.00	38

261	0.00	0.00	0.00	1
262	0.70	0.37	0.48	19
263	0.00	0.00	0.00	14
264	1.00	0.03	0.05	37
265	0.14	0.11	0.12	9
266	0.36	0.09	0.14	45
267	0.58	0.55	0.56	33
268	0.75	0.56	0.64	16
269	0.71	0.49	0.58	35
270	0.50	0.27	0.35	11
271	0.00	0.00	0.00	30
272	0.60	0.38	0.46	8
273	0.11	0.05	0.07	21
274	1.00	0.01	0.02	123
275	0.59	0.28	0.38	67
276	0.89	0.80	0.84	20
277	0.00	0.00	0.00	14
278	0.50	0.05	0.10	19
279	1.00	0.50	0.67	12
280	0.00	0.00	0.00	15
281	0.91	0.59	0.71	17
282	1.00	0.63	0.78	41
283	0.50	0.13	0.21	15
284	0.63	0.23	0.34	74
285	0.56	0.13	0.21	38
286	0.29	0.12	0.17	16
287	0.40	0.07	0.11	30
288	0.94	0.57	0.71	28
289	0.00	0.00	0.00	21
290	0.81	0.51	0.63	41
291	0.25	0.17	0.20	12
292	1.00	0.12	0.22	24
293	0.57	0.40	0.47	20
294	0.00	0.00	0.00	23
295	0.00	0.00	0.00	29
296	0.50	0.04	0.07	28
297	0.53	0.19	0.28	42
298	0.00	0.00	0.00	53
299	0.00	0.00	0.00	36
300	0.42	0.12	0.19	41
301	0.74	0.46	0.57	37
302	0.80	0.31	0.44	26
303	0.67	0.36	0.47	11
304	0.31	0.13	0.18	31
305	0.50	0.18	0.26	17
306	1.00	0.11	0.20	9
307	1.00	0.17	0.29	6
308	0.00	0.00	0.00	34

309	0.72	0.42	0.53	43
310	0.17	0.03	0.06	30
311	0.40	0.12	0.18	50
312	0.00	0.00	0.00	24
313	0.00	0.00	0.00	42
314	0.67	0.18	0.29	22
315	1.00	0.02	0.03	58
316	1.00	0.10	0.18	10
317	0.46	0.19	0.27	57
318	1.00	0.40	0.57	10
319	0.00	0.00	0.00	11
320	0.00	0.00	0.00	11
321	0.40	0.25	0.31	8
322	0.80	0.36	0.50	22
323	0.94	0.61	0.74	28
324	0.65	0.44	0.52	50
325	0.67	0.11	0.19	18
326	1.00	0.03	0.06	33
327	0.29	0.12	0.17	17
328	0.67	0.07	0.12	29
329	1.00	0.29	0.44	7
330	0.62	0.50	0.56	10
331	0.00	0.00	0.00	25
332	0.67	1.00	0.80	2
333	0.80	0.36	0.50	11
334	0.00	0.00	0.00	24
335	1.00	0.20	0.33	5
336	0.50	0.06	0.11	33
337	0.95	0.50	0.66	42
338	0.40	0.08	0.13	26
339	0.75	0.21	0.32	29
340	0.54	0.42	0.47	36
341	1.00	0.54	0.70	13
342	0.80	0.36	0.50	11
343	1.00	0.40	0.57	10
344	0.32	0.38	0.35	21
345	0.00	0.00	0.00	0
346	0.00	0.00	0.00	6
347	0.67	0.17	0.27	12
348	0.50	0.15	0.24	13
349	0.80	0.17	0.28	24
350	0.73	0.30	0.42	27
351	0.50	0.09	0.16	43
352	0.00	0.00	0.00	30
353	0.55	0.27	0.36	22
354	1.00	0.03	0.06	31
355	0.83	0.50	0.62	10
356	0.50	0.15	0.23	20

357	0.80	0.60	0.69	20
358	0.40	0.29	0.33	28
359	0.56	0.48	0.51	21
360	0.17	0.04	0.06	25
361	0.60	0.34	0.44	35
362	0.83	0.53	0.64	36
363	0.33	0.18	0.23	17
364	1.00	0.29	0.44	14
365	0.00	0.00	0.00	21
366	0.00	0.00	0.00	18
367	0.00	0.00	0.00	97
368	0.65	0.45	0.53	29
369	1.00	0.50	0.67	12
370	0.60	0.23	0.33	13
371	0.33	0.17	0.22	18
372	0.00	0.00	0.00	6
373	0.40	0.33	0.36	6
374	0.33	0.03	0.06	30
375	0.50	0.15	0.23	27
376	0.33	0.04	0.06	28
377	0.00	0.00	0.00	2
378	0.50	0.25	0.33	4
379	0.00	0.00	0.00	19
380	0.20	0.20	0.20	5
381	1.00	0.33	0.50	18
382	0.78	0.32	0.45	22
383	0.00	0.00	0.00	16
384	0.38	0.23	0.29	13
385	0.33	0.17	0.22	18
386	0.90	0.82	0.86	11
387	0.41	0.17	0.24	88
388	1.00	0.15	0.27	13
389	0.00	0.00	0.00	6
390	0.00	0.00	0.00	6
391	1.00	0.57	0.72	51
392	0.00	0.00	0.00	13
393	0.61	0.30	0.40	37
394	0.00	0.00	0.00	6
395	1.00	0.11	0.20	9
396	0.33	0.08	0.12	13
397	1.00	0.50	0.67	6
398	0.69	0.38	0.49	29
399	0.96	0.70	0.81	33
400	0.50	0.03	0.06	31
401	0.67	0.04	0.08	50
402	0.91	0.56	0.69	18
403	0.50	0.14	0.22	7
404	0.65	0.58	0.61	26

405	0.97	0.68	0.80	56
406	1.00	0.25	0.40	4
407	0.20	0.06	0.09	17
408	1.00	0.36	0.53	11
409	0.00	0.00	0.00	18
410	0.60	0.30	0.40	10
411	0.22	0.04	0.07	45
412	0.86	0.30	0.44	20
413	0.33	0.04	0.07	25
414	0.67	0.10	0.17	20
415	0.00	0.00	0.00	6
416	1.00	0.04	0.07	26
417	1.00	0.10	0.18	10
418	0.00	0.00	0.00	18
419	1.00	0.17	0.29	6
420	0.50	0.47	0.48	17
421	0.00	0.00	0.00	1
422	0.00	0.00	0.00	6
423	0.00	0.00	0.00	12
424	1.00	0.25	0.40	4
425	1.00	0.09	0.17	11
426	0.00	0.00	0.00	11
427	0.86	0.75	0.80	8
428	0.50	0.12	0.19	26
429	0.56	0.38	0.45	40
430	0.00	0.00	0.00	2
431	0.00	0.00	0.00	35
432	1.00	0.20	0.33	15
433	0.00	0.00	0.00	18
434	0.00	0.00	0.00	0
435	0.00	0.00	0.00	0
436	0.00	0.00	0.00	28
437	0.50	0.18	0.27	33
438	0.83	0.50	0.62	20
439	0.00	0.00	0.00	36
440	0.67	0.22	0.33	18
441	0.57	0.44	0.50	18
442	0.82	0.56	0.67	16
443	0.11	0.05	0.06	22
444	0.00	0.00	0.00	6
445	0.90	0.43	0.58	21
446	0.89	0.52	0.66	46
447	0.00	0.00	0.00	69
448	0.00	0.00	0.00	7
449	0.00	0.00	0.00	3
450	0.00	0.00	0.00	52
451	0.00	0.00	0.00	16
452	1.00	0.59	0.74	17

453	0.00	0.00	0.00	13
454	0.80	0.36	0.50	11
455	0.00	0.00	0.00	12
456	0.50	0.17	0.25	6
457	0.17	0.06	0.08	18
458	0.00	0.00	0.00	15
459	0.92	0.39	0.55	28
460	0.00	0.00	0.00	18
461	0.67	0.40	0.50	10
462	0.40	0.08	0.14	24
463	1.00	0.11	0.20	18
464	1.00	0.44	0.61	39
465	0.33	0.36	0.35	11
466	0.11	0.03	0.05	35
467	0.11	0.05	0.07	21
468	0.50	0.11	0.18	37
469	1.00	0.20	0.33	5
470	0.50	0.12	0.20	8
471	0.79	0.30	0.43	37
472	0.20	0.02	0.04	47
473	0.50	0.29	0.36	14
474	1.00	0.61	0.76	23
475	0.58	0.56	0.57	66
476	0.00	0.00	0.00	3
477	0.55	0.32	0.40	19
478	0.00	0.00	0.00	1
479	0.00	0.00	0.00	23
480	0.00	0.00	0.00	60
481	0.33	0.08	0.12	26
482	1.00	0.25	0.40	4
483	0.33	0.12	0.18	8
484	0.89	0.35	0.50	23
485	0.54	0.39	0.45	18
486	0.50	0.25	0.33	12
487	0.83	0.34	0.49	29
488	1.00	1.00	1.00	1
489	0.50	0.17	0.25	6
490	0.40	0.29	0.33	7
491	0.00	0.00	0.00	3
492	0.33	0.20	0.25	10
493	0.45	0.26	0.33	19
494	0.00	0.00	0.00	7
495	1.00	0.50	0.67	8
496	0.50	0.44	0.47	18
497	0.00	0.00	0.00	72
498	0.00	0.00	0.00	8
499	0.27	0.09	0.14	32

```
avg / total       0.64      0.30      0.39     37472
```

```
In [75]: start = datetime.now()

         if os.path.isfile('lr_with_more_title_weight_LR.pkl') == True:
             classifier = joblib.load('lr_with_more_title_weight_LR.pkl')

         if os.path.isfile('lr_with_more_title_weight_LR.pkl') == False:
             classifier = OneVsRestClassifier(LogisticRegression(penalty='l1'), n_jobs=-1)
             classifier.fit(x_train_multilabel, y_train)
             joblib.dump(classifier, 'lr_with_more_title_weight_LR.pkl')

         predictions = classifier.predict(x_test_multilabel)

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

```
Time taken to run this cell : 0:07:41.819447
```

```
In [76]: print("Accuracy :",metrics.accuracy_score(y_test, predictions))
         print("Hamming loss ",metrics.hamming_loss(y_test,predictions))

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

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

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

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

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

```
Accuracy : 0.18775
Hamming loss  0.0030755
Micro-average quality numbers
Precision: 0.7071, Recall: 0.3060, F1-measure: 0.4272
Macro-average quality numbers
Precision: 0.5249, Recall: 0.2485, F1-measure: 0.3163
             precision    recall  f1-score   support

          0       0.85      0.35      0.50       820
```

1	0.72	0.17	0.27	1931
2	0.60	0.12	0.20	544
3	0.64	0.21	0.32	222
4	0.82	0.46	0.59	1311
5	0.89	0.47	0.62	1014
6	0.77	0.37	0.50	1374
7	0.90	0.56	0.69	702
8	0.93	0.58	0.72	1424
9	0.72	0.47	0.57	1037
10	0.79	0.50	0.61	797
11	0.74	0.37	0.49	156
12	0.79	0.31	0.44	36
13	0.76	0.39	0.51	610
14	0.46	0.18	0.26	405
15	0.81	0.17	0.29	144
16	0.65	0.21	0.32	425
17	0.66	0.22	0.33	485
18	0.81	0.54	0.65	269
19	0.90	0.55	0.68	518
20	0.55	0.21	0.31	529
21	0.83	0.57	0.68	294
22	0.85	0.40	0.54	520
23	0.66	0.30	0.41	246
24	0.69	0.29	0.41	312
25	0.56	0.25	0.35	314
26	0.71	0.25	0.37	190
27	0.25	0.06	0.09	342
28	0.62	0.21	0.31	96
29	0.62	0.16	0.25	32
30	0.57	0.27	0.37	747
31	0.83	0.36	0.50	14
32	0.66	0.58	0.62	166
33	0.63	0.36	0.46	171
34	0.73	0.26	0.39	256
35	0.83	0.56	0.67	199
36	0.33	0.02	0.03	60
37	0.37	0.18	0.24	203
38	0.68	0.45	0.54	201
39	0.53	0.23	0.32	208
40	0.60	0.23	0.33	13
41	0.53	0.12	0.19	154
42	0.43	0.29	0.35	69
43	0.39	0.08	0.13	426
44	0.55	0.30	0.39	77
45	0.52	0.27	0.36	223
46	0.77	0.21	0.33	144
47	0.88	0.43	0.58	245
48	0.58	0.16	0.26	91

49	0.71	0.29	0.41	157
50	0.90	0.69	0.78	132
51	0.91	0.73	0.81	41
52	0.62	0.44	0.52	124
53	0.23	0.14	0.17	96
54	0.27	0.09	0.13	128
55	0.68	0.37	0.48	46
56	0.70	0.58	0.64	151
57	0.00	0.00	0.00	80
58	0.36	0.14	0.20	65
59	0.50	0.19	0.27	182
60	0.96	0.66	0.78	148
61	0.33	0.06	0.10	196
62	0.43	0.17	0.25	58
63	0.79	0.26	0.39	43
64	0.75	0.27	0.40	197
65	0.65	0.38	0.48	82
66	0.72	0.36	0.48	50
67	0.66	0.49	0.56	105
68	0.22	0.08	0.12	98
69	0.27	0.03	0.06	238
70	0.33	0.06	0.10	35
71	0.63	0.31	0.42	54
72	0.22	0.08	0.12	25
73	0.50	0.24	0.33	29
74	0.00	0.00	0.00	29
75	0.64	0.23	0.33	40
76	0.82	0.53	0.65	105
77	0.50	0.36	0.42	28
78	0.27	0.03	0.06	202
79	0.56	0.41	0.47	37
80	0.75	0.20	0.32	15
81	0.46	0.31	0.37	52
82	0.38	0.22	0.28	50
83	0.17	0.02	0.03	56
84	0.71	0.54	0.61	54
85	0.52	0.47	0.49	34
86	0.55	0.20	0.29	30
87	0.56	0.34	0.43	29
88	0.86	0.75	0.80	24
89	0.90	0.79	0.84	116
90	0.19	0.05	0.07	66
91	0.47	0.10	0.17	68
92	0.84	0.31	0.46	67
93	0.44	0.25	0.32	28
94	0.67	0.24	0.35	17
95	0.89	0.47	0.62	51
96	0.78	0.40	0.53	53

97	0.50	0.02	0.03	61
98	0.00	0.00	0.00	79
99	0.86	0.33	0.48	18
100	0.00	0.00	0.00	11
101	0.70	0.49	0.58	207
102	0.00	0.00	0.00	6
103	0.50	0.03	0.06	30
104	0.50	0.02	0.04	54
105	0.80	0.41	0.54	39
106	0.40	0.11	0.18	70
107	1.00	0.14	0.25	14
108	0.65	0.17	0.27	66
109	0.53	0.18	0.27	50
110	0.79	0.13	0.22	87
111	0.43	0.39	0.41	51
112	0.80	0.01	0.03	291
113	0.97	0.76	0.85	49
114	0.17	0.01	0.02	110
115	0.00	0.00	0.00	28
116	0.00	0.00	0.00	5
117	0.40	0.07	0.12	56
118	0.80	0.44	0.57	125
119	0.88	0.34	0.49	44
120	0.75	0.14	0.24	42
121	0.61	0.20	0.30	55
122	0.83	0.51	0.64	68
123	0.00	0.00	0.00	82
124	0.00	0.00	0.00	0
125	1.00	0.71	0.83	7
126	0.25	0.11	0.15	18
127	0.60	0.19	0.29	31
128	0.86	0.46	0.60	13
129	0.70	0.52	0.60	50
130	0.00	0.00	0.00	91
131	0.78	0.60	0.68	35
132	0.17	0.04	0.06	26
133	0.33	0.03	0.06	32
134	0.70	0.40	0.51	35
135	0.88	0.59	0.71	37
136	0.00	0.00	0.00	55
137	0.26	0.27	0.26	41
138	0.43	0.20	0.27	15
139	0.42	0.11	0.18	99
140	0.92	0.55	0.69	86
141	0.63	0.23	0.33	53
142	0.27	0.08	0.13	36
143	0.59	0.44	0.50	66
144	0.71	0.42	0.53	64

145	0.25	0.04	0.07	25
146	0.05	0.01	0.01	125
147	0.38	0.33	0.36	15
148	0.79	0.48	0.60	48
149	0.37	0.23	0.28	65
150	0.50	0.09	0.15	11
151	0.43	0.20	0.27	15
152	0.36	0.15	0.22	52
153	0.50	0.33	0.40	18
154	0.75	0.19	0.30	16
155	0.80	0.20	0.32	20
156	0.48	0.11	0.18	121
157	0.57	0.36	0.44	107
158	0.50	0.13	0.21	15
159	0.77	0.47	0.58	105
160	0.46	0.23	0.31	69
161	0.59	0.18	0.27	56
162	0.00	0.00	0.00	47
163	0.43	0.02	0.05	121
164	0.52	0.26	0.35	42
165	0.00	0.00	0.00	229
166	0.88	0.22	0.36	98
167	0.64	0.21	0.32	33
168	0.73	0.25	0.37	44
169	0.65	0.49	0.56	45
170	0.84	0.41	0.55	51
171	0.00	0.00	0.00	18
172	0.67	0.50	0.57	48
173	0.25	0.17	0.20	12
174	0.32	0.10	0.15	62
175	0.73	0.55	0.62	44
176	0.95	0.70	0.81	30
177	0.63	0.40	0.49	30
178	0.00	0.00	0.00	0
179	1.00	1.00	1.00	1
180	0.61	0.28	0.38	40
181	0.43	0.07	0.12	44
182	0.00	0.00	0.00	2
183	0.63	0.36	0.46	75
184	1.00	0.25	0.40	4
185	0.75	0.33	0.46	64
186	0.50	0.25	0.33	12
187	0.97	0.60	0.74	55
188	0.80	0.61	0.69	64
189	0.43	0.10	0.17	96
190	0.20	0.05	0.07	22
191	0.94	0.22	0.36	76
192	0.68	0.42	0.52	45

193	0.71	0.36	0.48	14
194	0.71	0.40	0.51	50
195	0.78	0.35	0.48	20
196	0.85	0.63	0.72	35
197	0.65	0.23	0.34	94
198	1.00	0.07	0.13	14
199	0.00	0.00	0.00	25
200	0.71	0.09	0.16	54
201	0.50	0.14	0.21	22
202	0.60	0.14	0.23	43
203	0.00	0.00	0.00	43
204	0.97	0.58	0.73	62
205	0.00	0.00	0.00	3
206	0.33	0.05	0.08	43
207	0.00	0.00	0.00	7
208	0.25	0.12	0.17	8
209	0.21	0.07	0.11	42
210	0.36	0.40	0.38	10
211	0.41	0.17	0.25	40
212	0.71	0.22	0.33	23
213	0.00	0.00	0.00	6
214	0.76	0.40	0.53	47
215	0.46	0.10	0.16	62
216	0.66	0.32	0.43	77
217	0.67	0.18	0.29	22
218	0.00	0.00	0.00	3
219	0.00	0.00	0.00	28
220	0.80	0.05	0.09	81
221	0.75	0.10	0.17	31
222	1.00	0.03	0.06	34
223	1.00	0.38	0.55	60
224	1.00	0.20	0.33	10
225	0.83	0.50	0.62	10
226	0.77	0.61	0.68	92
227	0.67	0.31	0.42	13
228	0.67	0.15	0.25	13
229	0.89	0.74	0.81	43
230	0.55	0.17	0.26	35
231	0.00	0.00	0.00	4
232	0.40	0.20	0.27	20
233	0.50	0.12	0.20	145
234	0.85	0.53	0.65	55
235	0.00	0.00	0.00	2
236	0.39	0.19	0.25	37
237	0.83	0.43	0.57	90
238	0.50	0.07	0.12	58
239	1.00	0.25	0.40	20
240	0.97	0.56	0.71	61

241	0.83	0.71	0.77	42
242	0.55	0.77	0.64	30
243	0.88	0.55	0.67	66
244	0.53	0.19	0.28	42
245	0.10	0.03	0.05	31
246	1.00	0.33	0.50	6
247	0.60	0.17	0.26	18
248	0.85	0.55	0.67	51
249	0.67	0.47	0.55	17
250	0.59	0.45	0.51	22
251	0.78	0.35	0.48	52
252	0.50	0.03	0.06	29
253	0.10	0.04	0.05	28
254	0.00	0.00	0.00	10
255	0.25	0.20	0.22	5
256	0.17	0.33	0.22	3
257	0.73	0.27	0.39	41
258	0.50	0.13	0.21	30
259	1.00	0.33	0.50	3
260	1.00	0.03	0.05	38
261	0.00	0.00	0.00	1
262	0.64	0.37	0.47	19
263	0.00	0.00	0.00	14
264	0.50	0.03	0.05	37
265	0.14	0.11	0.12	9
266	0.38	0.11	0.17	45
267	0.57	0.52	0.54	33
268	0.77	0.62	0.69	16
269	0.67	0.51	0.58	35
270	0.43	0.27	0.33	11
271	0.00	0.00	0.00	30
272	0.67	0.50	0.57	8
273	0.12	0.05	0.07	21
274	0.39	0.07	0.12	123
275	0.62	0.30	0.40	67
276	0.84	0.80	0.82	20
277	0.00	0.00	0.00	14
278	0.75	0.16	0.26	19
279	0.86	0.50	0.63	12
280	0.00	0.00	0.00	15
281	0.86	0.71	0.77	17
282	1.00	0.68	0.81	41
283	0.60	0.20	0.30	15
284	0.59	0.26	0.36	74
285	0.50	0.16	0.24	38
286	0.33	0.12	0.18	16
287	0.50	0.07	0.12	30
288	0.94	0.57	0.71	28

289	0.00	0.00	0.00	21
290	0.81	0.51	0.63	41
291	0.33	0.17	0.22	12
292	1.00	0.17	0.29	24
293	0.60	0.45	0.51	20
294	0.00	0.00	0.00	23
295	0.00	0.00	0.00	29
296	0.67	0.07	0.13	28
297	0.50	0.19	0.28	42
298	0.00	0.00	0.00	53
299	0.00	0.00	0.00	36
300	0.45	0.12	0.19	41
301	0.69	0.49	0.57	37
302	0.83	0.38	0.53	26
303	0.67	0.36	0.47	11
304	0.27	0.10	0.14	31
305	0.50	0.18	0.26	17
306	1.00	0.11	0.20	9
307	1.00	0.17	0.29	6
308	0.00	0.00	0.00	34
309	0.69	0.42	0.52	43
310	0.14	0.03	0.05	30
311	0.33	0.12	0.18	50
312	0.00	0.00	0.00	24
313	0.50	0.02	0.05	42
314	0.57	0.18	0.28	22
315	1.00	0.02	0.03	58
316	0.00	0.00	0.00	10
317	0.39	0.16	0.23	57
318	1.00	0.40	0.57	10
319	0.00	0.00	0.00	11
320	0.00	0.00	0.00	11
321	0.33	0.25	0.29	8
322	0.80	0.36	0.50	22
323	0.86	0.64	0.73	28
324	0.69	0.48	0.56	50
325	0.67	0.11	0.19	18
326	0.00	0.00	0.00	33
327	0.29	0.12	0.17	17
328	0.60	0.10	0.18	29
329	1.00	0.29	0.44	7
330	0.62	0.50	0.56	10
331	0.11	0.04	0.06	25
332	0.67	1.00	0.80	2
333	0.75	0.27	0.40	11
334	0.00	0.00	0.00	24
335	1.00	0.20	0.33	5
336	0.60	0.09	0.16	33

337	0.96	0.57	0.72	42
338	0.40	0.08	0.13	26
339	0.67	0.21	0.32	29
340	0.50	0.44	0.47	36
341	1.00	0.46	0.63	13
342	0.67	0.36	0.47	11
343	1.00	0.40	0.57	10
344	0.36	0.43	0.39	21
345	0.00	0.00	0.00	0
346	0.00	0.00	0.00	6
347	0.60	0.25	0.35	12
348	0.50	0.15	0.24	13
349	0.75	0.12	0.21	24
350	0.80	0.30	0.43	27
351	0.50	0.14	0.22	43
352	0.00	0.00	0.00	30
353	0.56	0.23	0.32	22
354	1.00	0.03	0.06	31
355	0.75	0.60	0.67	10
356	1.00	0.20	0.33	20
357	0.76	0.65	0.70	20
358	0.48	0.36	0.41	28
359	0.67	0.38	0.48	21
360	0.33	0.12	0.18	25
361	0.60	0.34	0.44	35
362	0.85	0.61	0.71	36
363	0.33	0.18	0.23	17
364	1.00	0.43	0.60	14
365	0.00	0.00	0.00	21
366	1.00	0.06	0.11	18
367	0.44	0.07	0.12	97
368	0.67	0.34	0.45	29
369	1.00	0.83	0.91	12
370	0.80	0.31	0.44	13
371	0.25	0.11	0.15	18
372	0.00	0.00	0.00	6
373	0.50	0.50	0.50	6
374	0.25	0.03	0.06	30
375	0.57	0.15	0.24	27
376	0.50	0.07	0.12	28
377	0.00	0.00	0.00	2
378	0.67	0.50	0.57	4
379	0.00	0.00	0.00	19
380	0.25	0.20	0.22	5
381	1.00	0.33	0.50	18
382	0.73	0.36	0.48	22
383	0.00	0.00	0.00	16
384	0.43	0.23	0.30	13

385	0.33	0.17	0.22	18
386	0.90	0.82	0.86	11
387	0.39	0.24	0.30	88
388	0.75	0.23	0.35	13
389	0.00	0.00	0.00	6
390	0.00	0.00	0.00	6
391	0.94	0.65	0.77	51
392	0.50	0.08	0.13	13
393	0.52	0.35	0.42	37
394	0.00	0.00	0.00	6
395	0.50	0.11	0.18	9
396	0.33	0.08	0.12	13
397	1.00	0.50	0.67	6
398	0.71	0.34	0.47	29
399	0.96	0.73	0.83	33
400	0.50	0.03	0.06	31
401	0.80	0.08	0.15	50
402	0.92	0.67	0.77	18
403	0.50	0.14	0.22	7
404	0.62	0.58	0.60	26
405	0.89	0.71	0.79	56
406	1.00	0.25	0.40	4
407	0.20	0.06	0.09	17
408	1.00	0.36	0.53	11
409	0.00	0.00	0.00	18
410	0.67	0.40	0.50	10
411	0.25	0.04	0.08	45
412	0.78	0.35	0.48	20
413	0.60	0.12	0.20	25
414	0.25	0.05	0.08	20
415	0.50	0.17	0.25	6
416	1.00	0.04	0.07	26
417	1.00	0.10	0.18	10
418	0.00	0.00	0.00	18
419	1.00	0.17	0.29	6
420	0.47	0.41	0.44	17
421	0.00	0.00	0.00	1
422	0.00	0.00	0.00	6
423	0.00	0.00	0.00	12
424	1.00	0.75	0.86	4
425	0.00	0.00	0.00	11
426	0.33	0.09	0.14	11
427	0.86	0.75	0.80	8
428	0.50	0.12	0.19	26
429	0.57	0.40	0.47	40
430	0.00	0.00	0.00	2
431	0.00	0.00	0.00	35
432	1.00	0.20	0.33	15

433	0.00	0.00	0.00	18
434	0.00	0.00	0.00	0
435	0.00	0.00	0.00	0
436	0.20	0.04	0.06	28
437	0.50	0.18	0.27	33
438	0.83	0.50	0.62	20
439	0.00	0.00	0.00	36
440	0.40	0.11	0.17	18
441	0.57	0.44	0.50	18
442	0.82	0.56	0.67	16
443	0.11	0.05	0.06	22
444	0.00	0.00	0.00	6
445	0.83	0.48	0.61	21
446	0.90	0.59	0.71	46
447	0.09	0.03	0.04	69
448	0.00	0.00	0.00	7
449	0.00	0.00	0.00	3
450	0.00	0.00	0.00	52
451	0.00	0.00	0.00	16
452	1.00	0.71	0.83	17
453	0.00	0.00	0.00	13
454	0.80	0.36	0.50	11
455	0.00	0.00	0.00	12
456	0.00	0.00	0.00	6
457	0.14	0.06	0.08	18
458	0.00	0.00	0.00	15
459	0.94	0.57	0.71	28
460	0.00	0.00	0.00	18
461	1.00	0.30	0.46	10
462	0.50	0.12	0.20	24
463	0.75	0.17	0.27	18
464	0.95	0.49	0.64	39
465	0.33	0.36	0.35	11
466	0.10	0.03	0.04	35
467	0.08	0.05	0.06	21
468	0.33	0.03	0.05	37
469	0.67	0.40	0.50	5
470	1.00	0.12	0.22	8
471	0.79	0.30	0.43	37
472	0.11	0.02	0.04	47
473	0.43	0.21	0.29	14
474	1.00	0.61	0.76	23
475	0.60	0.64	0.62	66
476	0.00	0.00	0.00	3
477	0.43	0.32	0.36	19
478	0.00	0.00	0.00	1
479	0.00	0.00	0.00	23
480	1.00	0.03	0.06	60

```
        481       0.43      0.12      0.18        26
        482       0.67      0.50      0.57         4
        483       0.50      0.12      0.20         8
        484       0.89      0.35      0.50        23
        485       0.62      0.44      0.52        18
        486       0.40      0.17      0.24        12
        487       0.85      0.38      0.52        29
        488       1.00      1.00      1.00         1
        489       0.50      0.17      0.25         6
        490       0.33      0.29      0.31         7
        491       0.00      0.00      0.00         3
        492       0.40      0.20      0.27        10
        493       0.50      0.42      0.46        19
        494       0.00      0.00      0.00         7
        495       0.80      0.50      0.62         8
        496       0.50      0.50      0.50        18
        497       0.00      0.00      0.00        72
        498       0.00      0.00      0.00         8
        499       0.50      0.25      0.33        32

avg / total       0.64      0.31      0.40     37472
```

5. Assignments

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

1.1 (5.1) Vectorize the data based on word frequencies (BOW)

Use countvectorizer to perform trigram vectorizing

```
In [77]: start = datetime.now()
         vectorizer = CountVectorizer(tokenizer=lambda x: x.split(),
                                      ngram_range=(1, 4),
                                      min_df=0.00009,
                                      max_features=200000,
                                      binary=False)
         x_train_multilabel = vectorizer.fit_transform(x_train['question'])
         x_test_multilabel = vectorizer.transform(x_test['question'])
         print("Time taken to run this cell :", datetime.now() - start)
```

```
Time taken to run this cell : 0:01:07.154276
```

```
In [78]: print("Dimensions of train data X:",x_train_multilabel.shape, "Y :",y_train.shape)
         print("Dimensions of test data X:",x_test_multilabel.shape,"Y:",y_test.shape)
```

```
Dimensions of train data X: (79999, 101734) Y : (79999, 500)
Dimensions of test data X: (20000, 101734) Y: (20000, 500)
```

```
In [79]: topfreq = list()
         dict(zip(vectorizer.get_feature_names(),np.array(np.sum(x_train_multilabel,axis=0))[0
         for freq in sorted(np.array(np.sum(x_train_multilabel,axis=0))[0],reverse=True)[0:21]
             if freq in dict(zip(vectorizer.get_feature_names(),np.array(np.sum(x_train_multila
                 topfreq.append(freq)
```

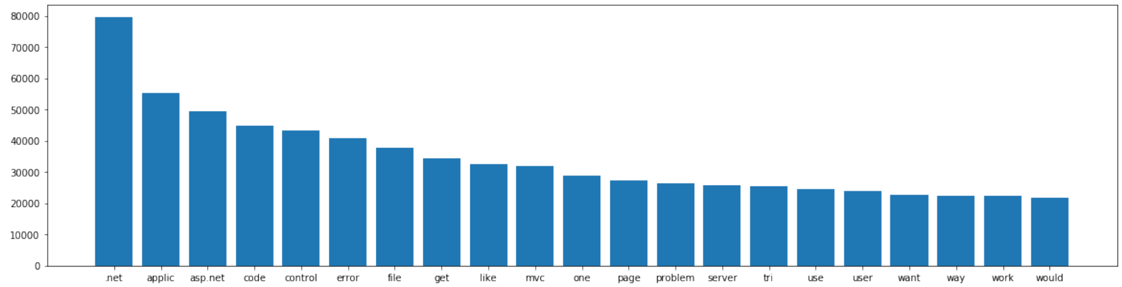
```
In [80]: topword = list()
         for key,value in dict(zip(vectorizer.get_feature_names(),np.array(np.sum(x_train_mult
             if value in topfreq:
                 topword.append(key)
```

```
In [81]: dict(zip(topword,topfreq))
```

```
Out[81]: {'.net': 79583,
          'applic': 55423,
          'asp.net': 49477,
          'code': 44744,
          'control': 43181,
          'error': 40957,
          'file': 37844,
          'get': 34408,
          'like': 32489,
          'mvc': 31785,
          'one': 28685,
          'page': 27333,
          'problem': 26413,
          'server': 25676,
          'tri': 25408,
          'use': 24677,
          'user': 23767,
          'want': 22699,
          'way': 22406,
          'work': 22295,
          'would': 21717}
```

The figure below shows the top 20 most frequently occuring words withing the questions

```
In [82]: plt.figure(figsize=(20,5))
         plt.bar(topword,topfreq);
```



(5.2) Build logistic regression model
Build a logistic regression model and pass it onto the one vs rest classifier

In [86]:
```
start = datetime.now()

if os.path.isfile('LR_on_BOW.pkl') == True:
    classifier = joblib.load('LR_on_BOW.pkl')

if os.path.isfile('LR_on_BOW.pkl') == False:
    classifier = OneVsRestClassifier(LogisticRegression(penalty='l1'), n_jobs=1)
    classifier.fit(x_train_multilabel,y_train)
    joblib.dump(classifier, 'LR_on_BOW.pkl')

predictions = classifier.predict(x_test_multilabel)

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

```
Time taken to run this cell : 0:18:52.419884
```

In [87]:
```
print("Accuracy: ",metrics.accuracy_score(y_test,predictions))
print("Hamming loss: ",metrics.hamming_loss(y_test,predictions))

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

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

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

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

```
Accuracy:  0.17315
Hamming loss:  0.0034015
Micro-average quality numbers
Precision: 0.5718, Recall: 0.3672, F1-measure: 0.4472
Macro-average quality numbers
Precision: 0.4130, Recall: 0.2887, F1-measure: 0.3282
```

1.2 (5.3) Peforming hyperparameter tuning

The code snippet below builds a SGD classifier with log loss and passes this function into the one vs rest model

In [88]:
```
# Here this code show what the name of the parameters are that must be used in grid s
SGD_model = SGDClassifier(loss='log')
model2 = OneVsRestClassifier(SGD_model)
model2.get_params().keys()
```

Out[88]:
```
dict_keys(['estimator__alpha', 'estimator__average', 'estimator__class_weight', 'estim
```

In [89]:
```
# set the various parameter values
params = {'estimator__alpha':[0.00001,0.001,0.01,0.1,100,1000,10000]}
```

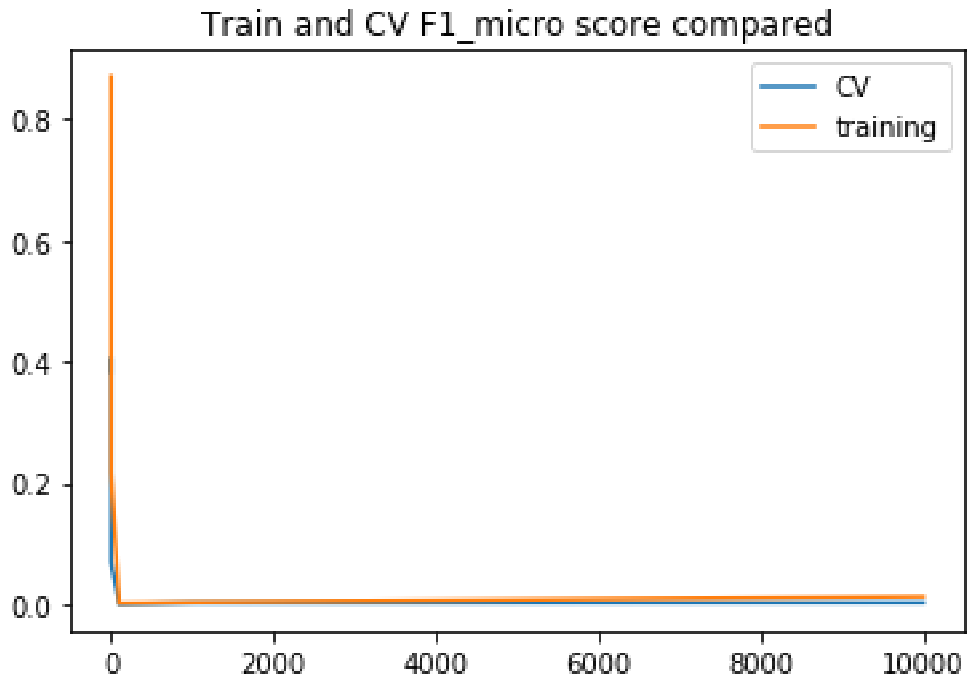
Use gridsearch to perform 3 fold CV for each of the various parameter values of alpha

In [90]:
```
start = datetime.now()

if os.path.isfile('Gridsearch_on_BOW_SGD_Logloss.pkl') == True:
    classifier = joblib.load('Gridsearch_on_BOW_SGD_Logloss.pkl')

if os.path.isfile('Gridsearch_on_BOW_SGD_Logloss.pkl') == False:
    grid = GridSearchCV(estimator=model2, param_grid=params, scoring='f1_micro',n_jobs
    grid.fit(x_train_multilabel,y_train)
    classifier = grid.best_estimator_
    joblib.dump(classifier, 'Gridsearch_on_BOW_SGD_Logloss.pkl')

predictions = classifier.predict(x_test_multilabel)

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

```
Time taken to run this cell : 0:15:10.839273
```

Plot the training and CV performance to ensure the model did not overfit

In [91]:
```
plt.plot(params['estimator__alpha'],dict(grid.cv_results_)['mean_test_score'],label='C
plt.plot(params['estimator__alpha'],dict(grid.cv_results_)['mean_train_score'],label=
plt.title("Train and CV F1_micro score compared")
plt.legend();
```



In [92]:
```
print("Accuracy :",metrics.accuracy_score(y_test, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test,predictions))

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

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

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

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

```
Accuracy : 0.16215
Hamming loss  0.0033435
Micro-average quality numbers
Precision: 0.6162, Recall: 0.2857, F1-measure: 0.3904
Macro-average quality numbers
Precision: 0.4561, Recall: 0.2060, F1-measure: 0.2640
```

1.3 (5.4) Build linear SVM model

Build SGD model with hinge loss, pass this model onto the one vs rest model. Perform grid search with 3 fold CV for the various parameter values of alpha

In [93]:
```
start = datetime.now()

if os.path.isfile('Gridsearch_on_BOW_SGD_Hingeloss.pkl') == True:
    classifier = joblib.load('Gridsearch_on_BOW_SGD_Hingeloss.pkl')

if os.path.isfile('Gridsearch_on_BOW_SGD_Hingeloss.pkl') == False:
    SGD_model = SGDClassifier(loss='hinge')
    model3 = OneVsRestClassifier(SGD_model)
    grid = GridSearchCV(estimator=model3, param_grid=params, scoring='f1_micro',n_jobs
    grid.fit(x_train_multilabel,y_train)
    classifier = grid.best_estimator_
    joblib.dump(classifier, 'Gridsearch_on_BOW_SGD_Hingeloss.pkl')

predictions = classifier.predict(x_test_multilabel)

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

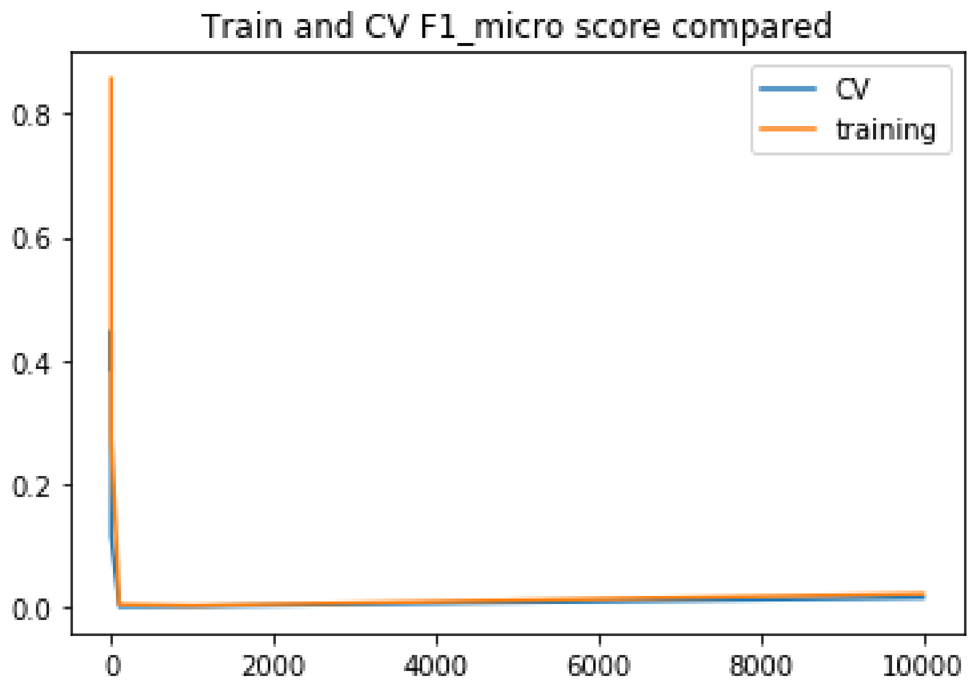
```
Time taken to run this cell : 0:13:52.546718
```

In [94]:
```
print("Accuracy :",metrics.accuracy_score(y_test, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test,predictions))

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

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

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

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

```
Accuracy : 0.19405
Hamming loss  0.0030403
Micro-average quality numbers
Precision: 0.7135, Recall: 0.3152, F1-measure: 0.4373
Macro-average quality numbers
Precision: 0.4954, Recall: 0.2437, F1-measure: 0.3027
```

Plot the training and CV performance to ensure the model did not overfit

```
In [95]: plt.plot(params['estimator__alpha'],dict(grid.cv_results_)['mean_test_score'],label='
         plt.plot(params['estimator__alpha'],dict(grid.cv_results_)['mean_train_score'],label=
         plt.title("Train and CV F1_micro score compared")
         plt.legend();
```




```
In [96]: table = PrettyTable(["Featurerisation", "Model", "Accuracy", "Hamming Loss", "F1-micr
```

```
In [97]: table.add_row(['TF-IDF + Bigram, equal title weight', 'SGD(log)', '22.42%', '0.0028',
         table.add_row(['TF-IDF + Bigram, more title weight', 'SGD(log)', '18.72%', '0.0030',
         table.add_row(['TF-IDF + Bigram, more title weight', 'LR', '18.77%', '0.0030', '0.43'
         table.add_row(['BOW + Trigram, more title weight', 'LR', '17.00%', '0.0034', '0.45',
         table.add_row(['BOW + Trigram, more title weight', 'SGD(log) + par tuning', '16.00%',
         table.add_row(['BOW + Trigram, more title weight', 'SGD(hinge) + par tuning', '16.00%
```

```
In [98]: print(table)
```

```
+-------------------------------------+-------------------------+----------+--------------+----
|           Featurerisation           |          Model          | Accuracy | Hamming Loss | F1-
+-------------------------------------+-------------------------+----------+--------------+----
| TF-IDF + Bigram, equal title weight |         SGD(log)        |  22.42%  |    0.0028    |    
|  TF-IDF + Bigram, more title weight |         SGD(log)        |  18.72%  |    0.0030    |    
|  TF-IDF + Bigram, more title weight |            LR           |  18.77%  |    0.0030    |    
|   BOW + Trigram, more title weight  |            LR           |  17.00%  |    0.0034    |    
```

BOW + Trigram, more title weight	SGD(log) + par tuning	16.00%	0.0030	[illegible]
BOW + Trigram, more title weight	SGD(hinge) + par tuning	16.00%	0.0030	[illegible]