Stack Overflow: Tag Prediction

1. Business Problem

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

1.1 Sources and Refernces:

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

Youtube : https://youtu.be/nNDqbUhtlRg (https://youtu.be/nNDqbUhtlRg)

Research paper : https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/tagging-1.pdf (https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/tagging-1.pdf)

Research paper : https://dl.acm.org/citation.cfm?id=2660970&dl=ACM&coll=DL (https://dl.acm.org/citation.cfm?id=2660970&dl=ACM&coll=DL)

1.2 Business 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:

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

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

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

Size of Train.csv - 6.75GB

Size of Test.csv - 2GB

Number of rows in Train.csv = 6034195

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

Data Field Explaination

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

Id - Unique identifier for each question

Title - The question's title

Body - The body of the question

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

3. Exploratory Data Analysis:

3.1 Data Loading and Cleaning

3.1.1 Using Pandas with SQLite to Load the data

In [1]:

```
from google.colab import files
files.upload()
```

Choose Files No file chosen

Upload widget is only available when the cell has been executed in the current browser session.
Please rerun this cell to enable.

```
Saving kaggle.json to kaggle.json
```

Out[1]:

```
{'kaggle.json': b'{"username":"shamimio","key":"e57ea3117a95a3194da0e8aaa7
da68b7"}'}
```

In [0]:

```
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/

#changeb the permmission
!chmod 600 ~/.kaggle/kaggle.json
```

In [3]:

```
!kaggle competitions download -c facebook-recruiting-iii-keyword-extraction
```

```
Downloading Train.zip to /content
100% 2.18G/2.19G [00:33<00:00, 61.9MB/s]
100% 2.19G/2.19G [00:33<00:00, 69.5MB/s]
Downloading Test.zip to /content
100% 722M/725M [00:12<00:00, 51.7MB/s]
100% 725M/725M [00:12<00:00, 58.8MB/s]
Downloading SampleSubmission.csv to /content
 98% 77.0M/78.7M [00:01<00:00, 47.9MB/s]
100% 78.7M/78.7M [00:01<00:00, 67.6MB/s]
```

In [4]:

```
from zipfile import ZipFile
file_name = "Train.zip"

with ZipFile(file_name, 'r') as zip_file:
  zip_file.extractall()
  print("Done")
```

```
Done
```

In [5]:

```
pip install scikit-multilearn
```

```
Collecting scikit-multilearn
  Downloading https://files.pythonhosted.org/packages/bb/1f/e6ff649c72a1cd
f2c7a1d31eb21705110ce1c5d3e7e26b2cc300e1637272/scikit_multilearn-0.2.0-py3
-none-any.whl (89kB)
    100% |████████████████████████████████| 92kB 7.2MB/s
Installing collected packages: scikit-multilearn
Successfully installed scikit-multilearn-0.2.0
```

In [1]:

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

In [2]:

```
#Creating db file from csv
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)
```

In [3]:

```
#Creating a new database with 500k Points
if not os.path.isfile('data_new.db'):
    con = sqlite3.connect('train.db')
    disk_dup = create_engine("sqlite:///data_new.db")
    #no_dup = pd.DataFrame(df_no_dup, columns=['Title', 'Body', 'Tags'])

    new = pd.read_sql_query('SELECT * FROM data LIMIT 500001', con)

    new.to_sql('data_new',disk_dup)
    con.close()
```

In [4]:

```
# Cointing number of rows

if os.path.isfile('data_new.db'):
    start = datetime.now()
    con = sqlite3.connect('data_new.db')
    num_rows = pd.read_sql_query("""SELECT count(*) FROM data_new""", 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
te train.db file")
```

```
Number of rows in the database :
 500001
Time taken to count the number of rows : 0:00:08.165739
```

3.1.2 Checking for duplicates

In [10]:

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

```
Time taken to run this cell : 0:00:06.500339
```

In [11]:

```
df_no_dup.head()
```

Out[11]:

	Title	Body	Tags	cnt_dup
0	"SQL Injection" issue preventing correct for...	<p>So I've been checking everything I can thin...	php forms	1
1	f a continuous function in [0,1], Show: $\l...	<p>Let f be a continuous function in [0,1] a...	calculus	1
2	*** Exception: Prelude.read: no parse in Hask...	<p>This portion of code should read in two or ...	parsing haskell expression	1
3	500 Internal Server Error in ASP.NET MVC	<p>I am working in ASP.NET MVC. I am using par...	asp.net-mvc	1
4	Accessing @Local Session Bean from an exposed...	<p>What I am trying to do should be very strai...	ejb resteasy	2

In [12]:

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

```
number of duplicate questions : 14164 ( 2.832794334411326 % )
```

In [13]:

```
# number of times each question appeared in our database
df_no_dup.cnt_dup.value_counts()
```

Out[13]:

```
1    471819
2     13872
3       146
Name: cnt_dup, dtype: int64
```

In [14]:

```
#adding a new feature number of tags per question

start = datetime.now()
df_no_dup["tag_count"] = df_no_dup["Tags"].apply(lambda text:len(text.split(" ")))
print("Time taken to run this cell :", datetime.now() - start)
df_no_dup.head()
```

```
Time taken to run this cell : 0:00:00.393995
```

Out[14]:

	Title	Body	Tags	cnt_dup	tag_count
0	"SQL Injection" issue preventing correct for...	<p>So I've been checking everything I can thin...	php forms	1	2
1	f a continuous function in [0,1], Show: $\l...	<p>Let f be a continuous function in [0,1] a...	calculus	1	1
2	*** Exception: Prelude.read: no parse in Hask...	<p>This portion of code should read in two or ...	parsing haskell expression	1	3
3	500 Internal Server Error in ASP.NET MVC	<p>I am working in ASP.NET MVC. I am using par...	asp.net-mvc	1	1
4	Accessing @Local Session Bean from an exposed...	<p>What I am trying to do should be very strai...	ejb resteasy	2	2

In [15]:

```
# number of tags per qurstion
df_no_dup.tag_count.value_counts()
```

Out[15]:

```
3    139065
2    129207
4     93550
1     66624
5     57391
Name: tag_count, dtype: int64
```

In [5]:

```
# creating a new datbse 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 [6]:

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

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

```
Time taken to run this cell : 0:00:07.836965
```

3.2 Analysis of Tags

3.2.1 Total number of unique tags

In [7]:

```
# 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 [8]:

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

```
Number of data points : 485836
Number of unique tags : 30429
```

In [9]:

```
#'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', '.aspxauth', '.bash-profile', '.
class-file', '.doc', '.each', '.emf', '.exe', '.hgtags']
```

3.2.3 Number of times a tag appeared

In [15]:

```
# 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 [10]:

```
#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[10]:

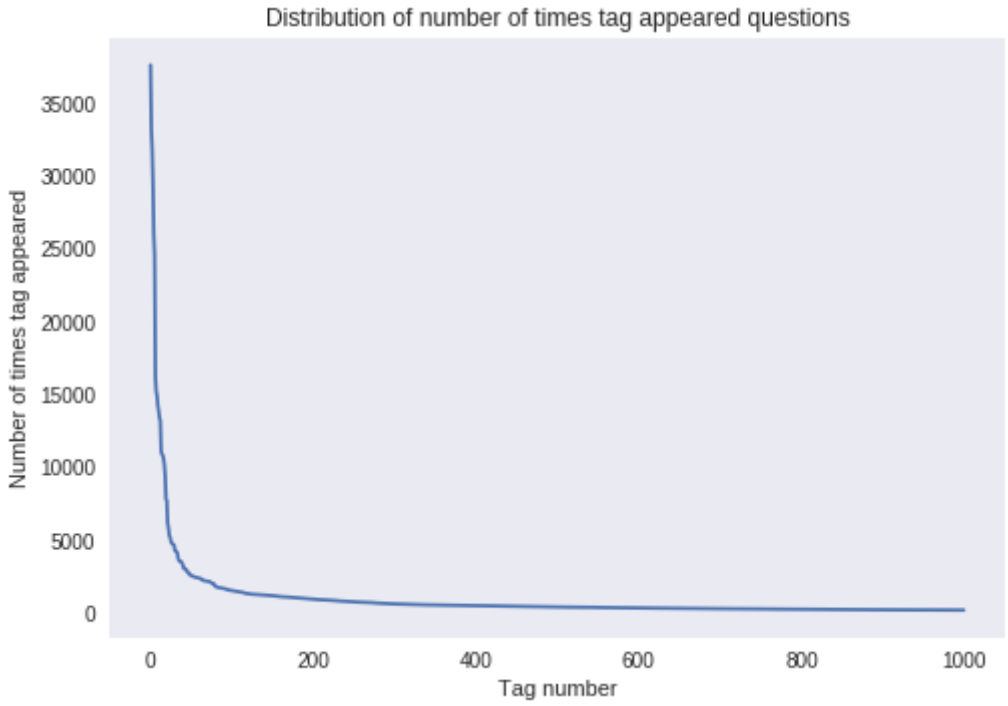
	Tags	Counts
0	.a	3
1	.app	8
2	.aspxauth	3
3	.bash-profile	13
4	.class-file	5

In [73]:

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

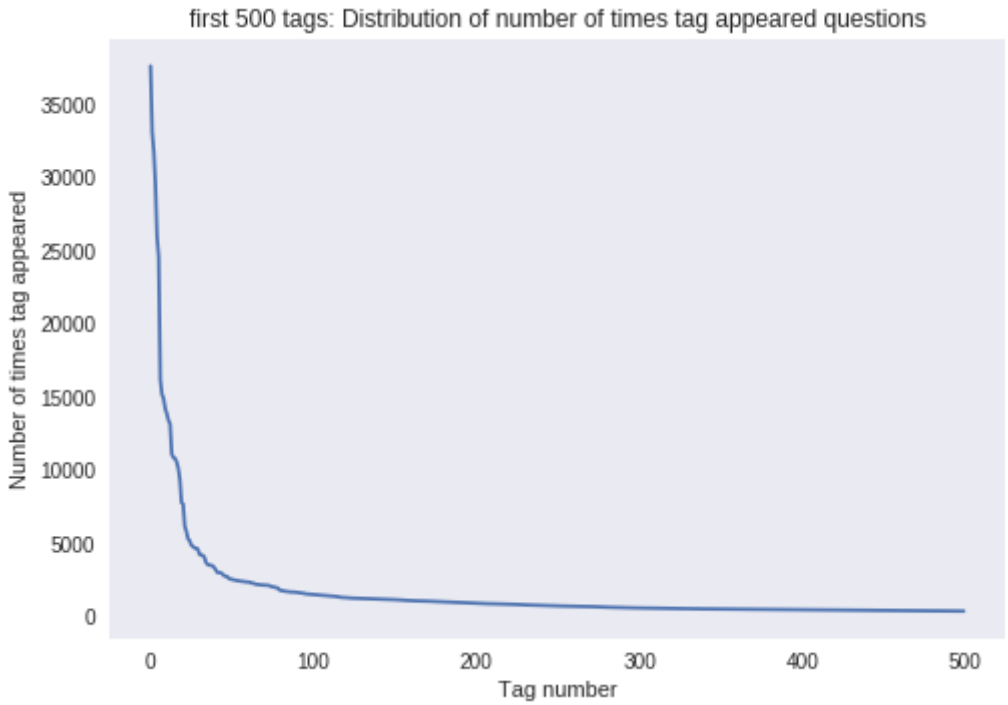
In [24]:

```
plt.plot(tag_counts[:1000])
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 [25]:

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



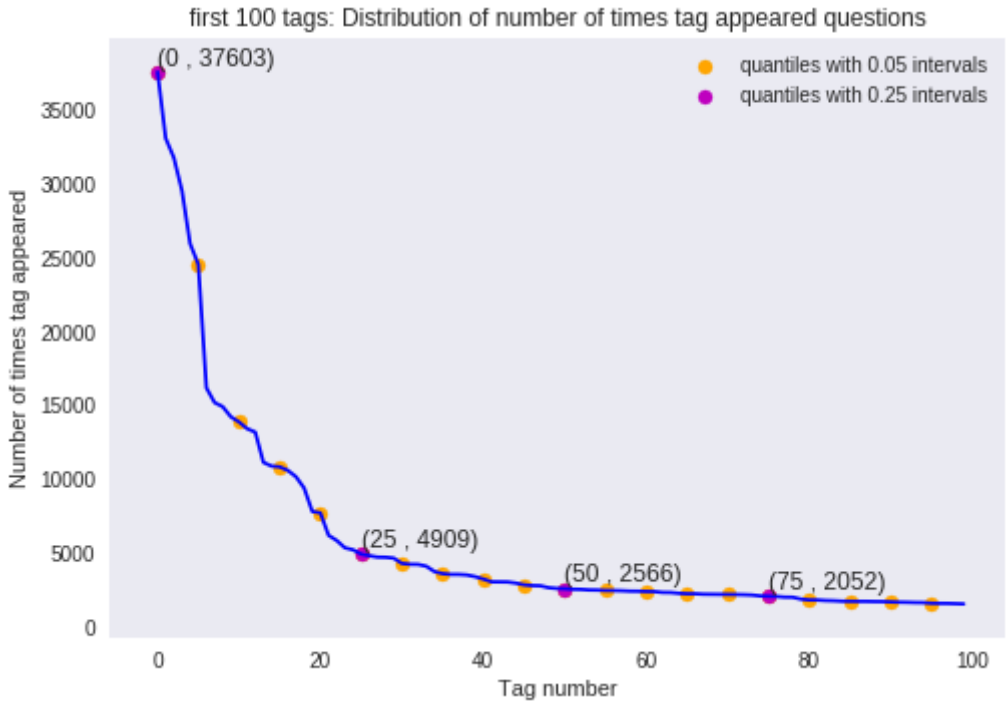

```
100 [37603 24567 13873 10826  7706  4909  4279  3585  3221  2827  2566  24
42
  2376  2222  2176  2052  1801  1708  1670  1576  1529  1480  1429  1370
  1315  1279  1257  1234  1215  1199  1178  1138  1109  1092  1072  1055
  1030   998   981   949   924   910   900   881   865   854   822   802
   790   769   747   739   725   716   701   686   664   646   636   620
   612   602   594   591   577   573   562   555   548   541   539   534
   525   514   506   502   496   494   487   484   479   475   470   462
   457   453   447   447   441   438   434   429   427   421   419   416
   407   404   402   400]
```

In [26]:

```
plt.plot(tag_counts[0:100], c='b')
plt.scatter(x=list(range(0,100,5)), y=tag_counts[0:100:5], c='orange', label="quantiles
with 0.05 intervals")
# quantiles with 0.25 difference
plt.scatter(x=list(range(0,100,25)), y=tag_counts[0:100:25], c='m', label = "quantiles
 with 0.25 intervals")

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

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




```
20 [37603 24567 13873 10826  7706  4909  4279  3585  3221  2827  2566  244
2
  2376  2222  2176  2052  1801  1708  1670  1576]
```

In [11]:

```
# Store tags greater than 10K in one list
lst_tags_gt_10k = tag_df[tag_df.Counts>5000].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>10000].Tags
#Print the length of the list.
print ('{} Tags are used more than 100000 times'.format(len(lst_tags_gt_100k)))
```

```
25 Tags are used more than 10000 times
18 Tags are used more than 100000 times
```

3.2.4 Tags Per Question

In [13]:

```
# Storing the count of tag in each question in the list 'tag_count
tag_quest_count = tag_dtm.sum(axis=1).tolist()

# converting list of lists into single list,we will get [[3], [4], [2], [2], [3]] and w
e are converting this to [3, 4, 2, 2, 3]
tag_quest_count=[int(j) for i in tag_quest_count for j in i]
print('We have total {} datapoints.'.format(len(tag_quest_count)))
print(tag_quest_count[:5])
```

```
We have total 485836 datapoints.
[1, 3, 1, 2, 4]
```

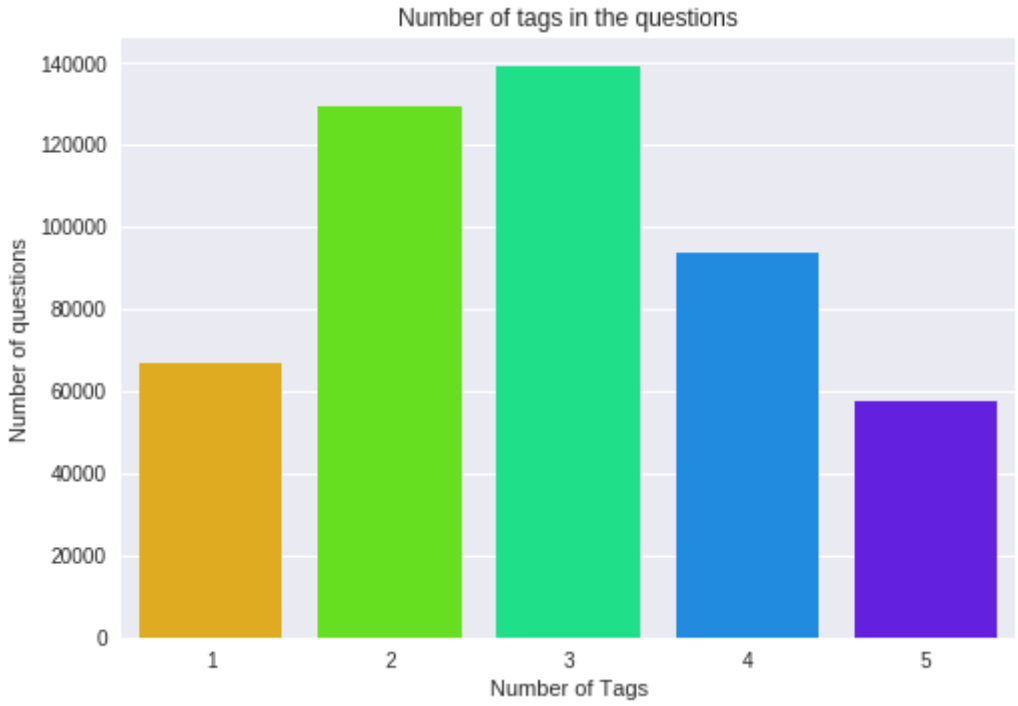
In [14]:

```
print ("Maximum no of tag per question: %d"%max(tag_quest_count))
print ("Minimum no of tags per quest: %d"%min(tag_quest_count))
print ("Avg number of tags per qudestion: %f"% ((sum(tag_quest_count)*1.0)/len(tag_ques
t_count)))
```

```
Maximum no of tag per question: 5
Minimum no of tags per quest: 1
Avg number of tags per qudestion: 2.888600
```

In [30]:

```
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

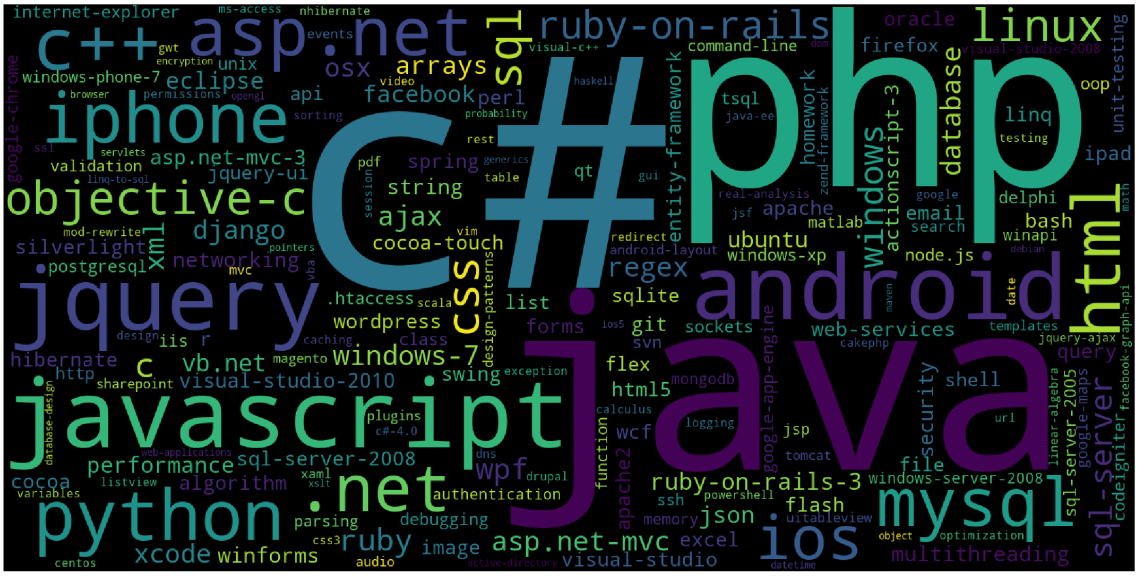
3.2.5 Most Frequent Tags

In [16]:

```
# 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)
```



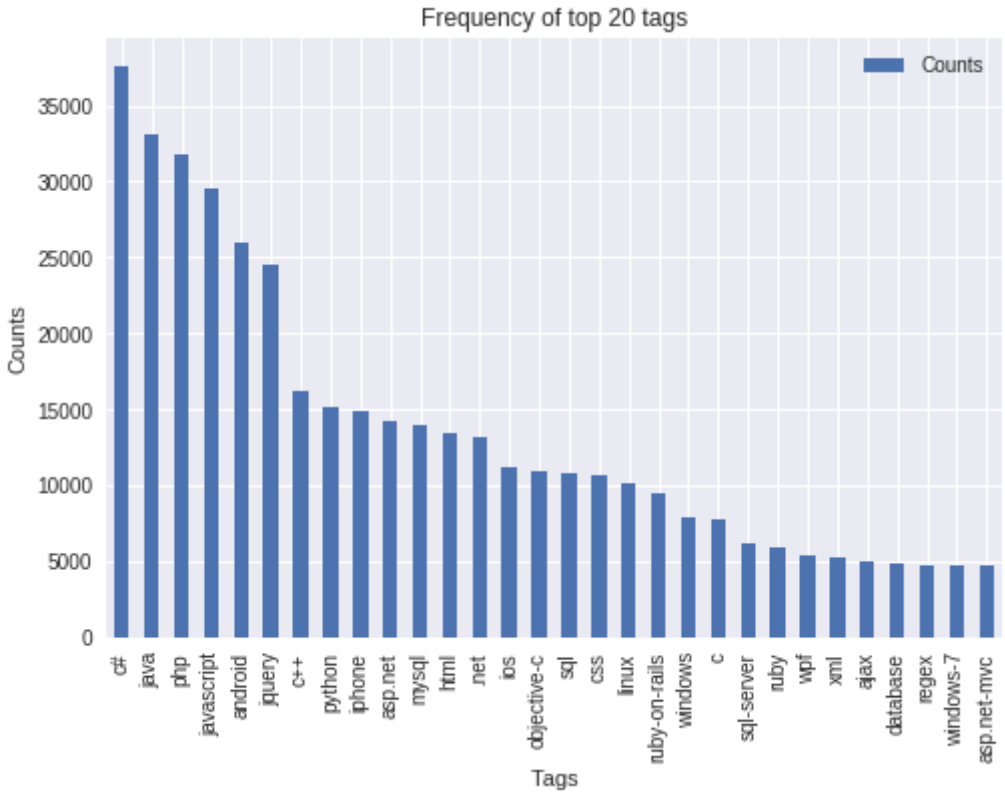
Time taken to run this cell : 0:00:06.151975

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 [32]:

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



Observations:

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

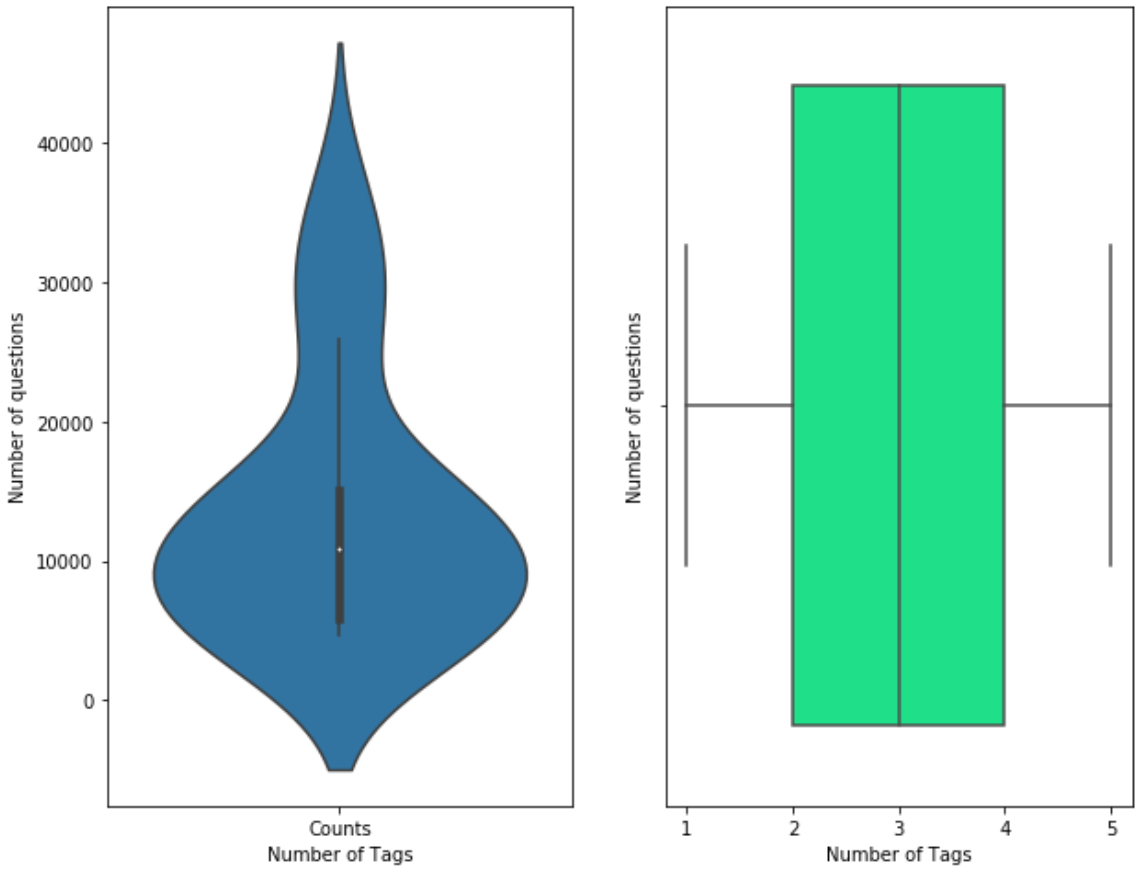
In [96]:

```
# Obsetving the quantiles using the violin plot and box .

plt.figure(figsize=(10, 8))

plt.subplot(1,2,1)
sns.violinplot(data = tag_df_sorted.head(30) , )
plt.xlabel("Number of Tags")
plt.ylabel("Number of questions")

plt.subplot(1,2,2)
sns.boxplot(tag_quest_count, palette='gist_rainbow')
plt.xlabel("Number of Tags")
plt.ylabel("Number of questions")
plt.show()
```

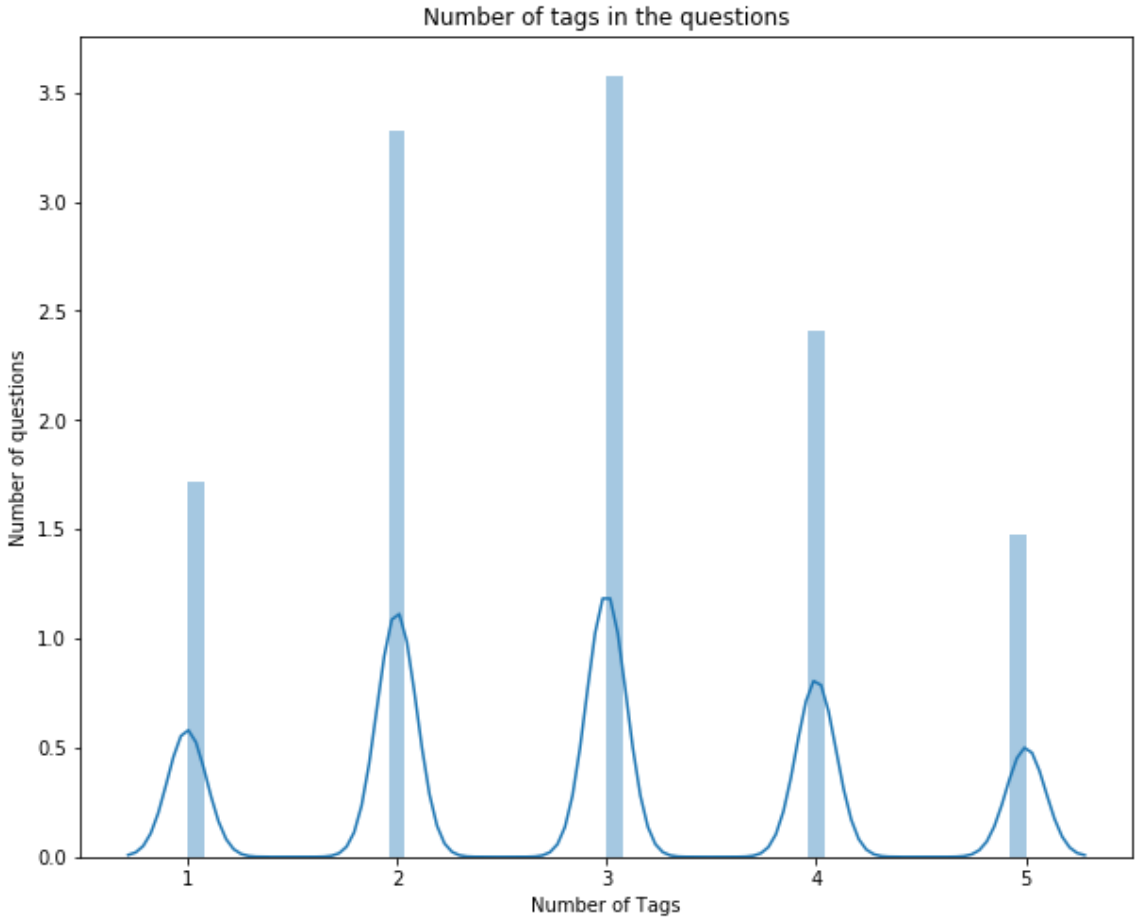


Observation: Most of the tags are acquired by 10000 questions and has 3 tags.

In [84]:

```
# plot for average no of tags per question

plt.figure(figsize=(10, 8))
plt.subplot()
sns.distplot(tag_quest_count)
plt.title("Number of tags in the questions")
plt.xlabel("Number of Tags")
plt.ylabel("Number of questions")
plt.show()
```



Observation: Most of the questions have 3 tags in average and there is an average of 10000 questions with 3 tags, 5000 questions with 1 tag and so on.

3.3 Cleaning and preprocessing of Questions

3.3.1 Preprocessing

In [33]:

```
import nltk
nltk.download('stopwords')
```

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

Out[33]:

```
True
```

In [2]:

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

In [3]:

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

    return None

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

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

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

In [4]:

```
sql_create_table = """CREATE TABLE IF NOT EXISTS QuestionsProcessed (question text NOT
 NULL, code text, tags text, words_pre integer, words_post integer, is_code intege
r);"""
create_database_table("Titlemoreweight.db", sql_create_table)
```

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

In [5]:

```
# 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 = '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 ORDER BY RANDOM();")
        # for selecting random points
        #reader.execute("SELECT Title, Body, Tags From no_dup_train ORDER BY RANDOM() L
IMIT 500001;")

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

```
Tables in the databse:
QuestionsProcessed
Cleared All the rows
Time taken to run this cell : 0:00:52.538222
```

Preprocessing of questions:

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

In [10]:

```
import nltk
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Out[10]:

True

In [6]:

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

    is_code = 0

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


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

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

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

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

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

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

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

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

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

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

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

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

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

```
number of questions completed= 100000
number of questions completed= 200000
number of questions completed= 300000
number of questions completed= 400000
Avg. length of questions(Title+Body) before processing: 1150
Avg. length of questions(Title+Body) after processing: 410
Percent of questions containing code: 56
Time taken to run this cell : 0:21:47.817606
```

In [7]:

```
# 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 [8]:

```
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
================================================================================
==========================
('loos object git gui refer loos object git gui refer loos object git gui
refer open git gui get popup messag refer loos object git gc remov messag
loos object could prevent occur',)
--------------------------------------------------------------------------------
-------------------------
('solv system equat unknown equat solv system equat unknown equat solv sys
tem equat unknown equat solv system equat first-year physic told abl find
solut system unknown one exist need least equat system look known quantiti
theta theta cos theta mg sin theta frac ell sin theta cos theta mg sin the
ta frac ell sin theta system equat unknown quantiti ell abl success solv f
our equat logic error understand',)
--------------------------------------------------------------------------------
-------------------------
('audio clip loop continu audio clip loop continu audio clip loop continu
anyon point right direct code play audio clip continu play stop',)
--------------------------------------------------------------------------------
-------------------------
('linqpad system.net trace linqpad system.net trace linqpad system.net tra
ce tri trace linqpad script setup system.net trace accord http msdn.micros
oft.com en-us librari ty48b824.aspx put follow line file file creat contai
n log linqpad version check howev run linqpad script use webrequest downlo
ad url request log linqpad script log possibl log',)
--------------------------------------------------------------------------------
-------------------------
('figur system approv post 10 user like figur system approv post 10 user l
ike figur system approv post 10 user like vote system articl articl store
stori tabl vote store vote tabl id stori tabl equal item name vote tabl th
erefor vote relat articl item name want make sum vote get 10 updat show fi
eld stori tabl valu queri use insert vote databas think add someth creat a
noth queri sum vote valu articl user vote see 10 yes set show databas stru
ctur stori tabl vote tabl',)
--------------------------------------------------------------------------------
-------------------------
('automat append execut line script file automat append execut line script
file automat append execut line script file host compani requir put top ev
eri rb file easi way use sed recurs someth go append top everi rb file',)
--------------------------------------------------------------------------------
-------------------------
('use runnabl android use runnabl android use runnabl android hi newb java
android develop use runnabl android look work nhere sourc code maintest.ja
va discovery.java main.xml execut discoveri class applic start updat textv
iew maintest class happen fix behaviour think runnabl problem',)
--------------------------------------------------------------------------------
-------------------------
('want develop android app base fbreader know use want develop android app
base fbreader know use want develop android app base fbreader know use exa
mpl obvious question nhow launch mainview filepath mmt sdcard ebook aaa.tx
t nwhere function nthank nif blog bbs learn mani thing fbreader pleas tel
l',)
--------------------------------------------------------------------------------
-------------------------
('error 404 form method post error 404 form method post error 404 form met
hod post form want sent googl api creat chart time click submit get error
404 messag see http bieresomnifiere.iblogger.org page id 14 nstrang site w
ork outsid wordpress http bieresomnifiere.iblogger.org yeast.html also che
ck list reserv word within wordpress non appear conflict variabl anyon ide
a caus problem nthank',)
```

```
------------------------------------------------------------------------
-------------------------
```

Saving Preprocessed data to a Database

In [9]:

```
# taking the data to tghe dataframe
write_db = 'Titlemoreweight.db'
if os.path.isfile(write_db):
    conn_r = create_connection(write_db)
    if conn_r is not None:
        preprocessed_data = pd.read_sql_query("""SELECT question, Tags FROM QuestionsPr
ocessed""", conn_r)
conn_r.commit()
conn_r.close()
```

In [10]:

```
preprocessed_data.head()
```

Out[10]:

	question	tags
0	select employe hql select employe hql select e...	hibernate hql
1	loos object git gui refer loos object git gui ...	git
2	solv system equat unknown equat solv system eq...	algebra-precalculus
3	audio clip loop continu audio clip loop contin...	java javasound
4	linqpad system.net trace linqpad system.net tr...	httpwebrequest linqpad tracing system.net

In [11]:

```
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 : 485836
number of dimensions : 2
```

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

Converting string Tags to multilable output variables :

In [12]:

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

In [13]:

```
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))
```

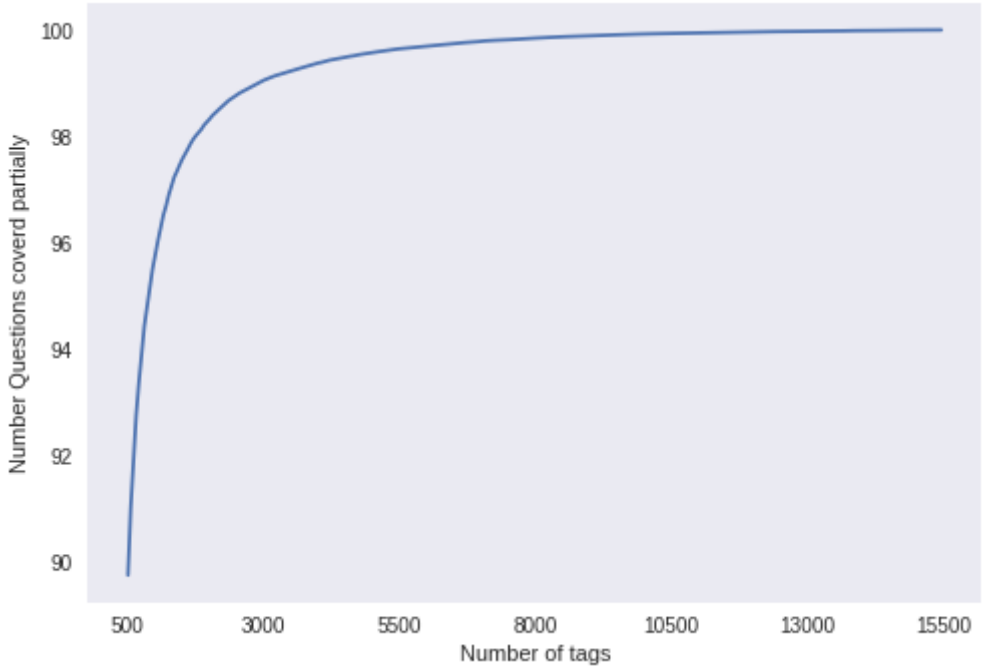
Selecting 500 Tags

In [14]:

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

In [48]:

```
fig, ax = plt.subplots()
ax.plot(questions_explained)
xlabel = list(500+np.array(range(-50,450,50))*50)
ax.set_xticklabels(xlabel)
plt.xlabel("Number of tags")
plt.ylabel("Number Questions coverd partially")
plt.grid()
plt.show()
# you can choose any number of tags based on your computing power, minimun is 500(it co
vers 90% of the tags)
print("with ",5500,"tags we are covering ",questions_explained[50],"% of questions")
print("with ",500,"tags we are covering ",questions_explained[0],"% of questions")
```




```
with  5500 tags we are covering  99.056 % of questions
with  500 tags we are covering  89.747 % of questions
```

In [15]:

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

```
number of questions that are not covered : 49812 out of  485836
```

In [16]:

```
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 [17]:

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

```
Number of data points in train data : (400000, 500)
Number of data points in test data : (85836, 500)
```

In [18]:

```
from sklearn.externals import joblib
joblib.dump(x_train, 'x_train_1.pkl')

joblib.dump(x_test, 'x_test_1.pkl')

joblib.dump(y_train, 'y_train_1.pkl')

joblib.dump(y_test, 'y_test_1.pkl')
```

Out[18]:

```
['y_test_1.pkl']
```

Featurization:

In [19]:

```
x_train=joblib.load( 'x_train_1.pkl')
x_test=joblib.load( 'x_test_1.pkl')

y_train=joblib.load( 'y_train_1.pkl')

y_test=joblib.load('y_test_1.pkl')
```

In [20]:

```
start = datetime.now()
vectorizer = CountVectorizer(min_df=0.00009, max_features=40000, tokenizer = lambda x:
x.split(), ngram_range=(1,2))
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 : 0:01:39.780679
```

In [21]:

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

```
Dimensions of train data X: (400000, 40000) Y : (400000, 500)
Dimensions of test data X: (85836, 40000) Y: (85836, 500)
```

Applying Logistic Regression with OneVsRest Classifier

In [29]:

```
#from sklearn.grid_search import GridSearchCV
from sklearn.model_selection import GridSearchCV

model_to_set = OneVsRestClassifier(SGDClassifier(loss='log', penalty='l1'))

parameters = {
    "estimator__alpha": [10**-5, 10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1]
}

model_tunning = GridSearchCV(model_to_set, param_grid=parameters, scoring='f1_micro',n_
jobs=-1)

model_tunning.fit(x_train_multilabel, y_train)

print (model_tunning.best_score_)
print (model_tunning.best_params_)
```

```
0.4466043958066712
{'estimator__alpha': 0.0001}
```

In [31]:

```
start = datetime.now()
#best_alpha = gsv.best_estimator_.get_params()['estimator__alpha']
classifier = OneVsRestClassifier(SGDClassifier(loss='log', alpha=0.0001, penalty='l1'),
n_jobs=-1)
classifier.fit(x_train_multilabel, y_train)
predictions = classifier.predict (x_test_multilabel)

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


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

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

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

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

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

```
Accuracy : 0.14983223822172514
Hamming loss  0.003888252015471364
Micro-average quality numbers
Precision: 0.4552, Recall: 0.4537, F1-measure: 0.4545
Macro-average quality numbers
Precision: 0.3705, Recall: 0.3692, F1-measure: 0.3638
              precision    recall  f1-score   support

           0       0.48      0.42      0.45      6820
           1       0.57      0.56      0.57      5931
           2       0.66      0.61      0.63      5555
           3       0.57      0.52      0.54      5232
           4       0.79      0.82      0.80      4612
           5       0.70      0.70      0.70      4352
           6       0.61      0.62      0.61      2870
           7       0.69      0.72      0.70      2636
           8       0.53      0.51      0.52      2710
           9       0.54      0.53      0.53      2508
          10       0.64      0.69      0.66      2416
          11       0.34      0.26      0.29      2384
          12       0.27      0.25      0.26      2355
          13       0.41      0.39      0.40      1974
          14       0.44      0.33      0.38      1956
          15       0.43      0.40      0.41      1850
          16       0.56      0.60      0.58      1923
          17       0.39      0.36      0.37      1711
          18       0.55      0.63      0.58      1572
          19       0.19      0.18      0.18      1374
          20       0.36      0.37      0.37      1360
          21       0.35      0.43      0.39      1041
          22       0.50      0.54      0.52      1002
          23       0.70      0.72      0.71       929
          24       0.51      0.55      0.53       914
          25       0.42      0.46      0.44       861
          26       0.24      0.17      0.20       831
          27       0.70      0.77      0.73       850
          28       0.33      0.37      0.35       841
          29       0.41      0.39      0.40       829
          30       0.36      0.31      0.33       743
          31       0.49      0.39      0.43       785
          32       0.76      0.81      0.78       756
          33       0.41      0.44      0.42       763
          34       0.53      0.50      0.51       650
          35       0.54      0.62      0.58       648
          36       0.56      0.70      0.62       625
          37       0.58      0.56      0.57       609
          38       0.29      0.31      0.30       595
          39       0.45      0.44      0.44       588
          40       0.15      0.22      0.18       571
          41       0.17      0.26      0.21       516
          42       0.50      0.49      0.49       551
          43       0.25      0.25      0.25       516
          44       0.43      0.44      0.43       534
          45       0.38      0.41      0.40       486
          46       0.42      0.37      0.39       510
          47       0.22      0.22      0.22       459
          48       0.55      0.55      0.55       452
          49       0.66      0.79      0.72       468
          50       0.14      0.17      0.16       455
          51       0.20      0.11      0.14       465
          52       0.58      0.42      0.49       424
```

53	0.24	0.18	0.20	469
54	0.44	0.54	0.48	437
55	0.26	0.27	0.26	422
56	0.30	0.42	0.35	425
57	0.26	0.29	0.28	397
58	0.22	0.25	0.24	418
59	0.52	0.57	0.54	420
60	0.58	0.61	0.59	420
61	0.78	0.83	0.81	422
62	0.83	0.77	0.80	414
63	0.13	0.13	0.13	391
64	0.61	0.62	0.62	401
65	0.22	0.22	0.22	364
66	0.54	0.59	0.56	377
67	0.42	0.55	0.48	393
68	0.41	0.44	0.42	394
69	0.64	0.36	0.47	395
70	0.45	0.56	0.50	388
71	0.37	0.35	0.36	404
72	0.33	0.47	0.39	380
73	0.36	0.29	0.32	359
74	0.10	0.05	0.06	364
75	0.70	0.58	0.63	349
76	0.71	0.76	0.73	371
77	0.48	0.46	0.47	369
78	0.58	0.50	0.54	339
79	0.08	0.11	0.09	318
80	0.19	0.18	0.18	325
81	0.47	0.59	0.52	331
82	0.30	0.34	0.32	275
83	0.24	0.25	0.25	307
84	0.32	0.47	0.38	272
85	0.50	0.57	0.54	313
86	0.58	0.54	0.56	279
87	0.33	0.37	0.35	291
88	0.73	0.70	0.72	305
89	0.72	0.66	0.69	293
90	0.29	0.31	0.30	290
91	0.77	0.72	0.74	306
92	0.70	0.63	0.66	274
93	0.37	0.43	0.40	270
94	0.60	0.62	0.61	302
95	0.28	0.33	0.31	285
96	0.16	0.17	0.16	266
97	0.35	0.32	0.34	301
98	0.37	0.47	0.41	261
99	0.66	0.78	0.71	273
100	0.27	0.18	0.22	269
101	0.70	0.77	0.73	264
102	0.57	0.72	0.64	240
103	0.39	0.29	0.33	281
104	0.40	0.34	0.37	262
105	0.76	0.79	0.78	262
106	0.35	0.46	0.40	264
107	0.11	0.12	0.11	287
108	0.27	0.37	0.31	257
109	0.27	0.31	0.29	239
110	0.59	0.59	0.59	237
111	0.22	0.26	0.24	245
112	0.16	0.07	0.10	237
113	0.79	0.79	0.79	248

114	0.30	0.18	0.22	253
115	0.25	0.25	0.25	247
116	0.29	0.31	0.30	235
117	0.37	0.28	0.32	243
118	0.56	0.52	0.54	262
119	0.19	0.12	0.15	253
120	0.27	0.23	0.25	212
121	0.72	0.81	0.76	208
122	0.43	0.56	0.49	228
123	0.12	0.13	0.13	241
124	0.32	0.34	0.33	235
125	0.39	0.38	0.39	226
126	0.72	0.73	0.73	234
127	0.22	0.21	0.21	234
128	0.40	0.33	0.36	220
129	0.14	0.17	0.15	255
130	0.19	0.19	0.19	223
131	0.70	0.83	0.76	205
132	0.26	0.30	0.28	205
133	0.37	0.44	0.40	209
134	0.50	0.59	0.54	214
135	0.46	0.36	0.40	224
136	0.60	0.68	0.64	212
137	0.53	0.50	0.52	217
138	0.24	0.34	0.28	204
139	0.15	0.08	0.10	211
140	0.50	0.57	0.53	182
141	0.30	0.29	0.30	183
142	0.86	0.67	0.75	241
143	0.12	0.15	0.13	220
144	0.35	0.45	0.39	195
145	0.18	0.20	0.19	253
146	0.15	0.12	0.13	201
147	0.58	0.56	0.57	204
148	0.09	0.14	0.11	194
149	0.30	0.46	0.36	213
150	0.32	0.45	0.38	195
151	0.36	0.35	0.35	208
152	0.19	0.24	0.21	187
153	0.72	0.88	0.79	174
154	0.03	0.01	0.02	206
155	0.88	0.84	0.86	230
156	0.21	0.32	0.25	182
157	0.55	0.52	0.53	208
158	0.10	0.12	0.11	197
159	0.50	0.31	0.38	207
160	0.51	0.13	0.21	167
161	0.14	0.16	0.15	208
162	0.56	0.78	0.65	196
163	0.17	0.22	0.19	199
164	0.88	0.76	0.81	186
165	0.48	0.60	0.53	182
166	0.20	0.31	0.25	179
167	0.46	0.46	0.46	203
168	0.35	0.24	0.29	205
169	0.82	0.77	0.79	195
170	0.28	0.38	0.32	195
171	0.12	0.17	0.14	192
172	0.38	0.41	0.40	206
173	0.22	0.20	0.21	192
174	0.18	0.15	0.17	181

175	0.37	0.35	0.36	187
176	0.27	0.33	0.30	178
177	0.24	0.38	0.29	171
178	0.67	0.55	0.61	173
179	0.53	0.66	0.59	183
180	0.11	0.04	0.06	186
181	0.38	0.55	0.45	152
182	0.19	0.23	0.21	170
183	0.11	0.12	0.12	177
184	0.31	0.05	0.09	207
185	0.47	0.51	0.49	172
186	0.52	0.53	0.53	182
187	0.48	0.55	0.51	165
188	0.27	0.23	0.25	179
189	0.61	0.77	0.68	166
190	0.39	0.45	0.42	177
191	0.12	0.09	0.10	173
192	0.13	0.15	0.14	179
193	0.54	0.69	0.60	153
194	0.48	0.38	0.42	183
195	0.61	0.55	0.58	181
196	0.45	0.50	0.48	163
197	0.39	0.44	0.41	154
198	0.84	0.69	0.76	147
199	0.37	0.19	0.25	189
200	0.13	0.16	0.15	177
201	0.25	0.29	0.26	154
202	0.65	0.75	0.69	179
203	0.88	0.85	0.86	158
204	0.19	0.28	0.22	162
205	0.20	0.26	0.22	162
206	0.86	0.88	0.87	157
207	0.19	0.15	0.17	155
208	0.42	0.34	0.38	147
209	0.47	0.24	0.32	136
210	0.09	0.10	0.10	146
211	0.13	0.04	0.06	170
212	0.32	0.47	0.38	137
213	0.05	0.08	0.06	143
214	0.23	0.20	0.21	163
215	0.80	0.85	0.82	162
216	0.24	0.24	0.24	147
217	0.19	0.11	0.14	163
218	0.32	0.34	0.33	175
219	0.21	0.16	0.18	147
220	0.17	0.17	0.17	144
221	0.40	0.46	0.42	148
222	0.02	0.02	0.02	167
223	0.26	0.31	0.28	154
224	0.20	0.20	0.20	147
225	0.20	0.14	0.16	146
226	0.23	0.32	0.27	157
227	0.45	0.40	0.42	152
228	0.66	0.66	0.66	146
229	0.17	0.27	0.20	138
230	0.52	0.64	0.57	127
231	0.58	0.75	0.65	151
232	0.25	0.29	0.27	139
233	0.56	0.48	0.52	160
234	0.56	0.50	0.53	151
235	0.29	0.26	0.28	141

236	0.08	0.13	0.10	138
237	0.15	0.19	0.16	134
238	0.50	0.60	0.54	131
239	0.05	0.05	0.05	149
240	0.78	0.78	0.78	152
241	0.38	0.21	0.27	139
242	0.29	0.39	0.33	127
243	0.25	0.27	0.26	143
244	0.23	0.20	0.21	130
245	0.10	0.16	0.12	119
246	0.35	0.39	0.37	160
247	0.08	0.19	0.11	120
248	0.16	0.21	0.18	137
249	0.20	0.11	0.14	132
250	0.10	0.06	0.07	121
251	0.32	0.24	0.27	134
252	0.23	0.33	0.27	130
253	0.50	0.39	0.44	118
254	0.52	0.58	0.55	145
255	0.68	0.68	0.68	145
256	0.30	0.40	0.34	136
257	0.06	0.09	0.08	131
258	0.37	0.48	0.42	126
259	0.71	0.66	0.69	136
260	0.58	0.68	0.63	133
261	0.43	0.37	0.40	132
262	0.56	0.76	0.64	127
263	0.67	0.68	0.67	126
264	0.59	0.58	0.59	120
265	0.06	0.07	0.06	128
266	0.61	0.71	0.66	123
267	0.11	0.09	0.10	108
268	0.24	0.27	0.25	137
269	0.48	0.51	0.49	129
270	0.57	0.61	0.59	112
271	0.30	0.32	0.31	124
272	0.59	0.50	0.54	121
273	0.08	0.11	0.09	104
274	0.05	0.05	0.05	124
275	0.45	0.41	0.43	114
276	0.19	0.26	0.22	133
277	0.57	0.43	0.49	117
278	0.68	0.72	0.70	102
279	0.25	0.20	0.22	117
280	0.89	0.79	0.84	114
281	0.42	0.14	0.21	103
282	0.03	0.04	0.04	129
283	0.28	0.21	0.24	120
284	0.21	0.25	0.23	101
285	0.46	0.58	0.51	119
286	0.03	0.02	0.02	127
287	0.81	0.92	0.86	100
288	0.45	0.50	0.48	113
289	0.12	0.16	0.14	86
290	0.17	0.10	0.12	92
291	0.19	0.28	0.23	113
292	0.42	0.35	0.38	124
293	0.29	0.42	0.35	114
294	0.09	0.06	0.07	102
295	0.06	0.05	0.06	120
296	0.60	0.47	0.52	105

297	0.15	0.14	0.14	116
298	0.16	0.14	0.15	127
299	0.22	0.37	0.28	100
300	0.40	0.41	0.40	115
301	0.29	0.30	0.30	109
302	0.65	0.54	0.59	112
303	0.52	0.67	0.59	95
304	0.31	0.44	0.36	105
305	0.08	0.07	0.07	104
306	0.37	0.40	0.38	122
307	0.38	0.28	0.33	102
308	0.07	0.01	0.02	95
309	0.37	0.45	0.40	101
310	0.25	0.16	0.19	95
311	0.82	0.66	0.73	109
312	0.48	0.24	0.32	115
313	0.52	0.39	0.45	120
314	0.85	0.75	0.79	119
315	0.11	0.06	0.08	98
316	0.66	0.77	0.71	109
317	0.09	0.06	0.07	85
318	0.18	0.12	0.14	94
319	0.27	0.37	0.31	102
320	0.26	0.25	0.26	87
321	0.05	0.07	0.06	101
322	0.27	0.08	0.13	109
323	0.00	0.00	0.00	101
324	0.82	0.72	0.77	96
325	0.69	0.70	0.70	104
326	0.12	0.12	0.12	98
327	0.25	0.33	0.28	85
328	0.85	0.81	0.83	98
329	0.27	0.33	0.30	93
330	0.03	0.03	0.03	89
331	0.51	0.47	0.49	100
332	0.08	0.09	0.09	108
333	0.18	0.08	0.11	100
334	0.10	0.13	0.11	99
335	0.23	0.12	0.16	106
336	0.22	0.18	0.20	87
337	0.80	0.73	0.76	107
338	0.05	0.05	0.05	88
339	0.39	0.28	0.32	87
340	0.42	0.53	0.47	111
341	0.26	0.19	0.22	108
342	0.12	0.16	0.14	89
343	0.37	0.41	0.39	79
344	0.16	0.20	0.18	102
345	0.26	0.33	0.29	104
346	0.32	0.29	0.31	83
347	0.08	0.10	0.09	94
348	0.40	0.42	0.41	109
349	0.19	0.05	0.07	109
350	0.50	0.25	0.33	104
351	0.27	0.42	0.33	80
352	0.24	0.21	0.22	97
353	0.39	0.47	0.43	87
354	0.10	0.07	0.08	100
355	0.10	0.19	0.13	80
356	0.23	0.20	0.22	104
357	0.83	0.57	0.68	101

358	0.24	0.13	0.17	77
359	0.15	0.15	0.15	84
360	0.57	0.43	0.49	107
361	0.39	0.42	0.41	106
362	0.32	0.41	0.36	90
363	0.14	0.09	0.11	87
364	0.36	0.38	0.37	100
365	0.25	0.12	0.16	82
366	0.16	0.14	0.15	83
367	0.34	0.37	0.35	97
368	0.18	0.18	0.18	93
369	0.22	0.29	0.25	84
370	0.39	0.33	0.36	97
371	0.14	0.12	0.13	84
372	0.25	0.34	0.29	83
373	0.77	0.81	0.79	93
374	0.75	0.72	0.73	81
375	0.36	0.18	0.24	77
376	0.27	0.21	0.24	98
377	0.64	0.77	0.70	100
378	0.50	0.65	0.57	89
379	0.30	0.37	0.33	78
380	0.48	0.64	0.55	85
381	0.35	0.41	0.37	101
382	0.16	0.20	0.18	96
383	0.00	0.00	0.00	75
384	0.51	0.31	0.39	86
385	0.24	0.33	0.28	83
386	0.65	0.59	0.62	90
387	0.43	0.37	0.40	82
388	0.39	0.51	0.44	87
389	0.67	0.58	0.62	86
390	0.24	0.32	0.28	73
391	0.14	0.17	0.16	80
392	0.77	0.53	0.62	95
393	0.76	0.82	0.79	83
394	0.18	0.11	0.14	83
395	0.15	0.11	0.12	76
396	0.32	0.24	0.27	88
397	0.21	0.23	0.22	100
398	0.31	0.31	0.31	98
399	0.12	0.11	0.11	94
400	0.09	0.14	0.11	88
401	0.30	0.26	0.28	93
402	0.93	0.86	0.90	66
403	0.31	0.19	0.23	81
404	0.58	0.70	0.63	89
405	0.54	0.53	0.54	83
406	0.12	0.13	0.12	71
407	0.11	0.02	0.04	82
408	0.08	0.10	0.09	73
409	0.50	0.49	0.50	79
410	0.33	0.45	0.38	82
411	0.62	0.68	0.65	78
412	0.64	0.81	0.71	77
413	0.14	0.12	0.13	88
414	0.07	0.01	0.02	86
415	0.11	0.15	0.13	81
416	0.65	0.60	0.62	89
417	0.16	0.16	0.16	87
418	0.25	0.30	0.27	88

419	0.08	0.11	0.09	82
420	0.17	0.14	0.15	72
421	0.45	0.30	0.36	93
422	0.23	0.27	0.25	64
423	0.16	0.16	0.16	79
424	0.02	0.01	0.02	75
425	0.17	0.18	0.17	88
426	0.36	0.44	0.40	84
427	0.20	0.20	0.20	88
428	0.14	0.20	0.17	74
429	0.44	0.30	0.36	79
430	0.18	0.21	0.19	73
431	0.14	0.11	0.12	75
432	0.13	0.19	0.15	78
433	0.34	0.27	0.30	82
434	0.49	0.65	0.56	83
435	0.85	0.59	0.70	95
436	0.49	0.52	0.50	77
437	0.05	0.06	0.06	80
438	0.85	0.49	0.62	82
439	0.70	0.68	0.69	91
440	0.85	0.62	0.71	89
441	0.02	0.03	0.02	77
442	0.33	0.31	0.32	67
443	0.08	0.06	0.07	96
444	0.55	0.61	0.58	69
445	0.25	0.11	0.15	83
446	0.25	0.14	0.18	90
447	0.52	0.32	0.39	97
448	0.26	0.35	0.30	74
449	0.11	0.17	0.14	72
450	0.32	0.42	0.36	62
451	0.44	0.15	0.23	91
452	0.33	0.21	0.26	70
453	0.81	0.74	0.78	70
454	0.07	0.10	0.08	68
455	0.02	0.01	0.01	93
456	0.16	0.11	0.13	71
457	0.22	0.34	0.27	73
458	0.33	0.42	0.37	77
459	0.71	0.66	0.68	76
460	0.29	0.16	0.20	88
461	0.22	0.26	0.24	76
462	0.16	0.16	0.16	85
463	0.52	0.16	0.24	69
464	0.69	0.70	0.69	69
465	0.19	0.15	0.17	66
466	0.69	0.82	0.75	67
467	0.08	0.04	0.05	79
468	0.00	0.00	0.00	78
469	0.46	0.39	0.42	71
470	0.24	0.39	0.29	72
471	0.70	0.67	0.68	87
472	0.54	0.60	0.56	62
473	0.13	0.19	0.15	64
474	0.61	0.61	0.61	79
475	0.69	0.64	0.66	74
476	0.41	0.45	0.43	67
477	0.18	0.23	0.20	69
478	0.04	0.05	0.04	61
479	0.09	0.08	0.08	64

```
          480       0.89      0.73      0.80        74
          481       0.24      0.32      0.28        72
          482       0.15      0.17      0.16        76
          483       0.15      0.24      0.19        67
          484       0.35      0.26      0.29        74
          485       0.31      0.40      0.35        60
          486       0.33      0.07      0.12        71
          487       0.00      0.00      0.00        72
          488       0.35      0.30      0.32        63
          489       0.43      0.57      0.49        63
          490       0.68      0.79      0.73        81
          491       0.43      0.61      0.50        67
          492       0.35      0.18      0.23        74
          493       0.64      0.51      0.57        63
          494       0.42      0.45      0.43        74
          495       0.39      0.44      0.41        66
          496       0.31      0.22      0.25        93
          497       0.50      0.34      0.40        74
          498       0.54      0.42      0.47        88
          499       0.11      0.13      0.12        67

   micro avg       0.46      0.45      0.45    153206
   macro avg       0.37      0.37      0.36    153206
weighted avg       0.46      0.45      0.45    153206
 samples avg       0.43      0.44      0.40    153206

Time taken to run this cell : 0:19:34.690894
```

Linear SVM :-

In [22]:

```
from sklearn.model_selection import GridSearchCV

model_to_set = OneVsRestClassifier(SGDClassifier(loss='hinge', penalty='l1'))

parameters = {
    "estimator__alpha": [10**-5, 10**-3, 10**-2, 10**1]
}

model_tunning = GridSearchCV(model_to_set, param_grid=parameters, scoring='f1_micro',n_
jobs=-1)

model_tunning.fit(x_train_multilabel, y_train)

print (model_tunning.best_score_)
print (model_tunning.best_params_)
```

```
0.43896966470627347
{'estimator__alpha': 0.001}
```

In [23]:

```
start = datetime.now()
#best_alpha = gsv.best_estimator_.get_params()['estimator__alpha']
classifier = OneVsRestClassifier(SGDClassifier(loss='hinge', alpha=0.001, penalty='l1'
), n_jobs=-1)
classifier.fit(x_train_multilabel, y_train)
predictions = classifier.predict (x_test_multilabel)

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


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

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

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

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

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

```
Accuracy : 0.19328719884430776
Hamming loss  0.003178945896826506
Micro-average quality numbers
Precision: 0.5944, Recall: 0.3438, F1-measure: 0.4356
Macro-average quality numbers
Precision: 0.3550, Recall: 0.2444, F1-measure: 0.2739
```

	precision	recall	f1-score	support
0	0.56	0.24	0.33	6489
1	0.71	0.43	0.53	5898
2	0.79	0.55	0.65	5527
3	0.66	0.42	0.51	5212
4	0.83	0.75	0.79	4600
5	0.84	0.63	0.72	4380
6	0.74	0.50	0.60	2910
7	0.82	0.64	0.72	2737
8	0.62	0.44	0.52	2665
9	0.69	0.44	0.54	2540
10	0.78	0.64	0.70	2448
11	0.43	0.16	0.24	2321
12	0.37	0.19	0.25	2244
13	0.59	0.28	0.38	1965
14	0.62	0.18	0.28	1918
15	0.58	0.27	0.37	1959
16	0.66	0.52	0.58	1820
17	0.58	0.23	0.33	1795
18	0.73	0.62	0.67	1733
19	0.05	0.00	0.00	1384
20	0.27	0.05	0.09	1428
21	0.54	0.28	0.37	1089
22	0.63	0.38	0.47	1027
23	0.75	0.67	0.71	945
24	0.48	0.49	0.49	880
25	0.55	0.48	0.51	845
26	0.00	0.00	0.00	828
27	0.79	0.66	0.72	824
28	0.20	0.08	0.12	817
29	0.54	0.37	0.44	789
30	0.58	0.29	0.38	766
31	0.48	0.41	0.44	719
32	0.90	0.81	0.85	745
33	0.38	0.36	0.37	745
34	0.79	0.38	0.51	657
35	0.60	0.61	0.61	591
36	0.67	0.70	0.68	646
37	0.54	0.60	0.57	574
38	0.28	0.17	0.21	637
39	0.64	0.32	0.43	624
40	0.00	0.00	0.00	575
41	0.00	0.00	0.00	541
42	0.44	0.46	0.45	549
43	0.00	0.00	0.00	556
44	0.63	0.25	0.35	521
45	0.80	0.28	0.41	505
46	0.56	0.29	0.38	509
47	0.35	0.13	0.19	495
48	0.70	0.52	0.59	465
49	0.81	0.76	0.79	439
50	0.16	0.19	0.17	434
51	0.00	0.00	0.00	442
52	0.70	0.40	0.51	448

53	0.33	0.01	0.02	465
54	0.54	0.52	0.53	457
55	0.33	0.03	0.05	427
56	0.46	0.41	0.44	444
57	0.57	0.08	0.15	431
58	0.32	0.15	0.20	439
59	0.72	0.62	0.66	427
60	0.55	0.53	0.54	382
61	0.76	0.81	0.78	400
62	0.80	0.71	0.76	421
63	0.00	0.00	0.00	389
64	0.72	0.63	0.67	394
65	0.23	0.26	0.24	392
66	0.64	0.51	0.57	416
67	0.61	0.37	0.46	417
68	0.43	0.30	0.35	385
69	0.48	0.26	0.33	358
70	0.43	0.38	0.41	413
71	0.54	0.37	0.44	365
72	0.33	0.34	0.33	384
73	0.17	0.06	0.09	393
74	0.00	0.00	0.00	347
75	0.44	0.24	0.31	372
76	0.78	0.73	0.75	349
77	0.51	0.50	0.51	336
78	0.66	0.53	0.59	334
79	0.04	0.04	0.04	358
80	0.22	0.14	0.17	313
81	0.71	0.60	0.65	302
82	0.71	0.12	0.21	331
83	0.00	0.00	0.00	292
84	0.57	0.27	0.37	287
85	0.42	0.50	0.46	323
86	0.57	0.55	0.56	311
87	0.55	0.10	0.16	320
88	0.78	0.65	0.71	306
89	0.82	0.59	0.69	288
90	0.66	0.13	0.22	280
91	0.86	0.62	0.72	285
92	0.74	0.53	0.62	278
93	0.39	0.41	0.40	277
94	0.43	0.54	0.48	253
95	0.44	0.04	0.08	281
96	0.00	0.00	0.00	273
97	0.43	0.27	0.33	296
98	0.54	0.40	0.45	281
99	0.78	0.66	0.71	287
100	0.23	0.07	0.11	277
101	0.70	0.57	0.63	285
102	0.76	0.61	0.67	257
103	0.71	0.12	0.21	258
104	0.31	0.28	0.30	271
105	0.86	0.69	0.76	251
106	0.52	0.49	0.51	255
107	0.08	0.01	0.02	259
108	0.77	0.46	0.58	261
109	0.00	0.00	0.00	256
110	0.50	0.64	0.56	263
111	0.00	0.00	0.00	256
112	0.08	0.06	0.07	277
113	0.82	0.73	0.78	241

114	0.25	0.24	0.24	224
115	0.18	0.19	0.19	233
116	0.53	0.24	0.33	269
117	0.54	0.06	0.11	226
118	0.65	0.56	0.60	235
119	0.00	0.00	0.00	255
120	0.30	0.11	0.16	249
121	0.85	0.72	0.78	220
122	0.61	0.40	0.48	235
123	0.00	0.00	0.00	244
124	0.00	0.00	0.00	232
125	0.48	0.28	0.36	224
126	0.76	0.58	0.66	223
127	0.00	0.00	0.00	244
128	0.58	0.13	0.22	215
129	0.00	0.00	0.00	192
130	0.29	0.12	0.17	228
131	0.92	0.85	0.89	239
132	0.29	0.30	0.29	205
133	0.33	0.36	0.34	218
134	0.56	0.35	0.43	227
135	0.78	0.11	0.19	199
136	0.67	0.65	0.66	239
137	0.43	0.52	0.47	216
138	0.16	0.21	0.18	198
139	0.09	0.09	0.09	227
140	0.59	0.57	0.58	221
141	0.00	0.00	0.00	204
142	0.92	0.68	0.78	198
143	0.00	0.00	0.00	208
144	0.14	0.31	0.19	193
145	0.00	0.00	0.00	206
146	0.00	0.00	0.00	220
147	0.70	0.48	0.57	193
148	0.00	0.00	0.00	208
149	0.31	0.36	0.34	188
150	0.28	0.22	0.25	199
151	0.65	0.15	0.24	191
152	0.20	0.21	0.21	211
153	0.85	0.78	0.81	197
154	1.00	0.01	0.02	197
155	0.81	0.79	0.80	190
156	0.34	0.27	0.30	192
157	0.79	0.52	0.62	222
158	0.00	0.00	0.00	208
159	0.43	0.20	0.28	196
160	0.35	0.06	0.10	200
161	0.00	0.00	0.00	199
162	0.88	0.74	0.81	176
163	0.00	0.00	0.00	191
164	0.89	0.64	0.74	170
165	0.57	0.48	0.52	216
166	0.57	0.14	0.23	213
167	0.60	0.52	0.56	189
168	0.01	0.01	0.01	191
169	0.84	0.57	0.68	191
170	0.35	0.23	0.28	189
171	0.00	0.00	0.00	198
172	0.31	0.29	0.30	192
173	0.22	0.17	0.19	187
174	0.00	0.00	0.00	187

175	0.44	0.37	0.40	183
176	0.00	0.00	0.00	176
177	0.84	0.12	0.21	179
178	0.43	0.33	0.37	191
179	0.79	0.66	0.72	185
180	0.00	0.00	0.00	191
181	0.67	0.40	0.50	176
182	0.37	0.18	0.24	161
183	0.04	0.01	0.02	175
184	0.00	0.00	0.00	177
185	0.35	0.28	0.31	172
186	0.61	0.57	0.59	196
187	0.56	0.46	0.50	167
188	0.45	0.13	0.21	179
189	0.83	0.65	0.73	177
190	0.57	0.33	0.42	166
191	0.16	0.05	0.08	193
192	0.11	0.09	0.10	175
193	0.47	0.40	0.43	164
194	0.00	0.00	0.00	188
195	0.48	0.28	0.35	181
196	0.62	0.55	0.59	166
197	0.44	0.39	0.41	155
198	0.88	0.60	0.71	173
199	0.00	0.00	0.00	159
200	0.00	0.00	0.00	164
201	0.19	0.23	0.21	151
202	0.29	0.42	0.34	176
203	0.82	0.71	0.76	163
204	0.26	0.03	0.05	189
205	0.00	0.00	0.00	165
206	0.95	0.70	0.80	164
207	0.12	0.05	0.07	155
208	0.52	0.16	0.25	155
209	0.23	0.11	0.15	175
210	0.00	0.00	0.00	155
211	0.00	0.00	0.00	150
212	0.39	0.49	0.44	150
213	0.03	0.01	0.01	155
214	0.06	0.09	0.07	153
215	0.82	0.84	0.83	167
216	0.15	0.16	0.16	140
217	0.00	0.00	0.00	155
218	0.69	0.27	0.39	152
219	0.00	0.00	0.00	151
220	0.00	0.00	0.00	141
221	0.51	0.36	0.42	146
222	0.00	0.00	0.00	154
223	0.13	0.13	0.13	155
224	0.00	0.00	0.00	143
225	0.00	0.00	0.00	157
226	0.24	0.20	0.22	137
227	0.42	0.40	0.41	160
228	0.67	0.51	0.58	151
229	0.54	0.10	0.16	135
230	0.28	0.21	0.24	154
231	0.63	0.73	0.68	141
232	0.48	0.25	0.33	151
233	0.58	0.53	0.55	148
234	0.53	0.51	0.52	149
235	0.40	0.03	0.05	147

236	0.00	0.00	0.00	131
237	0.00	0.00	0.00	145
238	0.53	0.65	0.58	126
239	0.00	0.00	0.00	138
240	0.82	0.69	0.75	134
241	0.00	0.00	0.00	131
242	0.33	0.09	0.15	137
243	0.00	0.00	0.00	140
244	0.31	0.16	0.21	128
245	0.00	0.00	0.00	131
246	0.32	0.19	0.24	134
247	0.16	0.13	0.14	129
248	0.00	0.00	0.00	132
249	0.00	0.00	0.00	130
250	0.00	0.00	0.00	123
251	0.39	0.12	0.18	113
252	0.16	0.21	0.18	143
253	0.35	0.34	0.34	138
254	0.62	0.50	0.55	139
255	0.49	0.50	0.50	119
256	0.39	0.10	0.16	120
257	0.00	0.00	0.00	141
258	0.42	0.41	0.42	132
259	0.54	0.54	0.54	102
260	0.52	0.33	0.41	141
261	0.32	0.36	0.34	137
262	0.77	0.54	0.64	116
263	0.79	0.39	0.53	142
264	0.54	0.44	0.49	140
265	0.00	0.00	0.00	118
266	0.74	0.58	0.65	105
267	0.00	0.00	0.00	132
268	0.27	0.12	0.17	136
269	0.26	0.45	0.33	121
270	0.64	0.33	0.43	128
271	0.40	0.14	0.21	123
272	0.00	0.00	0.00	145
273	0.27	0.03	0.05	133
274	0.00	0.00	0.00	121
275	0.70	0.29	0.41	121
276	0.00	0.00	0.00	134
277	0.42	0.48	0.45	98
278	0.72	0.42	0.53	122
279	0.00	0.00	0.00	106
280	0.92	0.57	0.71	107
281	0.00	0.00	0.00	125
282	0.00	0.00	0.00	117
283	0.25	0.16	0.20	111
284	0.00	0.00	0.00	115
285	0.43	0.56	0.49	118
286	0.00	0.00	0.00	124
287	0.76	0.85	0.80	96
288	0.53	0.39	0.45	117
289	0.00	0.00	0.00	107
290	1.00	0.01	0.02	93
291	0.30	0.11	0.17	96
292	0.23	0.19	0.21	109
293	0.39	0.29	0.33	120
294	0.02	0.01	0.01	127
295	0.00	0.00	0.00	96
296	0.66	0.25	0.37	114

297	0.17	0.14	0.15	103
298	0.40	0.07	0.12	110
299	0.45	0.17	0.25	98
300	0.56	0.43	0.49	104
301	0.37	0.08	0.14	131
302	0.78	0.34	0.47	103
303	0.43	0.54	0.47	112
304	0.25	0.01	0.02	111
305	0.00	0.00	0.00	107
306	0.38	0.34	0.36	95
307	0.30	0.14	0.19	103
308	0.00	0.00	0.00	107
309	0.00	0.00	0.00	80
310	0.00	0.00	0.00	109
311	0.85	0.44	0.58	100
312	0.54	0.34	0.42	99
313	0.47	0.30	0.37	102
314	0.70	0.65	0.67	96
315	0.00	0.00	0.00	98
316	0.64	0.72	0.68	105
317	0.00	0.00	0.00	142
318	0.00	0.00	0.00	96
319	0.34	0.42	0.38	104
320	0.00	0.00	0.00	98
321	0.00	0.00	0.00	91
322	0.00	0.00	0.00	97
323	0.00	0.00	0.00	104
324	0.74	0.60	0.66	97
325	0.43	0.58	0.50	104
326	0.00	0.00	0.00	92
327	0.24	0.21	0.22	94
328	0.56	0.65	0.60	93
329	0.29	0.19	0.23	106
330	0.00	0.00	0.00	104
331	0.75	0.13	0.22	93
332	0.00	0.00	0.00	100
333	0.00	0.00	0.00	86
334	0.00	0.00	0.00	91
335	0.00	0.00	0.00	106
336	0.17	0.05	0.07	106
337	0.73	0.59	0.66	96
338	0.00	0.00	0.00	100
339	0.24	0.23	0.23	102
340	0.38	0.44	0.41	81
341	0.53	0.07	0.13	112
342	0.00	0.00	0.00	101
343	0.61	0.44	0.51	101
344	0.00	0.00	0.00	99
345	0.00	0.00	0.00	94
346	1.00	0.03	0.06	95
347	0.00	0.00	0.00	95
348	0.36	0.28	0.31	86
349	0.00	0.00	0.00	111
350	0.00	0.00	0.00	95
351	0.00	0.00	0.00	76
352	0.00	0.00	0.00	115
353	0.42	0.33	0.37	98
354	0.00	0.00	0.00	106
355	0.00	0.00	0.00	84
356	0.18	0.20	0.19	102
357	0.65	0.27	0.38	88

358	0.08	0.11	0.10	105
359	0.00	0.00	0.00	89
360	0.43	0.27	0.33	93
361	0.40	0.36	0.38	110
362	0.00	0.00	0.00	88
363	0.00	0.00	0.00	96
364	0.48	0.16	0.25	85
365	0.31	0.17	0.22	82
366	0.00	0.00	0.00	83
367	0.40	0.21	0.27	86
368	0.00	0.00	0.00	79
369	0.16	0.12	0.14	102
370	0.24	0.15	0.18	81
371	0.00	0.00	0.00	85
372	0.29	0.19	0.23	77
373	0.92	0.74	0.82	112
374	0.77	0.51	0.62	78
375	0.15	0.18	0.16	90
376	0.00	0.00	0.00	85
377	0.71	0.42	0.53	86
378	0.96	0.29	0.45	78
379	0.00	0.00	0.00	89
380	0.77	0.48	0.59	85
381	0.52	0.18	0.27	94
382	0.00	0.00	0.00	89
383	0.00	0.00	0.00	88
384	0.00	0.00	0.00	84
385	0.12	0.14	0.13	70
386	0.62	0.60	0.61	88
387	0.32	0.27	0.30	73
388	0.52	0.36	0.42	98
389	0.55	0.57	0.56	81
390	0.80	0.04	0.07	104
391	0.00	0.00	0.00	99
392	0.92	0.14	0.25	84
393	0.88	0.75	0.81	80
394	0.00	0.00	0.00	81
395	0.00	0.00	0.00	81
396	0.15	0.02	0.04	90
397	0.00	0.00	0.00	82
398	0.18	0.20	0.19	85
399	0.00	0.00	0.00	89
400	0.00	0.00	0.00	85
401	0.31	0.07	0.12	69
402	0.98	0.79	0.87	80
403	0.00	0.00	0.00	81
404	0.52	0.46	0.49	74
405	0.64	0.37	0.47	92
406	0.10	0.15	0.12	87
407	0.00	0.00	0.00	96
408	0.00	0.00	0.00	80
409	0.29	0.37	0.33	89
410	0.40	0.32	0.36	81
411	0.75	0.51	0.61	84
412	0.61	0.61	0.61	70
413	0.00	0.00	0.00	86
414	0.00	0.00	0.00	77
415	0.05	0.09	0.06	77
416	0.90	0.57	0.69	92
417	0.00	0.00	0.00	82
418	0.22	0.23	0.23	81

419	0.00	0.00	0.00	83
420	0.00	0.00	0.00	72
421	0.39	0.27	0.32	88
422	0.00	0.00	0.00	79
423	0.14	0.13	0.14	84
424	0.00	0.00	0.00	74
425	0.00	0.00	0.00	74
426	0.52	0.16	0.25	74
427	0.00	0.00	0.00	83
428	0.00	0.00	0.00	79
429	0.00	0.00	0.00	69
430	0.20	0.14	0.17	85
431	0.00	0.00	0.00	74
432	0.00	0.00	0.00	80
433	0.24	0.42	0.31	60
434	0.54	0.50	0.52	70
435	0.82	0.31	0.45	74
436	0.75	0.33	0.46	91
437	0.00	0.00	0.00	68
438	0.81	0.29	0.42	91
439	0.78	0.34	0.47	91
440	0.78	0.35	0.48	81
441	0.00	0.00	0.00	77
442	0.80	0.04	0.08	92
443	0.00	0.00	0.00	82
444	0.75	0.61	0.67	74
445	0.00	0.00	0.00	84
446	0.00	0.00	0.00	73
447	0.47	0.25	0.32	69
448	0.34	0.31	0.33	67
449	0.00	0.00	0.00	71
450	0.00	0.00	0.00	67
451	0.00	0.00	0.00	69
452	0.36	0.20	0.25	81
453	0.74	0.54	0.62	65
454	0.00	0.00	0.00	65
455	0.00	0.00	0.00	81
456	0.00	0.00	0.00	72
457	0.21	0.24	0.22	72
458	0.60	0.21	0.31	85
459	0.66	0.61	0.64	80
460	0.00	0.00	0.00	70
461	0.00	0.00	0.00	83
462	0.00	0.00	0.00	82
463	0.00	0.00	0.00	69
464	0.47	0.47	0.47	72
465	0.17	0.29	0.21	72
466	0.53	0.54	0.54	63
467	0.00	0.00	0.00	78
468	0.00	0.00	0.00	74
469	0.00	0.00	0.00	81
470	0.22	0.12	0.16	80
471	0.71	0.34	0.46	74
472	0.49	0.39	0.44	74
473	0.00	0.00	0.00	78
474	0.60	0.56	0.58	80
475	0.57	0.48	0.52	79
476	0.40	0.39	0.39	69
477	0.00	0.00	0.00	58
478	0.12	0.15	0.13	74
479	0.00	0.00	0.00	78

```
          480       0.94      0.45      0.61        71
          481       0.50      0.13      0.20        71
          482       0.00      0.00      0.00        64
          483       0.00      0.00      0.00        58
          484       0.00      0.00      0.00        71
          485       0.24      0.22      0.23        76
          486       0.00      0.00      0.00        59
          487       0.00      0.00      0.00        74
          488       0.36      0.32      0.34        71
          489       0.48      0.22      0.30        63
          490       0.80      0.61      0.69        90
          491       0.67      0.56      0.61        75
          492       0.36      0.18      0.24        90
          493       0.59      0.52      0.55        64
          494       0.31      0.25      0.28        63
          495       0.38      0.17      0.23        54
          496       0.22      0.17      0.19        60
          497       0.32      0.10      0.16        68
          498       0.49      0.47      0.48        72
          499       0.00      0.00      0.00        63

avg / total       0.51      0.34      0.40    153155

Time taken to run this cell : 0:10:43.777699
```

In [27]:

```
from prettytable import PrettyTable
ptable = PrettyTable()

ptable.title = " Comparison of Performances "
ptable.field_names = ['Model','Hyperparameter', 'Macro Train F1-Score ','Macro Test F1-
Score']
ptable.add_row(["Logistic Regression","0.0001","0.45", "0.36"])
ptable.add_row(["Linear SVM", "0.001", "0.43", "0.27"])

print(ptable)
```

```
+---------------------+---------------+----------------------+----------
-----------+
|        Model        | Hyperparameter | Macro Train F1-Score  | Macro Tes
t F1-Score |
+---------------------+---------------+----------------------+----------
-----------+
| Logistic Regression |     0.0001     |          0.45         |
0.36        |
|      Linear SVM     |     0.001      |          0.43         |
0.27        |
+---------------------+---------------+----------------------+----------
-----------+
```

Observations:

EDA:

a. We have observed that there are total of 485836 tags present in our 500k data out of which 30429 are unique.

b. When we quantiled the tags we observed that 25 tags appeared 10000 times and 18 tags appeared more than 100000 times which is dramatic.

c. We observed that in each question there is an average of 3 tags.

d. The most common tags are C#, JAVA, php etc

e. And the most common tag C# apperaed more than 35000 times.

f. We have selected 500 most common tags which constitutes 89.47% question and further proceeded our operation.

Model Fabrication:

a. Logistic Regression: We build logistic regression model using SGDClassifier with log loss and we got macro f1-score as 0.36.

b. Linear SVM: We built the Linear SVM model using SGDClassifier with hinge loss and got a macro f1-score as 0.27

Few of the steps followed:

In [ ]:

```
1. After EDA i have selected top 500 features as they constitutes 89 % of the question
and also due to computation power.
2. Next we have done hyperparameter tuning using grid search and SGDclassifier with log
loss.
3. We had to train models for each of our tags so used one vs rest classifier with our
grid search and got our best hyperparameter.
4. Then i trained my logistic regression model and got macro f1-score of 0.36.
5. Similarly we trained our linear SVM model and got macro f1-score of 0.27.
6. We observe that our Linear SVM model performs better in this case.
7. I have added extra EDA on cell[96] and cell[84] which demonstrates most number of qu
estions and average number of tags per question.
```