Stack Overflow: Tag Prediction

Business Problem

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

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)

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.

Machine Learning problem

Data

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 Explaination

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

Id - Unique identifier for each question

Title - The question's title

Body - The body of the question

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

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


\n\n
The answer should come in the form of a table like
\n\n


           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


\n\n


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)
\n\n


    The output is not coming,can anyone correct the code or tell me what\'s wrong?
    \n'
    Tags : 'c++ c'
```

Mapping the real-world problem to a Machine Learning Problem

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

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)

Exploratory Data Analysis

Data Loading and Cleaning

Using Pandas with SQLite to Load the data

In [0]:
```
#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)
```

Counting the number of rows

In [0]:
```
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 : 0:01:15.750352
```

Checking for duplicates

In [0]:
```
#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 : 0:04:33.560122
```

In [0]:
```
df_no_dup.head()
# we can observe that there are duplicates
```

Out[6]:

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

In [0]:
```
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 ( 30.2920389063 % )
```

In [0]:
```
# number of times each question appeared in our database
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 [0]:
```
start = datetime.now()
df_no_dup["tag_count"] = df_no_dup["Tags"].apply(lambda text: len(text.split(" ")))
# adding a new feature number of tags per question
print("Time taken to run this cell :", datetime.now() - start)
df_no_dup.head()
```

```
Time taken to run this cell : 0:00:03.169523
```

Out[9]:

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

In [0]:
```
# distribution of number of tags per question
df_no_dup.tag_count.value_counts()
```

Out[10]:
```
3    1206157
2    1111706
4     814996
1     568298
5     505158
Name: tag_count, dtype: int64
```

In [0]:
```
#Creating a new database with no duplicates
if not os.path.isfile('train_no_dup.db'):
    disk_dup = create_engine("sqlite:///train_no_dup.db")
    no_dup = pd.DataFrame(df_no_dup, columns=['Title', 'Body', 'Tags'])
    no_dup.to_sql('no_dup_train',disk_dup)
```

In [0]:
```
#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)
    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 : 0:07:59.486007
```

In [0]:
```
tag_data.head()
```

Out[8]:

	Tags
1	c# silverlight data-binding
2	c# silverlight data-binding columns
3	jsp jstl
4	java jdbc
5	facebook api facebook-php-sdk

Analysis of Tags

Total number of unique tags

In [0]:
```
# Importing & Initializing the "CountVectorizer" object, which
#is scikit-learn's bag of words tool.

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

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

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

In [0]:
```
#'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']
```

Number of times a tag appeared

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

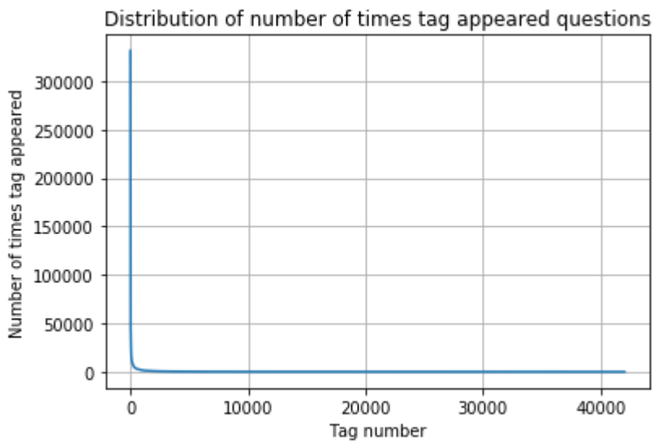
In [0]:
```
#Saving this dictionary to csv files.
if not os.path.isfile('tag_counts_dict_dtm.csv'):
    with open('tag_counts_dict_dtm.csv', 'w') as csv_file:
        writer = csv.writer(csv_file)
        for key, value in result.items():
            writer.writerow([key, value])
tag_df = pd.read_csv("tag_counts_dict_dtm.csv", names=['Tags', 'Counts'])
tag_df.head()
```

Out[17]:

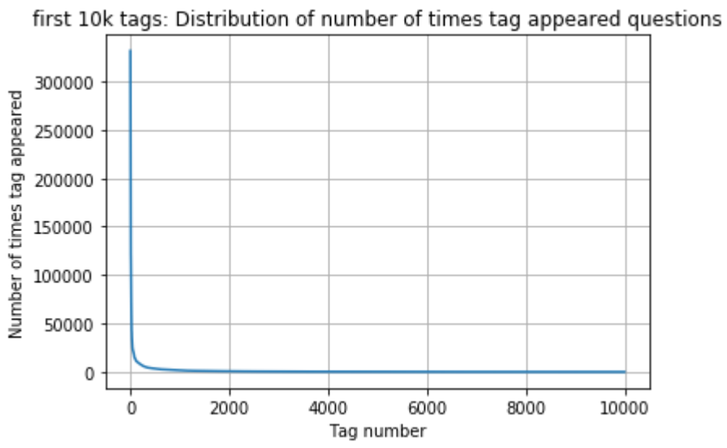
	Tags	Counts
0	.a	18
1	.app	37
2	.asp.net-mvc	1
3	.aspxauth	21
4	.bash-profile	138

In [0]:
```
tag_df_sorted = tag_df.sort_values(['Counts'], ascending=False)
tag_counts = tag_df_sorted['Counts'].values
```

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

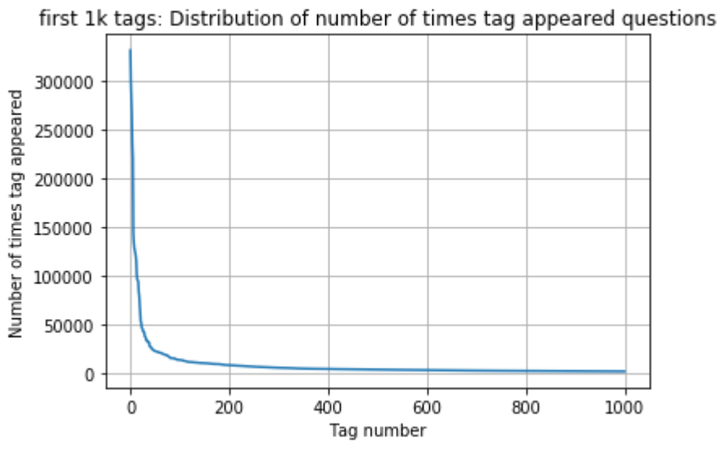


In [0]:
```
plt.plot(tag_counts[0:10000])
plt.title('first 10k tags: Distribution of number of times tag appeared questions')
plt.grid()
plt.xlabel("Tag number")
plt.ylabel("Number of times tag appeared")
plt.show()
print(len(tag_counts[0:10000:25]), tag_counts[0:10000:25])
```



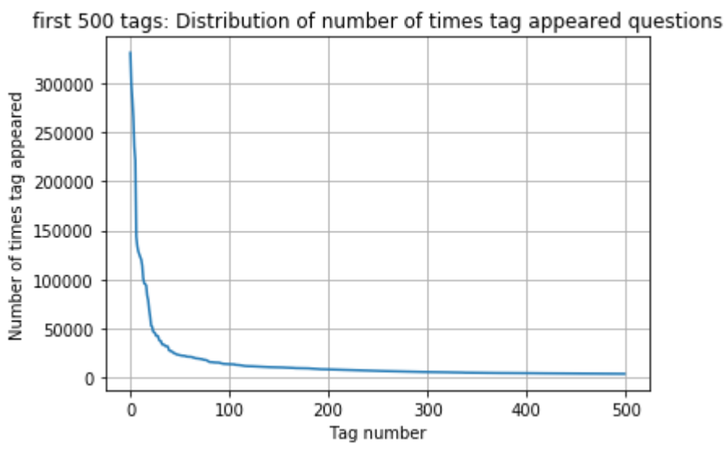

```
400 [331505  44829  22429  17728  13364  11162  10029   9148   8054   7151
   6466   5865   5370   4983   4526   4281   4144   3929   3750   3593
   3453   3299   3123   2989   2891   2738   2647   2527   2431   2331
   2259   2186   2097   2020   1959   1900   1828   1770   1723   1673
   1631   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 [0]:
```
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   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 [0]:
```
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])
```

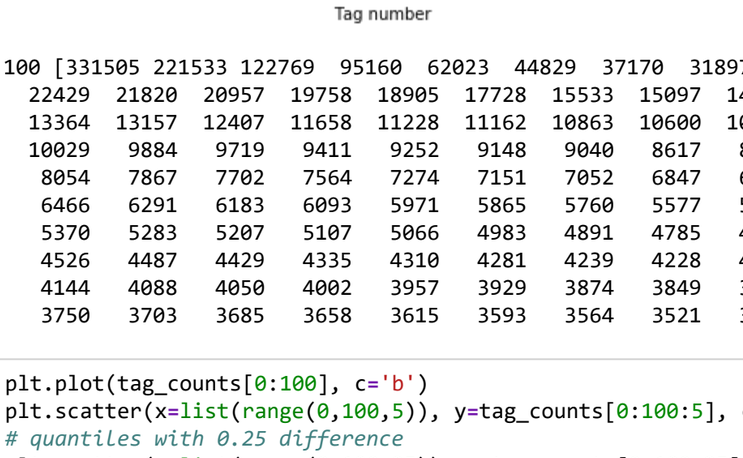



```
100 [331505 221533 122769  95160  62023  44829  37170  31897  26925  24537
  22429  21820  20957  19758  18905  17728  15533  15097  14884  13703
  13364  13157  12407  11658  11228  11162  10863  10600  10350  10224
  10029   9884   9719   9411   9252   9148   9040   8617   8361   8163
   8054   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]
```

In [0]:
```
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]
```

In [0]:
```
# 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.

Tags Per Question

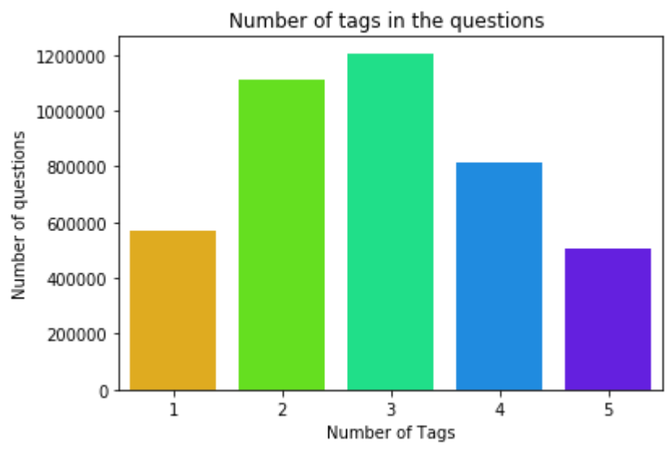
In [0]:
```
#Storing the count of tag in each question in list 'tag_count'
tag_quest_count = tag_dtm.sum(axis=1).tolist()
#Converting each value in the 'tag_quest_count' to integer.
tag_quest_count=[int(j) for i in tag_quest_count for j in i]
print ('We have total {} datapoints.'.format(len(tag_quest_count)))
print(tag_quest_count[:5])
```

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

In [0]:
```
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: 2.899440
```

In [0]:
```
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:

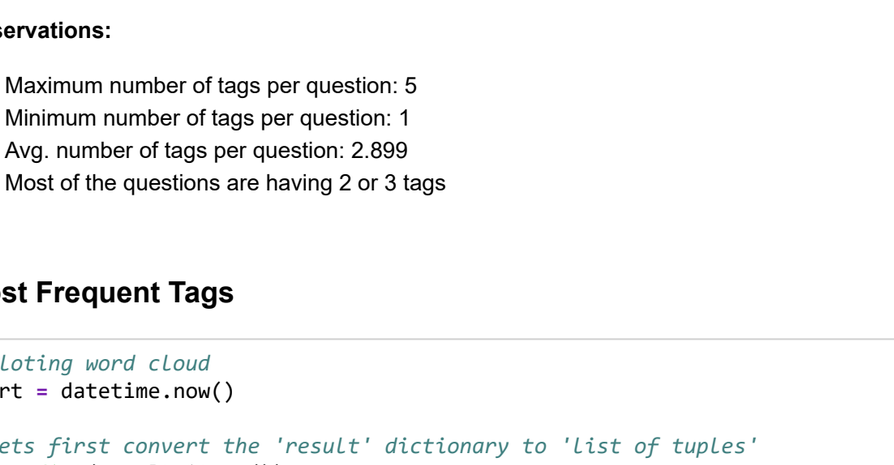
1. Maximum number of tags per question: 5
2. Minimum number of tags per question: 1
3. Avg. number of tags per question: 2.899
4. Most of the questions are having 2 or 3 tags

Most Frequent Tags

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

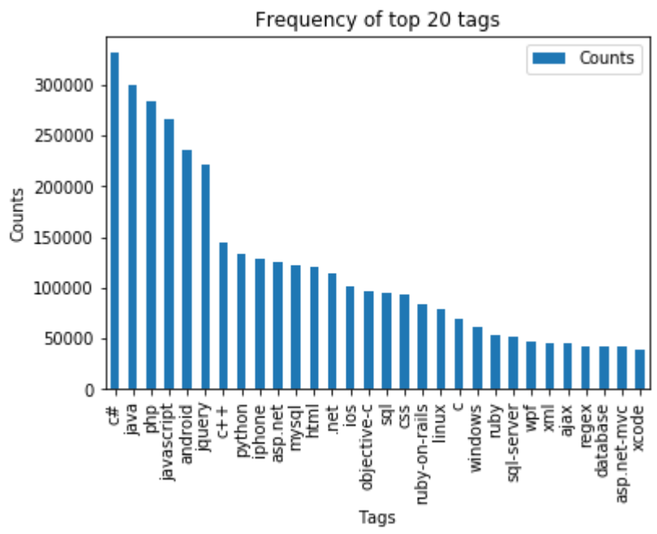



```
Time taken to run this cell : 0:00:03.232694
```

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

The top 20 tags

In [0]:
```
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.

Cleaning and preprocessing of Questions

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 [0]:
```
def striphtml(data):
    cleanr = re.compile('<.*?>')
    cleantext = re.sub(cleanr, ' ', str(data))
    return cleantext
stop_words = set(stopwords.words('english'))
stemmer = SnowballStemmer("english")
```

In [0]:
```
#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
```

In [0]:
```
# 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)
```

```
Tables in the databse:
QuestionsProcessed
Cleared All the rows
Time taken to run this cell : 0:06:32.806567
```

Observation: We create a new data base to store the sampled and preprocessed questions.

In [0]:
```
#http://www.bernzilla.com/2008/05/13/selecting-a-random-row-from-an-sqlite-table/

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

    is_code = 0

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

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

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

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

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

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

    #Removing all single letter and and stopwords from question exceptt for the 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
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: 1169
Avg. length of questions(Title+Body) after processing: 327
Percent of questions containing code: 57
Time taken to run this cell : 0:47:05.946582
```

In [0]:
```
# dont forget to close the connections, or else you will end up with locks
conn_r.commit()
conn_w.commit()
conn_r.close()
conn_w.close()
```

In [0]:
```
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
====================================================================================================
('ef code first defin one mani relationship differ key troubl defin one zero mani relationship entiti ef object model look like use fluent api object composit pk defin batch id bat
ch detail id use fluent api object composit pk defin batch detail id compani id map exist databas tpt basic idea submittedtransact zero mani submittedsplittransact associ navig rea
lli need one way submittedtransact submittedsplittransact need dbcontext class onmodelcr overrid map class lazi load occur submittedtransact submittedsplittransact help would much
appreci edit taken advic made follow chang dbcontext class ad follow onmodelcr overrid must miss someth get follow except thrown submittedtransact key batch id batch detail id zero
one mani submittedsplittransact key batch detail id compani id rather assum convent creat relationship two object configur requir sinc obvious wrong',)
----------------------------------------------------------------------------------------------------
('explan new statement review section c code came accross statement block come accross new oper use way someon explain new call way',)
----------------------------------------------------------------------------------------------------
('error function notat function solv logic riddl iloczyni list structur list possibl candid solut list possibl coordin matrix wan na choos one candid compar possibl candid element
equal wan na delet coordin call function skasuj look like ni knowledg haskel cant see what wrong',)
----------------------------------------------------------------------------------------------------
('step plan move one isp anoth one work busi plan switch isp realli soon need chang lot inform dns wan wan wifi question guy help mayb peopl plan correct chang current isp new one
first dns know receiv new ip isp major chang need take consider exchang server owa vpn two site link wireless connect km away citrix server vmware exchang domain control link place
import server crucial step inform need know avoid downtim busi regard ndavid',)
----------------------------------------------------------------------------------------------------
('use ef migrat creat databas googl migrat tutori af first run applic creat databas ef enabl migrat way creat databas migrat rune applic tri',)
----------------------------------------------------------------------------------------------------
('magento unit test problem magento site recent look way check integr magento site given point unit test jump one method would assum would big job write whole lot test check everyt
h site work anyon involv unit test magento advis follow possibl test whole site custom modul nis exampl test would amaz given site heavili link databas would nbe possibl fulli test
site without disturb databas better way automaticlli check integr magento site say integr realli mean fault site ship payment etc work correct',)
----------------------------------------------------------------------------------------------------
('find network devic without bonjour write mac applic need discov mac pcs iphon ipad connect wifi network bonjour seem reason choic turn problem mani type router mine exampl work b
lock bonjour servic need find ip devic tri connect applic specif port determin process run best approach accomplish task without violat app store sandbox',)
----------------------------------------------------------------------------------------------------
('send multipl row mysql databas want send user mysql databas column user skill time nnow want abl add one row user differ time etc would code send databas nthen use help schema',)
----------------------------------------------------------------------------------------------------
('insert data mysql php powerpoint event powerpoint present run continu way updat slide present automat data mysql databas websit',)
----------------------------------------------------------------------------------------------------
```

In [0]:
```
#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()
```

In [0]:
```
preprocessed_data.head()
```

Out[47]:

	question	tags
0	resiz root window tkinter resiz root window re...	python tkinter
1	ef code first defin one mani relationship diff...	entity-framework-4.1
2	explan new statement review section c code cam...	c++
3	error function notat function solv logic riddl...	haskell logic
4	step plan move one isp anoth one work busi pla...	dns isp

In [0]:
```
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 : 999999
number of dimensions : 2
```

Machine Learning Models

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 [0]:
```
# binary='true' will give a binary vectorizer
vectorizer = CountVectorizer(tokenizer = lambda x: x.split(), binary='true')
multilabel_y = vectorizer.fit_transform(preprocessed_data['tags'])
```

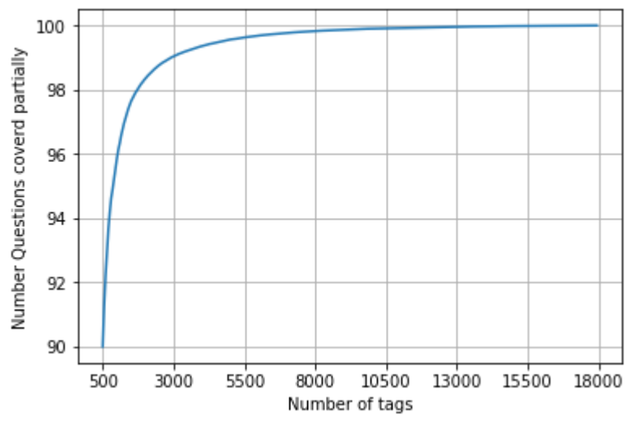
Lets sample the number of tags instead considering all of them (due to limitation of computing power).

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

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

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

In [0]:
```
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")
```




```
with  5500 tags we are covering  99.04 % of questions
```

In [0]:
```
multilabel_yx = tags_to_choose(5500)
print("number of questions that are not covered :", questions_explained_fn(5500),"out of ", total_qs)
```

```
number of questions that are not covered : 9599 out of  999999
```

In [0]:
```
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,"%)")
```

```
Number of tags in sample : 35422
number of tags taken : 5500 ( 15.527073570097679 %)
```

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

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

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

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

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

In [0]:
```
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 : (799999, 5500)
Number of data points in test data : (200000, 5500)
```

Featurizing data

In [0]:
```
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)
```

```
Time taken to run this cell : 0:09:50.460431
```

In [0]:
```
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)
```

```
Diamensions of train data X: (799999, 88244) Y : (799999, 5500)
Diamensions of test data X: (200000, 88244) Y: (200000, 5500)
```

In [0]:
```
# 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)
```

Out[92]:
```
"\nfrom skmultilearn.adapt import MLkNN\nclassifier = MLkNN(k=21)\n\n# train\nclassifier.fit(x_train_multilabel, y_train)\n\n# predict\npredictions = classifier.predict(x_test_mult
ilabel)\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"
```

Machine Learning Models

In [0]:
```
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, LogisticRegression
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 sklearn.naive_bayes import GaussianNB
from datetime import datetime
from sklearn.externals import joblib
from sklearn.model_selection import train_test_split
```

In [0]:
```
import sqlite3
import pandas as pd
# Create your connection.
cnx = sqlite3.connect('Titlemoreweight.db')

df = pd.read_sql_query("SELECT question, tags FROM QuestionsProcessed LIMIT 100000;", cnx)
```

Observations: We import an already created database with all the preprocessing, here the title is given 3 times the weight.

In [0]:
```
df.head()
```

Out[3]:

	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.noclassdeffounderror 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 [0]:
```
df.shape
```

Out[9]:
```
(100000, 2)
```

In [0]:
```
vectorizer = CountVectorizer(tokenizer=lambda x: x.split(), binary=True) #Perform bag of words and get vector form of data
multilabel_df = vectorizer.fit_transform(df['tags'])
```

In [0]:
```
multilabel_df.shape
```

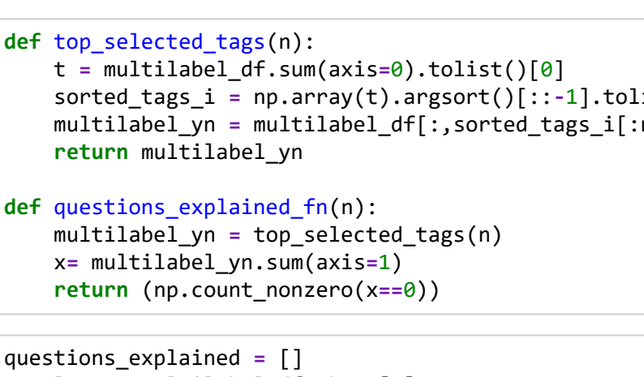
Out[5]:
```
(100000, 16321)
```

In [0]:
```
def top_selected_tags(n):
    t = multilabel_df.sum(axis=0).tolist()[0]
    sorted_tags_i = np.array(t).argsort()[::-1].tolist()
    multilabel_yn = multilabel_df[:,sorted_tags_i[:n]]
    return multilabel_yn

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

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

In [0]:
```
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(linestyle='-')
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")
print("With ",500,"tags we are covering ",questions_explained[0],"% of questions")
```




```
With  5500 tags we are covering  99.472 % of questions
With  500 tags we are covering  92.504 % of questions
```

In [0]:
```
multilabel_yn = top_selected_tags(500)
```

In [0]:
```
print("number of questions that are not covered :", questions_explained_fn(500),"out of ", total_qs)
```

```
number of questions that are not covered : 7496 out of  100000
```

In [0]:
```
print("Number of tags in sample :", multilabel_df.shape[1])
print("number of tags taken :", multilabel_yn.shape[1],"(",(multilabel_yn.shape[1]/multilabel_df.shape[1])*100,"%)")
```

```
Number of tags in sample : 16321
number of tags taken : 500 ( 3.0635377734207463 %)
```

In [0]:
```
train_size = int(df.shape[0]*0.80)
x_train = df[:train_size]
x_test = df[train_size:]
y_train = multilabel_yn[:train_size]
y_test = multilabel_yn[train_size:]
```

In [0]:
```
print('The shape of x_train is {}.'.format(x_train.shape))
print('The shape of y_train is {}.'.format(y_train.shape))
print('The shape of x_test is {}.'.format(x_test.shape))
print('The shape of y_test is {}.'.format(y_test.shape))
```

```
The shape of x_train is (80000, 2).
The shape of y_train is (80000, 500).
The shape of x_test is (20000, 2).
The shape of y_test is (20000, 500).
```

Bag Of Words

In [0]:
```
class MyCountVectorizer(CountVectorizer): #https://github.com/scikit-learn/scikit-learn/issues/6614#issuecomment-209922294
    def fit_transform(self, X):
        result = super(MyCountVectorizer, self).fit_transform(X)
        result.sort_indices()
        return result
```

Observations: The above function makes sure that indices are sorted after performing count vectorizer to avoid writebackifcopy error.

In [0]:
```
vectorizer = CountVectorizer(min_df=0.00009, max_features=25000, tokenizer = lambda x: x.split(), ngram_range=(1,4))
train_vect = vectorizer.fit_transform(x_train['question'])
test_vect = vectorizer.transform(x_test['question'])
```

In [0]:
```
print('The shape of train data is {}.'.format(train_vect.shape))
print('The shape of test data is {}.'.format(test_vect.shape))
```

```
The shape of train data is (80000, 25000).
The shape of test data is (20000, 25000).
```

Logistic Regression with One vs Rest

In [0]:
```
from sklearn.model_selection import ShuffleSplit
from sklearn.metrics import hamming_loss
from sklearn.metrics import make_scorer
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, hamming_loss
from sklearn.metrics import classification_report
from sklearn.model_selection import GridSearchCV
import warnings
warnings.filterwarnings("ignore")


param_grid = {'estimator__C':[0.0001,0.001,0.01,0.1,1,10,100]}
f1_scorer = make_scorer(f1_score, average='micro')

classifier = OneVsRestClassifier(LogisticRegression())
gsearch1 = GridSearchCV(estimator=classifier, param_grid=param_grid, cv=3, scoring=f1_scorer)
gsearch1.fit(train_vect,y_train)

print(gsearch1.best_estimator_)
```

```
OneVsRestClassifier(estimator=LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False),
          n_jobs=1)
```

In [0]:
```
print("The optimal alpha value for L2 penalty using GridSearchCV found is",gsearch1.best_params_['estimator__C'])
print('*'*50)
gpred1 = gsearch1.predict(test_vect)
print('\nThe accuracy of Logistic Regression for C = {} is {}'.format(gsearch1.best_params_['estimator__C'], accuracy_score(y_test, gpred1)))
print('\nThe precision of Logistic Regression for C = {} is {}'.format(gsearch1.best_params_['estimator__C'], precision_score(y_test,gpred1,average='micro')))
print('\nThe recall of Logistic Regression for C = {} is {}'.format(gsearch1.best_params_['estimator__C'], recall_score(y_test,gpred1,average='micro')))
print('\nThe f1 score of Logistic Regression for C = {} is {}'.format(gsearch1.best_params_['estimator__C'], f1_score(y_test,gpred1,average='micro')))
print('\nThe hamming of Logistic Regression for C = {} is {}\n'.format(gsearch1.best_params_['estimator__C'], hamming_loss(y_test,gpred1)*100))
print('*'*50)
#print(classification_report(y_test,gpred1))
#print("*"*50)
```

```
The optimal alpha value for L2 penalty using GridSearchCV found is 1

**************************************************

The accuracy of Logistic Regression for C = 1 is 0.1719

The precision of Logistic Regression for C = 1 is 0.5924203673564288

The recall of Logistic Regression for C = 1 is 0.33985114839811137

The f1 score of Logistic Regression for C = 1 is 0.4319229726064552

The hamming of Logistic Regression for C = 1 is 0.0033512

**************************************************
```

SGDClassifier with Log loss using One vs Rest classifier

In [0]:
```
param_grid = {'estimator__alpha':[0.0001,0.001,0.01,0.1,1,10,100]}
f1_scorer = make_scorer(f1_score, average='micro')

classifier = OneVsRestClassifier(SGDClassifier(loss='log'))
gsearch1 = GridSearchCV(estimator=classifier, param_grid=param_grid, cv=3, scoring=f1_scorer)
gsearch1.fit(train_vect,y_train)

print(gsearch1.best_estimator_)
```

```
OneVsRestClassifier(estimator=SGDClassifier(alpha=0.0001, average=False, class_weight=None, epsilon=0.1,
       eta0=0.0, fit_intercept=True, l1_ratio=0.15,
       learning_rate='optimal', loss='log', max_iter=None, n_iter=None,
       n_jobs=1, penalty='l2', power_t=0.5, random_state=None,
       shuffle=True, tol=None, verbose=0, warm_start=False),
          n_jobs=1)
```

In [0]:
```
print("The optimal alpha value for L2 penalty using GridSearchCV found is",gsearch1.best_params_['estimator__alpha'])
print('*'*50)
gpred1 = gsearch1.predict(test_vect)
print('\nThe accuracy of SGDClassifier with log loss for alpha = {} is {}'.format(gsearch1.best_params_['estimator__alpha'], accuracy_score(y_test, gpred1)))
print('\nThe precision of SGDClassifier with log loss for alpha = {} is {}'.format(gsearch1.best_params_['estimator__alpha'], precision_score(y_test,gpred1,average='micro')))
print('\nThe recall of SGDClassifier with log loss for alpha = {} is {}'.format(gsearch1.best_params_['estimator__alpha'], recall_score(y_test,gpred1,average='micro')))
print('\nThe f1 score of SGDClassifier with log loss for alpha = {} is {}'.format(gsearch1.best_params_['estimator__alpha'], f1_score(y_test,gpred1,average='micro')))
print('\nThe hamming of SGDClassifier with log loss for alpha = {} is {}\n'.format(gsearch1.best_params_['estimator__alpha'], hamming_loss(y_test,gpred1)))
print('*'*50)
#print(classification_report(y_test,gpred1))
#print("*"*50)
```

```
The optimal alpha value for L2 penalty using GridSearchCV found is 0.0001

**************************************************

The accuracy of SGDClassifier with log loss for alpha = 0.0001 is 0.1538

The precision of SGDClassifier with log loss for alpha = 0.0001 is 0.5350056919824362

The recall of SGDClassifier with log loss for alpha = 0.0001 is 0.35102835649691894

The f1 score of SGDClassifier with log loss for alpha = 0.0001 is 0.42391637002077864

The hamming of SGDClassifier with log loss for alpha = 0.0001 is 0.0035765

**************************************************
```

Linear SVM using One vs Rest Classifier

In [0]:
```
param_grid = {'estimator__alpha':[0.0001,0.001,0.01,0.1,1,10,100]}
f1_scorer = make_scorer(f1_score, average='micro')

classifier = OneVsRestClassifier(SGDClassifier(loss='hinge'))
#clf = Incremental(classifier)
gsearch1 = GridSearchCV(estimator=classifier, param_grid=param_grid, cv=3, scoring=f1_scorer)
gsearch1.fit(train_vect,y_train)

print(gsearch1.best_estimator_)
```

```
OneVsRestClassifier(estimator=SGDClassifier(alpha=0.001, average=False, class_weight=None, epsilon=0.1,
       eta0=0.0, fit_intercept=True, l1_ratio=0.15,
       learning_rate='optimal', loss='hinge', max_iter=None, n_iter=None,
       n_jobs=1, penalty='l2', power_t=0.5, random_state=None,
       shuffle=True, tol=None, verbose=0, warm_start=False),
          n_jobs=1)
```

In [0]:
```
print("The optimal alpha value for L2 penalty using GridSearchCV found is",gsearch1.best_params_['estimator__alpha'])
print('*'*50)
gpred1 = gsearch1.predict(test_vect)
print('\nThe accuracy of SGDClassifier with hinge loss for alpha = {} is {}'.format(gsearch1.best_params_['estimator__alpha'], accuracy_score(y_test, gpred1)))
print('\nThe precision of SGDClassifier with hinge loss for alpha = {} is {}'.format(gsearch1.best_params_['estimator__alpha'], precision_score(y_test,gpred1,average='micro')))
print('\nThe recall of SGDClassifier with hinge loss for alpha = {} is {}'.format(gsearch1.best_params_['estimator__alpha'], recall_score(y_test,gpred1,average='micro')))
print('\nThe f1 score of SGDClassifier with hinge loss for alpha = {} is {}'.format(gsearch1.best_params_['estimator__alpha'], f1_score(y_test,gpred1,average='micro')))
print('\nThe hamming of SGDClassifier with hinge loss for alpha = {} is {}\n'.format(gsearch1.best_params_['estimator__alpha'], hamming_loss(y_test,gpred1)))
print('*'*50)
#print(classification_report(y_test,gpred1))
#print("*"*50)
```

```
The optimal alpha value for L2 penalty using GridSearchCV found is 0.001

**************************************************

The accuracy of SGDClassifier with hinge loss for alpha = 0.001 is 0.19175

The precision of SGDClassifier with hinge loss for alpha = 0.001 is 0.7107849158580027

The recall of SGDClassifier with hinge loss for alpha = 0.001 is 0.3087203563902153

The f1 score of SGDClassifier with hinge loss for alpha = 0.001 is 0.43047108928936745

The hamming of SGDClassifier with hinge loss for alpha = 0.001 is 0.0030623

**************************************************
```

Results

In [0]:
```
from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ['Model','Hyperparameter','Accuracy', 'Precision', 'Recall', 'F1 Score', 'Hamming loss']
x.add_row(['Logistic Regression', '1', '0.1719', '0.5924', '0.3398', '0.4319', '0.0033'])
x.add_row(['','','','','','',''])
x.add_row(['SGD Classifier with Log loss', '0.001', '0.1538', '0.5350', '0.3510', '0.4239', '0.0035'])
x.add_row(['','','','','','',''])
x.add_row(['SGD Classifier with Hinge loss', '0.001', '0.1917', '0.7108', '0.3087', '0.4304', '0.0030'])
print(x.get_string())
```

```
+------------------------------+---------------+----------+-----------+--------+----------+--------------+
|            Model             | Hyperparameter | Accuracy | Precision | Recall | F1 Score | Hamming loss |
+------------------------------+---------------+----------+-----------+--------+----------+--------------+
|      Logistic Regression     |       1       |  0.1719  |   0.5924  | 0.3398 |  0.4319  |    0.0033    |
|                              |               |          |           |        |          |              |
|  SGD Classifier with Log loss |     0.001     |  0.1538  |   0.5350  | 0.3510 |  0.4239  |    0.0035    |
|                              |               |          |           |        |          |              |
| SGD Classifier with Hinge loss |     0.001     |  0.1917  |   0.7108  | 0.3087 |  0.4304  |    0.0030    |
+------------------------------+---------------+----------+-----------+--------+----------+--------------+
```