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

1.3 Real World / Business Objectives and Constraints

1. For the given Questions and its descriptions, Predict as many tags as possible with high precision and recall.
2. Precision is we have to be very sure that the tag is for that particular question and recall means if the tag is suppose to be present then it should be present most of the times
3. Incorrect tags could impact customer experience on StackOverflow.
4. If a incorrect tag is predicted, then the precision decreases and if any correct tag is missed then recall decreases. This impacts customer experience badly.
5. No strict latency constraints.

2. Machine Learning problem

2.1 Data

2.1.1 Data Overview

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

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

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

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

1. **Size of Train.csv** - 6.75GB (after unzipping)

1. **Size of Test.csv** - 2GB

1. **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:

1. **Id** - Unique identifier for each question

1. **Title** - The question's title

1. **Body** - The body of the question

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

2.1.2 Example Data point

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

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

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


    \n\n
```

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

2.2.1 Type of Machine Learning Problem

1. This Stack-overflow problem is multi-label classification problem, where each question have multiple labels/tags/classes.

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.

1. In binary classification and multi-class classification we only have one label for each Xi. When we have multiple labels for a given Xi then it multi-label classification problem.

Credit: http://scikit-learn.org/stable/modules/multiclass.html (http://scikit-learn.org/stable/modules/multiclass.html)

2.2.2 Performance metric

F1-score, Micro-averaged F1-score, Macro averaged F1-score

F1-score:- The F1 score can be interpreted as a weighted average of the precision and recall. To get both high precision and high recall, we can use F1-score which is a geometric mean of both precision and recall. where an F1 score reaches its best value at 1 and worst score at 0.

For F1-score minimum value is 0 and best value is 1.

1. The relative contribution of precision and recall to the F1 score are equal. The formula for the F1 score is:
2. F1 = 2 (precision * recall) / (precision + recall) . This is for binary classification.
3. Precision = True positives / True positives + False positives
4. Recall = True positives / True positives + False Negatives.

In multi-label classification we can modify F1-score into 2 types

a. Micro-averaged F1-score. It is more popular

b. Macro averaged F1-score

Micro-Averaged F1-Score (Mean F Score) :- This is the weighted average of the F1 score of each class.

1. Here weightage is given based on how frequently a label occurs.
2. It takes tag/label frequency of occurance into consideration when computing micro precision and micro recall.
3. Here we take individual TP, FP, FN, so we get weighted average of these and we get weighted F1-score.
4. Calculate metrics globally by counting the total true positives, false negatives and false positives.
5. This is a better metric when we have class imbalance.

Macro f1 score:-

1. Calculate metrics for each label, and find their unweighted mean. This does not take label imbalance into account.
2. It takes simple average of F1 scores of all labels/tags.
3. It doesn't take frequency of occurance of a tag into consideration. This is non-weighted, simple F1-score.

If we have a case where some tags occurs many times and some tags occurs very less times, then it is good to use micro averaged F1-score , not macro averaged F1-score.

Reference:-

https://www.kaggle.com/wiki/MeanFScore (https://www.kaggle.com/wiki/MeanFScore)

http://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html (http://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html)

Hamming loss : The Hamming loss is the fraction of labels that are incorrectly predicted. If the actual labels and predicted labels differ more, then the error also increases. This is Hamming loss.

https://www.kaggle.com/wiki/HammingLoss (https://www.kaggle.com/wiki/HammingLoss)

Loading all Libraries

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
```

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()
    # connecting to the database train.db . This is opening the database
    con = sqlite3.connect('train.db')
    # run SQL query to get number of rows in database table
    num_rows = pd.read_sql_query("""SELECT count(*) FROM data""", con)
    #Always remember to close the database
    print("Number of rows in the database :","\n",num_rows['count(*)'].values[
0])
    con.close()
    print("Time taken to count the number of rows :", datetime.now() - start)
else:
    print("Please download the train.db file from drive or run the above cell
 to genarate train.db file")
```

```
Number of rows in the database :
 6034196
Time taken to count the number of rows : 0:01:37.418065
```

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()
    # opening the database
    con = sqlite3.connect('train.db')
    # This query returns set of all the duplicates in the database , number of times each duplicate occurs and this data
    # is stored into df_no_dup.
    df_no_dup = pd.read_sql_query('SELECT Title, Body, Tags, COUNT(*) as cnt_dup FROM data GROUP BY Title, Body, Tags', con)
    # Closing the connection
    con.close()
    print("Time taken to run this cell :", datetime.now() - start)
else:
    print("Please download the train.db file from drive or run the first to genarate train.db file")
```

```
Time taken to run this cell : 0:02:49.075469
```

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

Out[5]:

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

In [6]:
```
print("number of duplicate questions :", num_rows['count(*)'].values[0]- df_no_dup.shape[0], "(",(1-((df_no_dup.shape[0])/(num_rows['count(*)'].values[0])))*100,"% )")
```

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

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

Out[7]:
```
1    2656284
2    1272336
3     277575
4         90
5         25
6          5
Name: cnt_dup, dtype: int64
```

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

Out[8]:

	Title	Body	Tags	cnt_dup
777547	Do we really need NULL?	<blockquote>\n <p><strong>Possible Duplicate:...	None	1
962680	Find all values that are not null and not in a...	<p>I am running into a problem which results i...	None	1
1126558	Handle NullObjects	<p>I have done quite a bit of research on best...	None	1
1256102	How do Germans call null	<p>In german null means 0, so how do they call...	None	1
2430668	Page cannot be null. Please ensure that this o...	<p>I get this error when i remove dynamically ...	None	1
3329908	What is the difference between NULL and "0"?	<p>What is the difference from NULL and "0"?</...	None	1
3551595	a bit of difference between null and space	I was just reading this quote \n\n<block...	None	2

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

In [10]:
```
# checking the tag count for each title and body before removing them

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

```
Time taken to run this cell : 0:00:03.259207
```

Out[10]:

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

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

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

In [12]:
```
#Creating a new database with no duplicates / after removing duplicates
# train_no_dup  -- train database with no duplicates . There are only non dupl
icate rows in this database

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 genarate train.db file")
```

```
Time taken to run this cell : 0:00:50.498138
```

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 string
s.
tag_dtm = vectorizer.fit_transform(tag_data['Tags'])
```

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

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

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

```
Some of the tags we have : ['.a', '.app', '.asp.net-mvc', '.aspxauth', '.bash
-profile', '.class-file', '.cs-file', '.doc', '.drv', '.ds-store']
```

3.2.3 Number of times a tag appeared

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

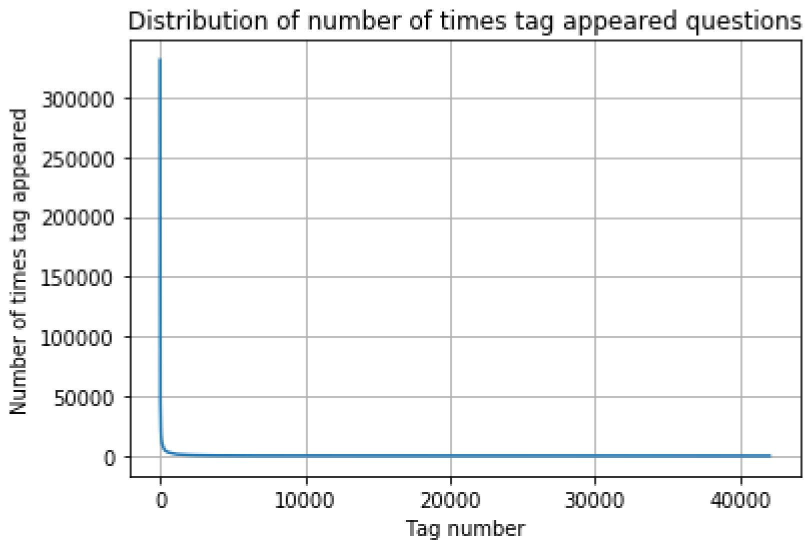
In [18]:
```
#Saving this dictionary to csv files.
if not os.path.isfile('tag_counts_dict_dtm.csv'):
    with open('tag_counts_dict_dtm.csv', 'w') as csv_file:
        writer = csv.writer(csv_file)
        for key, value in result.items():
            writer.writerow([key, value])
tag_df = pd.read_csv("tag_counts_dict_dtm.csv",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

Out[18]:

	Tags	Counts
0	paintbox	4
1	invoke-command	29
2	findbugs	248
3	cooking	15
4	netbeans-plugins	121

In [19]:
```
# Sort the tags in descending order of number of times it occurs

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

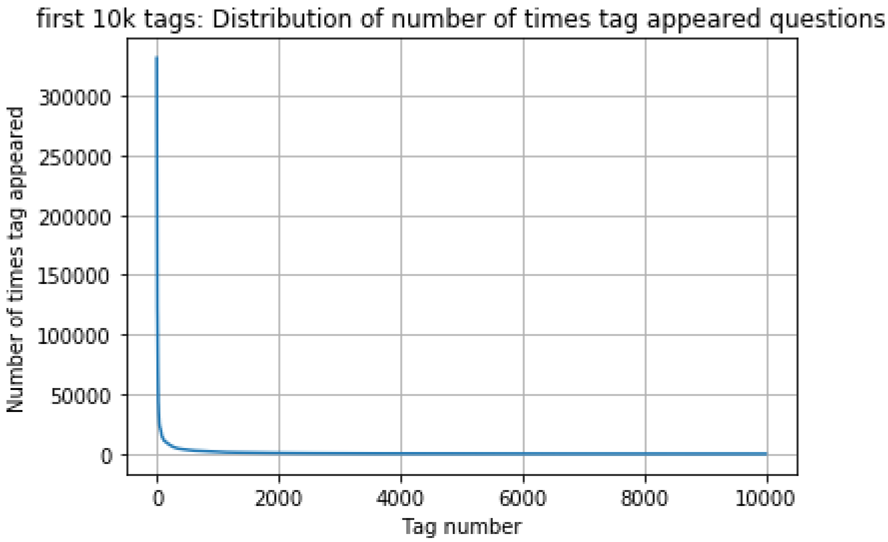
In [20]:
```
plt.plot(tag_counts)
plt.title("Distribution of number of times tag appeared questions")
plt.grid()
plt.xlabel("Tag number")
plt.ylabel("Number of times tag appeared")
plt.show()
```



Observation :-

This is highly skewed distribution , as we want to zoom in and see results clearly , taking first 10000 points in the plot below.

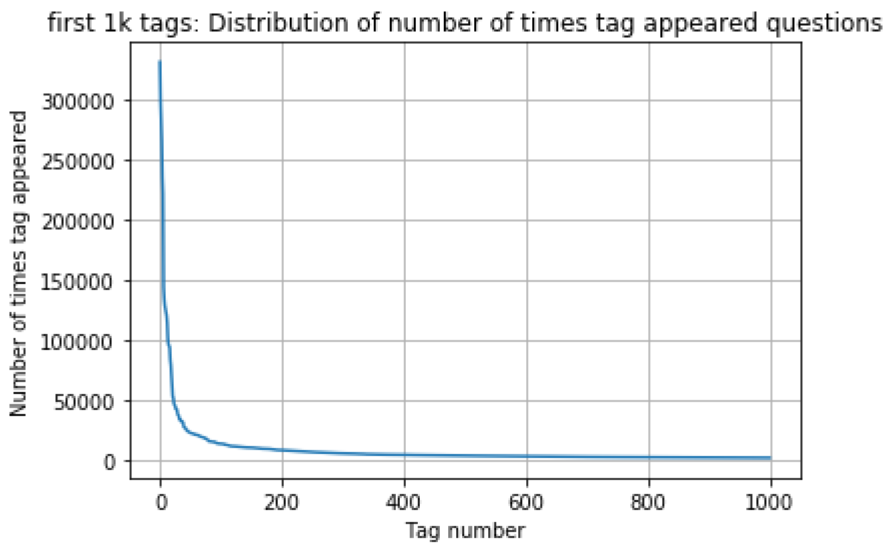
In [21]:
```
plt.plot(tag_counts[0:10000])
plt.title('first 10k tags: Distribution of number of times tag appeared questi
ons')
plt.grid()
plt.xlabel("Tag number")
plt.ylabel("Number of times tag appeared")
plt.show()
print(len(tag_counts[0:10000:25]), tag_counts[0:10000:25])
```



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

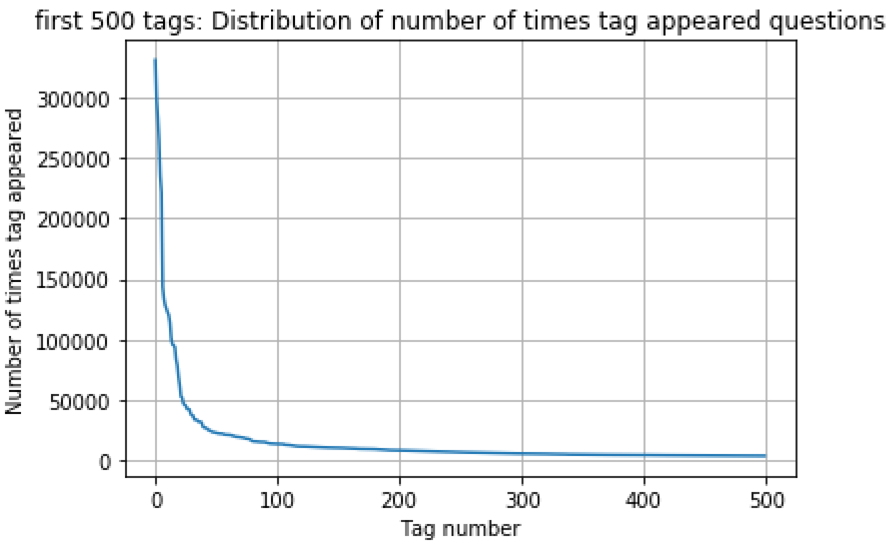
```
400 [331505  44829  22429  17728  13364  11162  10029   9148   8054   7151
   6466   5865   5370   4983   4526   4281   4144   3929   3750   3593
   3453   3299   3123   2986   2891   2738   2647   2527   2431   2331
   2259   2186   2097   2020   1959   1900   1828   1770   1723   1673
   1631   1574   1532   1479   1448   1406   1365   1328   1300   1266
   1245   1222   1197   1181   1158   1139   1121   1101   1076   1056
   1038   1023   1006    983    966    952    938    926    911    891
    882    869    856    841    830    816    804    789    779    770
    752    743    733    725    712    702    688    678    671    658
    650    643    634    627    616    607    598    589    583    577
    568    559    552    545    540    533    526    518    512    506
    500    495    490    485    480    477    469    465    457    450
    447    442    437    432    426    422    418    413    408    403
    398    393    388    385    381    378    374    370    367    365
    361    357    354    350    347    344    342    339    336    332
    330    326    323    319    315    312    309    307    304    301
    299    296    293    291    289    286    284    281    278    276
    275    272    270    268    265    262    260    258    256    254
    252    250    249    247    245    243    241    239    238    236
    234    233    232    230    228    226    224    222    220    219
    217    215    214    212    210    209    207    205    204    203
    201    200    199    198    196    194    193    192    191    189
    188    186    185    183    182    181    180    179    178    177
    175    174    172    171    170    169    168    167    166    165
    164    162    161    160    159    158    157    156    156    155
    154    153    152    151    150    149    149    148    147    146
    145    144    143    142    142    141    140    139    138    137
    137    136    135    134    134    133    132    131    130    130
    129    128    128    127    126    126    125    124    124    123
    123    122    122    121    120    120    119    118    118    117
    117    116    116    115    115    114    113    113    112    111
    111    110    109    109    108    108    107    106    106    106
    105    105    104    104    103    103    102    102    101    101
    100    100     99     99     98     98     97     97     96     96
     95     95     94     94     93     93     93     92     92     91
     91     90     90     89     89     88     88     87     87     86
     86     86     85     85     84     84     83     83     83     82
     82     82     81     81     80     80     80     79     79     78
     78     78     78     77     77     76     76     76     75     75
     75     74     74     74     73     73     73     73     72     72]
```

In [22]:
```
# first 1000 tags

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



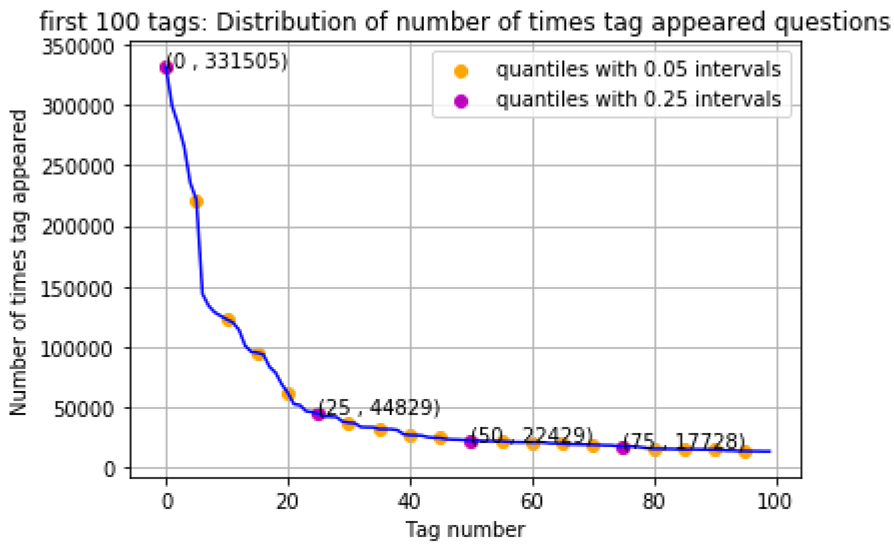

```
200 [331505 221533 122769  95160  62023  44829  37170  31897  26925  24537
  22429  21820  20957  19758  18905  17728  15533  15097  14884  13703
  13364  13157  12407  11658  11228  11162  10863  10600  10350  10224
  10029   9884   9719   9411   9252   9148   9040   8617   8361   8163
   8054   7867   7702   7564   7274   7151   7052   6847   6656   6553
   6466   6291   6183   6093   5971   5865   5760   5577   5490   5411
   5370   5283   5207   5107   5066   4983   4891   4785   4658   4549
   4526   4487   4429   4335   4310   4281   4239   4228   4195   4159
   4144   4088   4050   4002   3957   3929   3874   3849   3818   3797
   3750   3703   3685   3658   3615   3593   3564   3521   3505   3483
   3453   3427   3396   3363   3326   3299   3272   3232   3196   3168
   3123   3094   3073   3050   3012   2986   2983   2953   2934   2903
   2891   2844   2819   2784   2754   2738   2726   2708   2681   2669
   2647   2621   2604   2594   2556   2527   2510   2482   2460   2444
   2431   2409   2395   2380   2363   2331   2312   2297   2290   2281
   2259   2246   2222   2211   2198   2186   2162   2142   2132   2107
   2097   2078   2057   2045   2036   2020   2011   1994   1971   1965
   1959   1952   1940   1932   1912   1900   1879   1865   1855   1841
   1828   1821   1813   1801   1782   1770   1760   1747   1741   1734
   1723   1707   1697   1688   1683   1673   1665   1656   1646   1639]
```

In [23]:
```
# top 500 tags

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




```
100 [331505 221533 122769  95160  62023  44829  37170  31897  26925  24537
  22429  21820  20957  19758  18905  17728  15533  15097  14884  13703
  13364  13157  12407  11658  11228  11162  10863  10600  10350  10224
  10029   9884   9719   9411   9252   9148   9040   8617   8361   8163
   8054   7867   7702   7564   7274   7151   7052   6847   6656   6553
   6466   6291   6183   6093   5971   5865   5760   5577   5490   5411
   5370   5283   5207   5107   5066   4983   4891   4785   4658   4549
   4526   4487   4429   4335   4310   4281   4239   4228   4195   4159
   4144   4088   4050   4002   3957   3929   3874   3849   3818   3797
   3750   3703   3685   3658   3615   3593   3564   3521   3505   3483]
```

In [24]:
```
plt.plot(tag_counts[0:100], c='b')
plt.scatter(x=list(range(0,100,5)), y=tag_counts[0:100:5], c='orange', label=
"quantiles 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 = "q
uantiles 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 questi
ons')
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 [20]:
```
# 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

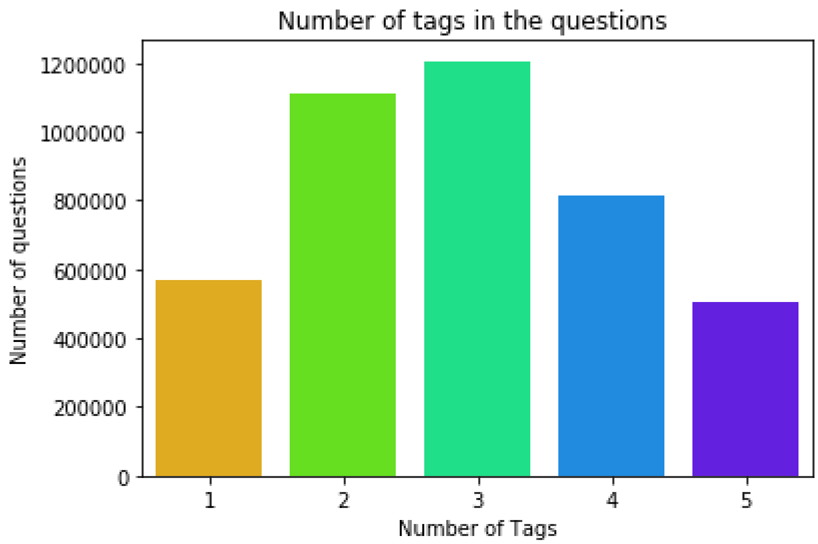
In [21]:
```
#Storing the count of tag in each question in list 'tag_count'
tag_quest_count = tag_dtm.sum(axis=1).tolist()
#Converting list of lists into single list, we will get [[3], [4], [2], [2],
 [3]] and we are converting this to [3, 4, 2, 2, 3]
tag_quest_count=[int(j) for i in tag_quest_count for j in i]
print ('We have total {} datapoints.'.format(len(tag_quest_count)))

print(tag_quest_count[:5])
```

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

In [22]:
```
print( "Maximum number of tags per question: %d"%max(tag_quest_count))
print( "Minimum number of tags per question: %d"%min(tag_quest_count))
print( "Avg. number of tags per question: %f"% ((sum(tag_quest_count)*1.0)/len
(tag_quest_count)))
```

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

In [67]:
```
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()
```

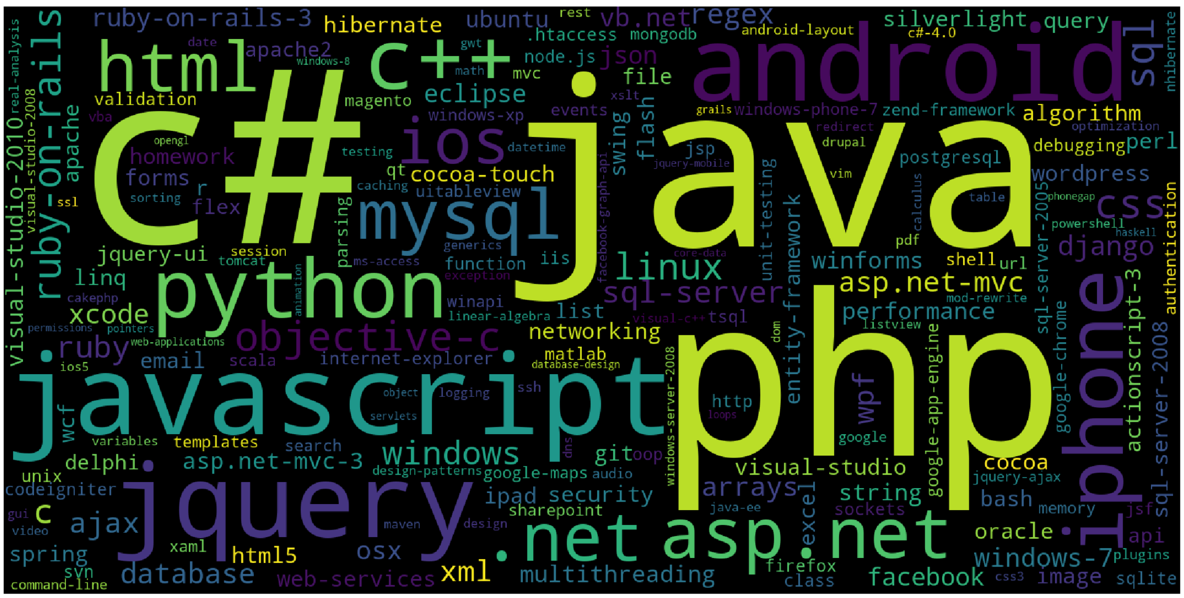


Observations:

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)
```

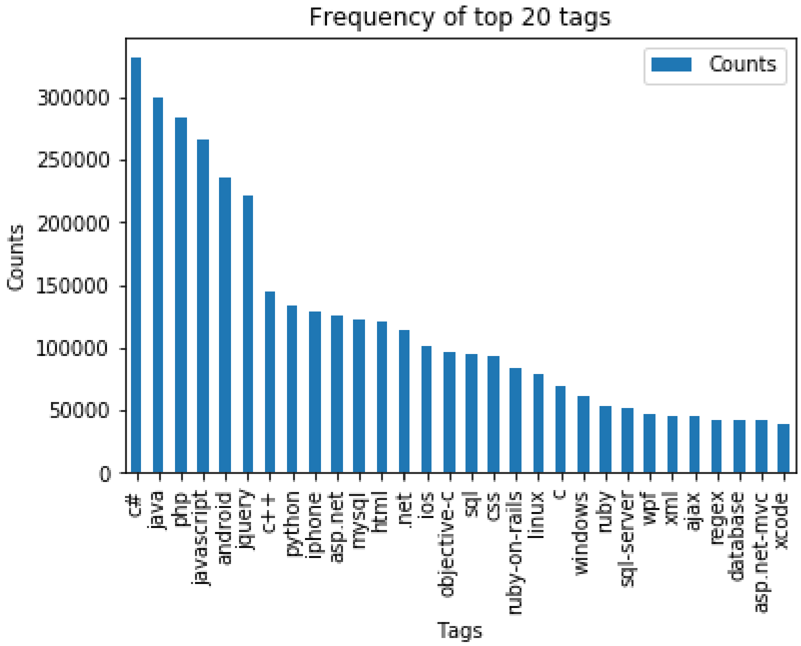



```
Time taken to run this cell : 0:00:05.177463
```

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 [28]:
```
i=np.arange(30)
tag_df_sorted.head(30).plot(kind='bar')
plt.title('Frequency of top 20 tags')
plt.xticks(i, tag_df_sorted['Tags'])
plt.xlabel('Tags')
plt.ylabel('Counts')
plt.show()
```



Observations:

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

3.3 Cleaning and preprocessing of Questions

3.3.1 Preprocessing

1. Sample 0.5M data points
2. Separate out code-snippets from Body
3. Remove Spcial characters from Question title and description (not in code)
4. Remove stop words (Except 'C')
5. Remove HTML Tags
6. Convert all the characters into small letters
7. Use SnowballStemmer to stem the words

In [23]:
```
import nltk
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     /home/pavana_paradesi5/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Out[23]: True

In [24]:
```
def striphtml(data):
    cleanr = re.compile('<.*?>')
    cleantext = re.sub(cleanr, ' ', str(data))
    return cleantext
stop_words = set(stopwords.words('english'))
stemmer = SnowballStemmer("english")
```

In [25]:

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

    return None

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

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

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

sql_create_table = """CREATE TABLE IF NOT EXISTS QuestionsProcessed (question
 text NOT NULL, code text, tags text, words_pre integer, words_post integer, i
s_code integer);"""
create_database_table("Processed.db", sql_create_table)
```

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

Modeling with less data points (0.2M data points) and more weight to title and 500 tags only.

So performing some hacks like:

Reducing 1 million data to 0.2 million data due to computational issues

1. Instead of using 5500 tags, we are taking 500 tags. as even 500 tags also covers 90% of questions.
2. As we have title and body text, the title has text with lot of meaning put in just some words/text. So we are giving more weightage of 3 times more to the text in title and keep the weightage for text in body as it is. Title plays an important role in answering the question. So we are implementing weighted models.
3. To give more weightage to the text in the title like 3 times more, we can just repeat the text in title 3 times.

In [26]:
```
sql_create_table = """CREATE TABLE IF NOT EXISTS QuestionsProcessed (question
 text NOT NULL, code text, tags text, words_pre integer, words_post integer, i
s_code integer);"""
create_database_table("Titlemoreweight.db", sql_create_table)
```

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

Creating new database called Processed db

In [27]:
```
# 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 = 160000
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 0.4M rows
        reader.execute("SELECT Title, Body, Tags From no_dup_train ORDER BY RANDOM() LIMIT 200001;")
        # for selecting random points
        #reader.execute("SELECT Title, Body, Tags From no_dup_train ORDER BY RANDOM() LIMIT 500001;")

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

```
Tables in the databse:
QuestionsProcessed
Cleared All the rows
Time taken to run this cell : 0:00:48.700549
```

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 [28]:
```
#http://www.bernzilla.com/2008/05/13/selecting-a-random-row-from-an-sqlite-tab
le/
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+" "+s
tr(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 th
e letter 'c'
    question=' '.join(str(stemmer.stem(j)) for j in words if j not in stop_wor
ds and (len(j)!=1 or j=='c'))

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

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

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

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

```
number of questions completed= 100000
number of questions completed= 200000
Avg. length of questions(Title+Body) before processing: 1173
Avg. length of questions(Title+Body) after processing: 407
Percent of questions containing code: 57
Time taken to run this cell : 0:07:02.416429
```

In [29]:
```
# never forget to close the conections or else we will end up with database lo
cks
conn_r.commit()
conn_w.commit()
conn_r.close()
conn_w.close()
```

Sample quesitons after preprocessing of data

In [30]:
```
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
==========================================================================================================
('reset radio group within tabl tr reset radio group within tabl tr reset rad
io group within tabl tr reset radio button mani row follow jqueri need user c
lick reset locat radio class reset locat set attr check fals radio tbl insid
row row hope help',)
----------------------------------------------------------------------------------------------------------
('creat object pool abl borrow return object creat object pool abl borrow ret
urn object creat object pool abl borrow return object want know possibl creat
pool object take object pool done work put pool',)
----------------------------------------------------------------------------------------------------------
('hard drive cooper hard drive cooper hard drive cooper run live disk verif r
estor macbook prompt disc repair tri partit close idea effect file back tri r
einstal lion comput came laptop bought know legal reinstal portion keep tell
contact appl though longer appl care anyth save file lost possibl corrupt har
d drive appreci help possibl',)
----------------------------------------------------------------------------------------------------------
('add exist element object add exist element object add exist element object
next object also function return element valu depend differ condit add valu r
eturn function',)
----------------------------------------------------------------------------------------------------------
('show result sql statement show result sql statement show result sql stateme
nt want display shirt alway get error code current place',)
----------------------------------------------------------------------------------------------------------
('programmat name filegroup sql server programmat name filegroup sql server p
rogrammat name filegroup sql server tri write store procedur creat new filegr
oup base upon given date paramet want see filegroup call someth like 2010 02
01 get filegroup call partitionnam',)
----------------------------------------------------------------------------------------------------------
('flex skin adob illustr flex skin adob illustr flex skin adob illustr develo
p skin flex use adob illustr ran problem design larger scrollbar skin use tou
ch screen applic flex seem appli default size regardless size symbol generat
.swf could achiev correct effect .png like know possibl use previous metho
d',)
----------------------------------------------------------------------------------------------------------
('primari key one entiti anoth entiti ef4 code-first primari key one entiti a
noth entiti ef4 code-first primari key one entiti anoth entiti ef4 code-first
say entiti want store chang post object also entiti right would defin histori
someth like problem figur get work ef4 code-first first model build fail prim
ari key defin context call follow except occur except talk object parameterle
ss constructor ad one help idea accomplish',)
----------------------------------------------------------------------------------------------------------
('sql movi databas search system sql movi databas search system sql movi data
bas search system build databas movi movi field genr actor director etc quest
ion design databas write sql movi multipl actor director etc right databas de
sign tabl movi movi id movi titl actor id director id genr id way seem harder
put multipl actor movi',)
```

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

Saving Preprocessed data to a Database

In [31]:
```
#Taking 0.2 Million entries to a dataframe.
write_db = 'Titlemoreweight.db'
if os.path.isfile(write_db):
    conn_r = create_connection(write_db)
    if conn_r is not None:
        preprocessed_data = pd.read_sql_query("""SELECT question, Tags FROM Qu
estionsProcessed""", conn_r)
conn_r.commit()
conn_r.close()
```

In [32]:
```
preprocessed_data.head()
```

Out[32]:

	question	tags
0	get asmx generat wsdl soap address use https g...	.net web-services wsdl asmx
1	reset radio group within tabl tr reset radio g...	jquery table dynamic radio
2	creat object pool abl borrow return object cre...	java pool
3	hard drive cooper hard drive cooper hard drive...	osx hard-drive
4	add exist element object add exist element obj...	javascript

In [33]:
```
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 : 200000
number of dimensions : 2
```

4. Machine Learning Models

4.1 Converting tags for multilabel problems

There are 4 methods to convert multi-label problem into single-label problem.

Binary Relevance:-

1. Converting multi-label classification into binary class classification / single class classification is called one vs rest.
2. We can use skmultilearn library when we have multi label classification task.
3. In practice this method is used a lot in multi-label problems and it is simplest of all 4 methods

Classifier chains:-

1. If we have X(X1,X2,X3) as input data points and Y1,Y2,Y3,Y4 as labels for these. We construct a classifier 1 with X and Y1. Next classifier 2 with X, Y1 and predict Y2. For classifier 3, it takes X,Y1,Y2 as training data and predict Y3 as output. Same process for classifier 4. These are called classifier chains.
2. When there is a relation between 2 labels / corelation between labels or if one label can predict other label, then classifier chains method is useful.

Label Powerset:-

Here if the binary strings are same, it gives them the same class. This converts multi-label problem into multi-class problem.

Adapted Algorithm:-

Here KNN algorithm is coverted as MLkNN which is a multi-label version of KNN which implements nearest neighbour method. skmultilearn has MLkNN implementation internally.

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

1. We are applying Binary Relevance method where instead of using all 42k tage, we can take subset of tags using Partial coverage method.
2. We take the subset of tags with high frequency of occurance or the top frequency tags which will occur in most set of questions. These subset of tags can partially cover the most number of questions.
3. We are converting the tags into binary vector.

Converting string Tags to multilable output variables

Using Count Vectorizer on tags to create binary vectors

In [34]:
```
# Converting string Tags to multilable output variables
# binary='true' will give a binary vectorizer
vectorizer = CountVectorizer(tokenizer = lambda x: x.split(), binary='true')
multilabel_y = vectorizer.fit_transform(preprocessed_data['tags'])
```

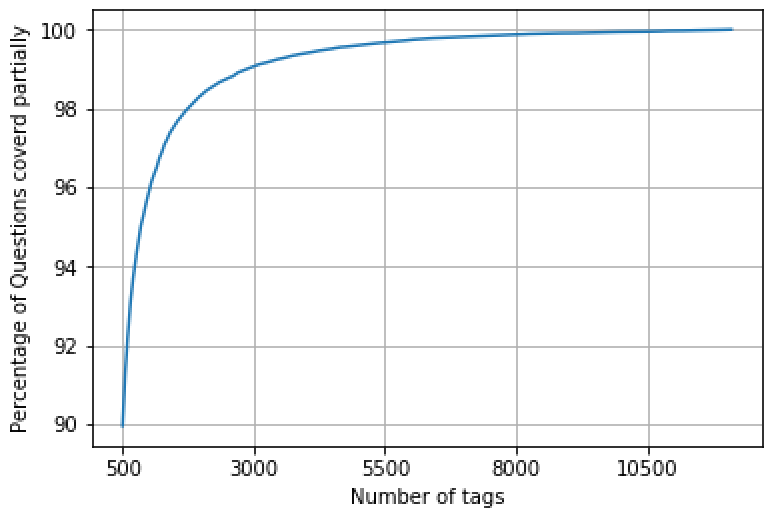
We will sample the number of tags instead considering all of them (due to limitation of computing power)

In [35]:
```
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 [36]:
```
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 [37]:
```
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("Percentage of Questions coverd partially")
plt.grid()
plt.show()
# you can choose any number of tags based on your computing power, minimun is
 500(it covers 90% of the tags)
print("with ",5500,"tags we are covering ",questions_explained[50],"% of quest
ions")
print("with ",500,"tags we are covering ",questions_explained[0],"% of questio
ns")
```




```
with  5500 tags we are covering  99.067 % of questions
with  500 tags we are covering  89.95 % of questions
```

In [38]:
```
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 : 20099 out of  200000
```

In [39]:
```
print("Number of tags in sample :", multilabel_y.shape[1])
print("number of tags taken :", multilabel_yx.shape[1],"(",(multilabel_yx.shap
e[1]/multilabel_y.shape[1])*100,"%)")
```

```
Number of tags in sample : 23719
number of tags taken : 500 ( 2.108014671782116 %)
```

We consider top 2.1% tags which covers 90% of the questions

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

There are no time stamps given. So this is not time based splitting. Here we are doing random splitting.

In [40]:
```
x_train=preprocessed_data.head(train_datasize)
x_test=preprocessed_data.tail(preprocessed_data.shape[0] - 160000)

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

In [41]:
```
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 : (160000, 500)
Number of data points in test data : (40000, 500)
```

4.3 Featurizing data using BoW

1. BoW is one of the popular representations of representing the text(processed text) into vectors. Here only title and body are the text and code is eliminated from this.
2. Tags are binary vectors.

In [42]:
```
# https://medium.com/@rnbrown/more-nlp-with-sklearns-countvectorizer-add577a0b
8c8
# max features is The CountVectorizer will choose the words/features that occu
r most frequently to be in its’ vocabulary
# and drop everything else.

start = datetime.now()
vectorizer = CountVectorizer(min_df=0.00009, max_features=40000, tokenizer = l
ambda x: x.split(), ngram_range=(1,4))
x_train_multilabel = vectorizer.fit_transform(x_train['question'])
x_test_multilabel = vectorizer.transform(x_test['question'])
print("Time taken to run this cell :", datetime.now() - start)
```

```
Time taken to run this cell : 0:04:26.057133
```

In [43]:
```
print("Dimensions of train data X:",x_train_multilabel.shape, "Y :",y_train.sh
ape)
print("Dimensions of test data X:",x_test_multilabel.shape,"Y:",y_test.shape)
```

```
Dimensions of train data X: (160000, 40000) Y : (160000, 500)
Dimensions of test data X: (40000, 40000) Y: (40000, 500)
```

Applying Logistic Regression with SGDClassifier (loss='log') with OneVsRest Classifier

1. Here we are using Logistic Regression model as training is cheap compared to complex models like SVM, RF, GBDT.
2. As there is high dimensional data, logistic regression performs very well. RF, GBDT may not work very well when we have high dimensional data.
3. Here we are using OneVsRestClassifier as it can be used for multi-label setting and multi-class setting. But here in multi-class, the class label becomes the vector.
4. SGDClassifier with log loss is Logistic Regression.

Using 0.2 million questions, 500 tags

Hyper parameter Tuning

In [44]:
```
# Hyper parameter tuning
from sklearn.model_selection import GridSearchCV
from sklearn.multiclass import OneVsRestClassifier

parameters = {'estimator__alpha':[0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 10
0]}
classifier = OneVsRestClassifier(SGDClassifier(loss='log', penalty='l1'))

model = GridSearchCV(estimator = classifier, param_grid=parameters, cv=3, verb
ose=0, scoring='f1_micro',n_jobs = -1)
model.fit(x_train_multilabel, y_train)

print(model.best_estimator_)
optimal_alpha = model.best_estimator_.get_params()['estimator__alpha']
print('best alpha value after hyperparameter tuning is : ',optimal_alpha)
```

```
OneVsRestClassifier(estimator=SGDClassifier(alpha=0.0001, average=False,
                                            class_weight=None,
                                            early_stopping=False, epsilon=0.
1,
                                            eta0=0.0, fit_intercept=True,
                                            l1_ratio=0.15,
                                            learning_rate='optimal', loss='lo
g',
                                            max_iter=1000, n_iter_no_change=
5,
                                            n_jobs=None, penalty='l1',
                                            power_t=0.5, random_state=None,
                                            shuffle=True, tol=0.001,
                                            validation_fraction=0.1, verbose=
0,
                                            warm_start=False),
                    n_jobs=None)
best alpha value after hyperparameter tuning is :  0.0001
```

Model with optimal alpha

In [46]:
```
start = datetime.now()
clf = OneVsRestClassifier(SGDClassifier(loss='log', alpha=optimal_alpha, penal
ty='l1'), n_jobs=-1)
clf.fit(x_train_multilabel, y_train)
predictions = clf.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 : 0.16095
Hamming loss  0.00369645
Micro-average quality numbers
Precision: 0.4876, Recall: 0.4392, F1-measure: 0.4621
Macro-average quality numbers
Precision: 0.3731, Recall: 0.3605, F1-measure: 0.3594
              precision    recall  f1-score   support

           0       0.53      0.39      0.45      3177
           1       0.67      0.51      0.58      2878
           2       0.71      0.62      0.66      2720
           3       0.60      0.47      0.53      2514
           4       0.83      0.81      0.82      2214
           5       0.77      0.70      0.73      2163
           6       0.64      0.57      0.60      1318
           7       0.75      0.69      0.72      1278
           8       0.61      0.45      0.52      1289
           9       0.62      0.49      0.55      1185
          10       0.72      0.67      0.69      1186
          11       0.39      0.27      0.32      1128
          12       0.30      0.20      0.24      1096
          13       0.50      0.33      0.40       974
          14       0.44      0.31      0.37       947
          15       0.43      0.38      0.40       917
          16       0.62      0.61      0.61       910
          17       0.62      0.62      0.62       781
          18       0.51      0.31      0.38       731
          19       0.52      0.34      0.42       676
          20       0.25      0.17      0.20       612
          21       0.64      0.49      0.56       522
          22       0.36      0.43      0.40       451
          23       0.67      0.73      0.70       399
          24       0.48      0.44      0.46       428
          25       0.50      0.51      0.50       428
          26       0.71      0.75      0.73       409
          27       0.40      0.43      0.42       367
          28       0.21      0.12      0.16       370
          29       0.42      0.28      0.33       379
          30       0.43      0.40      0.41       381
          31       0.83      0.84      0.83       350
          32       0.32      0.28      0.30       328
          33       0.35      0.40      0.37       331
          34       0.63      0.53      0.58       284
          35       0.48      0.59      0.53       290
          36       0.67      0.61      0.64       314
          37       0.27      0.23      0.25       301
          38       0.66      0.58      0.62       308
          39       0.29      0.18      0.22       269
          40       0.48      0.46      0.47       263
          41       0.51      0.45      0.48       279
          42       0.22      0.21      0.21       219
          43       0.42      0.33      0.37       253
          44       0.49      0.39      0.44       231
          45       0.21      0.09      0.12       255
          46       0.51      0.43      0.47       217
          47       0.27      0.27      0.27       199
          48       0.41      0.36      0.38       228
```

49	0.50	0.51	0.51	219
50	0.67	0.75	0.71	225
51	0.50	0.57	0.53	193
52	0.12	0.12	0.12	223
53	0.27	0.24	0.26	220
54	0.23	0.32	0.27	186
55	0.26	0.17	0.21	206
56	0.61	0.63	0.62	203
57	0.80	0.88	0.84	202
58	0.53	0.52	0.53	187
59	0.23	0.23	0.23	191
60	0.16	0.08	0.11	199
61	0.37	0.48	0.42	201
62	0.76	0.76	0.76	206
63	0.51	0.34	0.41	196
64	0.19	0.22	0.20	213
65	0.55	0.37	0.44	190
66	0.55	0.59	0.57	181
67	0.10	0.05	0.07	197
68	0.22	0.26	0.24	178
69	0.58	0.55	0.56	176
70	0.42	0.47	0.45	169
71	0.32	0.44	0.37	168
72	0.62	0.29	0.40	199
73	0.63	0.57	0.60	152
74	0.66	0.55	0.60	154
75	0.46	0.49	0.47	185
76	0.40	0.48	0.44	156
77	0.68	0.73	0.71	165
78	0.19	0.11	0.14	150
79	0.36	0.40	0.38	151
80	0.28	0.42	0.33	127
81	0.28	0.26	0.27	145
82	0.71	0.70	0.71	148
83	0.22	0.24	0.23	154
84	0.87	0.70	0.78	171
85	0.33	0.37	0.35	159
86	0.15	0.16	0.16	143
87	0.56	0.58	0.57	144
88	0.81	0.77	0.79	142
89	0.66	0.79	0.72	135
90	0.73	0.68	0.70	130
91	0.45	0.57	0.50	144
92	0.47	0.50	0.48	145
93	0.38	0.31	0.34	127
94	0.68	0.71	0.70	135
95	0.19	0.13	0.15	138
96	0.79	0.60	0.68	131
97	0.29	0.38	0.33	130
98	0.77	0.85	0.81	118
99	0.62	0.79	0.69	132
100	0.32	0.21	0.26	117
101	0.66	0.78	0.71	121
102	0.28	0.38	0.32	108
103	0.10	0.12	0.11	111
104	0.13	0.09	0.11	132
105	0.33	0.50	0.40	111

106	0.59	0.52	0.56	122
107	0.48	0.34	0.40	135
108	0.45	0.51	0.48	144
109	0.34	0.26	0.30	126
110	0.52	0.62	0.56	121
111	0.14	0.23	0.17	118
112	0.33	0.44	0.38	101
113	0.43	0.25	0.32	100
114	0.50	0.50	0.50	118
115	0.32	0.25	0.28	109
116	0.72	0.67	0.69	129
117	0.81	0.82	0.81	106
118	0.14	0.14	0.14	104
119	0.36	0.38	0.37	125
120	0.61	0.56	0.58	109
121	0.28	0.23	0.25	128
122	0.40	0.48	0.43	110
123	0.27	0.26	0.26	105
124	0.30	0.24	0.27	103
125	0.41	0.39	0.40	107
126	0.21	0.21	0.21	100
127	0.34	0.32	0.33	110
128	0.84	0.87	0.86	109
129	0.10	0.07	0.08	107
130	0.24	0.40	0.30	90
131	0.19	0.11	0.14	95
132	0.12	0.15	0.13	101
133	0.83	0.72	0.77	103
134	0.39	0.27	0.32	86
135	0.05	0.02	0.02	127
136	0.79	0.81	0.80	111
137	0.14	0.15	0.14	98
138	0.36	0.49	0.42	106
139	0.08	0.09	0.08	98
140	0.27	0.27	0.27	92
141	0.53	0.64	0.58	109
142	0.10	0.08	0.09	105
143	0.50	0.59	0.54	116
144	0.52	0.42	0.46	101
145	0.50	0.52	0.51	103
146	0.25	0.12	0.16	109
147	0.70	0.78	0.74	88
148	0.20	0.29	0.24	82
149	0.61	0.74	0.67	77
150	0.36	0.42	0.38	103
151	0.08	0.08	0.08	92
152	0.18	0.23	0.20	88
153	0.38	0.44	0.41	100
154	0.24	0.22	0.23	90
155	0.23	0.24	0.24	88
156	0.45	0.33	0.38	103
157	0.14	0.15	0.14	95
158	0.22	0.25	0.23	106
159	0.20	0.27	0.23	93
160	0.77	0.83	0.80	103
161	0.46	0.23	0.30	102
162	0.26	0.39	0.31	85

163	0.19	0.28	0.23	81
164	0.46	0.44	0.45	94
165	0.23	0.12	0.16	92
166	0.54	0.55	0.54	88
167	0.32	0.29	0.31	82
168	0.44	0.51	0.47	81
169	0.20	0.25	0.22	81
170	0.31	0.20	0.24	80
171	0.80	0.88	0.84	76
172	0.23	0.25	0.24	89
173	0.59	0.57	0.58	91
174	0.27	0.28	0.28	78
175	0.29	0.42	0.34	85
176	0.29	0.24	0.26	95
177	0.86	0.77	0.81	73
178	0.45	0.44	0.45	72
179	0.76	0.76	0.76	89
180	0.95	0.72	0.82	100
181	0.33	0.15	0.21	79
182	0.63	0.59	0.61	78
183	0.62	0.67	0.64	84
184	0.21	0.26	0.23	80
185	0.14	0.06	0.09	80
186	0.40	0.37	0.39	91
187	0.31	0.22	0.26	77
188	0.48	0.58	0.53	72
189	0.24	0.10	0.14	72
190	0.20	0.11	0.14	90
191	0.78	0.84	0.81	86
192	0.52	0.52	0.52	84
193	0.18	0.13	0.15	71
194	0.58	0.66	0.62	73
195	0.10	0.17	0.13	69
196	0.25	0.38	0.30	78
197	0.18	0.20	0.19	79
198	0.25	0.30	0.27	82
199	0.26	0.24	0.25	85
200	0.72	0.50	0.59	84
201	0.21	0.18	0.19	72
202	0.13	0.07	0.09	75
203	0.07	0.04	0.05	72
204	0.40	0.53	0.46	88
205	0.75	0.85	0.80	68
206	0.77	0.56	0.65	78
207	0.03	0.01	0.02	83
208	0.47	0.58	0.52	71
209	0.57	0.68	0.62	79
210	0.66	0.75	0.70	76
211	0.31	0.28	0.30	74
212	0.16	0.17	0.16	77
213	0.51	0.38	0.44	60
214	0.55	0.48	0.52	64
215	0.15	0.04	0.06	73
216	0.50	0.53	0.52	73
217	0.20	0.13	0.16	79
218	0.37	0.55	0.44	55
219	0.07	0.07	0.07	61

220	0.35	0.31	0.33	80
221	0.14	0.16	0.15	68
222	0.43	0.62	0.51	60
223	0.37	0.44	0.40	64
224	0.33	0.33	0.33	60
225	0.62	0.59	0.60	63
226	0.26	0.35	0.30	72
227	0.19	0.26	0.22	68
228	0.39	0.41	0.40	64
229	0.17	0.18	0.18	66
230	0.08	0.09	0.09	68
231	0.49	0.63	0.55	68
232	0.22	0.17	0.19	69
233	0.87	0.88	0.87	66
234	0.23	0.16	0.19	55
235	0.13	0.08	0.10	65
236	0.31	0.29	0.30	63
237	0.29	0.23	0.25	66
238	0.03	0.03	0.03	63
239	0.37	0.23	0.28	70
240	0.47	0.40	0.43	58
241	0.27	0.42	0.33	67
242	0.37	0.49	0.42	73
243	0.29	0.46	0.35	63
244	0.31	0.41	0.35	63
245	0.82	0.80	0.81	61
246	0.25	0.22	0.24	58
247	0.59	0.63	0.61	65
248	0.03	0.05	0.04	43
249	0.26	0.20	0.23	54
250	0.59	0.58	0.59	55
251	0.46	0.42	0.44	55
252	0.09	0.18	0.12	45
253	0.25	0.30	0.27	61
254	0.55	0.52	0.54	71
255	0.46	0.37	0.41	57
256	0.42	0.67	0.52	58
257	0.47	0.41	0.44	68
258	0.51	0.64	0.57	55
259	0.24	0.29	0.26	62
260	0.56	0.65	0.60	62
261	0.08	0.04	0.05	57
262	0.62	0.84	0.71	56
263	0.35	0.37	0.36	51
264	0.62	0.84	0.71	50
265	0.66	0.68	0.67	60
266	0.05	0.04	0.04	52
267	0.07	0.03	0.05	58
268	0.35	0.24	0.29	62
269	0.73	0.59	0.65	54
270	0.15	0.18	0.17	55
271	0.17	0.22	0.19	50
272	0.13	0.18	0.15	57
273	0.00	0.00	0.00	66
274	0.31	0.43	0.36	56
275	0.00	0.00	0.00	67
276	0.35	0.60	0.44	50

277	0.71	0.83	0.77	53
278	0.48	0.24	0.32	66
279	0.16	0.13	0.14	63
280	0.30	0.17	0.22	64
281	0.15	0.04	0.06	51
282	0.77	0.80	0.78	54
283	0.35	0.18	0.24	60
284	0.23	0.30	0.26	54
285	0.31	0.34	0.33	58
286	0.35	0.37	0.36	60
287	0.34	0.39	0.37	56
288	0.19	0.15	0.17	53
289	0.08	0.06	0.07	62
290	0.47	0.31	0.38	61
291	0.03	0.07	0.04	43
292	0.25	0.35	0.29	57
293	0.55	0.76	0.64	54
294	0.06	0.09	0.07	57
295	0.03	0.06	0.04	48
296	0.26	0.53	0.35	47
297	0.83	0.78	0.80	67
298	0.35	0.49	0.41	43
299	0.41	0.44	0.43	52
300	0.30	0.19	0.23	48
301	0.21	0.22	0.21	54
302	0.71	0.33	0.45	52
303	0.05	0.05	0.05	41
304	0.30	0.13	0.18	46
305	0.38	0.25	0.30	48
306	0.06	0.07	0.06	46
307	0.08	0.14	0.10	44
308	0.30	0.31	0.31	42
309	0.18	0.21	0.19	43
310	0.21	0.12	0.15	43
311	0.07	0.10	0.08	62
312	0.45	0.44	0.44	50
313	0.59	0.33	0.43	60
314	0.24	0.11	0.15	47
315	0.76	0.59	0.67	59
316	0.17	0.15	0.16	48
317	0.17	0.11	0.13	56
318	0.46	0.49	0.48	53
319	0.30	0.19	0.23	43
320	0.25	0.29	0.27	52
321	0.58	0.58	0.58	45
322	0.52	0.37	0.43	43
323	0.20	0.06	0.10	47
324	0.38	0.32	0.35	47
325	0.00	0.00	0.00	40
326	0.68	0.85	0.75	27
327	0.35	0.35	0.35	43
328	0.17	0.13	0.15	52
329	0.24	0.18	0.21	44
330	0.22	0.29	0.25	38
331	0.71	0.64	0.67	55
332	0.79	0.69	0.74	45
333	0.08	0.05	0.06	39

334	0.21	0.11	0.15	44
335	0.73	0.65	0.69	49
336	0.61	0.71	0.65	52
337	0.03	0.02	0.02	44
338	0.22	0.11	0.14	47
339	0.22	0.30	0.26	43
340	0.74	0.57	0.64	46
341	0.29	0.29	0.29	34
342	0.18	0.23	0.20	44
343	0.40	0.44	0.42	48
344	0.23	0.27	0.25	37
345	0.36	0.50	0.42	40
346	0.22	0.11	0.15	55
347	0.08	0.10	0.09	41
348	0.38	0.52	0.44	48
349	0.60	0.42	0.49	50
350	0.30	0.30	0.30	54
351	0.23	0.20	0.21	45
352	0.48	0.21	0.30	56
353	0.25	0.20	0.22	40
354	0.06	0.07	0.06	42
355	0.22	0.23	0.22	47
356	0.38	0.26	0.31	38
357	0.38	0.40	0.39	45
358	0.17	0.25	0.21	32
359	0.12	0.04	0.06	47
360	0.09	0.18	0.12	34
361	0.64	0.79	0.71	29
362	0.68	0.69	0.68	36
363	0.14	0.14	0.14	35
364	0.19	0.17	0.18	40
365	0.05	0.07	0.06	40
366	0.15	0.10	0.12	41
367	0.63	0.54	0.58	50
368	0.37	0.38	0.37	47
369	0.53	0.53	0.53	36
370	0.19	0.42	0.26	33
371	0.36	0.23	0.28	43
372	0.66	0.60	0.62	42
373	0.16	0.32	0.21	37
374	0.20	0.26	0.22	50
375	0.31	0.37	0.34	46
376	0.34	0.47	0.40	38
377	0.32	0.41	0.36	32
378	0.59	0.45	0.51	38
379	0.15	0.14	0.14	36
380	0.14	0.10	0.12	39
381	0.59	0.85	0.69	40
382	0.34	0.32	0.33	47
383	0.09	0.09	0.09	43
384	0.12	0.25	0.17	44
385	0.40	0.40	0.40	30
386	0.35	0.34	0.35	32
387	0.65	0.44	0.52	39
388	0.27	0.50	0.35	30
389	0.70	0.62	0.66	37
390	0.56	0.36	0.44	39

391	0.08	0.06	0.07	50
392	0.09	0.04	0.06	46
393	0.20	0.16	0.18	38
394	0.80	0.85	0.83	39
395	0.39	0.48	0.43	29
396	0.31	0.39	0.35	38
397	0.00	0.00	0.00	48
398	0.17	0.23	0.20	31
399	0.12	0.05	0.07	44
400	0.34	0.31	0.33	35
401	0.31	0.22	0.26	50
402	0.17	0.27	0.21	37
403	0.65	0.68	0.67	44
404	0.86	0.64	0.74	39
405	0.25	0.29	0.27	34
406	0.22	0.30	0.25	33
407	0.22	0.21	0.21	38
408	0.27	0.08	0.12	39
409	0.23	0.18	0.20	50
410	0.00	0.00	0.00	36
411	0.57	0.71	0.63	28
412	0.46	0.40	0.43	40
413	0.05	0.07	0.06	40
414	0.17	0.16	0.17	43
415	0.12	0.04	0.06	47
416	0.45	0.33	0.38	42
417	0.22	0.18	0.20	45
418	0.42	0.46	0.44	48
419	0.69	0.43	0.53	47
420	0.36	0.23	0.28	43
421	0.51	0.74	0.60	35
422	0.76	0.34	0.47	47
423	0.53	0.47	0.49	43
424	0.37	0.23	0.29	43
425	0.12	0.07	0.09	40
426	0.49	0.56	0.52	41
427	0.27	0.33	0.30	42
428	0.20	0.03	0.06	31
429	0.16	0.22	0.18	27
430	0.13	0.22	0.16	41
431	0.62	0.57	0.59	28
432	0.14	0.07	0.09	43
433	0.21	0.26	0.23	38
434	0.12	0.12	0.12	43
435	0.81	0.74	0.77	34
436	0.07	0.10	0.09	30
437	0.09	0.05	0.06	40
438	0.46	0.30	0.36	40
439	0.30	0.50	0.37	30
440	0.04	0.03	0.03	36
441	0.25	0.22	0.23	32
442	0.07	0.08	0.07	38
443	0.13	0.20	0.16	40
444	0.36	0.36	0.36	28
445	0.46	0.40	0.43	40
446	0.35	0.23	0.27	40
447	0.50	0.59	0.54	32

448	0.54	0.56	0.55	39
449	0.90	0.73	0.81	37
450	0.10	0.12	0.11	43
451	0.73	0.73	0.73	37
452	0.24	0.09	0.13	46
453	0.31	0.33	0.32	42
454	0.33	0.37	0.35	27
455	0.46	0.69	0.55	32
456	0.31	0.20	0.24	40
457	0.15	0.11	0.13	37
458	0.63	0.59	0.61	41
459	0.43	0.27	0.33	45
460	0.27	0.25	0.26	40
461	0.36	0.43	0.39	28
462	0.03	0.04	0.03	27
463	0.09	0.08	0.08	39
464	0.46	0.49	0.47	35
465	0.00	0.00	0.00	35
466	0.18	0.30	0.23	33
467	0.21	0.12	0.15	41
468	0.44	0.31	0.37	35
469	0.22	0.06	0.10	32
470	0.07	0.03	0.04	35
471	0.19	0.20	0.19	40
472	0.42	0.38	0.40	26
473	0.74	0.48	0.58	29
474	0.22	0.06	0.09	35
475	0.24	0.21	0.23	43
476	0.47	0.24	0.32	37
477	1.00	0.75	0.86	32
478	0.23	0.38	0.29	29
479	0.40	0.41	0.41	34
480	0.10	0.12	0.11	32
481	0.75	0.51	0.61	35
482	0.75	0.62	0.68	39
483	0.34	0.33	0.34	30
484	0.12	0.09	0.11	32
485	0.24	0.26	0.25	35
486	0.16	0.19	0.18	36
487	0.00	0.00	0.00	28
488	0.82	0.66	0.73	41
489	0.00	0.00	0.00	25
490	0.00	0.00	0.00	30
491	0.71	0.62	0.67	24
492	0.57	0.49	0.52	35
493	0.10	0.17	0.13	30
494	0.17	0.02	0.04	42
495	0.18	0.22	0.20	27
496	0.56	0.47	0.51	32
497	0.37	0.73	0.49	30
498	0.36	0.38	0.37	26
499	0.07	0.11	0.09	27
micro avg	0.49	0.44	0.46	72315
macro avg	0.37	0.36	0.36	72315
weighted avg	0.49	0.44	0.46	72315
samples avg	0.44	0.43	0.40	72315

```
Time taken to run this cell : 0:05:28.591649
```

Observation:-

1. The model with SGDClassifier with log loss, The Micro F1-score is 0.4621

Applying Logistic Regression with L1 penalty as it works well with sparse data.

In [47]:
```
start = datetime.now()
classifier_2 = OneVsRestClassifier(LogisticRegression(penalty='l1'), n_jobs=-1
)
classifier_2.fit(x_train_multilabel, y_train)
predictions_2 = classifier_2.predict(x_test_multilabel)
print("Accuracy :",metrics.accuracy_score(y_test, predictions_2))
print("Hamming loss ",metrics.hamming_loss(y_test,predictions_2))


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

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

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

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

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

```
Accuracy : 0.21365
Hamming loss  0.00318115
Micro-average quality numbers
Precision: 0.5804, Recall: 0.4339, F1-measure: 0.4965
Macro-average quality numbers
Precision: 0.4634, Recall: 0.3554, F1-measure: 0.3981
```

	precision	recall	f1-score	support
0	0.55	0.44	0.49	3177
1	0.68	0.52	0.59	2878
2	0.74	0.61	0.67	2720
3	0.63	0.48	0.54	2514
4	0.91	0.79	0.85	2214
5	0.79	0.68	0.73	2163
6	0.71	0.53	0.61	1318
7	0.82	0.64	0.72	1278
8	0.63	0.46	0.53	1289
9	0.65	0.48	0.55	1185
10	0.78	0.68	0.72	1186
11	0.41	0.32	0.36	1128
12	0.34	0.19	0.24	1096
13	0.51	0.37	0.43	974
14	0.45	0.30	0.36	947
15	0.47	0.37	0.42	917
16	0.71	0.60	0.65	910
17	0.74	0.58	0.65	781
18	0.48	0.34	0.40	731
19	0.53	0.36	0.43	676
20	0.33	0.20	0.25	612
21	0.59	0.41	0.49	522
22	0.46	0.38	0.42	451
23	0.81	0.67	0.73	399
24	0.58	0.46	0.51	428
25	0.60	0.49	0.54	428
26	0.84	0.71	0.77	409
27	0.54	0.43	0.48	367
28	0.22	0.12	0.16	370
29	0.46	0.28	0.35	379
30	0.50	0.38	0.43	381
31	0.90	0.79	0.84	350
32	0.44	0.31	0.37	328
33	0.53	0.40	0.45	331
34	0.60	0.44	0.51	284
35	0.68	0.57	0.62	290
36	0.78	0.64	0.70	314
37	0.36	0.25	0.29	301
38	0.76	0.58	0.66	308
39	0.30	0.19	0.23	269
40	0.62	0.47	0.53	263
41	0.59	0.36	0.45	279
42	0.23	0.21	0.22	219
43	0.51	0.33	0.40	253
44	0.52	0.38	0.44	231
45	0.24	0.16	0.19	255
46	0.56	0.39	0.46	217
47	0.32	0.25	0.28	199
48	0.53	0.31	0.39	228

49	0.64	0.47	0.54	219
50	0.83	0.75	0.79	225
51	0.67	0.58	0.62	193
52	0.22	0.10	0.14	223
53	0.31	0.24	0.27	220
54	0.42	0.37	0.39	186
55	0.28	0.17	0.21	206
56	0.72	0.63	0.67	203
57	0.86	0.82	0.84	202
58	0.73	0.59	0.65	187
59	0.23	0.14	0.17	191
60	0.15	0.09	0.11	199
61	0.54	0.49	0.51	201
62	0.87	0.68	0.77	206
63	0.60	0.37	0.45	196
64	0.32	0.20	0.24	213
65	0.47	0.36	0.41	190
66	0.72	0.59	0.65	181
67	0.12	0.05	0.07	197
68	0.37	0.24	0.29	178
69	0.73	0.50	0.59	176
70	0.58	0.43	0.49	169
71	0.47	0.36	0.41	168
72	0.62	0.25	0.35	199
73	0.71	0.56	0.62	152
74	0.64	0.50	0.56	154
75	0.58	0.47	0.52	185
76	0.59	0.54	0.56	156
77	0.78	0.68	0.73	165
78	0.13	0.08	0.10	150
79	0.56	0.36	0.44	151
80	0.56	0.35	0.43	127
81	0.42	0.31	0.36	145
82	0.78	0.60	0.68	148
83	0.32	0.20	0.25	154
84	0.82	0.67	0.74	171
85	0.51	0.45	0.48	159
86	0.26	0.22	0.24	143
87	0.70	0.56	0.63	144
88	0.91	0.72	0.80	142
89	0.78	0.72	0.75	135
90	0.79	0.60	0.68	130
91	0.67	0.57	0.62	144
92	0.66	0.49	0.56	145
93	0.34	0.27	0.30	127
94	0.84	0.65	0.73	135
95	0.32	0.17	0.22	138
96	0.80	0.59	0.68	131
97	0.44	0.33	0.38	130
98	0.82	0.79	0.81	118
99	0.81	0.80	0.81	132
100	0.42	0.27	0.33	117
101	0.86	0.72	0.78	121
102	0.38	0.31	0.34	108
103	0.27	0.15	0.20	111
104	0.21	0.11	0.15	132
105	0.46	0.42	0.44	111

106	0.67	0.48	0.56	122
107	0.45	0.20	0.28	135
108	0.59	0.47	0.53	144
109	0.47	0.28	0.35	126
110	0.70	0.60	0.65	121
111	0.29	0.14	0.18	118
112	0.44	0.38	0.41	101
113	0.31	0.24	0.27	100
114	0.55	0.50	0.52	118
115	0.52	0.31	0.39	109
116	0.82	0.72	0.77	129
117	0.91	0.76	0.83	106
118	0.24	0.19	0.21	104
119	0.48	0.40	0.43	125
120	0.72	0.55	0.62	109
121	0.37	0.23	0.29	128
122	0.52	0.39	0.45	110
123	0.28	0.18	0.22	105
124	0.31	0.21	0.25	103
125	0.40	0.31	0.35	107
126	0.29	0.22	0.25	100
127	0.49	0.30	0.37	110
128	0.93	0.86	0.90	109
129	0.11	0.05	0.07	107
130	0.38	0.36	0.37	90
131	0.25	0.16	0.19	95
132	0.33	0.22	0.26	101
133	0.85	0.71	0.77	103
134	0.45	0.34	0.39	86
135	0.07	0.02	0.03	127
136	0.92	0.86	0.89	111
137	0.24	0.14	0.18	98
138	0.50	0.42	0.45	106
139	0.14	0.08	0.10	98
140	0.32	0.20	0.24	92
141	0.76	0.61	0.68	109
142	0.10	0.06	0.07	105
143	0.70	0.54	0.61	116
144	0.61	0.48	0.53	101
145	0.68	0.43	0.52	103
146	0.26	0.14	0.18	109
147	0.85	0.76	0.80	88
148	0.37	0.40	0.39	82
149	0.67	0.60	0.63	77
150	0.50	0.42	0.46	103
151	0.20	0.13	0.16	92
152	0.31	0.23	0.26	88
153	0.51	0.35	0.41	100
154	0.37	0.26	0.30	90
155	0.33	0.24	0.28	88
156	0.53	0.30	0.39	103
157	0.22	0.15	0.17	95
158	0.31	0.16	0.21	106
159	0.33	0.28	0.30	93
160	0.92	0.76	0.83	103
161	0.41	0.18	0.25	102
162	0.44	0.36	0.40	85

163	0.42	0.27	0.33	81
164	0.77	0.56	0.65	94
165	0.27	0.12	0.17	92
166	0.65	0.55	0.59	88
167	0.26	0.23	0.25	82
168	0.49	0.43	0.46	81
169	0.36	0.35	0.35	81
170	0.35	0.24	0.28	80
171	0.90	0.86	0.88	76
172	0.39	0.28	0.33	89
173	0.67	0.56	0.61	91
174	0.32	0.29	0.30	78
175	0.42	0.39	0.40	85
176	0.34	0.19	0.24	95
177	0.84	0.58	0.68	73
178	0.61	0.39	0.47	72
179	0.84	0.71	0.77	89
180	0.92	0.72	0.81	100
181	0.30	0.15	0.20	79
182	0.59	0.53	0.55	78
183	0.68	0.64	0.66	84
184	0.29	0.21	0.24	80
185	0.11	0.05	0.07	80
186	0.45	0.35	0.40	91
187	0.29	0.23	0.26	77
188	0.61	0.61	0.61	72
189	0.12	0.08	0.10	72
190	0.20	0.10	0.13	90
191	0.90	0.74	0.82	86
192	0.68	0.45	0.54	84
193	0.22	0.17	0.19	71
194	0.70	0.62	0.66	73
195	0.50	0.29	0.37	69
196	0.32	0.29	0.31	78
197	0.33	0.20	0.25	79
198	0.43	0.26	0.32	82
199	0.30	0.20	0.24	85
200	0.56	0.48	0.52	84
201	0.24	0.14	0.18	72
202	0.17	0.11	0.13	75
203	0.07	0.04	0.05	72
204	0.58	0.55	0.56	88
205	0.92	0.84	0.88	68
206	0.77	0.55	0.64	78
207	0.05	0.02	0.03	83
208	0.57	0.51	0.54	71
209	0.90	0.66	0.76	79
210	0.93	0.71	0.81	76
211	0.39	0.23	0.29	74
212	0.40	0.22	0.29	77
213	0.43	0.35	0.39	60
214	0.61	0.39	0.48	64
215	0.12	0.05	0.08	73
216	0.60	0.51	0.55	73
217	0.28	0.13	0.17	79
218	0.56	0.60	0.58	55
219	0.03	0.02	0.02	61

220	0.57	0.49	0.53	80
221	0.30	0.21	0.25	68
222	0.62	0.55	0.58	60
223	0.50	0.33	0.40	64
224	0.37	0.37	0.37	60
225	0.67	0.48	0.56	63
226	0.41	0.33	0.37	72
227	0.23	0.16	0.19	68
228	0.59	0.42	0.49	64
229	0.31	0.17	0.22	66
230	0.33	0.12	0.17	68
231	0.62	0.53	0.57	68
232	0.21	0.12	0.15	69
233	0.90	0.85	0.88	66
234	0.20	0.09	0.13	55
235	0.30	0.22	0.25	65
236	0.40	0.35	0.37	63
237	0.33	0.26	0.29	66
238	0.12	0.08	0.10	63
239	0.27	0.20	0.23	70
240	0.56	0.48	0.52	58
241	0.59	0.52	0.56	67
242	0.59	0.48	0.53	73
243	0.34	0.30	0.32	63
244	0.61	0.52	0.56	63
245	0.94	0.82	0.88	61
246	0.21	0.16	0.18	58
247	0.72	0.74	0.73	65
248	0.03	0.02	0.03	43
249	0.32	0.24	0.27	54
250	0.60	0.47	0.53	55
251	0.63	0.47	0.54	55
252	0.25	0.22	0.24	45
253	0.58	0.31	0.40	61
254	0.77	0.58	0.66	71
255	0.45	0.26	0.33	57
256	0.57	0.48	0.52	58
257	0.66	0.49	0.56	68
258	0.64	0.65	0.65	55
259	0.35	0.19	0.25	62
260	0.62	0.50	0.55	62
261	0.16	0.12	0.14	57
262	0.85	0.73	0.79	56
263	0.47	0.51	0.49	51
264	0.81	0.86	0.83	50
265	0.72	0.57	0.64	60
266	0.16	0.12	0.13	52
267	0.23	0.09	0.13	58
268	0.51	0.37	0.43	62
269	0.69	0.54	0.60	54
270	0.21	0.15	0.17	55
271	0.26	0.20	0.22	50
272	0.29	0.18	0.22	57
273	0.00	0.00	0.00	66
274	0.50	0.41	0.45	56
275	0.11	0.03	0.05	67
276	0.73	0.64	0.68	50

277	0.92	0.83	0.87	53
278	0.47	0.32	0.38	66
279	0.18	0.11	0.14	63
280	0.58	0.23	0.33	64
281	0.06	0.04	0.05	51
282	0.88	0.83	0.86	54
283	0.30	0.22	0.25	60
284	0.34	0.26	0.29	54
285	0.47	0.40	0.43	58
286	0.50	0.30	0.37	60
287	0.45	0.30	0.36	56
288	0.19	0.13	0.16	53
289	0.19	0.10	0.13	62
290	0.38	0.30	0.33	61
291	0.07	0.05	0.05	43
292	0.56	0.40	0.47	57
293	0.71	0.65	0.68	54
294	0.23	0.19	0.21	57
295	0.12	0.04	0.06	48
296	0.66	0.49	0.56	47
297	0.92	0.72	0.81	67
298	0.44	0.35	0.39	43
299	0.54	0.52	0.53	52
300	0.31	0.23	0.27	48
301	0.34	0.24	0.28	54
302	0.55	0.23	0.32	52
303	0.16	0.15	0.15	41
304	0.23	0.11	0.15	46
305	0.41	0.29	0.34	48
306	0.10	0.09	0.09	46
307	0.22	0.16	0.18	44
308	0.28	0.21	0.24	42
309	0.21	0.14	0.17	43
310	0.16	0.09	0.12	43
311	0.26	0.10	0.14	62
312	0.53	0.40	0.45	50
313	0.44	0.23	0.30	60
314	0.17	0.09	0.11	47
315	0.70	0.56	0.62	59
316	0.27	0.12	0.17	48
317	0.24	0.11	0.15	56
318	0.62	0.53	0.57	53
319	0.22	0.19	0.20	43
320	0.38	0.29	0.33	52
321	0.62	0.62	0.62	45
322	0.51	0.42	0.46	43
323	0.12	0.04	0.06	47
324	0.34	0.30	0.32	47
325	0.00	0.00	0.00	40
326	0.77	0.74	0.75	27
327	0.41	0.42	0.41	43
328	0.22	0.12	0.15	52
329	0.42	0.30	0.35	44
330	0.23	0.24	0.23	38
331	0.82	0.60	0.69	55
332	0.71	0.56	0.63	45
333	0.11	0.08	0.09	39

```
334       0.18      0.09      0.12        44
335       0.88      0.59      0.71        49
336       0.87      0.77      0.82        52
337       0.00      0.00      0.00        44
338       0.29      0.13      0.18        47
339       0.56      0.44      0.49        43
340       0.82      0.59      0.68        46
341       0.30      0.24      0.26        34
342       0.33      0.20      0.25        44
343       0.49      0.44      0.46        48
344       0.48      0.38      0.42        37
345       0.56      0.50      0.53        40
346       0.30      0.13      0.18        55
347       0.21      0.10      0.13        41
348       0.56      0.38      0.45        48
349       0.53      0.46      0.49        50
350       0.39      0.24      0.30        54
351       0.42      0.33      0.37        45
352       0.50      0.34      0.40        56
353       0.18      0.17      0.18        40
354       0.33      0.26      0.29        42
355       0.48      0.26      0.33        47
356       0.36      0.37      0.36        38
357       0.67      0.58      0.62        45
358       0.16      0.16      0.16        32
359       0.17      0.04      0.07        47
360       0.42      0.29      0.34        34
361       0.74      0.79      0.77        29
362       0.72      0.64      0.68        36
363       0.22      0.17      0.19        35
364       0.31      0.28      0.29        40
365       0.08      0.05      0.06        40
366       0.19      0.12      0.15        41
367       0.91      0.58      0.71        50
368       0.68      0.53      0.60        47
369       0.50      0.44      0.47        36
370       0.41      0.55      0.47        33
371       0.30      0.16      0.21        43
372       0.80      0.57      0.67        42
373       0.47      0.46      0.47        37
374       0.29      0.14      0.19        50
375       0.47      0.37      0.41        46
376       0.44      0.42      0.43        38
377       0.41      0.53      0.47        32
378       0.64      0.47      0.55        38
379       0.23      0.19      0.21        36
380       0.33      0.10      0.16        39
381       0.91      0.78      0.84        40
382       0.44      0.30      0.35        47
383       0.27      0.16      0.20        43
384       0.38      0.20      0.26        44
385       0.50      0.43      0.46        30
386       0.36      0.31      0.33        32
387       0.67      0.41      0.51        39
388       0.42      0.37      0.39        30
389       0.66      0.62      0.64        37
390       0.52      0.38      0.44        39
```

391	0.24	0.08	0.12	50
392	0.18	0.09	0.12	46
393	0.27	0.18	0.22	38
394	0.88	0.77	0.82	39
395	0.40	0.41	0.41	29
396	0.34	0.37	0.35	38
397	0.06	0.02	0.03	48
398	0.17	0.13	0.15	31
399	0.10	0.05	0.06	44
400	0.37	0.31	0.34	35
401	0.48	0.28	0.35	50
402	0.23	0.19	0.21	37
403	0.79	0.70	0.75	44
404	1.00	0.69	0.82	39
405	0.41	0.38	0.39	34
406	0.34	0.30	0.32	33
407	0.24	0.13	0.17	38
408	0.14	0.05	0.08	39
409	0.30	0.12	0.17	50
410	0.09	0.03	0.04	36
411	0.53	0.71	0.61	28
412	0.24	0.20	0.22	40
413	0.18	0.10	0.13	40
414	0.25	0.12	0.16	43
415	0.16	0.06	0.09	47
416	0.64	0.55	0.59	42
417	0.27	0.16	0.20	45
418	0.50	0.44	0.47	48
419	0.68	0.49	0.57	47
420	0.64	0.33	0.43	43
421	0.77	0.69	0.73	35
422	0.75	0.38	0.51	47
423	0.57	0.37	0.45	43
424	0.52	0.28	0.36	43
425	0.33	0.23	0.27	40
426	0.77	0.56	0.65	41
427	0.42	0.31	0.36	42
428	0.15	0.06	0.09	31
429	0.15	0.19	0.17	27
430	0.29	0.17	0.22	41
431	0.55	0.57	0.56	28
432	0.11	0.05	0.06	43
433	0.38	0.21	0.27	38
434	0.33	0.14	0.20	43
435	0.79	0.76	0.78	34
436	0.26	0.20	0.23	30
437	0.00	0.00	0.00	40
438	0.54	0.35	0.42	40
439	0.35	0.40	0.38	30
440	0.36	0.25	0.30	36
441	0.14	0.16	0.15	32
442	0.05	0.03	0.04	38
443	0.27	0.15	0.19	40
444	0.36	0.18	0.24	28
445	0.50	0.30	0.37	40
446	0.25	0.10	0.14	40
447	0.68	0.59	0.63	32

448	0.87	0.67	0.75	39
449	0.82	0.76	0.79	37
450	0.21	0.14	0.17	43
451	0.89	0.65	0.75	37
452	0.12	0.04	0.06	46
453	0.48	0.36	0.41	42
454	0.31	0.30	0.30	27
455	0.73	0.59	0.66	32
456	0.46	0.28	0.34	40
457	0.31	0.14	0.19	37
458	0.74	0.56	0.64	41
459	0.50	0.31	0.38	45
460	0.35	0.17	0.23	40
461	0.35	0.29	0.31	28
462	0.12	0.07	0.09	27
463	0.20	0.08	0.11	39
464	0.67	0.34	0.45	35
465	0.00	0.00	0.00	35
466	0.26	0.24	0.25	33
467	0.19	0.10	0.13	41
468	0.50	0.40	0.44	35
469	0.31	0.16	0.21	32
470	0.18	0.11	0.14	35
471	0.57	0.33	0.41	40
472	0.52	0.46	0.49	26
473	0.88	0.72	0.79	29
474	0.29	0.11	0.16	35
475	0.48	0.33	0.39	43
476	0.47	0.24	0.32	37
477	0.91	0.66	0.76	32
478	0.40	0.28	0.33	29
479	0.54	0.38	0.45	34
480	0.29	0.19	0.23	32
481	0.72	0.51	0.60	35
482	0.71	0.64	0.68	39
483	0.32	0.27	0.29	30
484	0.27	0.22	0.24	32
485	0.17	0.11	0.14	35
486	0.39	0.33	0.36	36
487	0.18	0.11	0.13	28
488	0.78	0.61	0.68	41
489	0.10	0.08	0.09	25
490	0.06	0.03	0.04	30
491	0.65	0.54	0.59	24
492	0.51	0.57	0.54	35
493	0.19	0.10	0.13	30
494	0.15	0.05	0.07	42
495	0.62	0.30	0.40	27
496	0.68	0.53	0.60	32
497	0.67	0.73	0.70	30
498	0.44	0.46	0.45	26
499	0.22	0.19	0.20	27
micro avg	0.58	0.43	0.50	72315
macro avg	0.46	0.36	0.40	72315
weighted avg	0.56	0.43	0.49	72315
samples avg	0.48	0.42	0.41	72315

```
Time taken to run this cell : 0:13:16.265558
```

Observation:-

1. When trained Logistic regression in OneVsRestClassifier with l1 regularization, the performance of micro averaged F1-score improved from 0.46 to 0.49, which is better than the F1-score of the model with SGD Classifier and log loss.
2. Precision also improved from the previous model and Recall remained same.
3. Performace improved further after applying Logistic Regession with L1 penalty.

Applying Linear SVM with SGDClassifier (loss='hinge') with OneVsRestClassifier

Hyper parameter Tuning

In [48]:
```
# Hyper parameter tuning
from sklearn.model_selection import GridSearchCV
from sklearn.multiclass import OneVsRestClassifier

parameters = {'estimator__alpha':[0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 10
0]}
classifier_3 = OneVsRestClassifier(SGDClassifier(loss='hinge', penalty='l1'))

model_3 = GridSearchCV(estimator = classifier_3, param_grid=parameters, cv=3,
verbose=0, scoring='f1_micro',n_jobs = -1)
model_3.fit(x_train_multilabel, y_train)

print(model_3.best_estimator_)
optimal_alpha_3 = model_3.best_estimator_.get_params()['estimator__alpha']
print('best alpha value after hyperparameter tuning is : ',optimal_alpha_3)
```

```
OneVsRestClassifier(estimator=SGDClassifier(alpha=0.0001, average=False,
                                            class_weight=None,
                                            early_stopping=False, epsilon=0.
1,
                                            eta0=0.0, fit_intercept=True,
                                            l1_ratio=0.15,
                                            learning_rate='optimal',
                                            loss='hinge', max_iter=1000,
                                            n_iter_no_change=5, n_jobs=None,
                                            penalty='l1', power_t=0.5,
                                            random_state=None, shuffle=True,
                                            tol=0.001, validation_fraction=0.
1,
                                            verbose=0, warm_start=False),
                    n_jobs=None)
best alpha value after hyperparameter tuning is :  0.0001
```

Linear SVM model with optimal alpha

In [49]:
```
start = datetime.now()
clf_3 = OneVsRestClassifier(SGDClassifier(loss='hinge', alpha=optimal_alpha_3,
penalty='l1'), n_jobs=-1)
clf_3.fit(x_train_multilabel, y_train)
predictions_3 = clf_3.predict(x_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test, predictions_3))
print("Hamming loss ",metrics.hamming_loss(y_test,predictions_3))


precision = precision_score(y_test, predictions_3, average='micro')
recall = recall_score(y_test, predictions_3, average='micro')
f1 = f1_score(y_test, predictions_3, 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_3, average='macro')
recall = recall_score(y_test, predictions_3, average='macro')
f1 = f1_score(y_test, predictions_3, 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_3))
print("Time taken to run this cell :", datetime.now() - start)
```

```
Accuracy : 0.157775
Hamming loss  0.00378305
Micro-average quality numbers
Precision: 0.4749, Recall: 0.4378, F1-measure: 0.4556
Macro-average quality numbers
Precision: 0.3565, Recall: 0.3578, F1-measure: 0.3496
              precision    recall  f1-score   support

           0       0.54      0.38      0.44      3177
           1       0.68      0.51      0.59      2878
           2       0.71      0.62      0.66      2720
           3       0.60      0.47      0.53      2514
           4       0.81      0.82      0.81      2214
           5       0.75      0.69      0.72      2163
           6       0.64      0.58      0.61      1318
           7       0.72      0.67      0.70      1278
           8       0.58      0.48      0.52      1289
           9       0.58      0.49      0.53      1185
          10       0.71      0.68      0.70      1186
          11       0.39      0.28      0.33      1128
          12       0.33      0.19      0.24      1096
          13       0.48      0.35      0.40       974
          14       0.44      0.29      0.35       947
          15       0.45      0.36      0.40       917
          16       0.63      0.59      0.61       910
          17       0.65      0.63      0.64       781
          18       0.51      0.33      0.40       731
          19       0.48      0.36      0.41       676
          20       0.22      0.13      0.17       612
          21       0.53      0.50      0.51       522
          22       0.41      0.41      0.41       451
          23       0.69      0.72      0.71       399
          24       0.49      0.48      0.49       428
          25       0.48      0.45      0.47       428
          26       0.67      0.75      0.71       409
          27       0.46      0.36      0.40       367
          28       0.16      0.13      0.14       370
          29       0.39      0.29      0.33       379
          30       0.41      0.39      0.40       381
          31       0.74      0.82      0.78       350
          32       0.31      0.34      0.33       328
          33       0.45      0.39      0.42       331
          34       0.60      0.51      0.55       284
          35       0.57      0.62      0.59       290
          36       0.69      0.65      0.67       314
          37       0.31      0.23      0.26       301
          38       0.59      0.57      0.58       308
          39       0.21      0.16      0.18       269
          40       0.38      0.47      0.42       263
          41       0.46      0.44      0.45       279
          42       0.15      0.23      0.18       219
          43       0.33      0.34      0.33       253
          44       0.44      0.36      0.40       231
          45       0.18      0.15      0.16       255
          46       0.41      0.42      0.42       217
          47       0.24      0.28      0.26       199
          48       0.41      0.32      0.36       228
```

49	0.63	0.52	0.57	219
50	0.75	0.72	0.74	225
51	0.45	0.58	0.51	193
52	0.13	0.13	0.13	223
53	0.25	0.27	0.26	220
54	0.26	0.27	0.26	186
55	0.30	0.17	0.21	206
56	0.60	0.66	0.63	203
57	0.75	0.88	0.81	202
58	0.54	0.61	0.57	187
59	0.27	0.12	0.17	191
60	0.14	0.11	0.12	199
61	0.44	0.47	0.45	201
62	0.78	0.76	0.77	206
63	0.50	0.34	0.41	196
64	0.22	0.18	0.20	213
65	0.42	0.37	0.40	190
66	0.59	0.64	0.62	181
67	0.09	0.04	0.05	197
68	0.32	0.29	0.30	178
69	0.59	0.57	0.58	176
70	0.48	0.50	0.49	169
71	0.32	0.39	0.35	168
72	0.48	0.27	0.34	199
73	0.58	0.64	0.61	152
74	0.46	0.59	0.51	154
75	0.44	0.50	0.46	185
76	0.44	0.47	0.46	156
77	0.71	0.75	0.73	165
78	0.08	0.08	0.08	150
79	0.35	0.35	0.35	151
80	0.26	0.37	0.31	127
81	0.31	0.25	0.27	145
82	0.67	0.72	0.69	148
83	0.23	0.23	0.23	154
84	0.83	0.67	0.74	171
85	0.34	0.39	0.36	159
86	0.23	0.17	0.20	143
87	0.62	0.55	0.58	144
88	0.88	0.73	0.80	142
89	0.62	0.76	0.68	135
90	0.69	0.65	0.67	130
91	0.53	0.51	0.52	144
92	0.49	0.59	0.54	145
93	0.29	0.27	0.28	127
94	0.86	0.74	0.80	135
95	0.09	0.12	0.10	138
96	0.74	0.60	0.66	131
97	0.27	0.34	0.30	130
98	0.80	0.84	0.82	118
99	0.66	0.86	0.74	132
100	0.32	0.24	0.27	117
101	0.76	0.69	0.72	121
102	0.19	0.26	0.22	108
103	0.11	0.05	0.06	111
104	0.20	0.18	0.19	132
105	0.41	0.50	0.45	111

106	0.58	0.48	0.52	122
107	0.46	0.23	0.31	135
108	0.49	0.48	0.48	144
109	0.28	0.17	0.21	126
110	0.51	0.55	0.53	121
111	0.17	0.20	0.19	118
112	0.33	0.42	0.37	101
113	0.33	0.31	0.32	100
114	0.48	0.47	0.47	118
115	0.23	0.25	0.24	109
116	0.69	0.68	0.68	129
117	0.77	0.82	0.79	106
118	0.14	0.12	0.13	104
119	0.39	0.31	0.35	125
120	0.52	0.67	0.59	109
121	0.24	0.25	0.25	128
122	0.38	0.37	0.37	110
123	0.25	0.19	0.22	105
124	0.22	0.27	0.25	103
125	0.34	0.29	0.31	107
126	0.19	0.26	0.22	100
127	0.25	0.33	0.29	110
128	0.79	0.84	0.82	109
129	0.08	0.04	0.05	107
130	0.24	0.41	0.31	90
131	0.14	0.17	0.15	95
132	0.16	0.16	0.16	101
133	0.79	0.72	0.75	103
134	0.29	0.37	0.33	86
135	0.67	0.02	0.03	127
136	0.87	0.84	0.85	111
137	0.13	0.14	0.14	98
138	0.42	0.43	0.43	106
139	0.19	0.05	0.08	98
140	0.19	0.20	0.19	92
141	0.63	0.65	0.64	109
142	0.14	0.17	0.16	105
143	0.55	0.58	0.57	116
144	0.45	0.48	0.46	101
145	0.48	0.47	0.47	103
146	0.20	0.11	0.14	109
147	0.74	0.82	0.78	88
148	0.32	0.44	0.37	82
149	0.64	0.75	0.69	77
150	0.42	0.43	0.43	103
151	0.11	0.14	0.12	92
152	0.17	0.24	0.20	88
153	0.38	0.44	0.41	100
154	0.29	0.33	0.31	90
155	0.23	0.25	0.24	88
156	0.34	0.31	0.32	103
157	0.16	0.18	0.17	95
158	0.21	0.22	0.22	106
159	0.35	0.42	0.38	93
160	0.74	0.78	0.76	103
161	0.48	0.24	0.32	102
162	0.24	0.25	0.24	85

163	0.55	0.26	0.35	81
164	0.50	0.48	0.49	94
165	0.17	0.14	0.15	92
166	0.47	0.59	0.53	88
167	0.41	0.24	0.31	82
168	0.31	0.46	0.37	81
169	0.27	0.25	0.26	81
170	0.23	0.14	0.17	80
171	0.78	0.74	0.76	76
172	0.26	0.28	0.27	89
173	0.53	0.57	0.55	91
174	0.15	0.28	0.19	78
175	0.27	0.34	0.30	85
176	0.25	0.17	0.20	95
177	0.79	0.71	0.75	73
178	0.31	0.40	0.35	72
179	0.69	0.76	0.73	89
180	0.83	0.68	0.75	100
181	0.22	0.15	0.18	79
182	0.44	0.60	0.51	78
183	0.55	0.65	0.60	84
184	0.23	0.35	0.28	80
185	0.24	0.05	0.08	80
186	0.39	0.44	0.41	91
187	0.22	0.22	0.22	77
188	0.42	0.62	0.51	72
189	0.09	0.06	0.07	72
190	0.19	0.19	0.19	90
191	0.73	0.83	0.78	86
192	0.41	0.52	0.46	84
193	0.12	0.11	0.12	71
194	0.69	0.63	0.66	73
195	0.15	0.19	0.17	69
196	0.29	0.24	0.26	78
197	0.26	0.23	0.24	79
198	0.33	0.32	0.32	82
199	0.20	0.15	0.17	85
200	0.58	0.57	0.57	84
201	0.07	0.14	0.09	72
202	0.06	0.09	0.07	75
203	0.08	0.07	0.07	72
204	0.42	0.61	0.50	88
205	0.67	0.84	0.75	68
206	0.77	0.59	0.67	78
207	0.04	0.04	0.04	83
208	0.54	0.63	0.58	71
209	0.73	0.75	0.74	79
210	0.75	0.78	0.76	76
211	0.17	0.26	0.20	74
212	0.26	0.13	0.17	77
213	0.26	0.38	0.31	60
214	0.50	0.44	0.47	64
215	0.20	0.18	0.19	73
216	0.54	0.55	0.54	73
217	0.10	0.08	0.09	79
218	0.30	0.45	0.36	55
219	0.03	0.03	0.03	61

220	0.38	0.36	0.37	80
221	0.17	0.21	0.19	68
222	0.30	0.57	0.39	60
223	0.28	0.39	0.33	64
224	0.23	0.43	0.30	60
225	0.59	0.51	0.55	63
226	0.38	0.35	0.36	72
227	0.10	0.18	0.13	68
228	0.42	0.50	0.46	64
229	0.13	0.08	0.10	66
230	0.04	0.06	0.05	68
231	0.63	0.68	0.65	68
232	0.18	0.19	0.18	69
233	0.81	0.83	0.82	66
234	0.18	0.09	0.12	55
235	0.19	0.22	0.20	65
236	0.31	0.35	0.33	63
237	0.17	0.27	0.21	66
238	0.04	0.05	0.04	63
239	0.28	0.36	0.31	70
240	0.43	0.47	0.45	58
241	0.37	0.37	0.37	67
242	0.50	0.45	0.47	73
243	0.33	0.33	0.33	63
244	0.33	0.41	0.37	63
245	0.90	0.87	0.88	61
246	0.22	0.31	0.26	58
247	0.59	0.65	0.62	65
248	0.03	0.05	0.04	43
249	0.18	0.20	0.19	54
250	0.58	0.62	0.60	55
251	0.28	0.36	0.32	55
252	0.16	0.18	0.17	45
253	0.22	0.28	0.24	61
254	0.46	0.61	0.52	71
255	0.37	0.39	0.38	57
256	0.33	0.38	0.35	58
257	0.38	0.38	0.38	68
258	0.53	0.76	0.62	55
259	0.15	0.16	0.16	62
260	0.51	0.53	0.52	62
261	0.16	0.09	0.11	57
262	0.61	0.71	0.66	56
263	0.26	0.33	0.29	51
264	0.62	0.82	0.71	50
265	0.78	0.67	0.72	60
266	0.08	0.04	0.05	52
267	0.17	0.02	0.03	58
268	0.34	0.39	0.36	62
269	0.71	0.69	0.70	54
270	0.19	0.25	0.22	55
271	0.13	0.20	0.16	50
272	0.16	0.19	0.18	57
273	0.00	0.00	0.00	66
274	0.37	0.38	0.37	56
275	0.00	0.00	0.00	67
276	0.40	0.66	0.50	50

277	0.73	0.72	0.72	53
278	0.42	0.24	0.31	66
279	0.24	0.11	0.15	63
280	0.42	0.23	0.30	64
281	0.03	0.02	0.02	51
282	0.85	0.81	0.83	54
283	0.29	0.23	0.26	60
284	0.18	0.22	0.20	54
285	0.30	0.41	0.35	58
286	0.36	0.33	0.35	60
287	0.27	0.30	0.29	56
288	0.23	0.17	0.19	53
289	0.03	0.03	0.03	62
290	0.39	0.39	0.39	61
291	0.00	0.00	0.00	43
292	0.34	0.33	0.34	57
293	0.59	0.72	0.65	54
294	0.12	0.16	0.14	57
295	0.07	0.02	0.03	48
296	0.25	0.40	0.31	47
297	0.57	0.67	0.62	67
298	0.25	0.37	0.30	43
299	0.29	0.38	0.33	52
300	0.16	0.25	0.20	48
301	0.14	0.15	0.14	54
302	0.18	0.23	0.20	52
303	0.17	0.12	0.14	41
304	0.19	0.11	0.14	46
305	0.26	0.33	0.29	48
306	0.04	0.04	0.04	46
307	0.29	0.20	0.24	44
308	0.22	0.31	0.26	42
309	0.15	0.16	0.16	43
310	0.14	0.05	0.07	43
311	0.10	0.08	0.09	62
312	0.45	0.44	0.44	50
313	0.34	0.40	0.37	60
314	0.05	0.04	0.05	47
315	0.79	0.64	0.71	59
316	0.15	0.08	0.11	48
317	0.21	0.14	0.17	56
318	0.46	0.43	0.45	53
319	0.21	0.19	0.20	43
320	0.24	0.31	0.27	52
321	0.28	0.36	0.31	45
322	0.24	0.44	0.31	43
323	0.13	0.04	0.06	47
324	0.24	0.32	0.28	47
325	0.00	0.00	0.00	40
326	0.79	0.85	0.82	27
327	0.34	0.51	0.41	43
328	0.17	0.13	0.15	52
329	0.17	0.20	0.19	44
330	0.20	0.29	0.24	38
331	0.65	0.64	0.64	55
332	0.50	0.62	0.55	45
333	0.07	0.15	0.10	39

334	0.15	0.14	0.14	44
335	0.60	0.73	0.66	49
336	0.85	0.77	0.81	52
337	0.12	0.11	0.12	44
338	0.15	0.19	0.17	47
339	0.29	0.33	0.30	43
340	0.71	0.65	0.68	46
341	0.24	0.35	0.29	34
342	0.20	0.16	0.18	44
343	0.38	0.35	0.37	48
344	0.26	0.30	0.28	37
345	0.31	0.45	0.37	40
346	0.24	0.22	0.23	55
347	0.07	0.05	0.06	41
348	0.51	0.48	0.49	48
349	0.32	0.36	0.34	50
350	0.28	0.37	0.32	54
351	0.27	0.16	0.20	45
352	0.21	0.14	0.17	56
353	0.20	0.40	0.26	40
354	0.09	0.10	0.09	42
355	0.21	0.17	0.19	47
356	0.17	0.34	0.23	38
357	0.46	0.51	0.48	45
358	0.10	0.16	0.12	32
359	0.00	0.00	0.00	47
360	0.82	0.26	0.40	34
361	0.64	0.72	0.68	29
362	0.57	0.67	0.62	36
363	0.12	0.14	0.13	35
364	0.32	0.15	0.20	40
365	0.10	0.12	0.11	40
366	0.11	0.12	0.11	41
367	0.50	0.54	0.52	50
368	0.35	0.36	0.35	47
369	0.41	0.50	0.45	36
370	0.24	0.45	0.31	33
371	0.25	0.19	0.21	43
372	0.86	0.60	0.70	42
373	0.33	0.30	0.31	37
374	0.61	0.28	0.38	50
375	0.42	0.48	0.45	46
376	0.37	0.61	0.46	38
377	0.22	0.34	0.27	32
378	0.37	0.39	0.38	38
379	0.08	0.11	0.09	36
380	0.14	0.15	0.15	39
381	0.86	0.78	0.82	40
382	0.17	0.30	0.22	47
383	0.07	0.09	0.08	43
384	0.21	0.14	0.17	44
385	0.22	0.33	0.27	30
386	0.20	0.38	0.26	32
387	0.47	0.38	0.42	39
388	0.23	0.53	0.32	30
389	0.54	0.54	0.54	37
390	0.47	0.36	0.41	39

391	0.11	0.06	0.08	50
392	0.10	0.04	0.06	46
393	0.09	0.13	0.11	38
394	0.82	0.95	0.88	39
395	0.34	0.48	0.40	29
396	0.31	0.45	0.37	38
397	0.00	0.00	0.00	48
398	0.11	0.06	0.08	31
399	0.03	0.05	0.04	44
400	0.16	0.14	0.15	35
401	0.20	0.18	0.19	50
402	0.11	0.16	0.13	37
403	0.67	0.64	0.65	44
404	0.76	0.64	0.69	39
405	0.28	0.38	0.33	34
406	0.24	0.36	0.29	33
407	0.19	0.08	0.11	38
408	0.25	0.05	0.09	39
409	0.19	0.16	0.17	50
410	0.00	0.00	0.00	36
411	0.57	0.71	0.63	28
412	0.28	0.33	0.30	40
413	0.00	0.00	0.00	40
414	0.08	0.07	0.07	43
415	0.11	0.11	0.11	47
416	0.50	0.45	0.48	42
417	0.21	0.20	0.20	45
418	0.36	0.44	0.39	48
419	0.62	0.45	0.52	47
420	0.43	0.28	0.34	43
421	0.56	0.71	0.63	35
422	0.61	0.36	0.45	47
423	0.49	0.44	0.46	43
424	0.38	0.40	0.39	43
425	0.16	0.12	0.14	40
426	0.49	0.49	0.49	41
427	0.23	0.21	0.22	42
428	0.00	0.00	0.00	31
429	0.13	0.19	0.15	27
430	0.19	0.12	0.15	41
431	0.79	0.68	0.73	28
432	0.00	0.00	0.00	43
433	0.24	0.37	0.29	38
434	0.11	0.09	0.10	43
435	0.69	0.79	0.74	34
436	0.10	0.17	0.12	30
437	0.00	0.00	0.00	40
438	0.59	0.42	0.49	40
439	0.33	0.50	0.40	30
440	0.10	0.08	0.09	36
441	0.12	0.19	0.15	32
442	0.04	0.03	0.03	38
443	0.19	0.17	0.18	40
444	0.12	0.18	0.14	28
445	0.44	0.42	0.43	40
446	0.48	0.30	0.37	40
447	0.53	0.50	0.52	32

448	0.66	0.64	0.65	39
449	0.76	0.76	0.76	37
450	0.08	0.05	0.06	43
451	0.51	0.54	0.53	37
452	0.18	0.22	0.20	46
453	0.40	0.40	0.40	42
454	0.42	0.30	0.35	27
455	0.53	0.66	0.58	32
456	0.26	0.25	0.25	40
457	0.15	0.05	0.08	37
458	0.71	0.59	0.64	41
459	0.29	0.24	0.27	45
460	0.22	0.20	0.21	40
461	0.23	0.36	0.28	28
462	0.02	0.04	0.03	27
463	0.18	0.13	0.15	39
464	0.31	0.34	0.32	35
465	0.00	0.00	0.00	35
466	0.17	0.24	0.20	33
467	0.11	0.15	0.12	41
468	0.38	0.37	0.38	35
469	0.23	0.19	0.21	32
470	0.09	0.03	0.04	35
471	0.47	0.35	0.40	40
472	0.32	0.23	0.27	26
473	0.67	0.55	0.60	29
474	0.00	0.00	0.00	35
475	0.33	0.30	0.31	43
476	0.32	0.30	0.31	37
477	0.80	0.62	0.70	32
478	0.22	0.34	0.27	29
479	0.24	0.32	0.28	34
480	0.15	0.28	0.19	32
481	0.59	0.54	0.57	35
482	0.86	0.49	0.62	39
483	0.36	0.33	0.34	30
484	0.12	0.09	0.11	32
485	0.24	0.23	0.24	35
486	0.21	0.36	0.27	36
487	0.00	0.00	0.00	28
488	0.31	0.54	0.39	41
489	0.00	0.00	0.00	25
490	0.25	0.03	0.06	30
491	0.67	0.58	0.62	24
492	0.50	0.60	0.55	35
493	0.08	0.03	0.05	30
494	0.29	0.05	0.08	42
495	0.32	0.33	0.33	27
496	0.49	0.53	0.51	32
497	0.46	0.77	0.57	30
498	0.30	0.46	0.36	26
499	0.05	0.07	0.06	27
micro avg	0.47	0.44	0.46	72315
macro avg	0.36	0.36	0.35	72315
weighted avg	0.48	0.44	0.45	72315
samples avg	0.44	0.43	0.40	72315

```
Time taken to run this cell : 0:05:05.479173
```

Observation:-

1. The model with SGDClassifier(loss='hinge') gave Micro F1-score of 0.455 which is less than the logistic regression model with SGDClassifier(loss=log) .

Models Summarization

In [60]:
```
from pandas import DataFrame
StackOverflow = {'Vectorizer':['BoW', 'BoW','BoW'],
                 'Model':['LR with SGDClassifier with log loss','Logistic Regr
ession','Linear SVM'],
                 'Hyper parameter alpha':['0.0001','','0.0001'],
                 'Micro F1-score':['0.4621','0.4965','0.4556'],
                 'Precision':['0.4876','0.5804','0.4749'],
                 'Recall':['0.4392','0.4339','0.4378']}
```

In [61]:
```
Final_conclusions = DataFrame(StackOverflow)
Final_conclusions
```

Out[61]:

	Hyper parameter alpha	Micro F1-score	Model	Precision	Recall	Vectorizer
0	0.0001	0.4621	LR with SGDClassifier with log loss	0.4876	0.4392	BoW
1		0.4965	Logistic Regression	0.5804	0.4339	BoW
2	0.0001	0.4556	Linear SVM	0.4749	0.4378	BoW

Conclusions:-

In this assignment, for the given Questions and its descriptions, I have to predict tags for a particular question correctly with high precision and recall.

1. When we have high dimensional data, linear models perform very well. So Logistic Regression works well in this case.
2. Simple Logistic Regression gave high F1-score of 0.496 than the other two models.
3. Random Forests , GBDT, RBF kernel SVM does not work well with high dimensional data , so I tried Linear SVM with SGD Classifier with hinge loss.
4. Linear SVM gave similar F1-score approx same as to the F1-score of model with Logistic Regression using SGD Classifier with log loss.
5. Simple Logistic Regression has high Precision of 0.58 and the other two models have similar precision.
6. Logistic Regression using SGD Classifier with log loss has high Recall of 0.4392 than the other two models.
7. So, overall Simple Logistic Regression is working well out of 3 models.

In [ ]:

```

```