



```
In [1]: 1 import warnings
2 warnings.filterwarnings("ignore")
3 import pandas as pd
4 import sqlite3
5 import csv
6 import matplotlib.pyplot as plt
7 import seaborn as sns
8 import numpy as np
9 from wordcloud import WordCloud
10 import re
11 import os
12 from sqlalchemy import create_engine # database connection
13 import datetime as dt
14 from nltk.corpus import stopwords
15 from nltk.tokenize import word_tokenize
16 from nltk.stem.snowball import SnowballStemmer
17 from sklearn.feature_extraction.text import CountVectorizer
18 from sklearn.feature_extraction.text import TfidfVectorizer
19 from sklearn.multiclass import OneVsRestClassifier
20 from sklearn.linear_model import SGDClassifier
21 from sklearn import metrics
22 from sklearn.metrics import f1_score, precision_score, recall_score
23 from sklearn import svm
24 from sklearn.linear_model import LogisticRegression
25 from sklearnlearn.adapt import mlknn
26 from sklearnlearn.problem_transform import ClassifierChain
27 from sklearnlearn.problem_transform import BinaryRelevance
28 from sklearnlearn.problem_transform import LabelPowerset
29 from sklearn.naive_bayes import GaussianNB
30 from datetime import datetime
```

```
In [2]: 1 # !pip install scikit-multilearn
2 # !python -c "import sklearnlearn"
```

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

2. Machine Learning problem

2.1 Data

2.1.1 Data Overview

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

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

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

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

Size of Train.csv - 6.75GB

Size of Test.csv - 2GB

Number of rows in Train.csv = 6034195

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

Data Field Explanation

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

Id - Unique identifier for each question

Title - The question's title

Body - The body of the question

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

2.1.2 Example Data point

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

Body :

```

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

```

\n\n

<p>The answer should come in the form of a table like</p>\n\n

<pre><code>

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

</code></pre>\n\n

<p>if the no of inputs is 3 and their ranges are\n

1,100\n

1,100\n

1,100\n

(could be varied too)</p>\n\n

<p>The output is not coming,can anyone correct the code or tell me what\'s wrong?</p>\n'

Tags : 'c++ c'

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

2.2.1 Type of Machine Learning Problem

It is a multi-label classification problem

Multi-label Classification: Multilabel classification assigns to each sample a set of target labels. This can be thought as predicting properties of a data-point that are not mutually exclusive, such as topics that are relevant for a document. A question on Stackoverflow might be about any of C, Pointers, FileIO and/or memory-management at the same time or none of these.

___Credit___: <http://scikit-learn.org/stable/modules/multiclass.html>

2.2.2 Performance metric

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

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

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

'Micro f1 score':

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

'Macro f1 score':

Calculate metrics for each label, and find their unweighted mean. This does not take label imbalance into account.

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

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

Hamming loss : The Hamming loss is the fraction of labels that are incorrectly predicted.

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

3. Exploratory Data Analysis

3.1 Data Loading and Cleaning

3.1.1 Using Pandas with SQLite to Load the data

```
In [3]: ▶ 1 #Creating db file from csv
2 #Learn SQL: https://www.w3schools.com/sql/default.asp
3 if not os.path.isfile('train.db'):
4     start = datetime.now()
5     disk_engine = create_engine('sqlite:///train.db')
6     start = dt.datetime.now()
7     chunksize = 180000
8     j = 0
9     index_start = 1
10    for df in pd.read_csv('Train.csv', names=['Id', 'Title', 'Body', 'Tags'], chunksize=chunksize, iterator=True, encoding='utf-8', ):
11        df.index += index_start
12        j+=1
13        print('{} rows'.format(j*chunksize))
14        df.to_sql('data', disk_engine, if_exists='append')
15        index_start = df.index[-1] + 1
16    print("Time taken to run this cell :", datetime.now() - start)
```

3.1.2 Counting the number of rows

```
In [4]: ▶ 1 if os.path.isfile('train.db'):
2     start = datetime.now()
3     con = sqlite3.connect('train.db')
4     num_rows = pd.read_sql_query("""SELECT count(*) FROM data""", con)
5     #Always remember to close the database
6     print("Number of rows in the database :", "\n", num_rows['count(*)'].values[0])
7     con.close()
8     print("Time taken to count the number of rows :", datetime.now() - start)
9 else:
10    print("Please download the train.db file from drive or run the above cell to generate train.db file")
```

Number of rows in the database :

6034196

Time taken to count the number of rows : 0:00:04.748885

3.1.3 Checking for duplicates

```
In [5]: 1 #Learn SQL: https://www.w3schools.com/sql/default.asp
2 if os.path.isfile('train.db'):
3     start = datetime.now()
4     con = sqlite3.connect('train.db')
5     df_no_dup = pd.read_sql_query('SELECT Title, Body, Tags, COUNT(*) as cnt_dup FROM data GROUP BY Title, Body, Tags', con)
6     con.close()
7     print("Time taken to run this cell :", datetime.now() - start)
8 else:
9     print("Please download the train.db file from drive or run the first to generate train.db file")
```

Time taken to run this cell : 0:07:46.581660

```
In [6]: 1 df_no_dup.head()
2 # we can observe that there are duplicates
```

Out[6]:

	Title	Body	Tags	cnt_dup
0	Implementing Boundary Value Analysis of S...	<pre><code>#include<iosstream>\n#include<...</code></pre>	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...	<p>I use the following code</p>\n\n<pre><code>...</code></pre>	java jdbc	2

```
In [7]: 1 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 [8]: 1 # number of times each question appeared in our database
2 df_no_dup.cnt_dup.value_counts()
```

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

```
In [9]: 1 df_no_dup.drop(df_no_dup[df_no_dup['Tags'].isna()].index,inplace=True)
```



```
In [10]: 1 start = datetime.now()
2
3 df_no_dup["tag_count"] = df_no_dup['Tags'].apply(lambda x : len(x.split(" ")))
4 # adding a new feature number of tags per question
5 print("Time taken to run this cell :", datetime.now() - start)
6 df_no_dup.head()
```

Time taken to run this cell : 0:00:05.900910

Out[10]:

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

```
In [11]: 1 # distribution of number of tags per question
2 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]: 1 #Creating a new database with no duplicates
2 if not os.path.isfile('train_no_dup.db'):
3     disk_dup = create_engine("sqlite:///train_no_dup.db")
4     no_dup = pd.DataFrame(df_no_dup, columns=['Title', 'Body', 'Tags'])
5     no_dup.to_sql('no_dup_train',disk_dup)
```

```
In [13]: 1 #This method seems more appropriate to work with this much data.
2 #creating the connection with database file.
3 if os.path.isfile('train_no_dup.db'):
4     start = datetime.now()
5     con = sqlite3.connect('train_no_dup.db')
6     tag_data = pd.read_sql_query("""SELECT Tags FROM no_dup_train""", con)
7     #Always remember to close the database
8     con.close()
9
10    # Let's now drop unwanted column.
11    tag_data.drop(tag_data.index[0], inplace=True)
12    #Printing first 5 columns from our data frame
13    tag_data.head()
14    print("Time taken to run this cell :", datetime.now() - start)
15 else:
16    print("Please download the train.db file from drive or run the above cells to generate train.db file")
```

Time taken to run this cell : 0:00:31.251432

3.2 Analysis of Tags

3.2.1 Total number of unique tags

```
In [14]: 1 # Importing & Initializing the "CountVectorizer" object, which
2 #is scikit-learn's bag of words tool.
3
4 #by default 'split()' will tokenize each tag using space.
5 vectorizer = CountVectorizer(tokenizer = lambda x: x.split())
6 # fit_transform() does two functions: First, it fits the model
7 # and learns the vocabulary; second, it transforms our training data
8 # into feature vectors. The input to fit_transform should be a list of strings.
9 tag_dtm = vectorizer.fit_transform(tag_data['Tags'])
```

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

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

```
In [16]: 1 #'get_feature_name()' gives us the vocabulary.
2 tags = vectorizer.get_feature_names()
3 #Lets look at the tags we have.
4 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]: 1 # https://stackoverflow.com/questions/15115765/how-to-access-sparse-matrix-elements
2 #Lets now store the document term matrix in a dictionary.
3 freqs = tag_dtm.sum(axis=0).A1
4 result = dict(zip(tags, freqs))
```

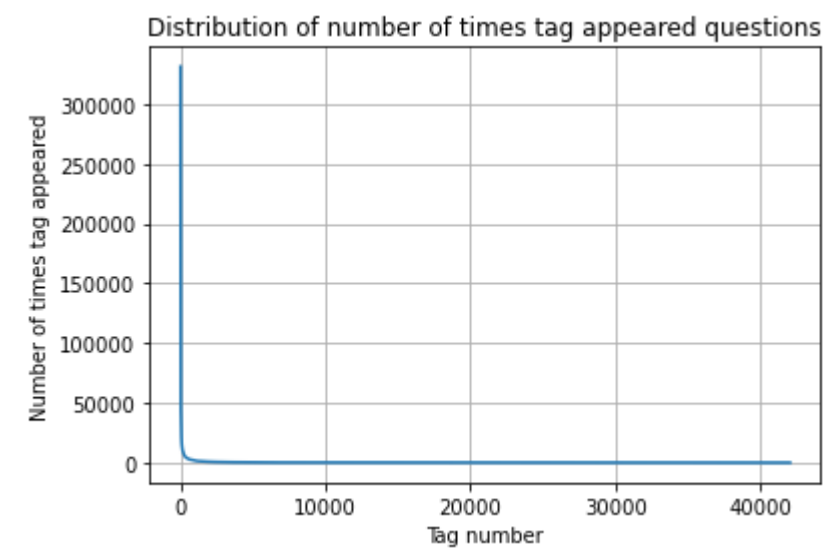
```
In [18]: 1 #Saving this dictionary to csv files.
2 if not os.path.isfile('tag_counts_dict_dtm.csv'):
3     with open('tag_counts_dict_dtm.csv', 'w') as csv_file:
4         writer = csv.writer(csv_file)
5         for key, value in result.items():
6             writer.writerow([key, value])
7 tag_df = pd.read_csv("tag_counts_dict_dtm.csv", names=['Tags', 'Counts'])
8 tag_df.head()
```

Out[18]:

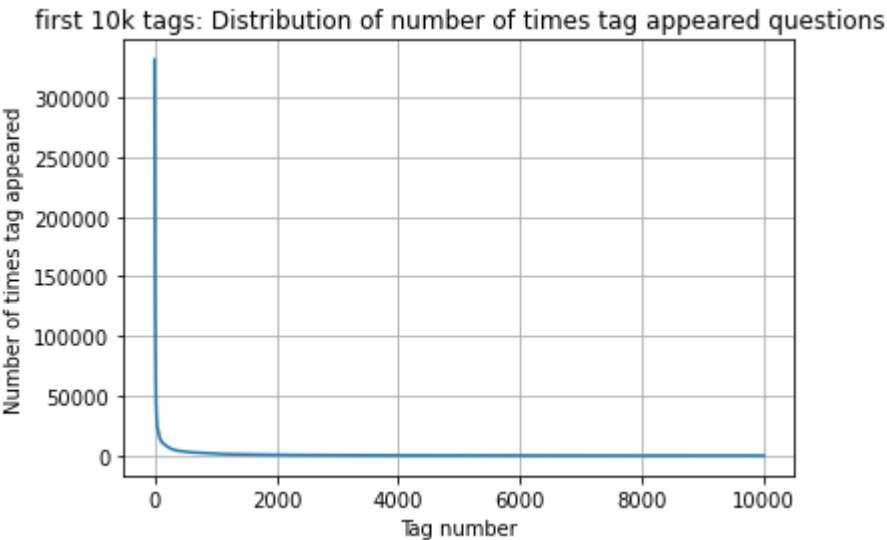
	Tags	Counts
0	.a	18
1	.app	37
2	.asp.net-mvc	1
3	.aspxauth	21
4	.bash-profile	138

```
In [19]: 1 tag_df_sorted = tag_df.sort_values(['Counts'], ascending=False)
2 tag_counts = tag_df_sorted['Counts'].values
```

```
In [20]: ▶ 1 plt.plot(tag_counts)
2 plt.title("Distribution of number of times tag appeared questions")
3 plt.grid()
4 plt.xlabel("Tag number")
5 plt.ylabel("Number of times tag appeared")
6 plt.show()
```



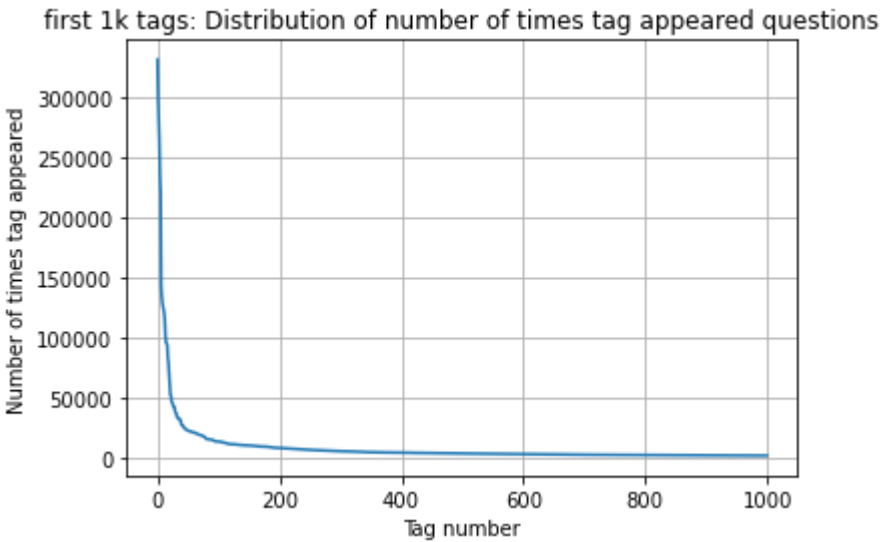
```
In [21]: ▶ 1 plt.plot(tag_counts[0:10000])
2 plt.title('first 10k tags: Distribution of number of times tag appeared questions')
3 plt.grid()
4 plt.xlabel("Tag number")
5 plt.ylabel("Number of times tag appeared")
6 plt.show()
7 print(len(tag_counts[0:10000:25]), tag_counts[0:10000:25])
```



400	[331505	44829	22429	17728	13364	11162	10029	9148	8054	7151
6466	5865	5370	4983	4526	4281	4144	3929	3750	3593	
3453	3299	3123	2989	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	

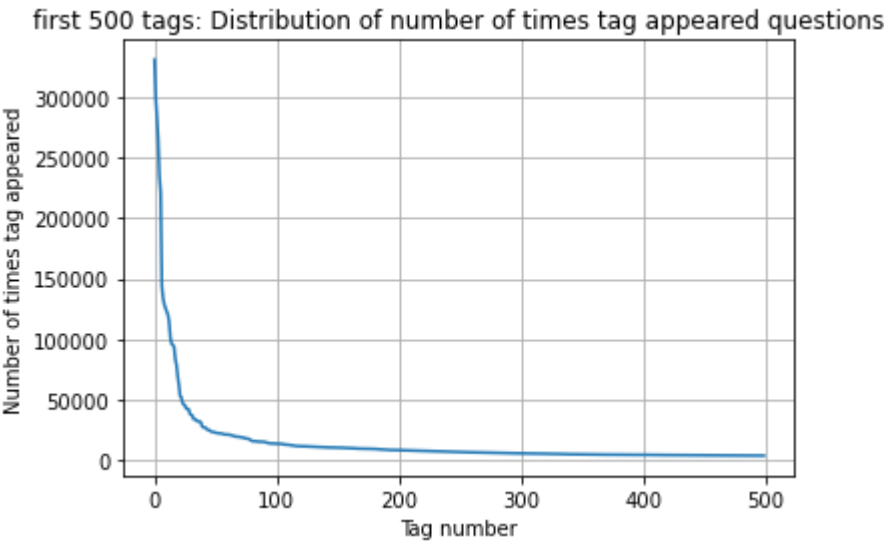
91	90	90	89	89	88	88	87	87	86
86	86	85	85	84	84	83	83	83	82
82	82	81	81	80	80	80	79	79	78
78	78	78	77	77	76	76	76	75	75
75	74	74	74	73	73	73	73	72	72]

```
In [22]: ▶ 1 plt.plot(tag_counts[0:1000])
2 plt.title('first 1k tags: Distribution of number of times tag appeared questions')
3 plt.grid()
4 plt.xlabel("Tag number")
5 plt.ylabel("Number of times tag appeared")
6 plt.show()
7 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	2989	2984	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]: ▶ 1 plt.plot(tag_counts[0:500])
2 plt.title('first 500 tags: Distribution of number of times tag appeared questions')
3 plt.grid()
4 plt.xlabel("Tag number")
5 plt.ylabel("Number of times tag appeared")
6 plt.show()
7 print(len(tag_counts[0:500:5]), tag_counts[0:500:5])
```

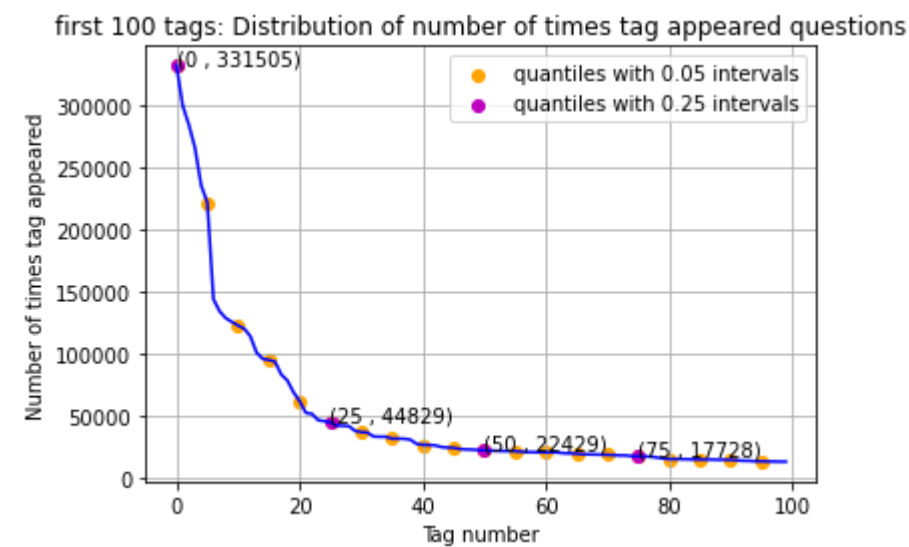


```
100 [331505 221533 122769 95160 62023 44829 37170 31897 26925 24537
22429 21820 20957 19758 18905 17728 15533 15097 14884 13703
13364 13157 12407 11658 11228 11162 10863 10600 10350 10224
10029 9884 9719 9411 9252 9148 9040 8617 8361 8163
8054 7867 7702 7564 7274 7151 7052 6847 6656 6553
6466 6291 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]: ▶ 1 plt.plot(tag_counts[0:100], c='b')
2 plt.scatter(x=list(range(0,100,5)), y=tag_counts[0:100:5], c='orange', label="quantiles with 0.05 intervals")
3 # quantiles with 0.25 difference
4 plt.scatter(x=list(range(0,100,25)), y=tag_counts[0:100:25], c='m', label = "quantiles with 0.25 intervals")
5
6 for x,y in zip(list(range(0,100,25)), tag_counts[0:100:25]):
7     plt.annotate(s="({} , {})".format(x,y), xy=(x,y), xytext=(x-0.05, y+500))
8
9 plt.title('first 100 tags: Distribution of number of times tag appeared questions')
10 plt.grid()
11 plt.xlabel("Tag number")
12 plt.ylabel("Number of times tag appeared")
13 plt.legend()
14 plt.show()
15 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 [25]: 1 # Store tags greater than 10K in one List
2 lst_tags_gt_10k = tag_df[tag_df.Counts>10000].Tags
3 #Print the length of the List
4 print ('{} Tags are used more than 10000 times'.format(len(lst_tags_gt_10k)))
5 # Store tags greater than 100K in one List
6 lst_tags_gt_100k = tag_df[tag_df.Counts>100000].Tags
7 #Print the length of the List.
8 print ('{} Tags are used more than 100000 times'.format(len(lst_tags_gt_100k)))
```

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

```
In [26]: 1 tag_df.iloc[tag_df.Counts.sort_values(ascending=False).index[0]]
```

```
Out[26]: Tags      c#
Counts    331505
Name: 4337, dtype: object
```

```
In [27]: 1 tag_df.iloc[tag_df.Counts.sort_values().index[0]]
```

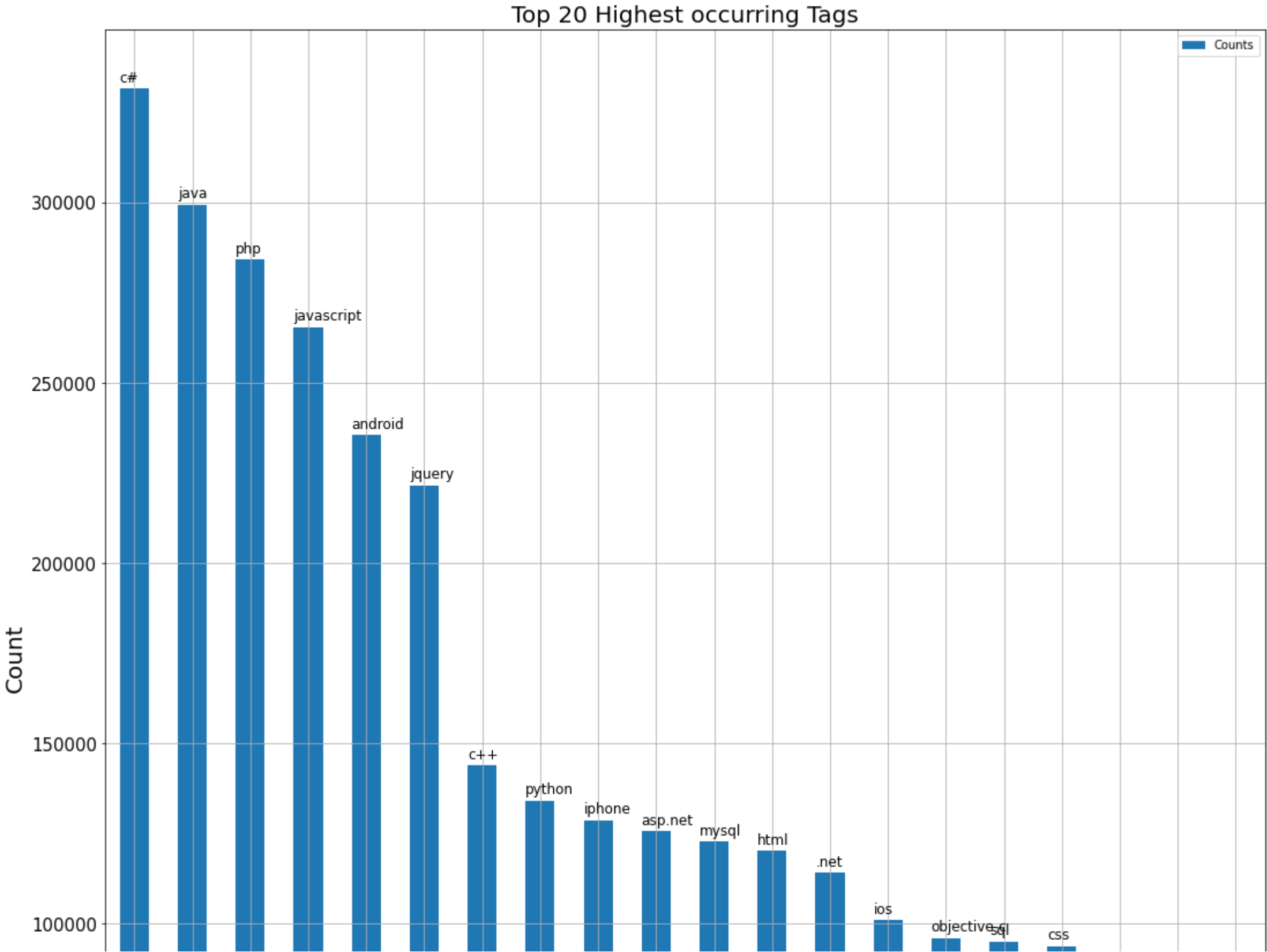
```
Out[27]: Tags      zzt-oop
Counts          1
Name: 42047, dtype: object
```

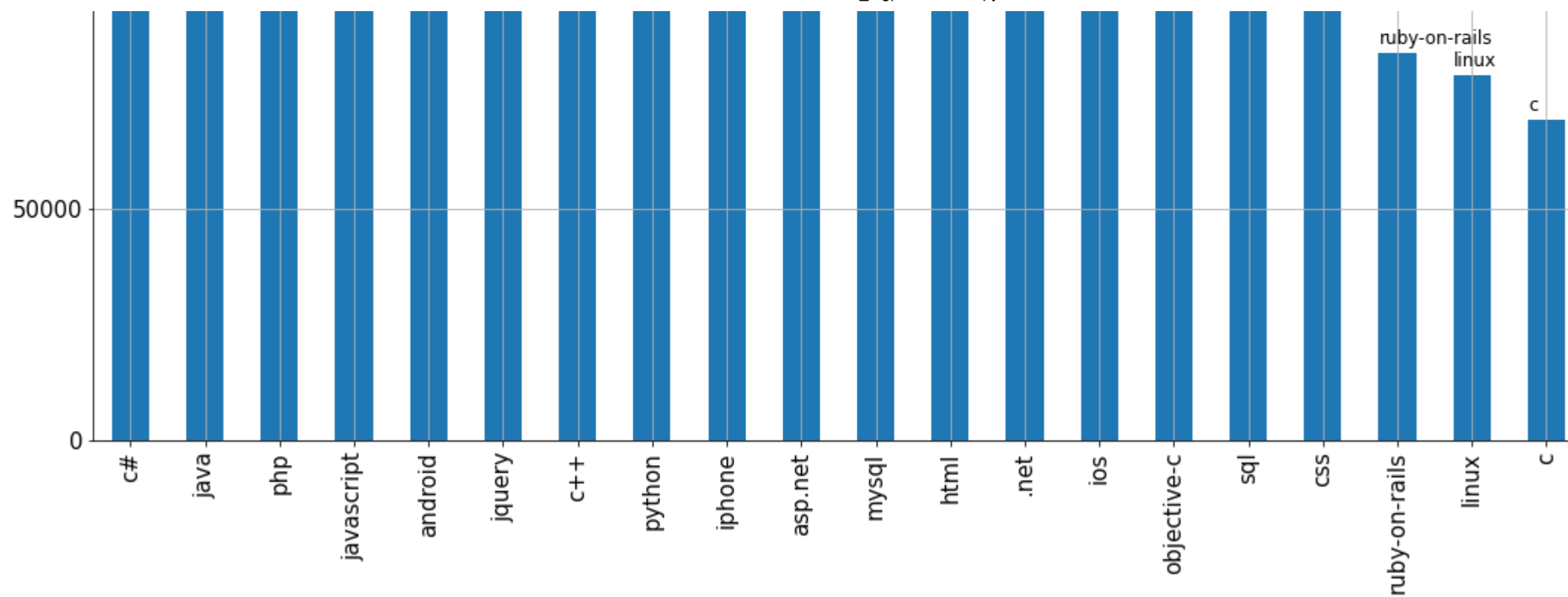
```
In [28]: 1 tag_df_sorted.head(10)['Tags'].reset_index(drop=True)
```

```
Out[28]: 0      c#
1      java
2      php
3  javascript
4      android
5      jquery
6      c++
7      python
8      iphone
9      asp.net
Name: Tags, dtype: object
```


In [29]:

```
1  ## Lets List out top 20 highest occuring tags
2
3  axes = tag_df_sorted.head(20).plot(x = 'Tags', y = 'Counts', kind = 'bar', figsize = (18, 20), fontsize = 15, grid = True,)
4  plt.xlabel("")
5  plt.ylabel("Count", fontsize = 20)
6  plt.title("Top 20 Highest occurring Tags", fontsize = 20)
7  cnt = 0
8  for i in axes.patches:
9      axes.text(i.get_x(), i.get_height()+2000, tag_df_sorted.head(20)['Tags'].reset_index(drop=True)[cnt], fontsize=12, color='black')
10     cnt +=1
```





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. Most least used tag (i.e. zzt-oop) is used 1 time ,possible to be an outlier
5. Since some tags occur much more frequently than others, Micro-averaged F1-score is the appropriate metric for this problem.

3.2.4 Tags Per Question

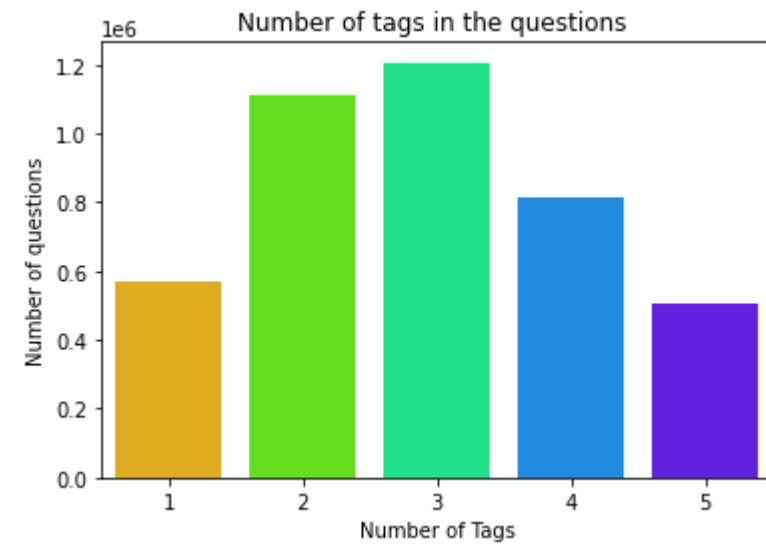
```
In [30]: 1 #Storing the count of tag in each question in List 'tag_count'
2 tag_quest_count = tag_dtm.sum(axis=1).tolist()
3 #Converting list of lists into single list, we will get [[3], [4], [2], [2], [3]] and we are converting this to [3, 4, 2, 2, 3]
4 tag_quest_count=[int(j) for i in tag_quest_count for j in i]
5 print ('We have total {} datapoints.'.format(len(tag_quest_count)))
6 print(tag_quest_count[:5])
```

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

```
In [31]: 1 print( "Maximum number of tags per question: %d"%max(tag_quest_count))
2 print( "Minimum number of tags per question: %d"%min(tag_quest_count))
3 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 [32]: 1 sns.countplot(tag_quest_count, palette='gist_rainbow')
2 plt.title("Number of tags in the questions ")
3 plt.xlabel("Number of Tags")
4 plt.ylabel("Number of questions")
5 plt.show()
```



Observations:

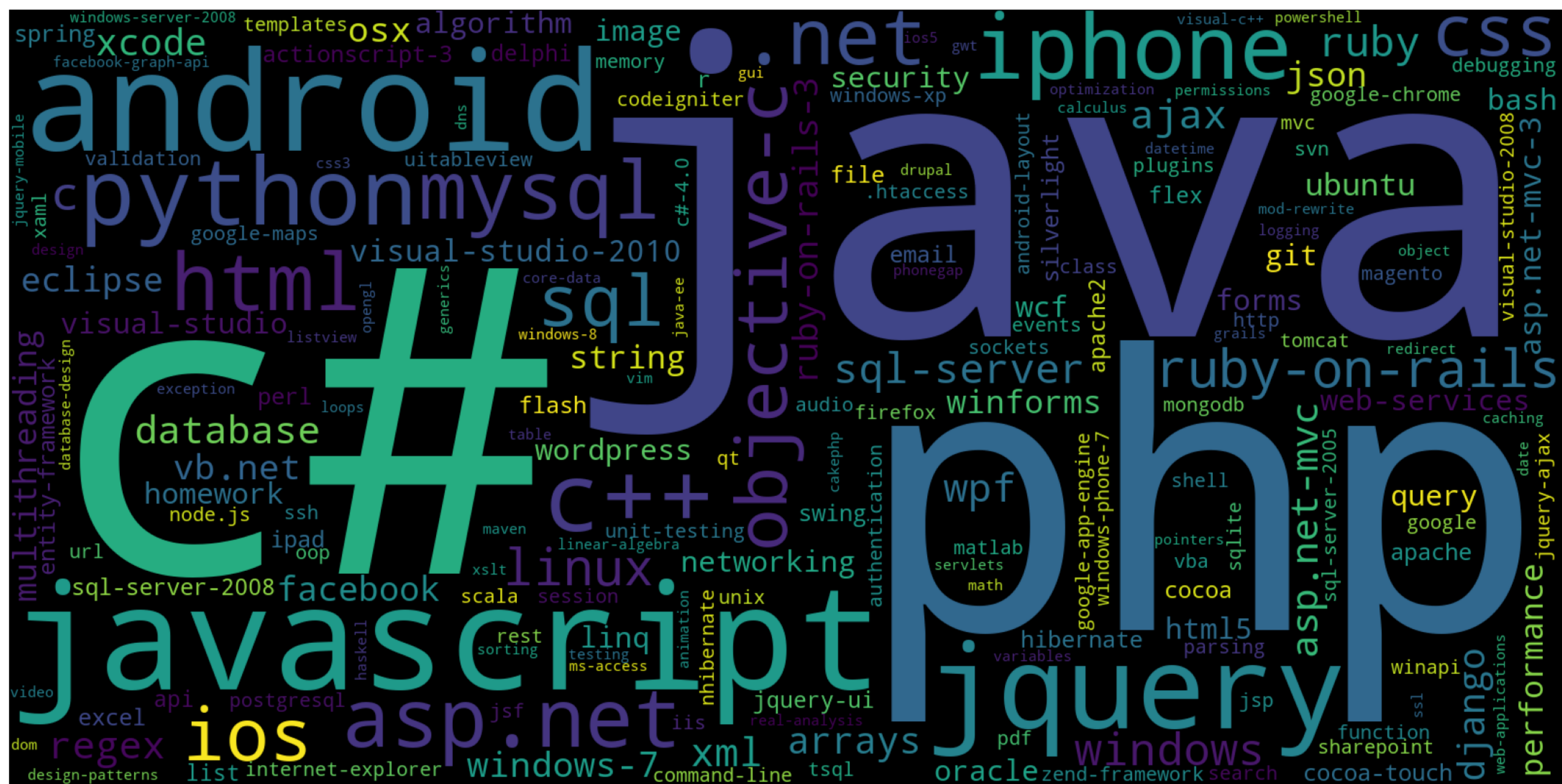
1. Maximum number of tags per question: 5
2. Minimum number of tags per question: 1
3. Avg. number of tags per question: 2.899
4. Most of the questions are having 2 or 3 tags

3.2.5 Most Frequent Tags

```

1  # Plotting word cloud
2  start = datetime.now()
3
4  # Lets first convert the 'result' dictionary to 'list of tuples'
5  tup = dict(result.items())
6  #Initializing WordCloud using frequencies of tags.
7  wordcloud = WordCloud(background_color='black',
8                          width=1600,
9                          height=800,
10                         ).generate_from_frequencies(tup)
11
12  fig = plt.figure(figsize=(30,20))
13  plt.imshow(wordcloud)
14  plt.axis('off')
15  plt.tight_layout(pad=0)
16  fig.savefig("tag.png")
17  plt.show()
18  print("Time taken to run this cell :", datetime.now() - start)

```



Time taken to run this cell : 0:00:04.587157

Observations:

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

Observations:

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

3.3 Cleaning and preprocessing of Questions

3.3.1 Preprocessing

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

```
In [34]: ► 1  '''We are going to create function to remove HTML tags'''
          2  def htmltag_remover(data) :
          3      pattern = re.compile("<.*?>")
          4      clean_text = re.sub(pattern," ",str(data))
          5      return clean_text
          6  stopwords = set(stopwords.words('english'))
          7  stemmer = SnowballStemmer("english")
```

```

In [35]: ▶ 1 #http://www.sqlitetutorial.net/sqlite-python/create-tables/
2 def create_connection(db_file):
3     """ create a database connection to the SQLite database
4         specified by db_file
5     :param db_file: database file
6     :return: Connection object or None
7     """
8     try:
9         conn = sqlite3.connect(db_file)
10        return conn
11    except Exception as e:
12        print(e)
13
14    return None
15
16 def create_table(conn,create_table_q,c):
17     """ create a table from the create_table_sql statement
18     param conn: Connection object
19     param create_table_sql: a CREATE TABLE statement
20     return: """
21     try:
22         c.execute(create_table_q)
23     except Exception as e:
24         print(e)
25
26
27 def checktable_exist(dbcon,c):
28     try:
29         q = '''select name from sqlite_master where type='table' '''
30         table_names = c.execute(q)
31         table_names = table_names.fetchall()
32         print("The tables in database are:")
33         print(table_names[0][0])
34         return len(table_names)
35     except Exception as e:
36         print(e)
37
38 def create_database_table(database, query):
39     conn = create_connection(database)
40     c = conn.cursor()
41     if conn is not None:
42         create_table(conn, query,c)
43         checktable_exist(conn,c)
44     else:
45         print("Error! cannot create the database connection.")
46     conn.close()
47
48 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);"""
49 create_database_table("Processed.db", sql_create_table)
50

```

The tables in database are:
QuestionsProcessed

```
In [36]: ▶ 1 ## Lets random sample of 1 million rows
2 start = datetime.now()
3 read_db = 'train_no_dup.db'
4 write_db = 'processed.db'
5 # Lets sample 1m rows
6 if os.path.isfile(read_db):
7     conn_r = create_connection(read_db)
8     if conn_r is not None:
9         r = conn_r.cursor()
10        r.execute('select Title,body,tags from no_dup_train order by Random() limit 1000000;')
11
12 ## Lets clear all the rows from the table Questions processed
13 if os.path.isfile(write_db):
14     conn_w = create_connection(write_db)
15     if conn_w is not None:
16         w = conn_w.cursor()
17         w.execute('' Delete from QuestionsProcessed where 1 '')
18         print(''Cleared all the rows!! '')
19 print("Time taken to run this cell :", datetime.now() - start)
```

Cleared all the rows!!

Time taken to run this cell : 0:10:44.641503

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

In [37]:

```

1 #http://www.bernzilla.com/2008/05/13/selecting-a-random-row-from-an-sqlite-table/
2 start = datetime.now()
3 preprocessed_data_list=[]
4 r.fetchone()
5 questions_with_code=0
6 len_pre=0
7 len_post=0
8 questions_proccesed = 0
9 for row in r:
10     is_code = 0
11     title, question, tags = row[0], row[1], row[2]
12
13     if '<code>' in question:
14         questions_with_code+=1
15         is_code = 1
16     x = len(question)+len(title)
17     len_pre+=x
18
19     code = str(re.findall(r'<code>(.*?)</code>', question, flags=re.DOTALL))
20
21     question=re.sub('<code>(.*?)</code>', '', question, flags=re.MULTILINE|re.DOTALL)
22     question=htmltag_remover(question.encode('utf-8'))
23
24     title=title.encode('utf-8')
25
26     question=str(title)+" "+str(question)
27     question=re.sub(r'^[A-Za-z]+', ' ', question)
28     words=word_tokenize(str(question.lower()))
29
30     #Removing all single letter and and stopwords from question exceptt for the letter 'c'
31     question=' '.join(str(stemmer.stem(j)) for j in words if j not in stopwords and (len(j)!=1 or j=='c'))
32
33     len_post+=len(question)
34     tup = (question,code,tags,x,len(question),is_code)
35     questions_proccesed += 1
36     w.execute("insert into QuestionsProccesed(question,code,tags,words_pre,words_post,is_code) values (?,?,?,?,?,?)",tup)
37     if (questions_proccesed%100000==0):
38         print("number of questions completed=",questions_proccesed)
39
40 no_dup_avg_len_pre=(len_pre*1.0)/questions_proccesed
41 no_dup_avg_len_post=(len_post*1.0)/questions_proccesed
42
43 print( "Avg. length of questions(Title+Body) before processing: %d"%no_dup_avg_len_pre)
44 print( "Avg. length of questions(Title+Body) after processing: %d"%no_dup_avg_len_post)
45 print( "Percent of questions containing code: %d"%((questions_with_code*100.0)/questions_proccesed))
46
47 print("Time taken to run this cell :", datetime.now() - start)

```

```

number of questions completed= 100000
number of questions completed= 200000
number of questions completed= 300000
number of questions completed= 400000
number of questions completed= 500000
number of questions completed= 600000
number of questions completed= 700000
number of questions completed= 800000
number of questions completed= 900000
Avg. length of questions(Title+Body) before processing: 1171
Avg. length of questions(Title+Body) after processing: 327

```


Percent of questions containing code: 57
 Time taken to run this cell : 0:26:55.884319

```
In [38]: 1 # import nltk
2 # nltk.download('averaged_perceptron_tagger')
3 # import nltk
4 # nltk.download('punkt')
```

```
In [39]: 1 # dont forget to close the connections, or else you will end up with locks
2 conn_r.commit()
3 conn_w.commit()
4 conn_r.close()
5 conn_w.close()
```

```
In [40]: 1 if os.path.isfile(write_db):
2     conn_r = create_connection(write_db)
3     if conn_r is not None:
4         reader = conn_r.cursor()
5         reader.execute("SELECT question From QuestionsProcessed LIMIT 10")
6         print("Questions after preprocessed")
7         print('='*100)
8         reader.fetchone()
9         for row in reader:
10            print(row)
11            print('-'*100)
12 conn_r.commit()
13 conn_r.close()
```

Questions after preprocessed

=====

```
('net use jit compil would like use direct net jit compil possibl mean would like thing like jit exe myfil msil myfil exe c code',)
```

```
-----
('set font style select text use custom typefac spannabl method want set style select text use custom typefac get error compil time code appli thank',)
```

```
-----
('downsid use wcf rout servic propos one team implement wcf rout servic simplifi configur servic architectur understand concept want make sure hidden gotcha bottleneck issu mov e forward use custom servic class error handler dispatch etc need make sure rout interfer get general feed back viabil complex',)
```

```
-----
('chang poco entiti framework templat inherit base class tri creat solut layer architectur self track entiti ni use entiti framework c poco entiti generat download extens onlin galleri ni separ context tt entiti tt class librari nmap entiti tt model need make class inherit base class creat anoth class librari unfortun know noth chang templat someone he lp problem continu thank',)
```

```
-----
('htaccess redirect url url like submit googl long time ago old work realli like kind url redirect anyon search googl redirect main url homepag rather page possibl htaccess tha nk',)
```

```
-----
('code write anyth text file c contest c grade code use code creat text file look like ad number averag',)
```

```
-----
('mail server limit connect port pc alon situat one pc network network behind singl public ip address connect mail server port send email use outlook perform success telnet tes t nprevious system could sudden one ni suspect limit host side abl confirm would like know possibl limit email way network one public ip isp come consum router share privat ip dhcp among pcs offic pcs thank help',)
```

```
-----
('show int frac frac frac operatornam artanh dt int ln ln frac sinh du would show explain int frac frac frac operatornam artanh dt int ln ln frac sinh du nand int frac frac fra c ln dv nare equival without evalu integr',)
```

```
-----
('c read jpeg file store imag read jpeg filesystem store system draw imag within c code',)
```

```
In [41]: 1 #Taking 1 Million entries to a dataframe.
2 write_db = 'Processed.db'
3 if os.path.isfile(write_db):
4     conn_r = create_connection(write_db)
5     if conn_r is not None:
6         preprocessed_data = pd.read_sql_query("""SELECT question, Tags FROM QuestionsProcessed""", conn_r)
7 conn_r.commit()
8 conn_r.close()
```

```
In [42]: 1 preprocessed_data.head()
```

Out[42]:

	question	tags
0	patent algorithm mean algorithm patent allow u...	algorithms
1	net use jit compil would like use direct net j...	.net compilation jit
2	set font style select text use custom typefac ...	java android edittext spannable
3	downsid use wcf rout servic propos one team im...	wcf .net-4.0 wcf-routing
4	chang poco entiti framework templat inherit ba...	entity-framework architecture c#-4.0 t4

```
In [43]: 1 print("number of data points in sample :", preprocessed_data.shape[0])
2 print("number of dimensions :", preprocessed_data.shape[1])
```

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

4. Machine Learning Models

4.1 Converting tags for multilabel problems

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

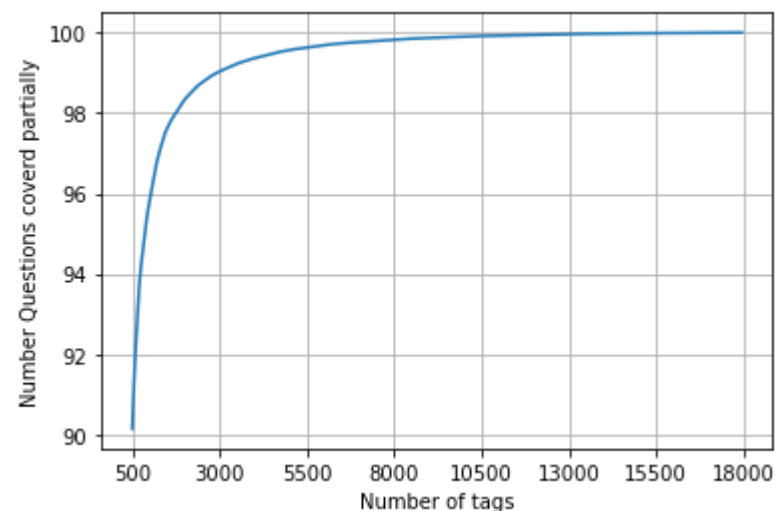
```
In [44]: 1 # binary='true' will give a binary vectorizer
2 vectorizer = CountVectorizer(tokenizer = lambda x: x.split(), binary='true')
3 multilabel_y = vectorizer.fit_transform(preprocessed_data['tags'])
```

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

```
In [45]: 1 def tags_to_choose(n):
2         t = multilabel_y.sum(axis=0).tolist()[0]
3         sorted_tags_i = sorted(range(len(t)), key=lambda i: t[i], reverse=True)
4         multilabel_yn=multilabel_y[:,sorted_tags_i[:n]]
5         return multilabel_yn
6
7 def questions_explained_fn(n):
8     multilabel_yn = tags_to_choose(n)
9     x= multilabel_yn.sum(axis=1)
10    return (np.count_nonzero(x==0))
```

```
In [46]: 1 questions_explained = []
2         total_tags=multilabel_y.shape[1]
3         total_qs=preprocessed_data.shape[0]
4         for i in range(500, total_tags, 100):
5             questions_explained.append(np.round(((total_qs-questions_explained_fn(i))/total_qs)*100,3))
```

```
In [47]: 1 fig, ax = plt.subplots()
2         ax.plot(questions_explained)
3         xlabel = list(500+np.array(range(-50,450,50))*50)
4         ax.set_xticklabels(xlabel)
5         plt.xlabel("Number of tags")
6         plt.ylabel("Number Questions covered partially")
7         plt.grid()
8         plt.show()
9         # you can choose any number of tags based on your computing power, minimum is 50(it covers 90% of the tags)
10        print("with ",5500,"tags we are covering ",questions_explained[50],"% of questions")
```



with 5500 tags we are covering 99.03 % of questions

```
In [48]: 1 multilabel_yx = tags_to_choose(5500)
2         print("number of questions that are not covered :", questions_explained_fn(5500),"out of ", total_qs)
```

number of questions that are not covered : 9701 out of 999999

```
In [49]: 1 print("Number of tags in sample :", multilabel_y.shape[1])
2         print("number of tags taken :", multilabel_yx.shape[1],"(",(multilabel_yx.shape[1]/multilabel_y.shape[1])*100,"%")")
```

Number of tags in sample : 35458
number of tags taken : 5500 (15.51130915449264 %)

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

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

```
In [50]: ▶ 1 total_size=preprocessed_data.shape[0]
2 train_size=int(0.80*total_size)
3
4 x_train=preprocessed_data.head(train_size)
5 x_test=preprocessed_data.tail(total_size - train_size)
6
7 y_train = multilabel_yx[0:train_size,:]
8 y_test = multilabel_yx[train_size:total_size,:]
```

```
In [51]: ▶ 1 print("Number of data points in train data :", y_train.shape)
2 print("Number of data points in test data :", y_test.shape)
```

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

4.3 Featurizing data

```
In [52]: ▶ 1 start = datetime.now()
2 vectorizer = TfidfVectorizer(min_df=0.00009, max_features=200000, smooth_idf=True, norm="l2", \
3                               tokenizer = lambda x: x.split(), sublinear_tf=False, ngram_range=(1,3))
4 x_train_multilabel = vectorizer.fit_transform(x_train['question'])
5 x_test_multilabel = vectorizer.transform(x_test['question'])
6 print("Time taken to run this cell :", datetime.now() - start)
```

Time taken to run this cell : 0:04:20.408655

```
In [53]: ▶ 1 print("Dimensions of train data X:",x_train_multilabel.shape, "Y :",y_train.shape)
2 print("Dimensions of test data X:",x_test_multilabel.shape,"Y:",y_test.shape)
```

Dimensions of train data X: (799999, 88046) Y : (799999, 5500)
Dimensions of test data X: (200000, 88046) Y: (200000, 5500)

```
In [54]: 1 # https://www.analyticsvidhya.com/blog/2017/08/introduction-to-multi-label-classification/
2 #https://stats.stackexchange.com/questions/117796/scikit-multi-label-classification
3 # classifier = LabelPowerset(GaussianNB())
4 """
5 from skmultilearn.adapt import MLkNN
6 classifier = MLkNN(k=21)
7
8 # train
9 classifier.fit(x_train_multilabel, y_train)
10
11 # predict
12 predictions = classifier.predict(x_test_multilabel)
13 print(accuracy_score(y_test,predictions))
14 print(metrics.f1_score(y_test, predictions, average = 'macro'))
15 print(metrics.f1_score(y_test, predictions, average = 'micro'))
16 print(metrics.hamming_loss(y_test,predictions))
17
18 """
19 # we are getting memory error because the multilearn package
20 # is trying to convert the data into dense matrix
21 # -----
22 #MemoryError                                Traceback (most recent call last)
23 #<ipython-input-170-f0e7c7f3e0be> in <module>()
24 #----> classifier.fit(x_train_multilabel, y_train)
```

```
Out[54]: "\nfrom skmultilearn.adapt import MLkNN\nnclassifier = MLkNN(k=21)\n\n# train\nnclassifier.fit(x_train_multilabel, y_train)\n\n# predict\nnpredictions = classifier.predict(x_test_multilabel)\n\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

```
In [55]: 1 # from sklearn.external import joblib
2 # joblib.dump(classifier, 'lr_with_equal_weight.pkl')
3 import joblib
4 import sys
5 sys.modules['sklearn.externals.joblib'] = joblib
```

```
In [56]: 1 # this will be taking so much time try not to run it, download the lr_with_equal_weight.pkl file and use to predict
2 # This takes about 6-7 hours to run.
3 if not os.path.isfile('lr_with_equal_weight.pkl'):
4     classifier = OneVsRestClassifier(SGDClassifier(loss='log', alpha=0.00001, penalty='l1'), n_jobs=-1)
5     classifier.fit(x_train_multilabel, y_train)
6     predictions = classifier.predict(x_test_multilabel)
7
8     print("accuracy :",metrics.accuracy_score(y_test,predictions))
9     print("macro f1 score :",metrics.f1_score(y_test, predictions, average = 'macro'))
10    print("micro f1 scoore :",metrics.f1_score(y_test, predictions, average = 'micro'))
11    print("hamming loss :",metrics.hamming_loss(y_test,predictions))
12    print("Precision recall report :\n",metrics.classification_report(y_test, predictions))
13    joblib.dump(classifier, 'lr_with_equal_weight.pkl')
14 else:
15     classifier = joblib.load('lr_with_equal_weight.pkl')
16
```

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

```
In [57]: 1 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);"""
2 create_database_table("Titlemoreweight.db", sql_create_table)
```

The tables in database are:
QuestionsProcessed

```
In [58]: 1 # http://www.sqlitetutorial.net/sqlite-delete/
2 # https://stackoverflow.com/questions/2279706/select-random-row-from-a-sqlite-table
3
4 read_db = 'train_no_dup.db'
5 write_db = 'Titlemoreweight.db'
6 train_datasize = 400000
7 if os.path.isfile(read_db):
8     conn_r = create_connection(read_db)
9     if conn_r is not None:
10         reader = conn_r.cursor()
11         # for selecting first 0.5M rows
12         reader.execute("SELECT Title, Body, Tags From no_dup_train LIMIT 500001;")
13         # for selecting random points
14         #reader.execute("SELECT Title, Body, Tags From no_dup_train ORDER BY RANDOM() LIMIT 500001;")
15
16 if os.path.isfile(write_db):
17     conn_w = create_connection(write_db)
18     if conn_w is not None:
19         writer = conn_w.cursor()
20         tables = checktable_exist(conn_w,writer)
21         if tables != 0:
22             writer.execute("DELETE FROM QuestionsProcessed WHERE 1")
23             print("Cleared All the rows")
24
```

The tables in database are:
QuestionsProcessed
Cleared All the rows

4.5.1 Preprocessing of questions

1. Separate Code from Body
2. Remove Special characters from Question title and description (not in code)
3. **Give more weightage to title : Add title three times to the question**

- Remove stop words (Except 'C')
- Remove HTML Tags
- Convert all the characters into small letters
- Use SnowballStemmer to stem the words

In [59]:

```

1 #http://www.bernzilla.com/2008/05/13/selecting-a-random-row-from-an-sqlite-table/
2 start = datetime.now()
3 preprocessed_data_list=[]
4 reader.fetchone()
5 questions_with_code=0
6 len_pre=0
7 len_post=0
8 questions_proccesed = 0
9 for row in reader:
10
11     is_code = 0
12
13     title, question, tags = row[0], row[1], str(row[2])
14
15     if '<code>' in question:
16         questions_with_code+=1
17         is_code = 1
18     x = len(question)+len(title)
19     len_pre+=x
20
21     code = str(re.findall(r'<code>(.*?)</code>', question, flags=re.DOTALL))
22
23     question=re.sub('<code>(.*?)</code>', '', question, flags=re.MULTILINE|re.DOTALL)
24     question=htmltag_remover(question.encode('utf-8'))
25
26     title=title.encode('utf-8')
27
28     # adding title three time to the data to increase its weight
29     # add tags string to the training data
30
31     ## here we are giving more weight to the title by repeating title in the question three times
32     question=str(title)+" "+str(title)+" "+str(title)+" "+question
33
34     # if questions_proccesed<=train_datasize:
35     #     question=str(title)+" "+str(title)+" "+str(title)+" "+question+" "+str(tags)
36     # else:
37     #     question=str(title)+" "+str(title)+" "+str(title)+" "+question
38
39     question=re.sub(r'^[A-Za-z0-9#+.\-]+', '', question)
40     words=word_tokenize(str(question.lower()))
41
42     #Removing all single letter and and stopwords from question exceptt for the letter 'c'
43     question=' '.join(str(stemmer.stem(j)) for j in words if j not in stopwords and (len(j)!=1 or j=='c'))
44
45     len_post+=len(question)
46     tup = (question,code,tags,x,len(question),is_code)
47     questions_proccesed += 1
48     writer.execute("insert into QuestionsProcessed(question,code,tags,words_pre,words_post,is_code) values (?,?,?,?,?,?)",tup)
49     if (questions_proccesed%100000==0):
50         print("number of questions completed=",questions_proccesed)
51
52 no_dup_avg_len_pre=(len_pre*1.0)/questions_proccesed
53 no_dup_avg_len_post=(len_post*1.0)/questions_proccesed
54
55 print( "Avg. length of questions(Title+Body) before processing: %d"%no_dup_avg_len_pre)
56 print( "Avg. length of questions(Title+Body) after processing: %d"%no_dup_avg_len_post)
57 print( "Percent of questions containing code: %d"%((questions_with_code*100.0)/questions_proccesed))
58
59 print("Time taken to run this cell :", datetime.now() - start)

```

```
number of questions completed= 100000
number of questions completed= 200000
number of questions completed= 300000
number of questions completed= 400000
number of questions completed= 500000
Avg. length of questions(Title+Body) before processing: 1239
Avg. length of questions(Title+Body) after processing: 424
Percent of questions containing code: 57
Time taken to run this cell : 0:16:52.881004
```

```
In [60]: ► 1 # never forget to close the conections or else we will end up with database locks
          2 conn_r.commit()
          3 conn_w.commit()
          4 conn_r.close()
          5 conn_w.close()
```


In [61]:

```

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

```

Questions after preprocessed

```

=====
('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 co
me column form come grid column although necessari bind nthank repli advanc ..',)
-----
('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 taglibraryval
id 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 o
dbc 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 met
hod 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 ad
d 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 su
re 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 e
xecut 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 sc
ript 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 foll
ow 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 ri
ght 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 i38
6 objc class skpsmtpmessag referenc error import framework send email applic background import framework i.e skpsmtpmessag somebody suggest get error collect2 ld return exit st
atus import framework correct sorc taken framework follow mfmcomposeviewcontrol question lock field updat answer drag drop folder project click copi nthat',)
-----

```

In [62]:

```

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

```

In [63]: ▶

1preprocessed_data.head()

Out[63]:

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

In [64]: ▶

1print("number of data points in sample :", preprocessed_data.shape[0])
2print("number of dimensions :", preprocessed_data.shape[1])

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

__ Converting string Tags to multilable output variables __

In [65]: ▶

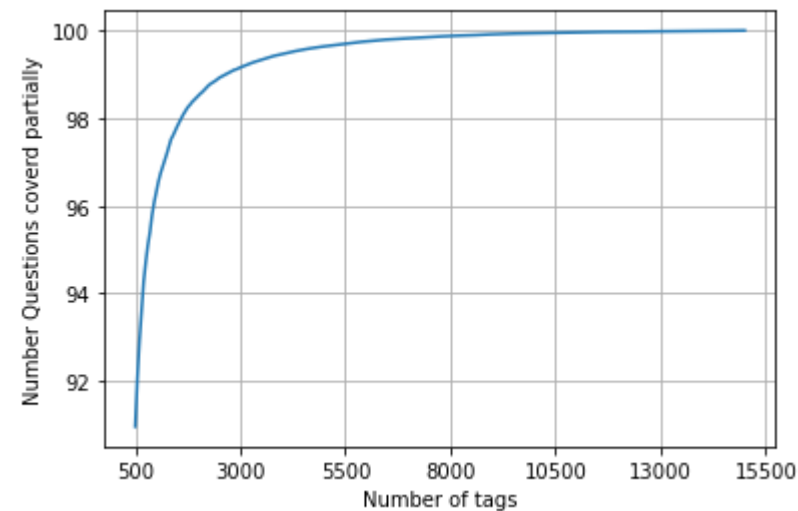
1vectorizer = CountVectorizer(tokenizer = lambda x: x.split(), binary='true')
2multilabel_y = vectorizer.fit_transform(preprocessed_data['tags'])

__ Selecting 500 Tags __

In [66]: ▶

1questions_explained = []
2total_tags=multilabel_y.shape[1]
3total_qs=preprocessed_data.shape[0]
4for i in range(500, total_tags, 100):
5 questions_explained.append(np.round(((total_qs-questions_explained_fn(i))/total_qs)*100,3))

```
In [67]: 1 fig, ax = plt.subplots()
2 ax.plot(questions_explained)
3 xlabel = list(500+np.array(range(-50,450,50))*50)
4 ax.set_xticklabels(xlabel)
5 plt.xlabel("Number of tags")
6 plt.ylabel("Number Questions covered partially")
7 plt.grid()
8 plt.show()
9 # you can choose any number of tags based on your computing power, minimum is 500(it covers 90% of the tags)
10 print("with ",5500,"tags we are covering ",questions_explained[50],"% of questions")
11 print("with ",500,"tags we are covering ",questions_explained[0],"% of questions")
```



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

```
In [68]: 1 # we will be taking 500 tags
2 multilabel_yx = tags_to_choose(500)
3 print("number of questions that are not covered :", questions_explained_fn(500),"out of ", total_qs)
```

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

```
In [69]: 1 x_train=preprocessed_data.head(train_datasize)
2 x_test=preprocessed_data.tail(preprocessed_data.shape[0] - 400000)
3
4 y_train = multilabel_yx[0:train_datasize,:]
5 y_test = multilabel_yx[train_datasize:preprocessed_data.shape[0],:]
```

```
In [70]: 1 print("Number of data points in train data :", y_train.shape)
2 print("Number of data points in test data :", y_test.shape)
```

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

4.5.2 Featurizing data with Tfidf vectorizer

```
In [71]: ▶ 1 start = datetime.now()
2 vectorizer = TfidfVectorizer(min_df=0.00009, max_features=200000, smooth_idf=True, norm="l2", \
3                             tokenizer = lambda x: x.split(), sublinear_tf=False, ngram_range=(1,3))
4 x_train_multilabel = vectorizer.fit_transform(x_train['question'])
5 x_test_multilabel = vectorizer.transform(x_test['question'])
6 print("Time taken to run this cell :", datetime.now() - start)
```

Time taken to run this cell : 0:01:51.807755

```
In [72]: ▶ 1 print("Dimensions of train data X:",x_train_multilabel.shape, "Y :",y_train.shape)
2 print("Dimensions of test data X:",x_test_multilabel.shape,"Y:",y_test.shape)
```

Dimensions of train data X: (400000, 95066) Y : (400000, 500)

Dimensions of test data X: (100000, 95066) Y: (100000, 500)

4.5.3 Applying Logistic Regression with OneVsRest Classifier

```

In [73]: ▶ 1 start = datetime.now()
2 if not os.path.isfile('lr_with_more_title_weight.pkl'):
3     classifier = OneVsRestClassifier(SGDClassifier(loss='log', alpha=0.00001, penalty='l1'), n_jobs=-1)
4     classifier.fit(x_train_multilabel, y_train)
5     predictions = classifier.predict (x_test_multilabel)
6
7     print("Accuracy :",metrics.accuracy_score(y_test, predictions))
8     print("Hamming loss ",metrics.hamming_loss(y_test,predictions))
9
10
11     precision = precision_score(y_test, predictions, average='micro')
12     recall = recall_score(y_test, predictions, average='micro')
13     f1 = f1_score(y_test, predictions, average='micro')
14
15     print("Micro-average quality numbers")
16     print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))
17
18     precision = precision_score(y_test, predictions, average='macro')
19     recall = recall_score(y_test, predictions, average='macro')
20     f1 = f1_score(y_test, predictions, average='macro')
21
22     print("Macro-average quality numbers")
23     print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))
24
25     print (metrics.classification_report(y_test, predictions))
26     joblib.dump(classifier, 'lr_with_more_title_weight.pkl')
27 else:
28     classifier =joblib.load('lr_with_more_title_weight.pkl')
29     predictions = classifier.predict (x_test_multilabel)
30
31     print("Accuracy :",metrics.accuracy_score(y_test, predictions))
32     print("Hamming loss ",metrics.hamming_loss(y_test,predictions))
33
34
35     precision = precision_score(y_test, predictions, average='micro')
36     recall = recall_score(y_test, predictions, average='micro')
37     f1 = f1_score(y_test, predictions, average='micro')
38
39     print("Micro-average quality numbers")
40     print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))
41
42     precision = precision_score(y_test, predictions, average='macro')
43     recall = recall_score(y_test, predictions, average='macro')
44     f1 = f1_score(y_test, predictions, average='macro')
45
46     print("Macro-average quality numbers")
47     print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))
48
49     print (metrics.classification_report(y_test, predictions))
50
51 print("Time taken to run this cell :", datetime.now() - start)

```

432	0.45	0.33	0.38	39
433	0.31	0.17	0.22	82
434	1.00	0.67	0.80	66
435	0.66	0.45	0.54	93
436	0.49	0.23	0.31	87
437	0.26	0.06	0.10	86
438	0.75	0.47	0.58	104
439	0.62	0.13	0.21	100
440	0.20	0.01	0.01	141

...	----	----	----	---
441	0.43	0.24	0.30	110
442	0.40	0.13	0.20	123
443	0.44	0.11	0.18	71
444	0.47	0.07	0.13	109
445	0.33	0.17	0.22	48
446	0.44	0.25	0.32	76
447	0.20	0.08	0.11	38
448	0.69	0.54	0.61	81
449	0.52	0.13	0.21	132
450	0.45	0.26	0.33	81
451	0.88	0.29	0.44	76

```

In [74]: ▶ 1 start = datetime.now()
2
3 # we are using solver as liblinear to avoid value error
4 if os.path.isfile('ovrclassifier2.pkl'):
5     classifier_2 = OneVsRestClassifier(LogisticRegression(penalty='l1', solver='liblinear'), n_jobs=-1)
6     classifier_2.fit(x_train_multilabel, y_train)
7     predictions_2 = classifier_2.predict(x_test_multilabel)
8     print("Accuracy :",metrics.accuracy_score(y_test, predictions_2))
9     print("Hamming loss ",metrics.hamming_loss(y_test,predictions_2))
10
11
12     precision = precision_score(y_test, predictions_2, average='micro')
13     recall = recall_score(y_test, predictions_2, average='micro')
14     f1 = f1_score(y_test, predictions_2, average='micro')
15
16     print("Micro-average quality numbers")
17     print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))
18
19     precision = precision_score(y_test, predictions_2, average='macro')
20     recall = recall_score(y_test, predictions_2, average='macro')
21     f1 = f1_score(y_test, predictions_2, average='macro')
22
23     print("Macro-average quality numbers")
24     print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))
25
26     print (metrics.classification_report(y_test, predictions_2))
27     joblib.dump(classifier_2, 'ovrclassifier2.pkl')
28 else:
29     classifier_2 = joblib.load("ovrclassifier2.pkl")
30     predictions_2 = classifier_2.predict(x_test_multilabel)
31     print("Accuracy :",metrics.accuracy_score(y_test, predictions_2))
32     print("Hamming loss ",metrics.hamming_loss(y_test,predictions_2))
33
34     precision = precision_score(y_test, predictions_2, average='micro')
35     recall = recall_score(y_test, predictions_2, average='micro')
36     f1 = f1_score(y_test, predictions_2, average='micro')
37
38     print("Micro-average quality numbers")
39     print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))
40
41     precision = precision_score(y_test, predictions_2, average='macro')
42     recall = recall_score(y_test, predictions_2, average='macro')
43     f1 = f1_score(y_test, predictions_2, average='macro')
44
45     print("Macro-average quality numbers")
46     print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))
47
48     print (metrics.classification_report(y_test, predictions_2))
49
50
51 print("Time taken to run this cell :", datetime.now() - start)

```

22	0.52	0.25	0.33	1513
23	0.89	0.55	0.68	1207
24	0.56	0.28	0.38	506
25	0.68	0.34	0.46	425
26	0.65	0.43	0.52	793
27	0.62	0.38	0.47	1291
28	0.74	0.39	0.51	1208
29	0.45	0.10	0.16	406
30	0.77	0.21	0.33	504

31	0.26	0.08	0.12	732
32	0.60	0.29	0.39	441
33	0.60	0.28	0.38	1645
34	0.69	0.26	0.38	1058
35	0.83	0.58	0.68	946
36	0.65	0.24	0.35	644
37	0.98	0.66	0.79	136
38	0.62	0.38	0.47	570
39	0.84	0.31	0.45	766
40	0.59	0.35	0.44	1132
41	0.47	0.18	0.26	174

4.6. Assignments

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

4.6.1 Randomly smapling 0.5 million rows

```
In [75]: ▶ 1  ## Lets random sample of 0.5 million rows
2  start = datetime.now()
3  read_db = 'train_no_dup.db'
4  write_db = 'processed.db'
5  # Lets sample 1m rows
6  if os.path.isfile(read_db):
7      conn_r = create_connection(read_db)
8      if conn_r is not None:
9          r = conn_r.cursor()
10         r.execute('select Title,body,tags from no_dup_train order by Random() limit 500001;')
11
12  ## Lets clear all the rows from the table Questions processed
13  if os.path.isfile(write_db):
14      conn_w = create_connection(write_db)
15      if conn_w is not None:
16          w = conn_w.cursor()
17          w.execute('' Delete from QuestionsProcessed where 1 '')
18          print(''Cleared all the rows!! '')
19  print("Time taken to run this cell :", datetime.now() - start)
```

Cleared all the rows!!
Time taken to run this cell : 0:05:25.458731

```
In [76]: ▶ 1  len(preprocessed_data)
2  #preprocessed_data.head(5)
```

Out[76]: 500000

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

4.6.2 Preprocessing the question

1. Separate Code from Body
2. Remove Spcial characters from Question title and description (not in code)

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

In [77]:

```

1  #http://www.bernzilla.com/2008/05/13/selecting-a-random-row-from-an-sqlite-table/
2
3  start = datetime.now()
4  preprocessed_data_list=[]
5  r.fetchone()
6  questions_with_code=0
7  len_pre=0
8  len_post=0
9  questions_proccesed = 0
10 for row in r:
11     is_code = 0
12     title, question, tags = row[0], row[1], row[2]
13
14     if '<code>' in question:
15         questions_with_code+=1
16         is_code = 1
17     x = len(question)+len(title)
18     len_pre+=x
19
20     code = str(re.findall(r'<code>(.*?)</code>', question, flags=re.DOTALL))
21
22     question=re.sub('<code>(.*?)</code>', '', question, flags=re.MULTILINE|re.DOTALL)
23     question=htmltag_remover(question.encode('utf-8'))
24
25     title=title.encode('utf-8')
26
27     question=str(title)+" "+str(question)
28     question=re.sub(r'^[A-Za-z]+', ' ', question)
29     words=word_tokenize(str(question.lower()))
30
31     #Removing all single letter and and stopwords from question exceptt for the letter 'c'
32     question=' '.join(str(stemmer.stem(j)) for j in words if j not in stopwords and (len(j)!=1 or j=='c'))
33
34     len_post+=len(question)
35     tup = (question,code,tags,x,len(question),is_code)
36     questions_proccesed += 1
37     w.execute("insert into QuestionsProcessed(question,code,tags,words_pre,words_post,is_code) values (?,?,,?,?,?)",tup)
38     if (questions_proccesed%100000==0):
39         print("number of questions completed=",questions_proccesed)
40
41 no_dup_avg_len_pre=(len_pre*1.0)/questions_proccesed
42 no_dup_avg_len_post=(len_post*1.0)/questions_proccesed
43
44 print( "Avg. length of questions(Title+Body) before processing: %d"%no_dup_avg_len_pre)
45 print( "Avg. length of questions(Title+Body) after processing: %d"%no_dup_avg_len_post)
46 print( "Percent of questions containing code: %d"%((questions_with_code*100.0)/questions_proccesed))
47
48 print("Time taken to run this cell :", datetime.now() - start)

```

```

number of questions completed= 100000
number of questions completed= 200000
number of questions completed= 300000
number of questions completed= 400000
number of questions completed= 500000
Avg. length of questions(Title+Body) before processing: 1172
Avg. length of questions(Title+Body) after processing: 327
Percent of questions containing code: 57
Time taken to run this cell : 0:11:05.593820

```

```
In [78]: 1 # never forget to close the conections or else we will end up with database Locks
2 conn_r.commit()
3 conn_w.commit()
4 conn_r.close()
5 conn_w.close()
```

```
In [79]: 1 if os.path.isfile(write_db):
2     conn_r = create_connection(write_db)
3     if conn_r is not None:
4         reader = conn_r.cursor()
5         reader.execute("SELECT question From QuestionsProcessed LIMIT 3")
6         print("Questions after preprocessed")
7         print('='*100)
8         reader.fetchone()
9         for row in reader:
10            print(row)
11            print('-'*100)
12 conn_r.commit()
13 conn_r.close()
```

Questions after preprocessed

```
=====
('someon explain follow js script work script work fine dont understand work espec follow section math floor math random somebodi pleas explain whole script work thank',)
-----
('print bind variabl java prepar statement suppos queri like conn connect object set countri code want see actual queri form like',)
-----
```

```
In [80]: 1 if os.path.isfile(write_db):
2     conn_r = create_connection(write_db)
3     if conn_r is not None:
4         preprocessed_data = pd.read_sql_query("""SELECT question, Tags FROM QuestionsProcessed""", conn_r)
5     conn_r.commit()
6     conn_r.close()
```

```
In [81]: 1 print("Number of sampled points: ",preprocessed_data.shape[0])
2 print("Number of dimensions: ",preprocessed_data.shape[1])
```

Number of sampled points: 500000
Number of dimensions: 2

```
In [82]: 1 ## creating a target
2 vect = CountVectorizer(tokenizer= lambda x :x.split() ,binary = True)
3 multilabel_y = vect.fit_transform(preprocessed_data['tags'])
```

```
In [83]: 1 ### choosing only 500 tags
2 multilabel_yx = tags_to_choose(500)
3 print(multilabel_yx.shape)
```

(500000, 500)

```
In [84]: 1 ### we already checked that using 500 tags we are covering 90% of our questions
```

4.6.3 Split data into train and test (80:20)

```
In [85]: 1 train_size = 400000
2 x_train = preprocessed_data.head(train_size)
3 x_test = preprocessed_data.tail(preprocessed_data.shape[0] - train_size)
4 y_train_multilabel = multilabel_yx[:train_size]
5 y_test_multilabel = multilabel_yx[train_size:preprocessed_data.shape[0]]
```

4.6.4 Vectorize using Bag of words upto 4 grams

```
In [86]: 1 # After preprocessing we are going to vectorize the question using bag of words
2 start = datetime.now()
3 bowvectorizer = CountVectorizer(min_df=0.00009, max_features=200000, tokenizer = lambda x: x.split(),
4                               binary='true', ngram_range=(1,4))
5 x_train_bow = bowvectorizer.fit_transform(x_train['question'])
6 x_test_bow = bowvectorizer.transform(x_test['question'])
7 print('shape of the train_data is', x_train_bow.shape)
8 print('shape of the test_data is', x_test_bow.shape)
9
10 print('Time taken to run cell :', datetime.now() - start)
```

shape of the train_data is (400000, 90390)
 shape of the test_data is (100000, 90390)
 Time taken to run cell : 0:02:48.717561

```
In [87]: 1 print("Dimensions of train data X:", x_train_bow.shape, "Y :", y_train.shape)
2 print("Dimensions of test data X:", x_test_bow.shape, "Y:", y_test.shape)
```

Dimensions of train data X: (400000, 90390) Y : (400000, 500)
 Dimensions of test data X: (100000, 90390) Y: (100000, 500)

4.6.5 Hyperparameter tuning using GridSearch (OvR)

```
In [94]: 1 ## let's initialize the alphas
2 start = datetime.now()
3 if not os.path.isfile('gs_bow.pkl'):
4     parameters = [{'estimator__C': [100, 10, 1, 0.1, 0.01, 0.001, 0.0001]}]
5     from sklearn.model_selection import GridSearchCV
6     logisticovr = OneVsRestClassifier(LogisticRegression())
7     gs_ovr = GridSearchCV(logisticovr, parameters, scoring='f1_micro', return_train_score=True, verbose=3, cv=2, n_jobs=-1)
8     gs_ovr.fit(x_train_bow, y_train_multilabel)
9     joblib.dump(gs_ovr, 'gs_bow.pkl')
10 else:
11     print("Hyperparameterized values are already present in the disk!!")
12     gs_ovr = joblib.load("gs_bow.pkl")
13 print('Time taken to run cell :', datetime.now() - start)
```

Hyperparameterized values are already present in the disk!!
 Time taken to run cell : 0:00:00.015956

```
In [95]: 1 print("the optimal C value after Grid Search: ", gs_ovr.best_estimator_)
```

the optimal C value after Grid Search: OneVsRestClassifier(estimator=LogisticRegression(C=1))

4.6.6 Training model with best C

```

In [97]: 1 if not os.path.isfile("bow_classifier.pkl"):
2         classifier = OneVsRestClassifier(LogisticRegression(C=1,penalty='l1', solver='liblinear'), n_jobs=-1)
3         classifier.fit(x_train_bow, y_train_multilabel)
4         predictions = classifier.predict(x_test_bow)
5         print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
6         print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))
7
8
9         precision = precision_score(y_test_multilabel, predictions, average='micro')
10        recall = recall_score(y_test_multilabel, predictions, average='micro')
11        f1 = f1_score(y_test_multilabel, predictions, average='micro')
12
13        print("Micro-average quality numbers")
14        print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))
15
16        precision = precision_score(y_test_multilabel, predictions, average='macro')
17        recall = recall_score(y_test_multilabel, predictions, average='macro')
18        f1 = f1_score(y_test_multilabel, predictions, average='macro')
19
20        print("Macro-average quality numbers")
21        print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))
22
23        print (metrics.classification_report(y_test_multilabel, predictions))
24        joblib.dump(classifier, 'bow_classifier.pkl')
25    else:
26        classifier = joblib.load("bow_classifier.pkl")
27        predictions = classifier.predict(x_test_bow)
28        print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
29        print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))
30
31
32        precision = precision_score(y_test_multilabel, predictions, average='micro')
33        recall = recall_score(y_test_multilabel, predictions, average='micro')
34        f1 = f1_score(y_test_multilabel, predictions, average='micro')
35
36        print("Micro-average quality numbers")
37        print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))
38
39        precision = precision_score(y_test_multilabel, predictions, average='macro')
40        recall = recall_score(y_test_multilabel, predictions, average='macro')
41        f1 = f1_score(y_test_multilabel, predictions, average='macro')
42
43        print("Macro-average quality numbers")
44        print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))
45
46        print (metrics.classification_report(y_test_multilabel, predictions))

```

38	0.35	0.26	0.30	700
39	0.30	0.14	0.19	695
40	0.34	0.18	0.24	624
41	0.53	0.41	0.46	623
42	0.60	0.38	0.47	628
43	0.53	0.41	0.46	630
44	0.44	0.24	0.31	632
45	0.32	0.16	0.21	557
46	0.25	0.10	0.14	551
47	0.34	0.20	0.25	552
48	0.50	0.23	0.32	550
49	0.54	0.46	0.50	535
50	0.33	0.12	0.17	524
51	0.68	0.50	0.58	537

52	0.51	0.31	0.38	532
53	0.38	0.19	0.25	535
54	0.52	0.34	0.41	545
55	0.78	0.76	0.77	528
56	0.71	0.60	0.65	511
57	0.86	0.78	0.82	541

Linear SVM : SGD classifier with hinge loss

4.6.6 Hyperparameter tuning using GridSearch (OvR) : SVM

```
In [98]: ► 1  ## let's initialize the alphas
2  start = datetime.now()
3  if not os.path.isfile('gs_bow_svm.pkl'):
4      parameters = [{'estimator__alpha': [100, 10, 1, 0.1, 0.01, 0.001, 0.0001]}]
5      SVM = OneVsRestClassifier(SGDClassifier(loss='hinge'))
6      gs_svm = GridSearchCV(SVM,parameters,scoring='f1_micro',return_train_score=True,verbose=3,cv=2)
7      gs_svm.fit(x_train_bow,y_train_multilabel)
8      joblib.dump(gs_svm,'gs_bow_svm.pkl')
9  else:
10     print("Hyperparameterized values are already present in the disk!!")
11     gs_svm = joblib.load("gs_bow_svm.pkl")
12     print('Time taken to run cell  :',datetime.now() - start)
```

Fitting 2 folds for each of 7 candidates, totalling 14 fits

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   4 out of  14 | elapsed: 31.1min remaining: 77.8min
[Parallel(n_jobs=-1)]: Done   9 out of  14 | elapsed: 91.1min remaining: 50.6min
[Parallel(n_jobs=-1)]: Done  14 out of  14 | elapsed: 93.1min remaining:    0.0s
[Parallel(n_jobs=-1)]: Done  14 out of  14 | elapsed: 93.1min finished
```

Time taken to run cell : 1:44:16.772311

```
In [12]: ► 1  print("the optimal alpha value after Grid Search: ",gs_svm.best_params_)
```

the optimal alpha value after Grid Search: {'estimator__alpha': 0.0001}

4.6.7 Training model with best Alpha

In [105]:

```

1  ### 4.6.6 Training model with best C
2
3  if not os.path.isfile("bow_classifier_svm.pkl"):
4      classifier_svm = OneVsRestClassifier(SGDClassifier(alpha=0.0001,penalty='l2'), n_jobs=-1)
5      classifier_svm.fit(x_train_bow, y_train_multilabel)
6      predictions = classifier_svm.predict(x_test_bow)
7      print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
8      print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))
9
10
11     precision = precision_score(y_test_multilabel, predictions, average='micro')
12     recall = recall_score(y_test_multilabel, predictions, average='micro')
13     f1 = f1_score(y_test_multilabel, predictions, average='micro')
14
15     print("Micro-average quality numbers")
16     print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))
17
18     precision = precision_score(y_test_multilabel, predictions, average='macro')
19     recall = recall_score(y_test_multilabel, predictions, average='macro')
20     f1 = f1_score(y_test_multilabel, predictions, average='macro')
21
22     print("Macro-average quality numbers")
23     print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))
24
25     print (metrics.classification_report(y_test_multilabel, predictions))
26     joblib.dump(classifier_svm, 'bow_classifier_svm.pkl')
27 else:
28     classifier_svm = joblib.load("bow_classifier_svm.pkl")
29     predictions = classifier_svm.predict(x_test_bow)
30     print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
31     print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))
32
33
34     precision = precision_score(y_test_multilabel, predictions, average='micro')
35     recall = recall_score(y_test_multilabel, predictions, average='micro')
36     f1 = f1_score(y_test_multilabel, predictions, average='micro')
37
38     print("Micro-average quality numbers")
39     print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))
40
41     precision = precision_score(y_test_multilabel, predictions, average='macro')
42     recall = recall_score(y_test_multilabel, predictions, average='macro')
43     f1 = f1_score(y_test_multilabel, predictions, average='macro')
44
45     print("Macro-average quality numbers")
46     print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))
47
48     print (metrics.classification_report(y_test_multilabel, predictions))

```

46	0.25	0.00	0.00	551
47	0.49	0.10	0.16	552
48	0.63	0.10	0.18	550
49	0.58	0.43	0.49	535
50	0.54	0.01	0.03	524
51	0.72	0.51	0.60	537
52	0.55	0.18	0.27	532
53	0.49	0.06	0.11	535
54	0.60	0.22	0.32	545
55	0.77	0.81	0.79	528
56	0.71	0.66	0.68	511
57	0.87	0.83	0.85	541

58	0.35	0.04	0.08	470
59	0.74	0.60	0.66	483
60	0.32	0.04	0.08	496
61	0.40	0.00	0.01	499
62	0.88	0.31	0.45	467
63	0.29	0.03	0.05	490
64	0.85	0.82	0.84	525

```
In [18]: 1 from prettytable import PrettyTable as pt
2 x = pt()
3 print("Model results for sampled data of size 0.5M")
4 x.field_names = ["Model Name", "Vectorizer", "Accuracy", "Hamming loss", "Precision", "Recall", "Micro-F1"]
5 x.add_row(['SGD (OvR)', 'TF-IDF', 0.23673, 0.00278, 0.7212, 0.3262, 0.4493])
6 x.add_row(['LogisticRegression (OvR)', 'TF-IDF', 0.25147, 0.00270, 0.7173, 0.3674, 0.4859])
7 x.add_row(['Logistic Regression(OvR)', 'BOW', 0.21906, 0.00307, 0.6097, 0.4166, 0.4950])
8 x.add_row(['Linear SVM(OvR)', 'BOW', 0.24303, 0.0028, 0.7286, 0.3646, 0.4860])
9 print(x)
```

***** Model results for sampled data of size 0.5M *****

Model Name	Vectorizer	Accuracy	Hamming loss	Precision	Recall	Micro-F1
SGD (OvR)	TF-IDF	0.23673	0.00278	0.7212	0.3262	0.4493
LogisticRegression (OvR)	TF-IDF	0.25147	0.0027	0.7173	0.3674	0.4859
Logistic Regression(OvR)	BOW	0.21906	0.00307	0.6097	0.4166	0.495
Linear SVM(OvR)	BOW	0.24303	0.0028	0.7286	0.3646	0.486

we can observe that Logistic Regression with BagOfWords as the vectorizer gives highest F1 micro average of 0.4950