SO_Tag_Predictor

August 3, 2019

[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
import nltk
from sklearn.model_selection import GridSearchCV
```

1 Stack Overflow: Tag Prediction

1. Business Problem

1.1 Description

Description

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

Problem Statemtent

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

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

1.2 Source / useful links

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

1.3 Real World / Business Objectives and Constraints

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

2. Machine Learning problem

2.1 Data

2.1.1 Data Overview

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

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

Data Field Explaination

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

2.1.2 Example Data point

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

2.2.1 Type of Machine Learning Problem

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

2.2.2 Performance metric

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

The relative contribution of precision and recall to the F1 score are equal. The formula for the F1 score is:

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

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

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

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

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

3. Exploratory Data Analysis

3.1 Data Loading and Cleaning

3.1.1 Using Pandas with SQLite to Load the data

[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

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

```
Number of rows in the database :
 6034196
Time taken to count the number of rows : 0:01:50.894625
```

3.1.3 Checking for duplicates

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

```
Time taken to run this cell : 0:03:36.667965
```

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

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

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

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

[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 % )
```

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

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

[8]:
```
df_no_dup = df_no_dup[df_no_dup["Tags"].isnull() != True]
```

[9]:
```
start = datetime.now()
df_no_dup["tag_count"] = df_no_dup["Tags"].apply(lambda text: len(text.split("␣
↪")) )
# adding a new feature number of tags per question
print("Time taken to run this cell :", datetime.now() - start)
df_no_dup.head()
```

```
Time taken to run this cell : 0:00:02.843259
```

[9]:
```
                                               Title  \
0      Implementing Boundary Value Analysis of S...
1          Dynamic Datagrid Binding in Silverlight?
2          Dynamic Datagrid Binding in Silverlight?
3     java.lang.NoClassDefFoundError: javax/serv...
4     java.sql.SQLException:[Microsoft][ODBC Dri...

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

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

[10]:
```
# distribution of number of tags per question
df_no_dup.tag_count.value_counts()
```

[10]:
```
3    1206157
2    1111706
4     814996
1     568291
```

```
5    505158
Name: tag_count, dtype: int64
```

[11]:
```
#Creating a new database with no duplicates
if not os.path.isfile('train_no_dup.db'):
    disk_dup = create_engine("sqlite:///train_no_dup.db")
    no_dup = pd.DataFrame(df_no_dup, columns=['Title', 'Body', 'Tags'])
    no_dup.to_sql('no_dup_train',disk_dup)
```

[12]:
```
#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 : 0:01:12.543393
```

3.2 Analysis of Tags

3.2.1 Total number of unique tags

[13]:
```
# Importing & Initializing the "CountVectorizer" object, which
#is scikit-learn's bag of words tool.

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

[14]:
```
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
```

[15]:
```
#'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

[16]:
```
# https://stackoverflow.com/questions/15115765/
↪how-to-access-sparse-matrix-elements
#Lets now store the document term matrix in a dictionary.
freqs = tag_dtm.sum(axis=0).A1
result = dict(zip(tags, freqs))
```

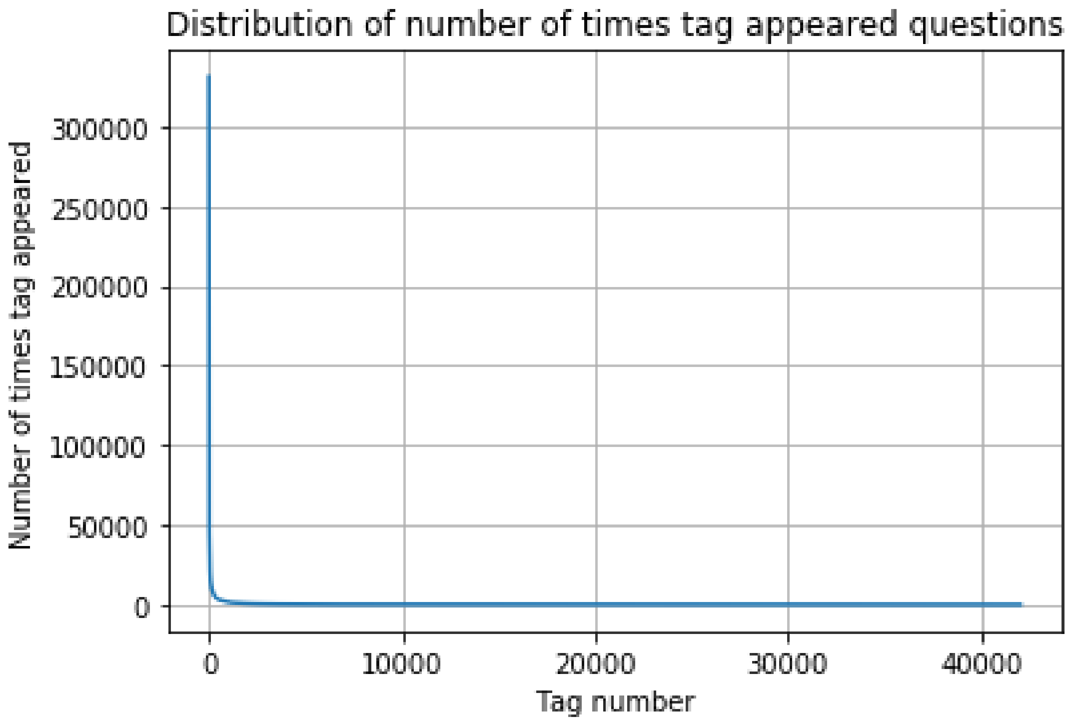
[17]:
```
#Saving this dictionary to csv files.
if not os.path.isfile('tag_counts_dict_dtm.csv'):
    with open('tag_counts_dict_dtm.csv', 'w') as csv_file:
        writer = csv.writer(csv_file)
        for key, value in result.items():
            writer.writerow([key, value])
tag_df = pd.read_csv("tag_counts_dict_dtm.csv", names=['Tags', 'Counts'])
tag_df.head()
```

[17]:

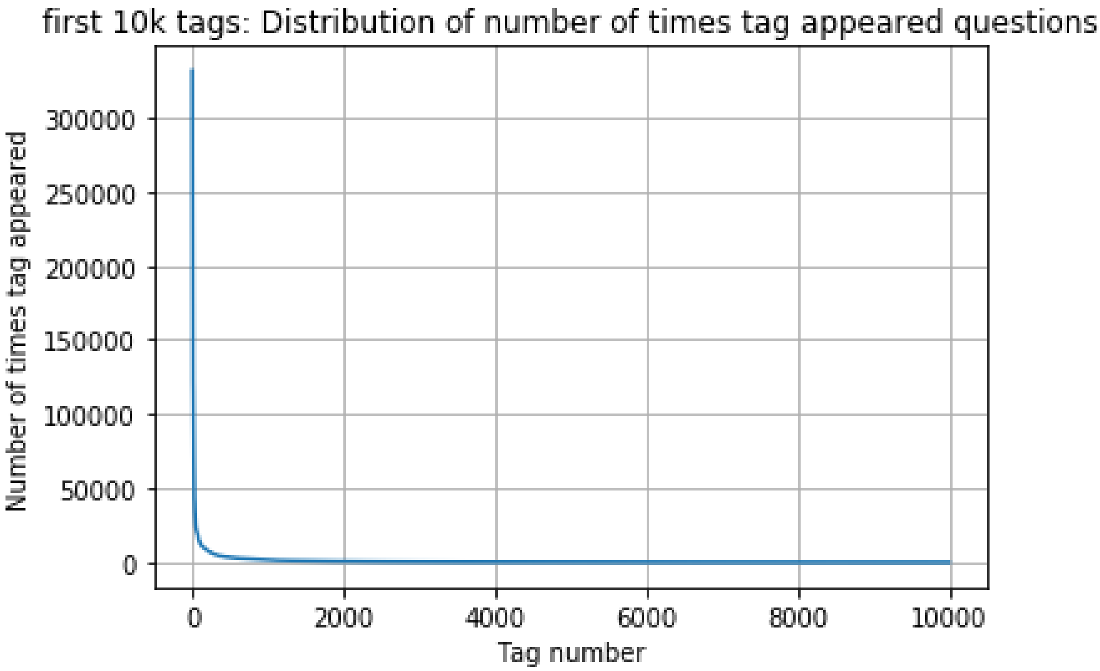
	Tags	Counts
0	.a	18
1	.app	37
2	.asp.net-mvc	1
3	.aspxauth	21
4	.bash-profile	138

[18]:
```
tag_df_sorted = tag_df.sort_values(['Counts'], ascending=False)
tag_counts = tag_df_sorted['Counts'].values
```

[19]:
```
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()
```



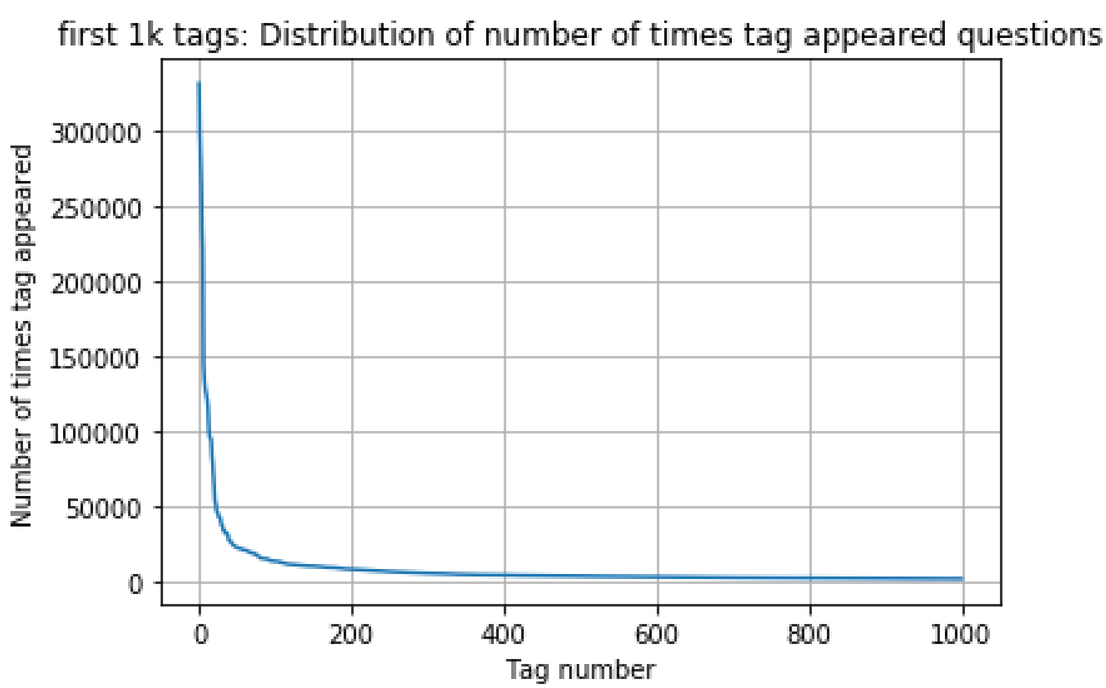
[20]:
```
plt.plot(tag_counts[0:10000])
plt.title('first 10k tags: Distribution of number of times tag appeared␣
 ↪questions')
plt.grid()
plt.xlabel("Tag number")
plt.ylabel("Number of times tag appeared")
plt.show()
print(len(tag_counts[0:10000:25]), tag_counts[0:10000:25])
```




```
400 [331505  44829  22429  17728  13364  11162  10029   9148   8054   7151
   6466   5865   5370   4983   4526   4281   4144   3929   3750   3593
   3453   3299   3123   2986   2891   2738   2647   2527   2431   2331
   2259   2186   2097   2020   1959   1900   1828   1770   1723   1673
   1631   1574   1532   1479   1448   1406   1365   1328   1300   1266
   1245   1222   1197   1181   1158   1139   1121   1101   1076   1056
   1038   1023   1006    983    966    952    938    926    911    891
    882    869    856    841    830    816    804    789    779    770
    752    743    733    725    712    702    688    678    671    658
    650    643    634    627    616    607    598    589    583    577
    568    559    552    545    540    533    526    518    512    506
    500    495    490    485    480    477    469    465    457    450
    447    442    437    432    426    422    418    413    408    403
    398    393    388    385    381    378    374    370    367    365
    361    357    354    350    347    344    342    339    336    332
    330    326    323    319    315    312    309    307    304    301
    299    296    293    291    289    286    284    281    278    276
    275    272    270    268    265    262    260    258    256    254
    252    250    249    247    245    243    241    239    238    236
    234    233    232    230    228    226    224    222    220    219
    217    215    214    212    210    209    207    205    204    203
    201    200    199    198    196    194    193    192    191    189
    188    186    185    183    182    181    180    179    178    177
    175    174    172    171    170    169    168    167    166    165
    164    162    161    160    159    158    157    156    156    155
```

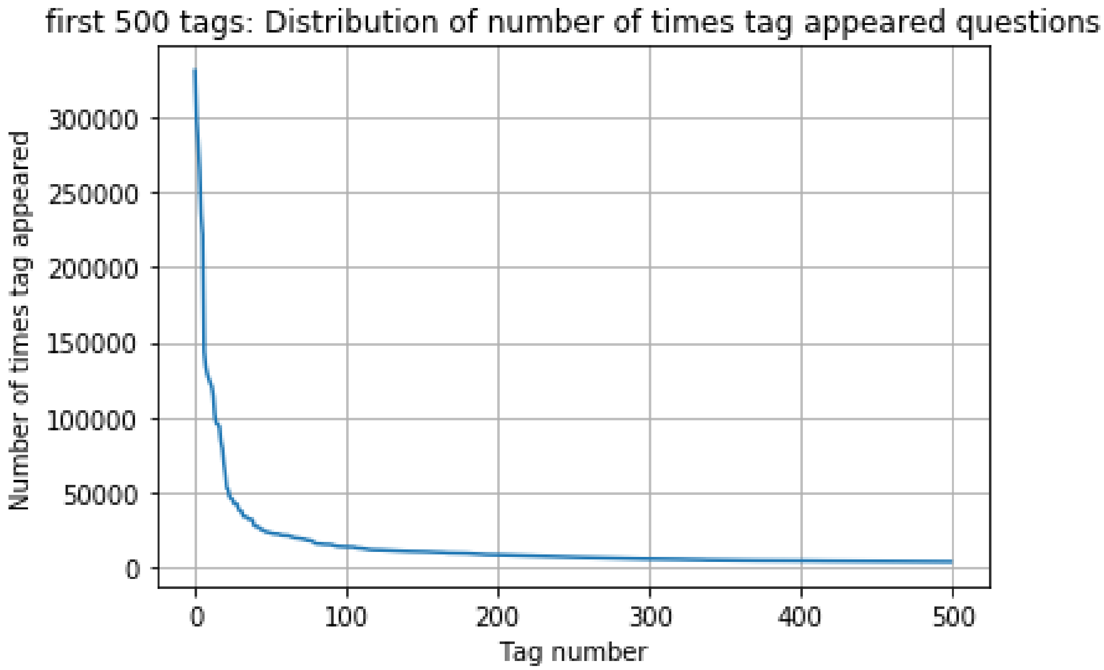
```
 154   153   152   151   150   149   149   148   147   146
 145   144   143   142   142   141   140   139   138   137
 137   136   135   134   134   133   132   131   130   130
 129   128   128   127   126   126   125   124   124   123
 123   122   122   121   120   120   119   118   118   117
 117   116   116   115   115   114   113   113   112   111
 111   110   109   109   108   108   107   106   106   106
 105   105   104   104   103   103   102   102   101   101
 100   100    99    99    98    98    97    97    96    96
  95    95    94    94    93    93    93    92    92    91
  91    90    90    89    89    88    88    87    87    86
  86    86    85    85    84    84    83    83    83    82
  82    82    81    81    80    80    80    79    79    78
  78    78    78    77    77    76    76    76    75    75
  75    74    74    74    73    73    73    73    72    72]
```

[21]:
```
plt.plot(tag_counts[0:1000])
plt.title('first 1k tags: Distribution of number of times tag appeared␣
 ↪questions')
plt.grid()
plt.xlabel("Tag number")
plt.ylabel("Number of times tag appeared")
plt.show()
print(len(tag_counts[0:1000:5]), tag_counts[0:1000:5])
```



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

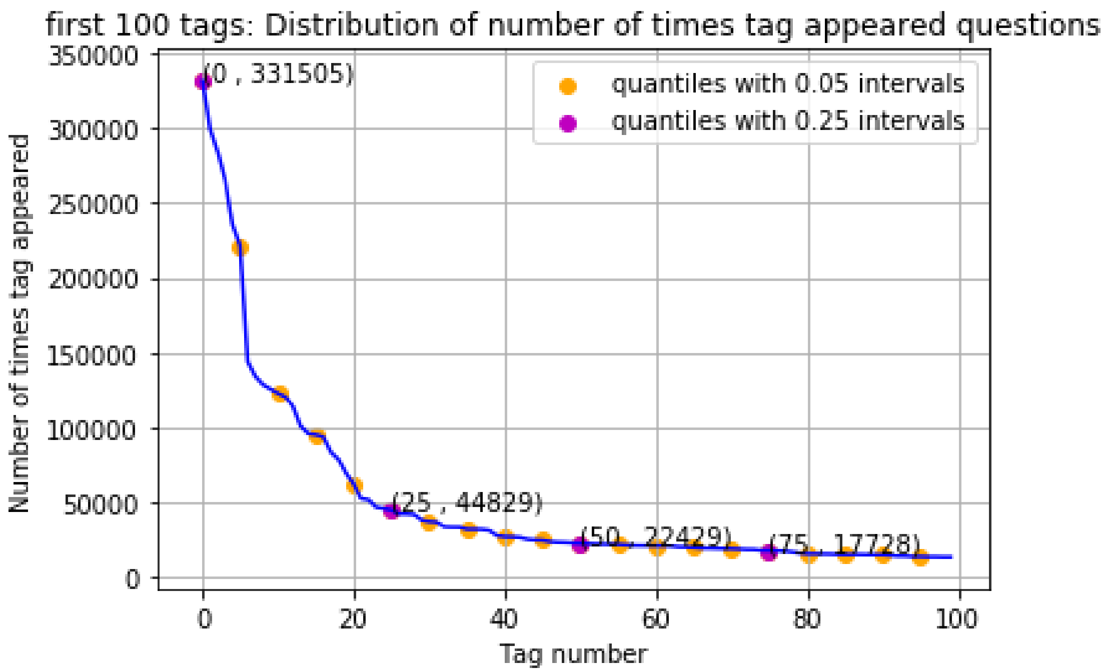
[22]:
```
plt.plot(tag_counts[0:500])
plt.title('first 500 tags: Distribution of number of times tag appeared␣
 ↪questions')
plt.grid()
plt.xlabel("Tag number")
plt.ylabel("Number of times tag appeared")
plt.show()
print(len(tag_counts[0:500:5]), tag_counts[0:500:5])
```




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

[23]:
```
plt.plot(tag_counts[0:100], c='b')
plt.scatter(x=list(range(0,100,5)), y=tag_counts[0:100:5], c='orange',␣
 ↪label="quantiles with 0.05 intervals")
# quantiles with 0.25 difference
plt.scatter(x=list(range(0,100,25)), y=tag_counts[0:100:25], c='m', label =␣
 ↪"quantiles with 0.25 intervals")

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

plt.title('first 100 tags: Distribution of number of times tag appeared␣
 ↪questions')
plt.grid()
plt.xlabel("Tag number")
```

```
plt.ylabel("Number of times tag appeared")
plt.legend()
plt.show()
print(len(tag_counts[0:100:5]), tag_counts[0:100:5])
```




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

[24]:
```
# 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

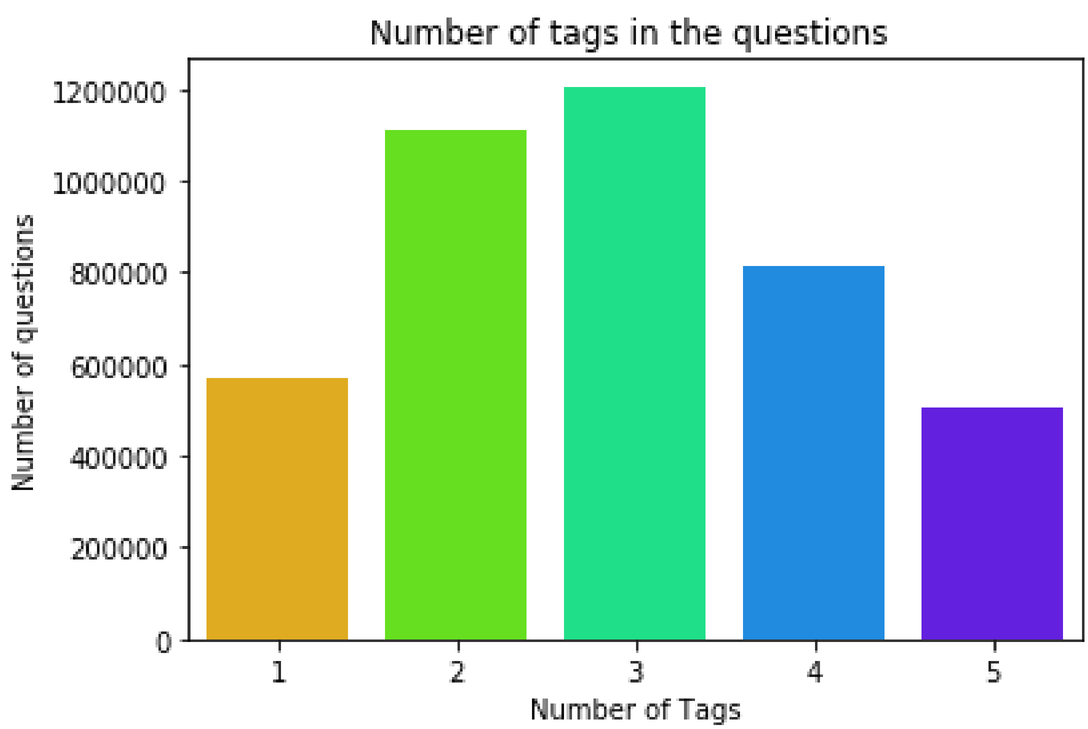
[25]:
```
#Storing the count of tag in each question in list 'tag_count'
tag_quest_count = tag_dtm.sum(axis=1).tolist()
#Converting list of lists into single list, we will get [[3], [4], [2], [2],␣
↪[3]] and we are converting this to [3, 4, 2, 2, 3]
tag_quest_count=[int(j) for i in tag_quest_count for 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]
```

[26]:
```
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
```

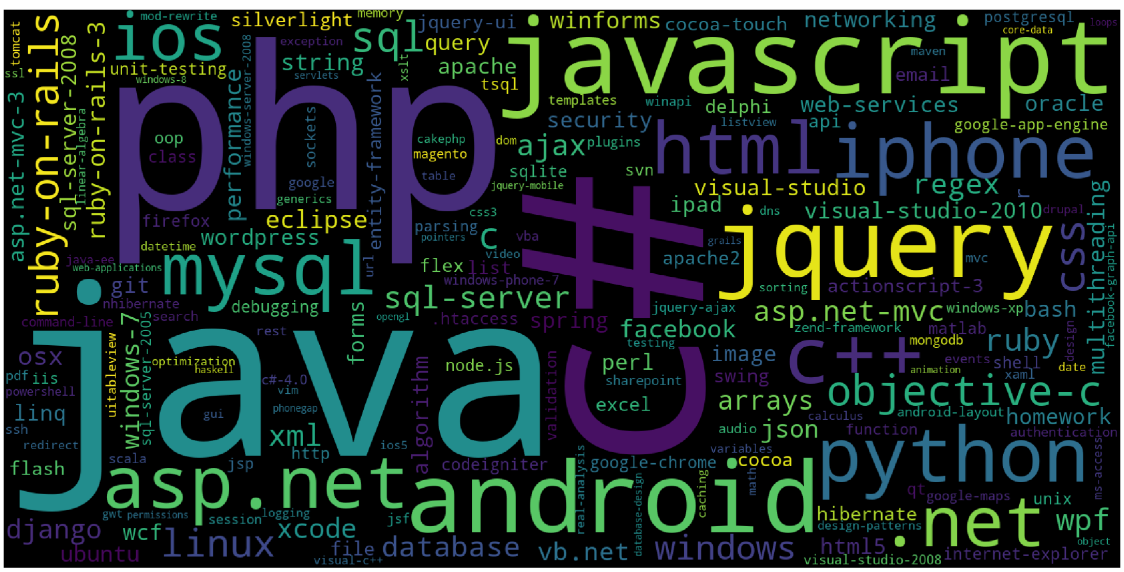
[27]:
```
sns.countplot(tag_quest_count, palette='gist_rainbow')
plt.title("Number of tags in the questions ")
plt.xlabel("Number of Tags")
plt.ylabel("Number of questions")
plt.show()
```



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

[28]:
```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)
```

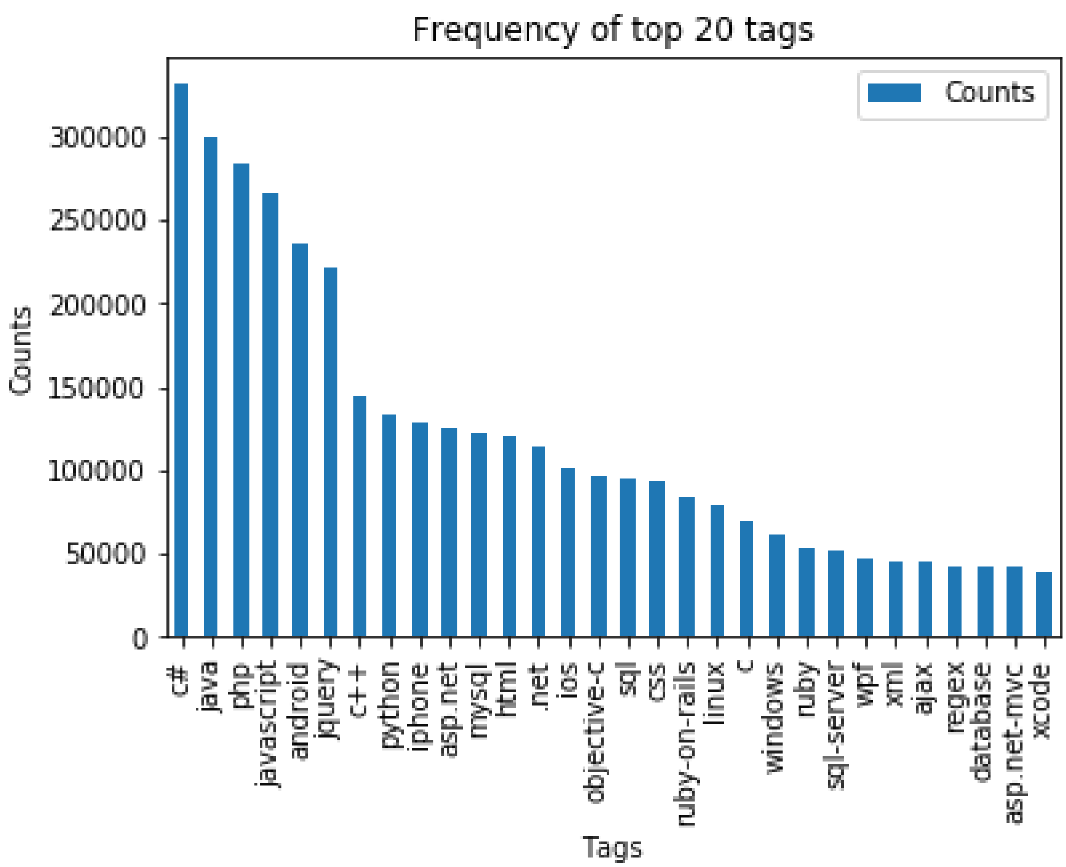



```
Time taken to run this cell : 0:00:04.340505
```

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

[29]:
```
%matplotlib inline
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

Sample 1M data points

Separate out code-snippets from Body

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

Remove stop words (Except 'C')

Remove HTML Tags

Convert all the characters into small letters

Use SnowballStemmer to stem the words

[30]:
```
nltk.download('stopwords')
def striphtml(data):
    cleanr = re.compile('<.*?>')
    cleantext = re.sub(cleanr, ' ', str(data))
    return cleantext
stop_words = set(stopwords.words('english'))
stemmer = SnowballStemmer("english")
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /home/m_nekkalapudi111_gmail_com/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

[31]:
```
#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,␣
 ↪is_code integer);"""
create_database_table("Processed.db", sql_create_table)
```

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

[32]:
```
# # http://www.sqlitetutorial.net/sqlite-delete/
# # https://stackoverflow.com/questions/2279706/
 ↪select-random-row-from-a-sqlite-table
# start = datetime.now()
# read_db = 'train_no_dup.db'
# write_db = 'Processed.db'
# if os.path.isfile(read_db):
#     conn_r = create_connection(read_db)
#     if conn_r is not None:
#         reader =conn_r.cursor()
#         reader.execute("SELECT Title, Body, Tags From no_dup_train ORDER BY␣
 ↪RANDOM() LIMIT 1000000;")

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

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

[33]:
```
# #http://www.bernzilla.com/2008/05/13/
 ↪selecting-a-random-row-from-an-sqlite-table/

# start = datetime.now()
```

```
# nltk.download('punkt')
# preprocessed_data_list=[]
# reader.fetchone()
# questions_with_code=0
# len_pre=0
# len_post=0
# questions_proccesed = 0
# for row in reader:

#     is_code = 0

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

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

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

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

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

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

#     #Removing all single letter and and stopwords from question exceptt for␣
↪the letter 'c'
#     question=' '.join(str(stemmer.stem(j)) for j in words if j not in␣
↪stop_words 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_pre,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)
```

[34]:
```
# # dont forget to close the connections, or else you will end up with locks
# conn_r.commit()
# conn_w.commit()
# conn_r.close()
# conn_w.close()
```

[35]:
```
# 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()
```

[36]:
```
# #Taking 1 Million entries to a dataframe.
# write_db = 'Processed.db'
# if os.path.isfile(write_db):
#     conn_r = create_connection(write_db)
#     if conn_r is not None:
#         preprocessed_data = pd.read_sql_query("""SELECT question, Tags FROM␣
↪QuestionsProcessed""", conn_r)
# conn_r.commit()
# conn_r.close()
```

[37]:
```
# preprocessed_data.head()
```

[38]:
```
# print("number of data points in sample :", preprocessed_data.shape[0])
# print("number of dimensions :", preprocessed_data.shape[1])
```

4. Machine Learning Models

4.1 Converting tags for multilabel problems
X
y1

y2
y3
y4
x1
0
1
1
0
x1
1
0
0
0
x1
0
1
0
0

[39]:
```
# # 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) __

[40]:
```
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))
```

[41]:
```
# 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))
```

[42]:
```
# fig, ax = plt.subplots()
# ax.plot(questions_explained)
# xlabel = list(500+np.array(range(-50,450,50))*50)
# ax.set_xticklabels(xlabel)
# plt.xlabel("Number of tags")
# plt.ylabel("Number Questions coverd partially")
```

```
# plt.grid()
# plt.show()
# # you can choose any number of tags based on your computing power, minimun is␣
↪50(it covers 90% of the tags)
# print("with ",5500,"tags we are covering ",questions_explained[50],"% of␣
↪questions")
```

[43]:
```
# multilabel_yx = tags_to_choose(5500)
# print("number of questions that are not covered :",␣
↪questions_explained_fn(5500),"out of ", total_qs)
```

[44]:
```
# print("Number of tags in sample :", multilabel_y.shape[1])
# print("number of tags taken :", multilabel_yx.shape[1],"(",(multilabel_yx.
↪shape[1]/multilabel_y.shape[1])*100,"%)")
```

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

[45]:
```
# total_size=preprocessed_data.shape[0]
# train_size=int(0.80*total_size)

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

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

[46]:
```
# print("Number of data points in train data :", y_train.shape)
# print("Number of data points in test data :", y_test.shape)
```

4.3 Featurizing data

[47]:
```
# start = datetime.now()
# vectorizer = TfidfVectorizer(min_df=0.00009, max_features=200000,␣
↪smooth_idf=True, norm="l2", \
#                              tokenizer = lambda x: x.split(),␣
↪sublinear_tf=False, ngram_range=(1,3))
# 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)
```

[48]:
```
# print("Dimensions of train data X:",x_train_multilabel.shape, "Y :",y_train.
↪shape)
# print("Dimensions of test data X:",x_test_multilabel.shape,"Y:",y_test.shape)
```

[49]:
```
# https://www.analyticsvidhya.com/blog/2017/08/
↪introduction-to-multi-label-classification/
#https://stats.stackexchange.com/questions/117796/
↪scikit-multi-label-classification
# classifier = LabelPowerset(GaussianNB())
"""
```

```
from skmultilearn.adapt import MLkNN
classifier = MLkNN(k=21)

# train
classifier.fit(x_train_multilabel, y_train)

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

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

[49]:
```
"\nfrom skmultilearn.adapt import MLkNN\nclassifier = MLkNN(k=21)\n\n#
train\nclassifier.fit(x_train_multilabel, y_train)\n\n# predict\npredictions = c
lassifier.predict(x_test_multilabel)\nprint(accuracy_score(y_test,predictions))\
nprint(metrics.f1_score(y_test, predictions, average =
'macro'))\nprint(metrics.f1_score(y_test, predictions, average =
'micro'))\nprint(metrics.hamming_loss(y_test,predictions))\n\n"
```

4.4 Applying Logistic Regression with OneVsRest Classifier

[50]:
```
# # this will be taking so much time try not to run it, download the␣
↪lr_with_equal_weight.pkl file and use to predict
# # This takes about 6-7 hours to run.
# classifier = OneVsRestClassifier(SGDClassifier(loss='log', alpha=0.00001,␣
↪penalty='l1'), n_jobs=-1)
# classifier.fit(x_train_multilabel, y_train)
# predictions = classifier.predict(x_test_multilabel)

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

[51]:
```
# from sklearn.externals import joblib
# joblib.dump(classifier, 'lr_with_equal_weight.pkl')
```

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

[52]:
```
sql_create_table = """CREATE TABLE IF NOT EXISTS QuestionsProcessed (question␣
↪text NOT NULL, code text, tags text, words_pre integer, words_post integer,␣
↪is_code integer);"""
create_database_table("Titlemoreweight.db", sql_create_table)
```

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

[100]:
```
# http://www.sqlitetutorial.net/sqlite-delete/
# https://stackoverflow.com/questions/2279706/
↪select-random-row-from-a-sqlite-table

read_db = 'train_no_dup.db'
write_db = 'Titlemoreweight.db'
train_datasize = 200000
if os.path.isfile(read_db):
    conn_r = create_connection(read_db)
    if conn_r is not None:
        reader =conn_r.cursor()
        # for selecting first 0.5M rows
        reader.execute("SELECT Title, Body, Tags From no_dup_train LIMIT 300001;
↪")
        # for selecting random points
        #reader.execute("SELECT Title, Body, Tags From no_dup_train ORDER BY␣
↪RANDOM() LIMIT 300001;")

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

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

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

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

[101]:
```
#http://www.bernzilla.com/2008/05/13/
↪selecting-a-random-row-from-an-sqlite-table/
start = datetime.now()
preprocessed_data_list=[]
reader.fetchone()
questions_with_code=0
len_pre=0
len_post=0
questions_proccesed = 0
for row in reader:

    is_code = 0

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

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

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

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

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

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

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

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

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

    #Removing all single letter and and stopwords from question exceptt for the␣
↪letter 'c'
```

```
    question=' '.join(str(stemmer.stem(j)) for j in words if j not in␣
↪stop_words 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_pre,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
number of questions completed= 300000
Avg. length of questions(Title+Body) before processing: 1266
Avg. length of questions(Title+Body) after processing: 433
Percent of questions containing code: 55
Time taken to run this cell : 0:09:44.480580
```

[102]:
```
# never forget to close the conections or else we will end up with database␣
↪locks
conn_r.commit()
conn_w.commit()
conn_r.close()
conn_w.close()
```

__ Sample quesitons after preprocessing of data __

[103]:
```
if os.path.isfile(write_db):
    conn_r = create_connection(write_db)
    if conn_r is not None:
        reader =conn_r.cursor()
        reader.execute("SELECT question From QuestionsProcessed LIMIT 10")
        print("Questions after preprocessed")
        print('='*100)
```

```
        reader.fetchone()
        for row in reader:
            print(row)
            print('-'*100)
conn_r.commit()
conn_r.close()
```

```
Questions after preprocessed
================================================================================
====================
('dynam datagrid bind silverlight dynam datagrid bind silverlight dynam datagrid
bind silverlight bind datagrid dynam code wrote code debug code block seem bind
correct grid come column form come grid column although necessari bind nthank
repli advance..',)
--------------------------------------------------------------------------------
--------------------
('java.lang.noclassdeffounderror javax servlet jsp tagext taglibraryvalid
java.lang.noclassdeffounderror javax servlet jsp tagext taglibraryvalid
java.lang.noclassdeffounderror javax servlet jsp tagext taglibraryvalid follow
guid link instal jstl got follow error tri launch jsp page
java.lang.noclassdeffounderror javax servlet jsp tagext taglibraryvalid taglib
declar instal jstl 1.1 tomcat webapp tri project work also tri version 1.2 jstl
still messag caus solv',)
--------------------------------------------------------------------------------
--------------------
('java.sql.sqlexcept microsoft odbc driver manag invalid descriptor index
java.sql.sqlexcept microsoft odbc driver manag invalid descriptor index
java.sql.sqlexcept microsoft odbc driver manag invalid descriptor index use
follow code display caus solv',)
--------------------------------------------------------------------------------
--------------------
('better way updat feed fb php sdk better way updat feed fb php sdk better way
updat feed fb php sdk novic facebook api read mani tutori still confused.i find
post feed api method like correct second way use curl someth like way better',)
--------------------------------------------------------------------------------
--------------------
('btnadd click event open two window record ad btnadd click event open two
window record ad btnadd click event open two window record ad open window
search.aspx use code hav add button search.aspx nwhen insert record btnadd click
event open anoth window nafter insert record close window',)
--------------------------------------------------------------------------------
--------------------
('sql inject issu prevent correct form submiss php sql inject issu prevent
correct form submiss php sql inject issu prevent correct form submiss php check
everyth think make sure input field safe type sql inject good news safe bad news
one tag mess form submiss place even touch life figur exact html use templat
file forgiv okay entir php script get execut see data post none forum field post
```

```
problem use someth titl field none data get post current use print post see
submit noth work flawless statement though also mention script work flawless
local machin use host come across problem state list input test mess',)
--------------------------------------------------------------------------------
--------------------
('countabl subaddit lebesgu measur countabl subaddit lebesgu measur countabl
subaddit lebesgu measur let lbrace rbrace sequenc set sigma -algebra mathcal
want show left bigcup right leq sum left right countabl addit measur defin set
sigma algebra mathcal think use monoton properti somewher proof start appreci
littl help nthank ad han answer make follow addit construct given han answer
clear bigcup bigcup cap emptyset neq left bigcup right left bigcup right sum
left right also construct subset monoton left right leq left right final would
sum leq sum result follow',)
--------------------------------------------------------------------------------
--------------------
('hql equival sql queri hql equival sql queri hql equival sql queri hql queri
replac name class properti name error occur hql error',)
--------------------------------------------------------------------------------
--------------------
('undefin symbol architectur i386 objc class skpsmtpmessag referenc error
undefin symbol architectur i386 objc class skpsmtpmessag referenc error undefin
symbol architectur i386 objc class skpsmtpmessag referenc error import framework
send email applic background import framework i.e skpsmtpmessag somebodi suggest
get error collect2 ld return exit status import framework correct sorc taken
framework follow mfmailcomposeviewcontrol question lock field updat answer drag
drop folder project click copi nthat',)
--------------------------------------------------------------------------------
--------------------
```

__ Saving Preprocessed data to a Database __

[104]:
```python
#Taking 0.5 Million entries to a dataframe.
write_db = 'Titlemoreweight.db'
if os.path.isfile(write_db):
    conn_r = create_connection(write_db)
    if conn_r is not None:
        preprocessed_data = pd.read_sql_query("""SELECT question, Tags FROM␣
↪QuestionsProcessed""", conn_r)
conn_r.commit()
conn_r.close()
```

[105]:
```python
preprocessed_data.head()
```

[105]:
```
                                            question  \
0  dynam datagrid bind silverlight dynam datagrid...
1  dynam datagrid bind silverlight dynam datagrid...
2  java.lang.noclassdeffounderror javax servlet j...
3  java.sql.sqlexcept microsoft odbc driver manag...
4  better way updat feed fb php sdk better way up...
```

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

[106]:
```
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 : 300000
number of dimensions : 2
```

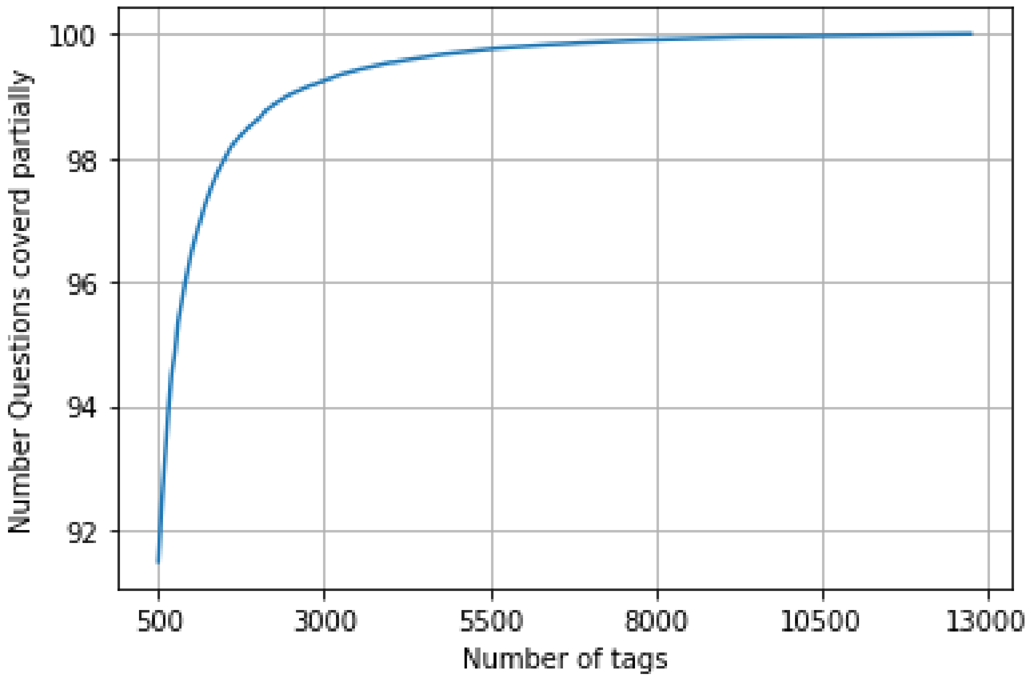
__ Converting string Tags to multilable output variables __

[107]:
```
vectorizer = CountVectorizer(tokenizer = lambda x: x.split(), binary='true')
multilabel_y = vectorizer.fit_transform(preprocessed_data['tags'])
```

__ Selecting 500 Tags __

[108]:
```
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))
```

[109]:
```
fig, ax = plt.subplots()
ax.plot(questions_explained)
xlabel = list(500+np.array(range(-50,450,50))*50)
ax.set_xticklabels(xlabel)
plt.xlabel("Number of tags")
plt.ylabel("Number Questions coverd partially")
plt.grid()
plt.show()
# you can choose any number of tags based on your computing power, minimun is␣
 ↪500(it covers 90% of the tags)
print("with ",5500,"tags we are covering ",questions_explained[50],"% of␣
 ↪questions")
print("with ",500,"tags we are covering ",questions_explained[0],"% of␣
 ↪questions")
```




```
with  5500 tags we are covering  99.244 % of questions
with  500 tags we are covering  91.492 % of questions
```

[110]:
```
# we will be taking 500 tags
multilabel_yx = tags_to_choose(500)
print("number of questions that are not covered :",␣
 ↪questions_explained_fn(500),"out of ", total_qs)
```

```
number of questions that are not covered : 25523 out of  300000
```

[115]:
```
x_train=preprocessed_data.head(train_datasize)
x_test=preprocessed_data.tail(preprocessed_data.shape[0] - 200000)

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

[116]:
```
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 : (200000, 500)
Number of data points in test data : (100000, 500)
```

4.5.2 Featurizing data with BOW vectorizer

[117]:
```
start = datetime.now()
vectorizer = CountVectorizer(min_df=0.00009, max_features=200000, tokenizer =␣
↪lambda x: x.split(), ngram_range=(1,3))
x_train_multilabel = vectorizer.fit_transform(x_train['question'])
x_test_multilabel = vectorizer.transform(x_test['question'])
print("Time taken to run this cell :", datetime.now() - start)
```

```
Time taken to run this cell : 0:03:07.652468
```

[118]:
```
print("Dimensions of train data X:",x_train_multilabel.shape, "Y :",y_train.
↪shape)
print("Dimensions of test data X:",x_test_multilabel.shape,"Y:",y_test.shape)
```

```
Dimensions of train data X: (200000, 98488) Y : (200000, 500)
Dimensions of test data X: (100000, 98488) Y: (100000, 500)
```

4.5.3 Applying Logistic Regression with OneVsRest Classifier

[119]:
```
start = datetime.now()
sgd_clf = SGDClassifier(loss='hinge', alpha=0.00001, penalty='l1', n_jobs=-1)
classifier = OneVsRestClassifier(sgd_clf)
classifier.fit(x_train_multilabel, y_train)
predictions = classifier.predict (x_test_multilabel)


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


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

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

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

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

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

```
Accuracy : 0.07129
Hamming loss  0.00819042
Micro-average quality numbers
Precision: 0.2112, Recall: 0.4685, F1-measure: 0.2911
Macro-average quality numbers
Precision: 0.1515, Recall: 0.3979, F1-measure: 0.2071
             precision    recall  f1-score   support

          0       0.65      0.77      0.70      4633
          1       0.38      0.37      0.38      7549
          2       0.47      0.49      0.48      7112
          3       0.47      0.61      0.53      3919
          4       0.43      0.49      0.46      5009
          5       0.50      0.58      0.54      5204
          6       0.47      0.61      0.53      3229
          7       0.21      0.29      0.24      3097
          8       0.41      0.51      0.45      3142
          9       0.30      0.48      0.37      1549
         10       0.44      0.61      0.51      2888
         11       0.25      0.32      0.28      2157
         12       0.35      0.46      0.40      2575
         13       0.18      0.43      0.25       852
         14       0.27      0.41      0.33      2094
         15       0.49      0.60      0.54      1829
         16       0.52      0.63      0.57      2649
         17       0.33      0.65      0.44      1614
         18       0.42      0.62      0.50      1869
         19       0.25      0.35      0.29      2268
         20       0.19      0.56      0.28       388
         21       0.22      0.33      0.27      1350
         22       0.17      0.28      0.21      1798
         23       0.32      0.54      0.41      2380
         24       0.52      0.65      0.58      2896
         25       0.26      0.43      0.33      1431
         26       0.17      0.41      0.24       695
         27       0.27      0.55      0.36       480
         28       0.26      0.48      0.33       717
         29       0.78      0.85      0.81      2408
         30       0.35      0.51      0.42      1528
         31       0.15      0.28      0.20      1095
         32       0.23      0.50      0.32       632
         33       0.28      0.50      0.36      1046
         34       0.14      0.41      0.21       579
         35       0.18      0.41      0.25       632
         36       0.24      0.45      0.31       986
         37       0.04      0.14      0.06       151
         38       0.19      0.35      0.24       929
         39       0.13      0.42      0.20       385
```

40	0.43	0.64	0.52	888
41	0.21	0.44	0.29	719
42	0.22	0.50	0.30	516
43	0.21	0.47	0.29	728
44	0.22	0.46	0.29	964
45	0.17	0.37	0.23	652
46	0.11	0.46	0.17	138
47	0.35	0.53	0.42	531
48	0.88	0.90	0.89	1622
49	0.19	0.32	0.24	848
50	0.12	0.27	0.16	790
51	0.33	0.67	0.44	716
52	0.09	0.28	0.14	410
53	0.11	0.29	0.16	499
54	0.18	0.38	0.25	535
55	0.11	0.59	0.18	203
56	0.44	0.78	0.57	640
57	0.07	0.20	0.11	423
58	0.13	0.35	0.19	455
59	0.20	0.42	0.27	702
60	0.21	0.39	0.27	839
61	0.32	0.76	0.45	546
62	0.11	0.29	0.15	444
63	0.12	0.41	0.19	237
64	0.10	0.27	0.14	485
65	0.30	0.65	0.42	492
66	0.14	0.32	0.20	517
67	0.06	0.28	0.09	101
68	0.11	0.24	0.15	480
69	0.15	0.56	0.23	264
70	0.17	0.37	0.23	473
71	0.08	0.19	0.11	350
72	0.28	0.48	0.35	568
73	0.17	0.44	0.24	283
74	0.05	0.41	0.09	74
75	0.07	0.29	0.11	224
76	0.67	0.78	0.72	854
77	0.11	0.52	0.19	144
78	0.12	0.38	0.18	325
79	0.03	0.12	0.04	111
80	0.31	0.62	0.41	525
81	0.04	0.09	0.06	413
82	0.15	0.54	0.24	205
83	0.74	0.91	0.82	905
84	0.14	0.33	0.20	306
85	0.18	0.40	0.25	307
86	0.10	0.78	0.18	32
87	0.15	0.33	0.20	469

88	0.05	0.13	0.07	374
89	0.19	0.49	0.28	490
90	0.06	0.20	0.10	353
91	0.13	0.29	0.18	196
92	0.73	0.86	0.79	745
93	0.07	0.19	0.10	374
94	0.05	0.24	0.09	107
95	0.11	0.30	0.16	383
96	0.45	0.58	0.51	866
97	0.05	0.12	0.07	403
98	0.12	0.63	0.20	81
99	0.02	0.13	0.04	116
100	0.12	0.34	0.18	416
101	0.09	0.39	0.14	127
102	0.19	0.58	0.28	284
103	0.38	0.50	0.43	824
104	0.16	0.25	0.19	606
105	0.35	0.49	0.40	401
106	0.04	0.26	0.07	126
107	0.05	0.17	0.08	233
108	0.14	0.40	0.20	260
109	0.22	0.52	0.30	629
110	0.23	0.46	0.30	375
111	0.04	0.25	0.06	69
112	0.11	0.31	0.16	353
113	0.06	0.21	0.10	457
114	0.02	0.23	0.04	86
115	0.09	0.28	0.14	283
116	0.17	0.65	0.27	172
117	0.13	0.60	0.22	100
118	0.10	0.42	0.16	113
119	0.10	0.37	0.16	216
120	0.45	0.75	0.56	360
121	0.18	0.55	0.27	263
122	0.08	0.24	0.12	238
123	0.43	0.76	0.55	480
124	0.11	0.37	0.17	427
125	0.07	0.25	0.11	255
126	0.19	0.48	0.28	281
127	0.04	0.15	0.06	225
128	0.13	0.35	0.19	530
129	0.22	0.42	0.29	352
130	0.06	0.39	0.10	119
131	0.23	0.50	0.32	405
132	0.19	0.50	0.27	159
133	0.14	0.52	0.22	296
134	0.32	0.77	0.45	311
135	0.05	0.19	0.08	237

136	0.18	0.43	0.26	220
137	0.10	0.33	0.16	273
138	0.06	0.24	0.10	216
139	0.20	0.46	0.27	363
140	0.04	0.39	0.08	38
141	0.01	0.12	0.02	88
142	0.12	0.42	0.18	219
143	0.09	0.32	0.15	238
144	0.23	0.54	0.32	186
145	0.50	0.70	0.58	408
146	0.33	0.61	0.43	343
147	0.08	0.39	0.14	125
148	0.06	0.30	0.10	183
149	0.25	0.55	0.35	292
150	0.02	0.10	0.03	86
151	0.38	0.42	0.40	595
152	0.08	0.27	0.12	265
153	0.20	0.54	0.30	219
154	0.09	0.34	0.14	201
155	0.12	0.25	0.16	369
156	0.29	0.63	0.40	280
157	0.15	0.40	0.21	234
158	0.25	0.56	0.35	255
159	0.13	0.42	0.20	175
160	0.32	0.69	0.44	401
161	0.16	0.39	0.22	222
162	0.07	0.35	0.11	208
163	0.04	0.14	0.07	332
164	0.05	0.18	0.08	213
165	0.16	0.35	0.22	234
166	0.08	0.23	0.12	271
167	0.03	0.21	0.06	52
168	0.23	0.58	0.33	229
169	0.11	0.33	0.17	228
170	0.09	0.32	0.14	224
171	0.02	0.33	0.04	30
172	0.11	0.26	0.16	559
173	0.03	0.13	0.05	211
174	0.08	0.29	0.12	189
175	0.32	0.66	0.43	153
176	0.08	0.25	0.13	234
177	0.21	0.48	0.30	292
178	0.14	0.38	0.21	206
179	0.15	0.40	0.21	345
180	0.11	0.26	0.16	364
181	0.09	0.50	0.16	103
182	0.02	0.05	0.02	232
183	0.08	0.25	0.12	240

184	0.05	0.18	0.08	205
185	0.39	0.72	0.51	254
186	0.06	0.21	0.09	199
187	0.05	0.43	0.10	109
188	0.02	0.33	0.05	42
189	0.25	0.59	0.35	259
190	0.12	0.41	0.19	229
191	0.32	0.71	0.44	278
192	0.04	0.17	0.07	160
193	0.25	0.65	0.36	305
194	0.11	0.32	0.16	228
195	0.22	0.52	0.31	192
196	0.31	0.50	0.39	441
197	0.14	0.59	0.23	87
198	0.05	0.21	0.09	270
199	0.11	0.40	0.17	228
200	0.03	0.19	0.05	118
201	0.26	0.65	0.37	201
202	0.29	0.64	0.40	129
203	0.08	0.24	0.12	246
204	0.09	0.33	0.15	308
205	0.04	0.12	0.06	293
206	0.17	0.44	0.24	180
207	0.19	0.61	0.29	99
208	0.06	0.15	0.09	227
209	0.06	0.17	0.09	384
210	0.34	0.73	0.46	208
211	0.17	0.51	0.25	187
212	0.12	0.39	0.19	199
213	0.07	0.16	0.10	370
214	0.02	0.13	0.03	108
215	0.11	0.38	0.17	199
216	0.07	0.19	0.10	289
217	0.03	0.17	0.05	86
218	0.17	0.56	0.26	177
219	0.07	0.35	0.12	142
220	0.05	0.20	0.08	172
221	0.10	0.29	0.15	259
222	0.08	0.22	0.12	256
223	0.07	0.20	0.11	319
224	0.22	0.58	0.32	207
225	0.14	0.48	0.22	167
226	0.36	0.78	0.49	207
227	0.11	0.61	0.18	79
228	0.04	0.56	0.07	16
229	0.10	0.28	0.15	225
230	0.31	0.58	0.40	279
231	0.03	0.13	0.04	116

232	0.16	0.63	0.25	79
233	0.06	0.30	0.11	186
234	0.02	0.14	0.04	80
235	0.08	0.34	0.13	209
236	0.20	0.47	0.28	224
237	0.04	0.25	0.07	152
238	0.04	0.47	0.08	34
239	0.13	0.38	0.19	143
240	0.08	0.34	0.13	144
241	0.01	0.20	0.02	40
242	0.02	0.09	0.03	118
243	0.67	0.81	0.73	439
244	0.04	0.24	0.07	113
245	0.06	0.37	0.10	82
246	0.05	0.20	0.09	191
247	0.21	0.46	0.29	208
248	0.10	0.30	0.15	248
249	0.28	0.62	0.39	191
250	0.04	0.15	0.06	142
251	0.07	0.64	0.12	14
252	0.04	0.30	0.07	81
253	0.04	0.46	0.08	37
254	0.09	0.43	0.14	147
255	0.33	0.69	0.45	100
256	0.04	0.50	0.07	14
257	0.04	0.39	0.07	49
258	0.07	0.36	0.12	153
259	0.25	0.50	0.33	117
260	0.07	0.30	0.12	183
261	0.17	0.38	0.23	238
262	0.16	0.44	0.23	156
263	0.05	0.46	0.10	76
264	0.16	0.61	0.25	171
265	0.04	0.19	0.07	193
266	0.13	0.60	0.21	140
267	0.30	0.53	0.39	201
268	0.06	0.23	0.10	164
269	0.01	0.04	0.02	216
270	0.15	0.60	0.24	114
271	0.06	0.40	0.11	85
272	0.14	0.46	0.22	112
273	0.14	0.38	0.21	169
274	0.08	0.38	0.14	95
275	0.03	0.21	0.06	107
276	0.11	0.37	0.17	152
277	0.02	0.08	0.03	156
278	0.11	0.37	0.17	160
279	0.03	0.33	0.05	27

280	0.14	0.43	0.21	100
281	0.17	0.40	0.24	84
282	0.05	0.23	0.09	169
283	0.02	0.10	0.03	63
284	0.06	0.26	0.09	47
285	0.01	0.05	0.02	167
286	0.07	0.23	0.11	119
287	0.03	0.40	0.05	20
288	0.07	0.34	0.12	50
289	0.08	0.37	0.13	141
290	0.14	0.50	0.22	172
291	0.03	0.28	0.05	47
292	0.14	0.52	0.22	160
293	0.21	0.49	0.29	92
294	0.41	0.64	0.50	172
295	0.04	0.21	0.06	91
296	0.12	0.29	0.17	267
297	0.22	0.76	0.34	114
298	0.12	0.49	0.19	138
299	0.12	0.40	0.19	224
300	0.08	0.30	0.12	200
301	0.25	0.65	0.36	111
302	0.25	0.63	0.35	199
303	0.14	0.33	0.20	298
304	0.17	0.54	0.25	153
305	0.08	0.47	0.14	80
306	0.11	0.37	0.17	136
307	0.03	0.19	0.05	95
308	0.21	0.72	0.33	170
309	0.13	0.46	0.20	134
310	0.30	0.74	0.43	157
311	0.18	0.50	0.27	217
312	0.08	0.27	0.12	108
313	0.34	0.75	0.47	159
314	0.12	0.52	0.20	111
315	0.02	0.29	0.04	31
316	0.35	0.91	0.51	94
317	0.06	0.18	0.09	109
318	0.07	0.41	0.12	101
319	0.10	0.31	0.15	158
320	0.15	0.52	0.24	138
321	0.70	0.89	0.79	316
322	0.05	0.31	0.09	71
323	0.04	0.25	0.06	65
324	0.18	0.55	0.27	120
325	0.08	0.30	0.13	186
326	0.05	0.18	0.07	202
327	0.85	0.90	0.87	351

328	0.10	0.45	0.16	106
329	0.08	0.24	0.12	160
330	0.45	0.75	0.57	192
331	0.06	0.30	0.09	91
332	0.16	0.41	0.23	232
333	0.01	0.09	0.02	93
334	0.77	0.70	0.73	336
335	0.06	0.36	0.11	87
336	0.20	0.65	0.30	161
337	0.07	0.27	0.11	186
338	0.28	0.71	0.40	149
339	0.35	0.46	0.40	297
340	0.10	0.33	0.16	136
341	0.04	0.58	0.07	24
342	0.12	0.47	0.19	129
343	0.09	0.41	0.14	119
344	0.42	0.74	0.54	202
345	0.23	0.59	0.33	141
346	0.10	0.28	0.14	174
347	0.03	0.30	0.05	37
348	0.22	0.64	0.33	107
349	0.08	0.50	0.14	76
350	0.77	0.93	0.85	269
351	0.43	0.82	0.56	143
352	0.24	0.60	0.35	143
353	0.12	0.43	0.19	123
354	0.02	0.07	0.03	121
355	0.06	0.26	0.10	66
356	0.04	0.19	0.07	189
357	0.04	0.21	0.06	52
358	0.09	0.35	0.14	181
359	0.05	0.20	0.07	154
360	0.09	0.31	0.13	138
361	0.05	0.23	0.08	114
362	0.05	0.23	0.09	62
363	0.15	0.37	0.22	214
364	0.01	0.18	0.01	11
365	0.05	0.26	0.09	142
366	0.03	0.24	0.05	38
367	0.06	0.33	0.10	82
368	0.08	0.29	0.13	83
369	0.21	0.58	0.31	110
370	0.12	0.46	0.19	81
371	0.01	0.04	0.01	99
372	0.30	0.79	0.44	115
373	0.08	0.64	0.14	22
374	0.04	0.20	0.06	81
375	0.06	0.31	0.10	68

376	0.02	0.08	0.03	142
377	0.10	0.37	0.16	139
378	0.04	0.36	0.08	45
379	0.06	0.40	0.10	42
380	0.03	0.16	0.05	124
381	0.04	0.50	0.07	12
382	0.28	0.45	0.34	247
383	0.04	0.38	0.07	37
384	0.06	0.37	0.11	90
385	0.01	0.11	0.02	65
386	0.15	0.43	0.22	124
387	0.17	0.49	0.26	110
388	0.18	0.51	0.26	74
389	0.13	0.42	0.20	126
390	0.05	0.13	0.07	143
391	0.06	0.17	0.08	120
392	0.20	0.53	0.29	190
393	0.11	0.35	0.16	123
394	0.08	0.43	0.13	99
395	0.55	0.82	0.66	214
396	0.12	0.53	0.20	83
397	0.01	0.15	0.02	40
398	0.04	0.22	0.07	83
399	0.04	0.14	0.06	121
400	0.06	0.40	0.11	62
401	0.06	0.31	0.11	95
402	0.01	0.08	0.02	101
403	0.13	0.42	0.20	116
404	0.17	0.49	0.25	135
405	0.21	0.51	0.30	71
406	0.06	0.25	0.09	115
407	0.05	0.31	0.09	95
408	0.03	0.09	0.04	126
409	0.02	0.24	0.04	29
410	0.24	0.66	0.36	132
411	0.02	0.11	0.04	98
412	0.04	0.17	0.07	136
413	0.01	0.12	0.02	33
414	0.06	0.20	0.09	127
415	0.10	0.41	0.16	76
416	0.38	0.83	0.52	108
417	0.06	0.29	0.10	112
418	0.10	0.31	0.15	128
419	0.05	0.29	0.09	111
420	0.08	0.65	0.15	34
421	0.04	0.21	0.06	75
422	0.08	0.29	0.13	170
423	0.09	0.30	0.14	162

424	0.04	0.23	0.06	86
425	0.05	0.37	0.10	71
426	0.09	0.50	0.15	109
427	0.11	0.34	0.16	200
428	0.05	0.31	0.09	89
429	0.01	0.22	0.02	36
430	0.02	0.25	0.04	16
431	0.04	0.14	0.06	122
432	0.00	0.06	0.01	16
433	0.24	0.53	0.33	127
434	0.06	0.30	0.10	100
435	0.05	0.67	0.09	12
436	0.02	0.22	0.03	27
437	0.13	0.44	0.20	135
438	0.12	0.45	0.19	121
439	0.20	0.82	0.33	34
440	0.04	0.22	0.06	85
441	0.19	0.49	0.27	83
442	0.02	0.15	0.03	78
443	0.03	0.23	0.05	87
444	0.40	0.72	0.51	134
445	0.04	0.27	0.07	56
446	0.10	0.41	0.16	85
447	0.04	0.42	0.08	26
448	0.05	0.20	0.08	83
449	0.11	0.49	0.17	107
450	0.32	0.65	0.43	114
451	0.04	0.23	0.07	90
452	0.06	0.37	0.10	59
453	0.05	0.33	0.08	66
454	0.05	0.22	0.09	120
455	0.03	0.12	0.05	83
456	0.07	0.19	0.11	80
457	0.04	0.41	0.07	17
458	0.02	0.36	0.04	14
459	0.17	0.37	0.24	148
460	0.04	0.29	0.07	31
461	0.07	0.25	0.11	149
462	0.08	0.32	0.13	53
463	0.11	0.34	0.17	113
464	0.37	0.84	0.52	94
465	0.03	0.25	0.05	28
466	0.03	0.12	0.05	78
467	0.06	0.21	0.09	67
468	0.04	0.26	0.07	70
469	0.16	0.49	0.25	69
470	0.04	0.24	0.07	97
471	0.15	0.56	0.23	115

```
         472       0.11      0.44      0.17        75
         473       0.02      0.07      0.03        97
         474       0.06      0.20      0.09       105
         475       0.01      0.43      0.02         7
         476       0.15      0.57      0.24       112
         477       0.05      0.45      0.10        42
         478       0.15      0.47      0.23        91
         479       0.03      0.19      0.06        74
         480       0.72      0.82      0.77       208
         481       0.08      0.19      0.11        73
         482       0.03      0.13      0.05       100
         483       0.04      0.23      0.07        84
         484       0.07      0.31      0.12        87
         485       0.12      0.59      0.20        54
         486       0.01      0.15      0.02        27
         487       0.01      0.12      0.02        48
         488       0.10      0.46      0.16        70
         489       0.13      0.56      0.22        88
         490       0.04      0.31      0.07        29
         491       0.15      0.53      0.23       115
         492       0.10      0.34      0.16       110
         493       0.18      0.43      0.25       119
         494       0.04      0.28      0.07        39
         495       0.11      0.40      0.18        85
         496       0.09      0.32      0.15       139
         497       0.05      0.35      0.09        34
         498       0.21      0.55      0.31       129
         499       0.03      0.24      0.05        33

   micro avg       0.21      0.47      0.29    179520
   macro avg       0.15      0.40      0.21    179520
weighted avg       0.30      0.47      0.35    179520
 samples avg       0.31      0.45      0.32    179520

Time taken to run this cell : 2:23:14.946314
```

1.1 Logistic Regression with One vs Rest Classifier and Hyperprameter Tuning

[120]:
```
start = datetime.now()
hyper_param = {'estimator__C': [10**-5,10**-4, 10**-3, 10**-2, 10**-1, 1,␣
 ↪10**1, 10**2, 10**3, 10**4,10**5]}

classifier = OneVsRestClassifier(LogisticRegression(penalty='l1'))

clf = GridSearchCV(classifier, hyper_param, scoring = 'f1_micro', cv=3,␣
 ↪n_jobs=-1)
```

```
clf.fit(x_train_multilabel, y_train)

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

```
Time taken to run this cell : 6:06:10.528862
```

[125]:
```
start = datetime.now()

best_C = clf.best_params_['estimator__C']

best_clf = OneVsRestClassifier(LogisticRegression(C= best_C, penalty='l1'),␣
 ↪n_jobs=-1)

best_clf.fit(x_train_multilabel, y_train)

predictions = best_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 : 0.20191
Hamming loss  0.00328342
Micro-average quality numbers
Precision: 0.5589, Recall: 0.4054, F1-measure: 0.4700
Macro-average quality numbers
Precision: 0.4295, Recall: 0.3206, F1-measure: 0.3626
              precision    recall  f1-score   support
```

0	0.88	0.73	0.80	4633
1	0.48	0.32	0.39	7549
2	0.61	0.45	0.52	7112
3	0.72	0.57	0.63	3919
4	0.60	0.46	0.52	5009
5	0.68	0.53	0.60	5204
6	0.69	0.58	0.63	3229
7	0.38	0.22	0.28	3097
8	0.60	0.45	0.52	3142
9	0.64	0.45	0.53	1549
10	0.67	0.53	0.59	2888
11	0.40	0.27	0.32	2157
12	0.52	0.40	0.45	2575
13	0.53	0.39	0.45	852
14	0.44	0.32	0.37	2094
15	0.71	0.58	0.63	1829
16	0.79	0.59	0.67	2649
17	0.70	0.57	0.63	1614
18	0.72	0.56	0.63	1869
19	0.42	0.31	0.35	2268
20	0.64	0.49	0.56	388
21	0.42	0.30	0.35	1350
22	0.30	0.16	0.21	1798
23	0.41	0.45	0.43	2380
24	0.65	0.52	0.58	2896
25	0.49	0.37	0.42	1431
26	0.46	0.29	0.36	695
27	0.54	0.45	0.49	480
28	0.60	0.45	0.52	717
29	0.91	0.87	0.89	2408
30	0.52	0.41	0.46	1528
31	0.30	0.18	0.22	1095
32	0.69	0.45	0.55	632
33	0.61	0.41	0.49	1046
34	0.40	0.31	0.35	579
35	0.56	0.32	0.40	632
36	0.49	0.35	0.41	986
37	0.14	0.08	0.10	151
38	0.40	0.25	0.31	929
39	0.51	0.34	0.41	385
40	0.79	0.57	0.66	888
41	0.52	0.43	0.47	719
42	0.66	0.49	0.56	516
43	0.51	0.39	0.44	728
44	0.51	0.34	0.41	964
45	0.35	0.25	0.29	652
46	0.51	0.30	0.38	138

47	0.69	0.51	0.59	531
48	0.98	0.90	0.94	1622
49	0.44	0.24	0.31	848
50	0.31	0.19	0.23	790
51	0.72	0.54	0.62	716
52	0.26	0.17	0.20	410
53	0.25	0.20	0.22	499
54	0.56	0.29	0.38	535
55	0.52	0.52	0.52	203
56	0.88	0.72	0.79	640
57	0.19	0.11	0.14	423
58	0.46	0.26	0.34	455
59	0.51	0.31	0.39	702
60	0.42	0.26	0.32	839
61	0.75	0.72	0.73	546
62	0.32	0.24	0.28	444
63	0.44	0.31	0.36	237
64	0.31	0.18	0.23	485
65	0.67	0.56	0.61	492
66	0.31	0.21	0.25	517
67	0.23	0.11	0.15	101
68	0.32	0.20	0.25	480
69	0.64	0.49	0.56	264
70	0.60	0.34	0.43	473
71	0.29	0.14	0.19	350
72	0.54	0.39	0.45	568
73	0.56	0.45	0.50	283
74	0.20	0.15	0.17	74
75	0.34	0.18	0.24	224
76	0.96	0.89	0.92	854
77	0.62	0.45	0.52	144
78	0.34	0.22	0.27	325
79	0.13	0.06	0.09	111
80	0.74	0.59	0.66	525
81	0.12	0.04	0.06	413
82	0.63	0.47	0.54	205
83	0.91	0.89	0.90	905
84	0.43	0.17	0.24	306
85	0.62	0.44	0.51	307
86	0.68	0.59	0.63	32
87	0.45	0.27	0.34	469
88	0.03	0.01	0.02	374
89	0.54	0.36	0.43	490
90	0.20	0.07	0.10	353
91	0.30	0.25	0.27	196
92	0.94	0.88	0.91	745
93	0.21	0.12	0.15	374
94	0.21	0.14	0.17	107

95	0.32	0.20	0.24	383
96	0.69	0.45	0.55	866
97	0.14	0.07	0.10	403
98	0.59	0.57	0.58	81
99	0.14	0.05	0.07	116
100	0.45	0.26	0.33	416
101	0.50	0.31	0.38	127
102	0.76	0.49	0.59	284
103	0.52	0.43	0.47	824
104	0.33	0.20	0.24	606
105	0.73	0.47	0.57	401
106	0.17	0.13	0.15	126
107	0.13	0.08	0.10	233
108	0.47	0.35	0.40	260
109	0.47	0.38	0.42	629
110	0.68	0.37	0.48	375
111	0.25	0.14	0.18	69
112	0.36	0.24	0.29	353
113	0.25	0.13	0.17	457
114	0.22	0.15	0.18	86
115	0.27	0.17	0.21	283
116	0.63	0.47	0.54	172
117	0.58	0.51	0.54	100
118	0.43	0.33	0.37	113
119	0.41	0.31	0.35	216
120	0.83	0.76	0.79	360
121	0.55	0.44	0.49	263
122	0.21	0.11	0.14	238
123	0.80	0.69	0.74	480
124	0.44	0.28	0.34	427
125	0.31	0.20	0.25	255
126	0.60	0.40	0.48	281
127	0.15	0.08	0.10	225
128	0.52	0.32	0.40	530
129	0.53	0.41	0.46	352
130	0.49	0.35	0.41	119
131	0.57	0.48	0.52	405
132	0.32	0.36	0.34	159
133	0.46	0.45	0.45	296
134	0.76	0.60	0.67	311
135	0.24	0.14	0.18	237
136	0.34	0.28	0.31	220
137	0.47	0.28	0.35	273
138	0.24	0.16	0.19	216
139	0.59	0.45	0.51	363
140	0.29	0.26	0.27	38
141	0.03	0.02	0.02	88
142	0.34	0.29	0.31	219

143	0.34	0.20	0.25	238
144	0.57	0.46	0.51	186
145	0.78	0.66	0.71	408
146	0.65	0.55	0.59	343
147	0.40	0.27	0.33	125
148	0.34	0.15	0.21	183
149	0.55	0.50	0.52	292
150	0.27	0.08	0.13	86
151	0.57	0.34	0.43	595
152	0.28	0.15	0.19	265
153	0.60	0.44	0.51	219
154	0.32	0.24	0.27	201
155	0.32	0.21	0.25	369
156	0.86	0.61	0.72	280
157	0.55	0.30	0.39	234
158	0.71	0.56	0.63	255
159	0.48	0.27	0.35	175
160	0.69	0.69	0.69	401
161	0.66	0.40	0.50	222
162	0.31	0.29	0.30	208
163	0.29	0.11	0.15	332
164	0.13	0.07	0.09	213
165	0.50	0.26	0.34	234
166	0.24	0.13	0.17	271
167	0.26	0.10	0.14	52
168	0.64	0.54	0.59	229
169	0.33	0.26	0.29	228
170	0.38	0.28	0.32	224
171	0.27	0.33	0.30	30
172	0.25	0.14	0.18	559
173	0.20	0.10	0.13	211
174	0.33	0.19	0.24	189
175	0.83	0.62	0.71	153
176	0.32	0.19	0.24	234
177	0.71	0.44	0.55	292
178	0.46	0.36	0.41	206
179	0.50	0.30	0.37	345
180	0.31	0.23	0.26	364
181	0.56	0.41	0.47	103
182	0.19	0.05	0.08	232
183	0.38	0.25	0.30	240
184	0.16	0.09	0.11	205
185	0.77	0.71	0.74	254
186	0.22	0.12	0.16	199
187	0.53	0.39	0.44	109
188	0.26	0.26	0.26	42
189	0.55	0.47	0.51	259
190	0.45	0.31	0.37	229

191	0.71	0.64	0.67	278
192	0.15	0.08	0.10	160
193	0.81	0.61	0.69	305
194	0.48	0.27	0.35	228
195	0.46	0.46	0.46	192
196	0.57	0.40	0.47	441
197	0.70	0.53	0.60	87
198	0.27	0.17	0.21	270
199	0.42	0.32	0.36	228
200	0.14	0.10	0.12	118
201	0.73	0.57	0.64	201
202	0.58	0.53	0.56	129
203	0.19	0.09	0.12	246
204	0.41	0.27	0.32	308
205	0.21	0.09	0.13	293
206	0.69	0.40	0.51	180
207	0.57	0.59	0.58	99
208	0.24	0.11	0.15	227
209	0.14	0.07	0.09	384
210	0.82	0.61	0.70	208
211	0.68	0.51	0.59	187
212	0.58	0.36	0.44	199
213	0.14	0.10	0.11	370
214	0.26	0.09	0.14	108
215	0.48	0.32	0.38	199
216	0.19	0.11	0.14	289
217	0.16	0.07	0.10	86
218	0.68	0.50	0.58	177
219	0.35	0.31	0.33	142
220	0.14	0.05	0.07	172
221	0.31	0.21	0.25	259
222	0.39	0.20	0.26	256
223	0.29	0.13	0.18	319
224	0.82	0.56	0.66	207
225	0.41	0.28	0.33	167
226	0.84	0.71	0.77	207
227	0.74	0.54	0.63	79
228	0.67	0.25	0.36	16
229	0.36	0.21	0.27	225
230	0.67	0.43	0.53	279
231	0.06	0.06	0.06	116
232	0.78	0.54	0.64	79
233	0.27	0.13	0.18	186
234	0.09	0.06	0.07	80
235	0.35	0.20	0.26	209
236	0.54	0.39	0.45	224
237	0.14	0.12	0.13	152
238	0.47	0.26	0.34	34

239	0.37	0.25	0.30	143
240	0.21	0.16	0.18	144
241	0.16	0.20	0.18	40
242	0.12	0.05	0.07	118
243	0.97	0.87	0.92	439
244	0.33	0.22	0.26	113
245	0.40	0.30	0.35	82
246	0.24	0.12	0.16	191
247	0.62	0.46	0.53	208
248	0.38	0.21	0.27	248
249	0.83	0.60	0.70	191
250	0.12	0.06	0.08	142
251	0.61	0.79	0.69	14
252	0.42	0.22	0.29	81
253	0.54	0.41	0.46	37
254	0.34	0.33	0.33	147
255	0.69	0.61	0.65	100
256	0.55	0.43	0.48	14
257	0.19	0.16	0.17	49
258	0.40	0.31	0.35	153
259	0.52	0.44	0.48	117
260	0.32	0.18	0.23	183
261	0.46	0.31	0.37	238
262	0.54	0.28	0.37	156
263	0.49	0.46	0.48	76
264	0.73	0.61	0.67	171
265	0.18	0.08	0.11	193
266	0.50	0.48	0.49	140
267	0.65	0.57	0.60	201
268	0.41	0.23	0.30	164
269	0.03	0.01	0.02	216
270	0.63	0.49	0.55	114
271	0.54	0.45	0.49	85
272	0.34	0.38	0.36	112
273	0.57	0.34	0.42	169
274	0.22	0.20	0.21	95
275	0.21	0.14	0.17	107
276	0.52	0.35	0.42	152
277	0.06	0.03	0.04	156
278	0.46	0.28	0.34	160
279	0.20	0.11	0.14	27
280	0.49	0.49	0.49	100
281	0.56	0.39	0.46	84
282	0.37	0.21	0.27	169
283	0.23	0.10	0.13	63
284	0.11	0.09	0.10	47
285	0.00	0.00	0.00	167
286	0.22	0.11	0.15	119

287	0.29	0.30	0.29	20
288	0.39	0.30	0.34	50
289	0.21	0.16	0.18	141
290	0.50	0.35	0.41	172
291	0.16	0.13	0.14	47
292	0.47	0.49	0.48	160
293	0.62	0.54	0.58	92
294	0.93	0.63	0.75	172
295	0.29	0.18	0.22	91
296	0.37	0.25	0.30	267
297	0.81	0.77	0.79	114
298	0.66	0.43	0.52	138
299	0.46	0.31	0.37	224
300	0.38	0.28	0.32	200
301	0.65	0.60	0.63	111
302	0.64	0.57	0.60	199
303	0.21	0.11	0.14	298
304	0.59	0.45	0.51	153
305	0.39	0.36	0.37	80
306	0.45	0.22	0.30	136
307	0.18	0.12	0.14	95
308	0.81	0.68	0.74	170
309	0.40	0.39	0.39	134
310	0.84	0.69	0.76	157
311	0.49	0.42	0.46	217
312	0.24	0.19	0.21	108
313	0.77	0.67	0.72	159
314	0.51	0.50	0.50	111
315	0.13	0.06	0.09	31
316	0.94	0.84	0.89	94
317	0.26	0.17	0.20	109
318	0.32	0.35	0.33	101
319	0.44	0.22	0.29	158
320	0.58	0.46	0.51	138
321	0.99	0.92	0.95	316
322	0.27	0.23	0.25	71
323	0.15	0.08	0.10	65
324	0.86	0.55	0.67	120
325	0.23	0.12	0.16	186
326	0.17	0.08	0.11	202
327	0.96	0.93	0.95	351
328	0.40	0.34	0.37	106
329	0.27	0.14	0.18	160
330	0.86	0.78	0.82	192
331	0.22	0.21	0.21	91
332	0.41	0.43	0.42	232
333	0.09	0.03	0.05	93
334	0.86	0.80	0.83	336

335	0.31	0.23	0.26	87
336	0.82	0.57	0.67	161
337	0.24	0.20	0.22	186
338	0.77	0.68	0.72	149
339	0.50	0.36	0.42	297
340	0.17	0.06	0.09	136
341	0.50	0.50	0.50	24
342	0.48	0.39	0.43	129
343	0.56	0.42	0.48	119
344	0.92	0.70	0.80	202
345	0.62	0.59	0.60	141
346	0.30	0.21	0.24	174
347	0.18	0.08	0.11	37
348	0.83	0.61	0.70	107
349	0.55	0.39	0.46	76
350	0.97	0.95	0.96	269
351	0.75	0.78	0.76	143
352	0.73	0.64	0.68	143
353	0.47	0.27	0.34	123
354	0.07	0.03	0.05	121
355	0.23	0.17	0.19	66
356	0.27	0.11	0.16	189
357	0.44	0.13	0.21	52
358	0.40	0.17	0.24	181
359	0.14	0.14	0.14	154
360	0.27	0.20	0.23	138
361	0.20	0.10	0.13	114
362	0.20	0.15	0.17	62
363	0.55	0.36	0.43	214
364	0.08	0.09	0.08	11
365	0.42	0.25	0.32	142
366	0.26	0.13	0.18	38
367	0.26	0.17	0.21	82
368	0.41	0.29	0.34	83
369	0.67	0.47	0.55	110
370	0.43	0.23	0.30	81
371	0.12	0.05	0.07	99
372	0.77	0.77	0.77	115
373	0.48	0.50	0.49	22
374	0.20	0.14	0.16	81
375	0.30	0.19	0.23	68
376	0.16	0.06	0.08	142
377	0.43	0.29	0.34	139
378	0.32	0.27	0.29	45
379	0.11	0.10	0.10	42
380	0.36	0.14	0.20	124
381	0.29	0.17	0.21	12
382	0.54	0.39	0.45	247

383	0.24	0.16	0.19	37
384	0.31	0.21	0.25	90
385	0.07	0.05	0.06	65
386	0.47	0.38	0.42	124
387	0.61	0.56	0.59	110
388	0.43	0.43	0.43	74
389	0.53	0.40	0.45	126
390	0.13	0.03	0.06	143
391	0.15	0.04	0.06	120
392	0.45	0.35	0.40	190
393	0.29	0.22	0.25	123
394	0.39	0.29	0.34	99
395	0.91	0.86	0.89	214
396	0.43	0.36	0.39	83
397	0.24	0.12	0.16	40
398	0.17	0.07	0.10	83
399	0.28	0.16	0.20	121
400	0.26	0.19	0.22	62
401	0.24	0.17	0.20	95
402	0.13	0.08	0.10	101
403	0.62	0.42	0.50	116
404	0.45	0.46	0.46	135
405	0.53	0.51	0.52	71
406	0.30	0.16	0.20	115
407	0.25	0.14	0.18	95
408	0.22	0.06	0.10	126
409	0.17	0.10	0.13	29
410	0.79	0.65	0.71	132
411	0.12	0.06	0.08	98
412	0.24	0.13	0.17	136
413	0.09	0.06	0.07	33
414	0.23	0.13	0.17	127
415	0.32	0.24	0.27	76
416	0.84	0.81	0.82	108
417	0.31	0.25	0.28	112
418	0.36	0.28	0.31	128
419	0.29	0.20	0.24	111
420	0.50	0.56	0.53	34
421	0.16	0.09	0.12	75
422	0.34	0.19	0.25	170
423	0.39	0.20	0.27	162
424	0.26	0.20	0.22	86
425	0.39	0.28	0.33	71
426	0.43	0.29	0.35	109
427	0.23	0.20	0.22	200
428	0.36	0.24	0.28	89
429	0.23	0.17	0.19	36
430	0.19	0.19	0.19	16

431	0.18	0.04	0.07	122
432	0.00	0.00	0.00	16
433	0.84	0.53	0.65	127
434	0.21	0.14	0.17	100
435	0.38	0.42	0.40	12
436	0.12	0.07	0.09	27
437	0.43	0.28	0.34	135
438	0.47	0.47	0.47	121
439	0.81	0.76	0.79	34
440	0.24	0.12	0.16	85
441	0.57	0.42	0.49	83
442	0.20	0.13	0.16	78
443	0.13	0.06	0.08	87
444	0.88	0.70	0.78	134
445	0.20	0.14	0.16	56
446	0.36	0.21	0.27	85
447	0.28	0.31	0.29	26
448	0.19	0.11	0.14	83
449	0.34	0.30	0.32	107
450	0.77	0.61	0.68	114
451	0.08	0.03	0.05	90
452	0.35	0.25	0.29	59
453	0.31	0.29	0.30	66
454	0.22	0.13	0.17	120
455	0.13	0.04	0.06	83
456	0.42	0.20	0.27	80
457	0.38	0.47	0.42	17
458	0.40	0.29	0.33	14
459	0.31	0.13	0.18	148
460	0.38	0.32	0.35	31
461	0.29	0.17	0.21	149
462	0.31	0.26	0.29	53
463	0.42	0.27	0.32	113
464	0.84	0.82	0.83	94
465	0.29	0.18	0.22	28
466	0.13	0.08	0.10	78
467	0.23	0.15	0.18	67
468	0.15	0.09	0.11	70
469	0.58	0.43	0.50	69
470	0.24	0.11	0.15	97
471	0.61	0.53	0.57	115
472	0.24	0.11	0.15	75
473	0.09	0.03	0.05	97
474	0.26	0.15	0.19	105
475	0.33	0.57	0.42	7
476	0.69	0.49	0.57	112
477	0.18	0.14	0.16	42
478	0.48	0.35	0.41	91

```
         479       0.03      0.01      0.02        74
         480       0.89      0.87      0.88       208
         481       0.22      0.18      0.20        73
         482       0.23      0.11      0.15       100
         483       0.13      0.12      0.13        84
         484       0.37      0.24      0.29        87
         485       0.67      0.41      0.51        54
         486       0.09      0.07      0.08        27
         487       0.00      0.00      0.00        48
         488       0.53      0.37      0.44        70
         489       0.56      0.49      0.52        88
         490       0.08      0.03      0.05        29
         491       0.54      0.50      0.52       115
         492       0.36      0.24      0.28       110
         493       0.66      0.47      0.55       119
         494       0.20      0.13      0.16        39
         495       0.55      0.35      0.43        85
         496       0.40      0.22      0.28       139
         497       0.28      0.32      0.30        34
         498       0.67      0.53      0.59       129
         499       0.00      0.00      0.00        33

   micro avg       0.56      0.41      0.47    179520
   macro avg       0.43      0.32      0.36    179520
weighted avg       0.54      0.41      0.46    179520
 samples avg       0.44      0.39      0.38    179520

Time taken to run this cell : 0:21:55.234731
```

5. Conclusions

[134]:
```
from prettytable import PrettyTable

model_metric = PrettyTable()

#Micro-Precision-> Micro-Pr, Micro-Recall-> Micro-Re, Macro-Precision->␣
↪Macro-Pr, Macro-Recall-> Macro-Re
model_metric = PrettyTable(["Model Name", 'Accuracy', 'Hamming loss',␣
↪'Micro-Pr', 'Micro-Re',
                            'Micro-F1','Macro-Pr', 'Macro-Re', 'Macro-F1'])

model_metric.add_row(["LR with OvsR- Hyperparam", 0.20191, 0.00328342, 0.5589,␣
↪0.4054, 0.4700, 0.4295, 0.3206, 0.3626])
model_metric.add_row(["Linear-SVM",0.07129,  0.00819042, 0.2112, 0.4685, 0.
↪2911, 0.1515, 0.3979, 0.2071])

print(model_metric.get_string(start=0, end=8))
```

Model Name	Accuracy	Hamming loss	Micro-Pr	Micro-Re	Micro-F1	Macro-Pr	Macro-Re	Macro-F1
LR with OvsR- Hyperparam	0.20191	0.00328342	0.5589	0.4054	0.47	0.4295	0.3206	0.3626
Linear-SVM	0.07129	0.00819042	0.2112	0.4685	0.2911	0.1515	0.3979	0.2071

1. As part of the problem statement, this is a multi-label classification problem and the performance metrics will be 'Micro f1 score' and 'Macro f1 score'.

2. After the EDA, the following conculsions were made:

   a) Only a few no. of tags appeared most times for the given data.
   b) First 500 tags covers almost 91.492% of question. Hence the for this case study 500 tags will be considered as Y.

3. For featurization we've used BoW vectorizer with the max no. of features as 200000.

4. Linear-SVM along with OneVsRest Classifier is applied as it suits the multi-label classification and high dimensional data

5. Logistic Regression with One vs Rest Classifier and Hyperprameter Tuning is performed on the data.

6. Logistic Regression with One vs Rest Classifier performed significanlty well compared to Linear-SVM.

7. Time complexity for Logistic Regression with One vs Rest Classifier and Hyperprameter Tuning is very very high. So, less no. of data points were used for the task.

[ ]:
```
```