In [12]:
```
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
import joblib
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem.snowball import SnowballStemmer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.multiclass import OneVsRestClassifier
from sklearn.linear_model import SGDClassifier
from sklearn import metrics
from sklearn.metrics import f1_score,precision_score,recall_score
from sklearn import svm
from sklearn.linear_model import LogisticRegression
from skmultilearn.adapt import mlknn
from skmultilearn.problem_transform import ClassifierChain
from skmultilearn.problem_transform import BinaryRelevance
from skmultilearn.problem_transform import LabelPowerset
from sklearn.naive_bayes import GaussianNB
from datetime import datetime
```

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 in Stackoverflow.

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

1.2 Sources / 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)
yourtube : 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 man labels as possible correctly.
2. No strict latency constaraint.
3. Cost of errors would be a bad customer experience.

2. Machine Learning problem

2.1 Data

2.1.1 Data Overview

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 6034195 rows. The column 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 (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
```

```
                    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 te
    ll 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 multilable classification problem
Multilable 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 text might be about any of religion, politics, finance or education 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.

'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)

2.2.3 Machine Learning Objectives and Constraints

1. Minimize Micro avg F1 Score.
2. Try out multiple startegies for Multi-label classification.

3. Exploratory Data Analysis

3.1 Data Loading and Cleaning

3.1.1 Using Pandas with SQLite to Load the data

In [18]:
```
#Creating db file from csv
if not os.path.isfile('D:/Data Science/DataSets/Stackoverflow_Tag_Predictor/trai
    start = datetime.now()
    disk_engine = create_engine('sqlite:///D:/Data Science/DataSets/Stackoverflo
    start = dt.datetime.now()
    chunksize = 180000
    j = 0
    index_start = 1
    for df in pd.read_csv('D:/Data Science/DataSets/Stackoverflow_Tag_Predictor/
        df.index += index_start
        j+=1
        print('{} rows'.format(j*chunksize))
        df.to_sql('data', disk_engine, if_exists='append')
        index_start = df.index[-1] + 1
    print("Time taken to run this cell :", datetime.now() - start)
```

3.1.2 Counting the number of rows

In [19]:
```
if os.path.isfile('D:/Data Science/DataSets/Stackoverflow_Tag_Predictor/train.db
    start = datetime.now()
    con = sqlite3.connect('D:/Data Science/DataSets/Stackoverflow_Tag_Predictor/t
    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
```

```
Number of rows in the database :
 6034196
Time taken to count the number of rows : 0:01:41.385089
```

3.1.3 Checking for duplicates

In [20]:
```
if os.path.isfile('D:/Data Science/DataSets/Stackoverflow_Tag_Predictor/train.db
    start = datetime.now()
    con = sqlite3.connect('D:/Data Science/DataSets/Stackoverflow_Tag_Predictor/t
    df_no_dup = pd.read_sql_query('SELECT Title, Body, Tags, COUNT(*) as cnt_dup
    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 genar
```

```
Time taken to run this cell : 0:36:12.827058
```

In [21]:
```
df_no_dup.head()
# we can observe that there are duplicates
```

Out[21]:

	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 [22]:
```
print("number of duplicate questions :", num_rows['count(*)'].values[0]- df_no_du
```

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

In [23]:
```
# number of times each question appeared in our database
df_no_dup.cnt_dup.value_counts()
```

Out[23]:
```
1    2656284
2    1272336
3     277575
4         90
5         25
6          5
Name: cnt_dup, dtype: int64
```

In [24]:
```
# creating bool series True for NaN values
bool_series = pd.isnull(df_no_dup["Tags"])

# filtering data
# displayind data only with team = NaN
print("Number of rows with NaN values = ",len(df_no_dup[bool_series]))
df_no_dup[bool_series]
```

```
Number of rows with NaN values =  7
```

Out[24]:

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

In [25]:
```
# Deleting rows with NaN values
df_no_dup = df_no_dup.drop([777547,962680,1126558,1256102,2430668,3329908,355159
```

In [26]:
```
# creating bool series True for NaN values
bool_series = pd.isnull(df_no_dup["Tags"])

# filtering data
# displayind data only with team = NaN
print("Number of rows with NaN values = ",len(df_no_dup[bool_series]))
```

```
Number of rows with NaN values =  0
```

In [27]:
```
df_no_dup['index_col'] = df_no_dup.index
df_no_dup = df_no_dup.reset_index()
df_no_dup[777543:777548]
```

Out[27]:

	index	Title	Body	Tags	cnt_dup	index_col
777543	777543	Do we really need Automapper?	<p>I was learning AutoMapper and understand it...	automapper	2	777543
777544	777544	Do we really need Continuous Integration if we...	<p>Each of our team members runs all kind of t...	continuous-integration	1	777544
777545	777545	Do we really need Hahn-Banach that much?	<p>In the texts on functional analysis I'm rea...	abstract-algebra functional-analysis normed-sp...	2	777545
777546	777546	Do we really need MVC user controls, when movi...	<p>We are migrating and asp.NET application to...	asp.net asp.net-mvc	1	777546
777547	777548	Do we really need VOLATILE keyword in C#?	<p>Here is the code that I was trying on my wo...	c# multithreading volatile	2	777548

In [28]:
```
df_no_dup = df_no_dup.drop("index", axis=1)
df_no_dup = df_no_dup.drop("index_col", axis=1)
df_no_dup[777543:777548]
```

Out[28]:

	Title	Body	Tags	cnt_dup
777543	Do we really need Automapper?	<p>I was learning AutoMapper and understand it...	automapper	2
777544	Do we really need Continuous Integration if we...	<p>Each of our team members runs all kind of t...	continuous-integration	1
777545	Do we really need Hahn-Banach that much?	<p>In the texts on functional analysis I'm rea...	abstract-algebra functional-analysis normed-sp...	2
777546	Do we really need MVC user controls, when movi...	<p>We are migrating and asp.NET application to...	asp.net asp.net-mvc	1
777547	Do we really need VOLATILE keyword in C#?	<p>Here is the code that I was trying on my wo...	c# multithreading volatile	2

In [29]:
```
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 : 0:00:24.581189
```

Out[29]:

	Title	Body	Tags	cnt_dup	tag_
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 [30]:
```
# distribution of number of tags per question
df_no_dup.tag_count.value_counts()
```

Out[30]:
```
3    1206157
2    1111706
4     814996
1     568291
5     505158
Name: tag_count, dtype: int64
```

In [31]:
```
#Creating a new database with no duplicates
if not os.path.isfile('D:/Data Science/DataSets/Stackoverflow_Tag_Predictor/trai
    disk_dup = create_engine("sqlite:///D:/Data Science/DataSets/Stackoverflow_Ta
    no_dup = pd.DataFrame(df_no_dup, columns=['Title', 'Body', 'Tags'])
    no_dup.to_sql('no_dup_train',disk_dup)
```

In [32]:
```
#This method seems more appropriate to work with this much data.
#creating the connection with database file.
if os.path.isfile('D:/Data Science/DataSets/Stackoverflow_Tag_Predictor/train_no_
    start = datetime.now()
    con = sqlite3.connect('D:/Data Science/DataSets/Stackoverflow_Tag_Predictor/t
    tag_data = pd.read_sql_query("""SELECT Tags FROM no_dup_train""", con)
    #Always remember to close the database
    con.close()

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

```
Time taken to run this cell : 0:03:16.984086
```

3.2 Analysis of Tags

3.2.1 Total number of unique tags

In [33]:
```
# 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 [34]:
```
print("Number of data points :", tag_dtm.shape[0])
print("Number of unique tags :", tag_dtm.shape[1])
```

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

In [35]:
```
#'get_feature_name()' gives us the vocabulary.
tags = vectorizer.get_feature_names()
#Lets look at the tags we have.
print("Some of the tages we have :", tags[:10])
```

```
Some of the tages 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 [36]:
```
# https://stackoverflow.com/questions/15115765/how-to-access-sparse-matrix-elemer
#Lets now store the document term matrix in a dictionary.
freqs = tag_dtm.sum(axis=0).A1
result = dict(zip(tags, freqs))
```

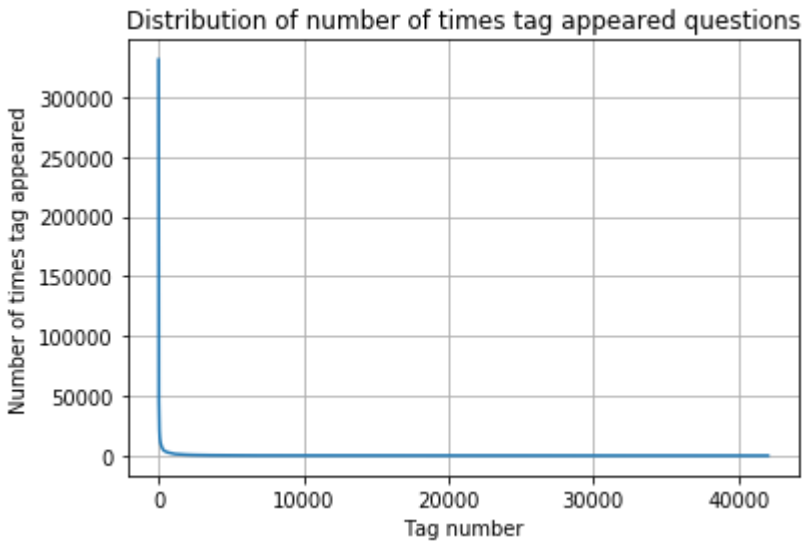
In [37]:
```
#Saving this dictionary to csv files.
if not os.path.isfile('D:/Data Science/DataSets/Stackoverflow_Tag_Predictor/tag_
    with open('D:/Data Science/DataSets/Stackoverflow_Tag_Predictor/tag_counts_d
        writer = csv.writer(csv_file)
        for key, value in result.items():
            writer.writerow([key, value])
tag_df = pd.read_csv("D:/Data Science/DataSets/Stackoverflow_Tag_Predictor/tag_c
tag_df.head()
```

Out[37]:

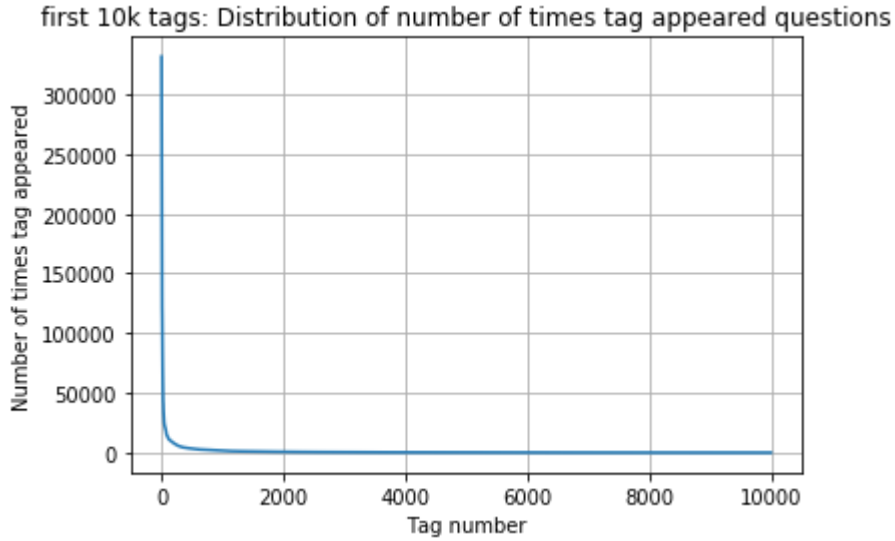
	Tags	Counts
0	.a	18
1	.app	37
2	.asp.net-mvc	1
3	.aspxauth	21
4	.bash-profile	138

In [38]:
```
tag_df_sorted = tag_df.sort_values(['Counts'], ascending=False)
tag_counts = tag_df_sorted['Counts'].values
```

In [39]:
```
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()
```



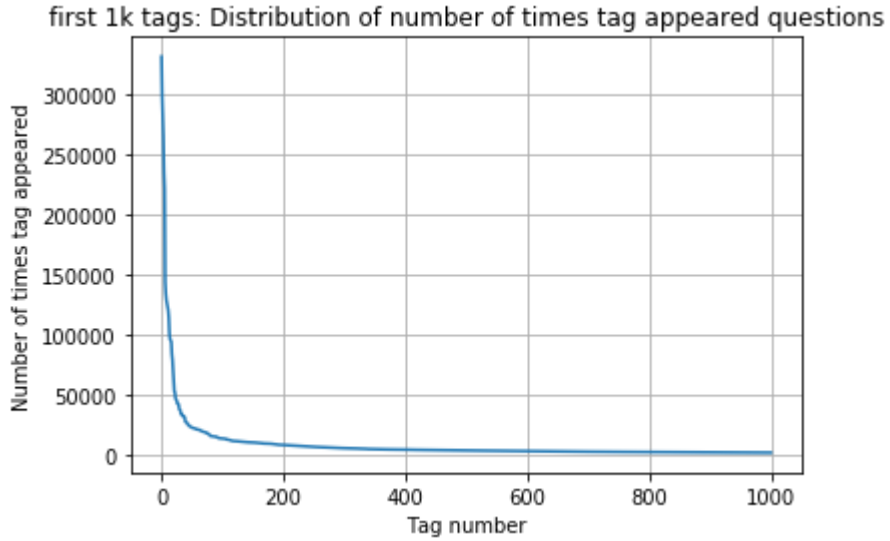
In [40]:
```
plt.plot(tag_counts[0:10000])
plt.title('first 10k tags: Distribution of number of times tag appeared questions
plt.grid()
plt.xlabel("Tag number")
plt.ylabel("Number of times tag appeared")
plt.show()
print(len(tag_counts[0:10000:25]), tag_counts[0:10000:25])
```




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

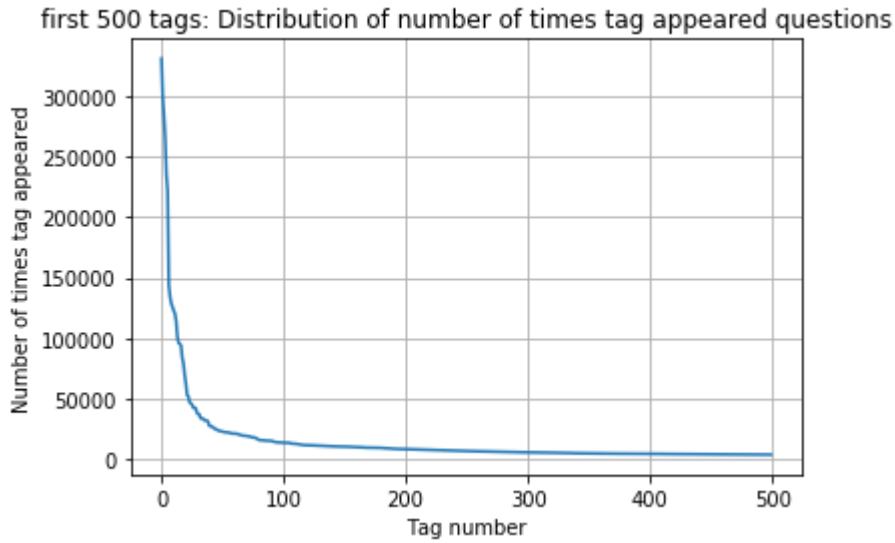
```
 105   105   104   104   103   103   102   102   101   101
 100   100    99    99    98    98    97    97    96    96
  95    95    94    94    93    93    93    92    92    91
  91    90    90    89    89    88    88    87    87    86
  86    86    85    85    84    84    83    83    83    82
  82    82    81    81    80    80    80    79    79    78
  78    78    78    77    77    76    76    76    75    75
  75    74    74    74    73    73    73    73    72    72]
```

In [41]:
```
plt.plot(tag_counts[0:1000])
plt.title('first 1k tags: Distribution of number of times tag appeared questions
plt.grid()
plt.xlabel("Tag number")
plt.ylabel("Number of times tag appeared")
plt.show()
print(len(tag_counts[0:1000:5]), tag_counts[0:1000:5])
```



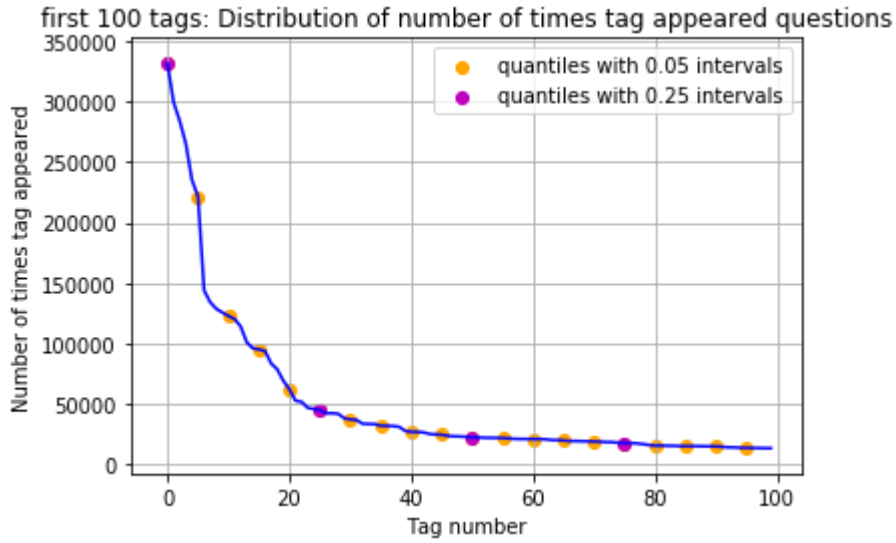

```
200 [331505 221533 122769  95160  62023  44829  37170  31897  26925  24537
  22429  21820  20957  19758  18905  17728  15533  15097  14884  13703
  13364  13157  12407  11658  11228  11162  10863  10600  10350  10224
  10029   9884   9719   9411   9252   9148   9040   8617   8361   8163
   8054 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

In [42]:
```
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 [44]:
```
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="qua
# quantiles with 0.25 difference
plt.scatter(x=list(range(0,100,25)), y=tag_counts[0:100:25], c='m', label = "quar

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 [45]:
```
# 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.

3.2.4 Tags Per Question

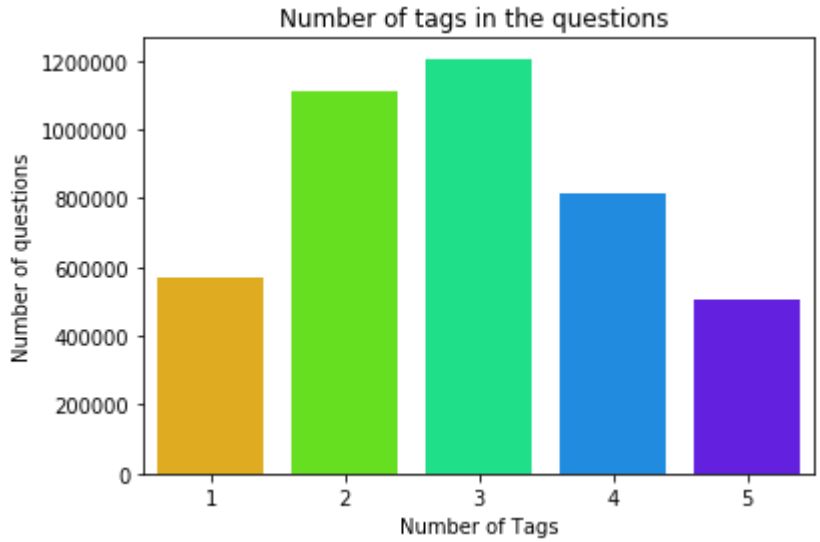
In [46]:
```
#Storing the count of tag in each question in list 'tag_count'
tag_quest_count = tag_dtm.sum(axis=1).tolist()
#Converting each value in the 'tag_quest_count' to integer.
tag_quest_count=[int(j) for i in tag_quest_count for j in i]
print ('We have total {} datapoints.'.format(len(tag_quest_count)))

print(tag_quest_count[:5])
```

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

In [47]:
```
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(t
```

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

In [48]:
```
sns.countplot(tag_quest_count, palette='gist_rainbow')
plt.title("Number of tags in the questions ")
plt.xlabel("Number of Tags")
plt.ylabel("Number of questions")
plt.show()
```



Observations:

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

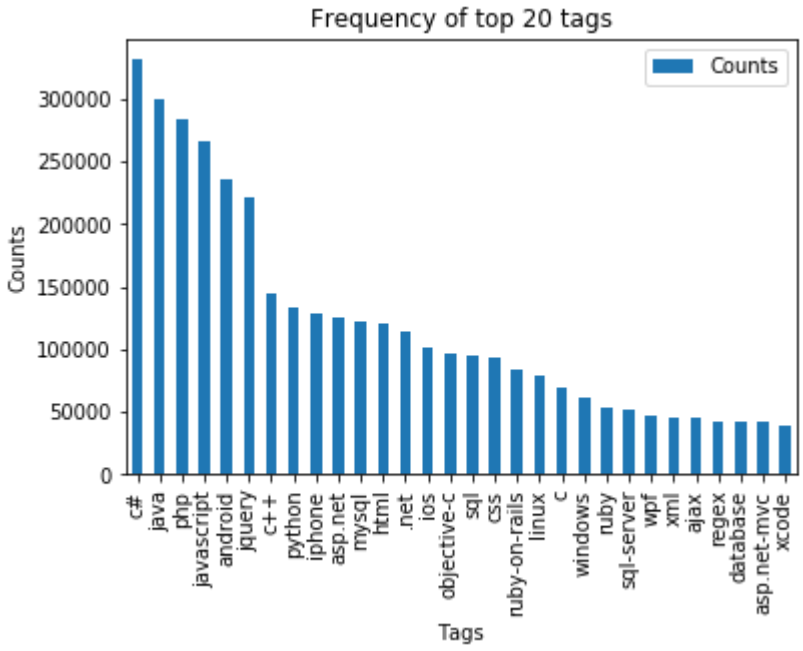
3.2.5 Most Frequent Tags

In [ ]:
```
# Ploting word cloud
start = datetime.now()

# Lets first convert the 'result' dictionary to 'list of tuples'
tup = dict(result.items())
#Initializing WordCloud using frequencies of tags.
wordcloud = WordCloud(    background_color='black',
                          width=1600,
                          height=800,
                    ).generate_from_frequencies(tup)

fig = plt.figure(figsize=(30,20))
plt.imshow(wordcloud)
plt.axis('off')
plt.tight_layout(pad=0)
fig.savefig("tag.png")
plt.show()
print("Time taken to run this cell :", datetime.now() - start)
```

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

3.2.6 The top 20 tags

In [50]:
```
i=np.arange(30)
tag_df_sorted.head(30).plot(kind='bar')
plt.title('Frequency of top 20 tags')
plt.xticks(i, tag_df_sorted['Tags'])
plt.xlabel('Tags')
plt.ylabel('Counts')
plt.show()
```



Observations:

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

3.3 Cleaning and preprocessing of Questions

3.3.1 Preprocessing

1. Sample 1M data points
2. Separate Code 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 [51]:
```
def striphtml(data):
    cleanr = re.compile('<.*?>')
    cleantext = re.sub(cleanr, ' ', str(data))
    return cleantext
stop_words = set(stopwords.words('english'))
stemmer = SnowballStemmer("english")
```

In [52]:
```
#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
create_database_table("D:/Data Science/DataSets/Stackoverflow_Tag_Predictor/Proce
```

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

In [ ]:
```
# http://www.sqlitetutorial.net/sqlite-delete/
# https://stackoverflow.com/questions/2279706/select-random-row-from-a-sqlite-tal
start = datetime.now()
read_db = 'D:/Data Science/DataSets/Stackoverflow_Tag_Predictor/train_no_dup.db'
write_db = 'D:/Data Science/DataSets/Stackoverflow_Tag_Predictor/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 RANDO

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

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

In [ ]:
```
#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.DO
    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
    question=' '.join(str(stemmer.stem(j)) for j in words if j not in stop_words

    len_post+=len(question)
    tup = (question,code,tags,x,len(question),is_code)
    questions_proccesed += 1
    conn_w = create_connection(write_db)
    writer =conn_w.cursor()
    writer.execute("insert into QuestionsProcessed(question,code,tags,words_pre,
    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_l
print( "Avg. length of questions(Title+Body) after processing: %d"%no_dup_avg_le
print ("Percent of questions containing code: %d"%((questions_with_code*100.0)/q

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

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

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

In [ ]:
```
#Taking 1 Million entries to a dataframe.
write_db = 'D:/Data Science/DataSets/Stackoverflow_Tag_Predictor/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 Ques
conn_r.commit()
conn_r.close()
```

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

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

4. Machine Learning Models

4.1 Converting tags for multilable problems

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

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

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

In [ ]:
```
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 [ ]:
```
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))/to
```

In [ ]:
```
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
print("with ",5500,"tags we are covering ",questions_explained[50],"% of questio
```

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

In [ ]:
```
print("Number of tags in sample :", multilabel_y.shape[1])
print("number of tags taken :", multilabel_yx.shape[1],"(",(multilabel_yx.shape[
```

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

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

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

4.3 Featurizing data

In [ ]:
```
start = datetime.now()
vectorizer = TfidfVectorizer(min_df=0.00009, max_features=200000, smooth_idf=Tru
                             tokenizer = lambda x: x.split(), sublinear_tf=False
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)
```

In [ ]:
```
print("Diamensions of train data X:",x_train_multilabel.shape, "Y :",y_train.sha
print("Diamensions of test data X:",x_test_multilabel.shape,"Y:",y_test.shape)
```

In [ ]:
```
# https://www.analyticsvidhya.com/blog/2017/08/introduction-to-multi-label-class
#https://stats.stackexchange.com/questions/117796/scikit-multi-label-classificat
# 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 conver
# ---------------------------------------------------------------------------
#MemoryError                               Traceback (most recent call last)
#<ipython-input-170-f0e7c7f3e0be> in <module>()
#----> classifier.fit(x_train_multilabel, y_train)
```

4.4 Applying Logistic Regression with OneVsRest

Classifier

In [ ]:
```
classifier = OneVsRestClassifier(SGDClassifier(loss='log', alpha=0.00001, penalty
classifier.fit(x_train_multilabel, y_train)
predictions = classifier.predict(x_test_multilabel)

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

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

4.5 Modeling with less data points (0.5M data points) and more weight to title

In [ ]:
```
sql_create_table = """CREATE TABLE IF NOT EXISTS QuestionsProcessed (question te
create_database_table("D:/Data Science/DataSets/Stackoverflow_Tag_Predictor/Title
```

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

read_db = 'D:/Data Science/DataSets/Stackoverflow_Tag_Predictor/train_no_dup.db'
write_db = 'D:/Data Science/DataSets/Stackoverflow_Tag_Predictor/Titlemoreweight
train_datasize = 400000
if os.path.isfile(read_db):
    conn_r = create_connection(read_db)
    if conn_r is not None:
        reader =conn_r.cursor()
        # for selecting first 0.5M rows
        reader.execute("SELECT Title, Body, Tags From no_dup_train LIMIT 500001;
        # for selecting random points
        #reader.execute("SELECT Title, Body, Tags From no_dup_train ORDER BY RANL

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")
```

4.5.1 Preprocessing of questions

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

3. **Give more weightage to title : Add title three times to the question**
4. **Add 'Tags' string to the training data**
5. Remove stop words (Except 'C')
6. Remove HTML Tags
7. Convert all the characters into small letters
8. Use SnowballStemmer to stem the words

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

    is_code = 0

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

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

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

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

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

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

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

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

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

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

    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,
    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_l
print( "Avg. length of questions(Title+Body) after processing: %d"%no_dup_avg_le
print ("Percent of questions containing code: %d"%((questions_with_code*100.0)/q
```

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

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

__ Sample quesitons after preprocessing of data __

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

__ Saving Preprocessed data to a Database __

In [ ]:
```
#Taking 1 Million entries to a dataframe.
write_db = 'D:/Data Science/DataSets/Stackoverflow_Tag_Predictor/Titlemoreweight
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 Ques
conn_r.commit()
conn_r.close()
```

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

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

__ Converting string Tags to multilable output variables __

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

__ Selecting 500 Tags __

In [ ]:
```
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))/to
```

In [ ]:
```
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
print("with ",5500,"tags we are covering ",questions_explained[50],"% of questio
```

In [ ]:
```
multilabel_yx = tags_to_choose(500)
print("number of questions that are not covered :", questions_explained_fn(500),
```

In [ ]:
```
import joblib

x_train = preprocessed_data.head(train_datasize)
x_test = preprocessed_data.tail(preprocessed_data.shape[0] - 400000)

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

joblib.dump(x_train,"D:/Data Science/DataSets/Stackoverflow_Tag_Predictor/x_trai
joblib.dump(x_test,"D:/Data Science/DataSets/Stackoverflow_Tag_Predictor/x_test.
joblib.dump(y_train,"D:/Data Science/DataSets/Stackoverflow_Tag_Predictor/y_trai
joblib.dump(y_test,"D:/Data Science/DataSets/Stackoverflow_Tag_Predictor/y_test.
```

In [2]:
```
import joblib
x_train = joblib.load("D:/Data Science/DataSets/Stackoverflow_Tag_Predictor/x_tra
x_test = joblib.load("D:/Data Science/DataSets/Stackoverflow_Tag_Predictor/x_test
y_train = joblib.load("D:/Data Science/DataSets/Stackoverflow_Tag_Predictor/y_tra
y_test = joblib.load("D:/Data Science/DataSets/Stackoverflow_Tag_Predictor/y_test
```

In [3]:
```
print("Number of data points in train data :", y_train.shape)
print("Number of data points in test data :", y_test.shape)
```

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

In [4]:
```
# Taking only 100k data points
x_train_1 = x_train
y_train_1 = y_train

x_train = x_train_1[0:80000]
x_test = x_train_1[80000:100000]
y_train = y_train_1[0:80000]
y_test = y_train_1[80000:100000]

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 : (80000, 500)
Number of data points in test data : (20000, 500)
```

4.5.2 Featurizing data with Tfidf vectorizer

In [ ]:
```
start = datetime.now()
vectorizer = TfidfVectorizer(min_df=0.00009, max_features=200000, smooth_idf=Tru
                             tokenizer = lambda x: x.split(), sublinear_tf=False
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)
```

In [ ]:
```
print("Diamensions of train data X:",x_train_multilabel.shape, "Y :",y_train.sha
print("Diamensions of test data X:",x_test_multilabel.shape,"Y:",y_test.shape)
```

4.5.3 Applying Logistric Regression with OneVsRest Classifier

In [ ]:
```
start = datetime.now()
classifier = OneVsRestClassifier(SGDClassifier(loss='log', alpha=0.00001, penalty
classifier.fit(x_train_multilabel, y_train)
predictions = classifier.predict (x_test_multilabel)


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


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

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

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

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

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

In [ ]:
```
joblib.dump(classifier, 'lr_with_more_title_weight.pkl')
```

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


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

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

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

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

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

5. Assignments

1. Use bag of words upto 4 grams and compute the micro f1 score with Logistic regression(OvR)
2. Use tdidf vectorizer upto 4 grams and compute the micro f1 score with Knearest (OvR)
3. Add some extra features and try to get micro f1 score > 0.5

Featurizing data with BoW vectorizer

In [5]:
```
start = datetime.now()
print("Starting time: ",start)
vectorizer = CountVectorizer(max_features=500,tokenizer = lambda x: x.split(), ng
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)
```

```
Starting time:  2019-04-12 18:01:26.979977
Time taken to run this cell : 0:04:43.896785
```

In [6]:
```
print("Diamensions of train data X:",x_train_multilabel.shape, "Y :",y_train.shap
print("Diamensions of test data X:",x_test_multilabel.shape,"Y:",y_test.shape)
```

```
Diamensions of train data X: (80000, 500) Y : (80000, 500)
Diamensions of test data X: (20000, 500) Y: (20000, 500)
```

Logistic Regression with BoW

In [28]:
```
#BoW
start = datetime.now()
print("Starting time: ",start)

classifier_2 = OneVsRestClassifier(LogisticRegression(penalty='l1'))
classifier_2.fit(x_train_multilabel, y_train)
predictions_2 = classifier_2.predict(x_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test, predictions_2))
print("Hamming loss ",metrics.hamming_loss(y_test,predictions_2))


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

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

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

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

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

```
Starting time:  2019-04-10 20:25:10.380943
Accuracy : 0.13055
Hamming loss  0.0036048
Micro-average quality numbers
Precision: 0.5372, Recall: 0.1861, F1-measure: 0.2765
```

In [7]:
```
print(np.shape(x_train_multilabel))
print(np.shape(y_train))
```

```
(80000, 500)
(80000, 500)
```

In [8]:
```
from sklearn.model_selection import GridSearchCV
import datetime

#hyper-paramater tuning
startTime3 = datetime.datetime.now()
print("Current Time = ",startTime3)

#Using GridSearchCV with L1 Regularizer
tuned_parameters = [{'estimator__C': [100,0.1, 0.0001]}]

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

model_l2 = GridSearchCV(classifier,tuned_parameters,scoring = 'f1_micro', cv=3)
model_l2.fit(x_train_multilabel, y_train)
```

```
C:\Users\AbhiShek\Anaconda3\lib\site-packages\sklearn\multiclass.py:76: UserW
arning: Label not 429 is present in all training examples.
  str(classes[c]))
C:\Users\AbhiShek\Anaconda3\lib\site-packages\sklearn\multiclass.py:76: UserW
arning: Label not 432 is present in all training examples.
  str(classes[c]))
C:\Users\AbhiShek\Anaconda3\lib\site-packages\sklearn\svm\base.py:922: Conver
genceWarning: Liblinear failed to converge, increase the number of iteration
```

In [10]:
```
GS_OPTIMAL_clf_sgd = model_l2.best_estimator_
print("GS_OPTIMAL_clf_sgd = ",GS_OPTIMAL_clf_sgd)
best_score_model_l2 = model_l2.best_score_
print("\nBest score: ",best_score_model_l2)
test_score_l2 = model_l2.score(x_test_multilabel, y_test)
print("test_score_l2 = ",test_score_l2)
param_C = model_l2.best_params_["estimator__C"]
print("Best C= ",param_C)
```

```
GS_OPTIMAL_clf_sgd =  OneVsRestClassifier(estimator=LogisticRegression(C=0.1, c
lass_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l1', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False),
          n_jobs=None)

Best score:  0.34329031831081575
test_score_l2 =  0.25842721112152184
Best C=  0.1
```

Applying Logistic regression with best C value

In [17]:
```
#BoW
start = datetime.now()
print("Starting time: ",start)

classifier_2 = OneVsRestClassifier(LogisticRegression(penalty='l1',C=0.1,random_
classifier_2.fit(x_train_multilabel, y_train)
predictions_2 = classifier_2.predict(x_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test, predictions_2))
print("Hamming loss ",metrics.hamming_loss(y_test,predictions_2))


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

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

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

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

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

```
Starting time:  2019-04-13 14:44:03.450358
Accuracy : 0.13855
Hamming loss  0.0034033
Micro-average quality numbers
Precision: 0.6669, Recall: 0.1603, F1-measure: 0.2584
Macro-average quality numbers
Precision: 0.1743, Recall: 0.0493, F1-measure: 0.0706
              precision    recall  f1-score   support

           0       0.92      0.47      0.62      1424
           1       0.64      0.08      0.14      1931
           2       0.82      0.32      0.46      1374
           3       0.83      0.36      0.50       797
           4       0.84      0.41      0.55      1311
           5       0.80      0.27      0.41       820
           6       0.90      0.45      0.60      1014
           7       0.87      0.52      0.65       702
           8       0.60      0.12      0.21       544
           9       0.83      0.54      0.65       269
          10       0.50      0.16      0.25       405
```

OneVsRestClassifier with Linear SVM with BoW

In [34]:
```
#BoW
start = datetime.now()
print("Starting time: ",start)

classifier = OneVsRestClassifier(SGDClassifier(loss='hinge', alpha=0.00001, penal
classifier.fit(x_train_multilabel, y_train)
predictions = classifier.predict (x_test_multilabel)


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


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

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

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

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

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

```
Starting time:  2019-04-11 11:36:23.508911
Accuracy : 0.0671
Hamming loss  0.0074196
Micro-average quality numbers
Precision: 0.1675, Recall: 0.2532, F1-measure: 0.2016
Macro-average quality numbers
Precision: 0.0701, Recall: 0.1155, F1-measure: 0.0785

C:\Users\AbhiShek\Anaconda3\lib\site-packages\sklearn\metrics\classification.
py:1145: UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 i
n labels with no true samples.
  'recall', 'true', average, warn_for)
C:\Users\AbhiShek\Anaconda3\lib\site-packages\sklearn\metrics\classification.
py:1145: UndefinedMetricWarning: F-score is ill-defined and being set to 0.0
in labels with no true samples.
  'recall', 'true', average, warn_for)
C:\Users\AbhiShek\Anaconda3\lib\site-packages\sklearn\metrics\classification.
py:1145: UndefinedMetricWarning: Recall and F-score are ill-defined and being
set to 0.0 in labels with no true samples.
  'recall', 'true', average, warn_for)
```

Featurizing data with tfidf vectorizer

In [32]:
```
start = datetime.now()
print("Starting time: ",start)
vectorizer_tfidf = TfidfVectorizer(min_df=0.00009, max_features=500, smooth_idf=
                               tokenizer = lambda x: x.split(), sublinear_tf=False
x_train_multilabel_tfidf = vectorizer_tfidf.fit_transform(x_train['question'])
x_test_multilabel_tfidf = vectorizer_tfidf.transform(x_test['question'])
print("Time taken to run this cell :", datetime.now() - start)
```

```
Starting time:  2019-04-11 11:30:14.504213
Time taken to run this cell : 0:05:11.062140
```

In [33]:
```
print("Diamensions of train data X:",x_train_multilabel_tfidf.shape, "Y :",y_tra
print("Diamensions of test data X:",x_test_multilabel_tfidf.shape,"Y:",y_test.sh
```

```
Diamensions of train data X: (80000, 500) Y : (80000, 500)
Diamensions of test data X: (20000, 500) Y: (20000, 500)
```

In [20]:
```
#tfidf
start = datetime.now()
print("Starting time: ",start)

classifier_2 = OneVsRestClassifier(LogisticRegression(penalty='l1'), n_jobs=-1)
classifier_2.fit(x_train_multilabel_tfidf, y_train)
predictions_2 = classifier_2.predict(x_test_multilabel_tfidf)

print("Accuracy :",metrics.accuracy_score(y_test, predictions_2))
print("Hamming loss ",metrics.hamming_loss(y_test,predictions_2))


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

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

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

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

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

```
Starting time:  2019-04-10 18:13:28.793510
Accuracy : 0.1435
Hamming loss  0.0033437
Micro-average quality numbers
Precision: 0.7030, Recall: 0.1668, F1-measure: 0.2696
Macro-average quality numbers
Precision: 0.1871, Recall: 0.0537, F1-measure: 0.0753
```

In [26]:
```
#tfidf
start = datetime.now()
print("Starting time: ",start)

classifier = OneVsRestClassifier(SGDClassifier(loss='hinge', alpha=0.00001, penal
classifier.fit(x_train_multilabel, y_train)
predictions = classifier.predict (x_test_multilabel)


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


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

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

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

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

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

```
Starting time:  2019-04-10 19:59:50.760367
Accuracy : 0.14565
Hamming loss  0.0032959
Micro-average quality numbers
Precision: 0.7996, Recall: 0.1458, F1-measure: 0.2466
Macro-average quality numbers
Precision: 0.1008, Recall: 0.0355, F1-measure: 0.0468
              precision    recall  f1-score   support

           0       0.94      0.51      0.67      1424
           1       0.77      0.01      0.01      1931
           2       0.85      0.31      0.46      1374
           3       0.82      0.34      0.48       797
           4       0.88      0.35      0.50      1311
           5       0.80      0.32      0.46       820
           6       0.89      0.45      0.60      1014
           7       0.88      0.55      0.68       702
           8       0.61      0.13      0.22       544
           9       0.84      0.51      0.63       269
          10       0.00      0.00      0.00       405
```

In [2]:
```
from prettytable import PrettyTable
x = PrettyTable()

x.field_names = ["Vectorizer","Model", "n_gram range" ,"F1_Score"]

x.add_row(["BoW" ,"Logistic Regression", 4, 0.2765])
x.add_row(["BoW" ,"Log. Regression (Grid Search)",4, 0.2584])
x.add_row(["BoW", "Linear SVM", 4, 0.2016])

x.add_row(["TFIDF" ,"Logistic Regression", 4, 0.2696])
x.add_row(["TFIDF" ,"Lr. SVM",4, 0.2466])

print(x)
```

```
+------------+------------------------------+--------------+----------+
| Vectorizer |             Model            | n_gram range | F1_Score |
+------------+------------------------------+--------------+----------+
|    BoW     |      Logistic Regression     |      4       |  0.2765  |
|    BoW     | Log. Regression (Grid Search) |      4       |  0.2584  |
|    BoW     |           Linear SVM         |      4       |  0.2016  |
|   TFIDF    |      Logistic Regression     |      4       |  0.2696  |
|   TFIDF    |            Lr. SVM           |      4       |  0.2466  |
+------------+------------------------------+--------------+----------+
```

In [ ]: