In [1]:

```
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import sqlite3
import csv
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from wordcloud import WordCloud
import re
import os
from sqlalchemy import create_engine # database connection
import datetime as dt
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem.snowball import SnowballStemmer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.multiclass import OneVsRestClassifier
from sklearn.linear_model import SGDClassifier
from sklearn import metrics
from sklearn.metrics import f1_score,precision_score,recall_score
from sklearn import svm
from sklearn.linear_model import LogisticRegression
from skmultilearn.adapt import mlknn
from skmultilearn.problem_transform import ClassifierChain
from skmultilearn.problem_transform import BinaryRelevance
from skmultilearn.problem_transform import LabelPowerset
from sklearn.naive_bayes import GaussianNB
from datetime import datetime
```

Stack Overflow: Tag Prediction

1. Business Problem

1.1 Description

Description

Stack Overflow is the largest, most trusted online community for developers to learn, share their programming knowledge, and build their careers.

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

Problem Statemtent

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

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

1.2 Source / useful links

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

1.3 Real World / Business Objectives and Constraints

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

2. Machine Learning problem

2.1 Data

2.1.1 Data Overview

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

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

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

Size of Train.csv - 6.75GB

Size of Test.csv - 2GB

Number of rows in Train.csv = 6034195

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

Data Field 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 '&')

2.1.2 Example Data point

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

Body :

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

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

2.2.1 Type of Machine Learning Problem

It is a multi-label classification problem
Multi-label Classification: Multilabel classification assigns to each sample a set of target labels. This can be thought as predicting properties of a data-point that are not mutually exclusive, such as topics that are relevant for a document. A question on Stackoverflow might be about any of C, Pointers, FileIO and/or memory-management at the same time or none of these.
https://stats.stackexchange.com/questions/11859/what-is-the-difference-between-multiclass-and-multilabel-problem
__Credit__: http://scikit-learn.org/stable/modules/multiclass.html

2.2.2 Performance metric

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

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

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

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

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

https://www.kaggle.com/wiki/MeanFScore (https://www.kaggle.com/wiki/MeanFScore)
http://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html (http://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html)

Hamming loss : The Hamming loss is the fraction of labels that are incorrectly predicted.
https://www.kaggle.com/wiki/HammingLoss (https://www.kaggle.com/wiki/HammingLoss)

3. Exploratory Data Analysis

3.1 Data Loading and Cleaning

3.1.1 Using Pandas with SQLite to Load the data

In [0]:

```
#Learn SQL: https://www.w3schools.com/sql/default.asp https://www.sqlite.org/index.html

#Creating db file from csv. The Train.csv was downloaded from Kaggle

if not os.path.isfile('train.db'):
    start = datetime.now()
    disk_engine = create_engine('sqlite:///train.db')
    start = dt.datetime.now()
    chunksize = 180000
    j = 0
    index_start = 1
    for df in pd.read_csv('Train.csv', names=['Id', 'Title', 'Body', 'Tags'], chunksize
=chunksize, iterator=True, encoding='utf-8', ):
        df.index += index_start
        j+=1
        print('{} rows'.format(j*chunksize))
        df.to_sql('data', disk_engine, if_exists='append')
        index_start = df.index[-1] + 1
    print("Time taken to run this cell :", datetime.now() - start)
```

3.1.2 Counting the number of rows

In [0]:

```
if os.path.isfile('train.db'):
    start = datetime.now()
    con = sqlite3.connect('train.db')  #start the connection with the data base
    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()  #close the connection
    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 genara
te train.db file")
```

```
Number of rows in the database :
 6034196
Time taken to count the number of rows : 0:01:15.750352
```

3.1.3 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 d
ata 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 tr
ain.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[0]:

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

In [0]:

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

```
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[0]:

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

In [0]:

```
# distribution of number of tags per question
df_no_dup.tag_count.value_counts()
```

Out[0]:

```
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)
    #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 genar
ate train.db file")
```

```
Time taken to run this cell : 0:00:52.992676
```

3.2 Analysis of Tags

3.2.1 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 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 [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[0]:

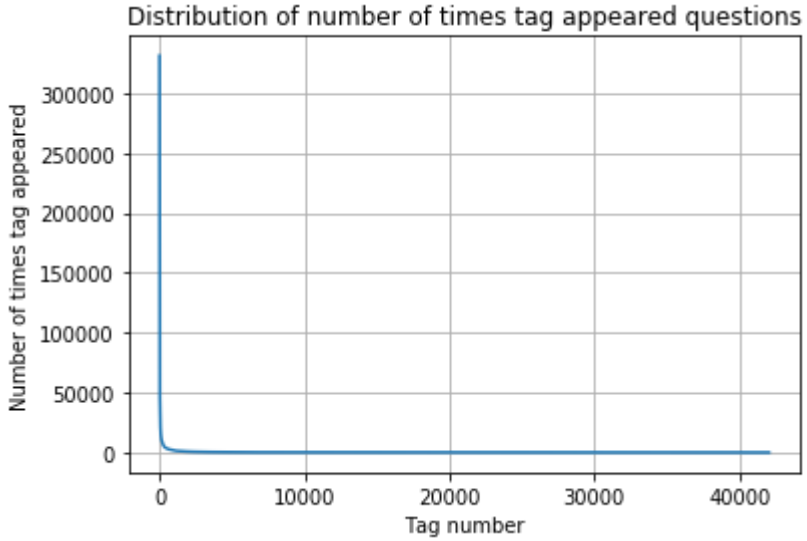
	Tags	Counts
0	.a	18
1	.app	37
2	.asp.net-mvc	1
3	.aspxauth	21
4	.bash-profile	138

In [0]:

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

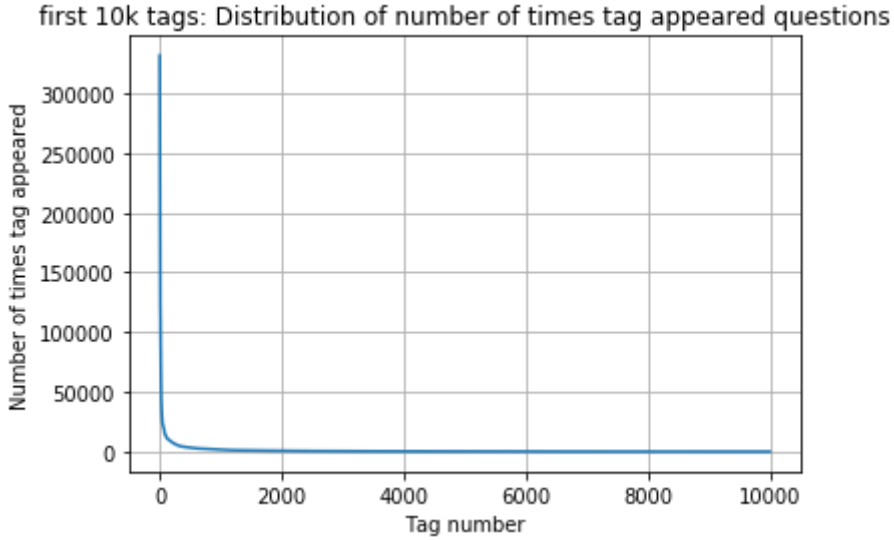
In [0]:

```
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]:

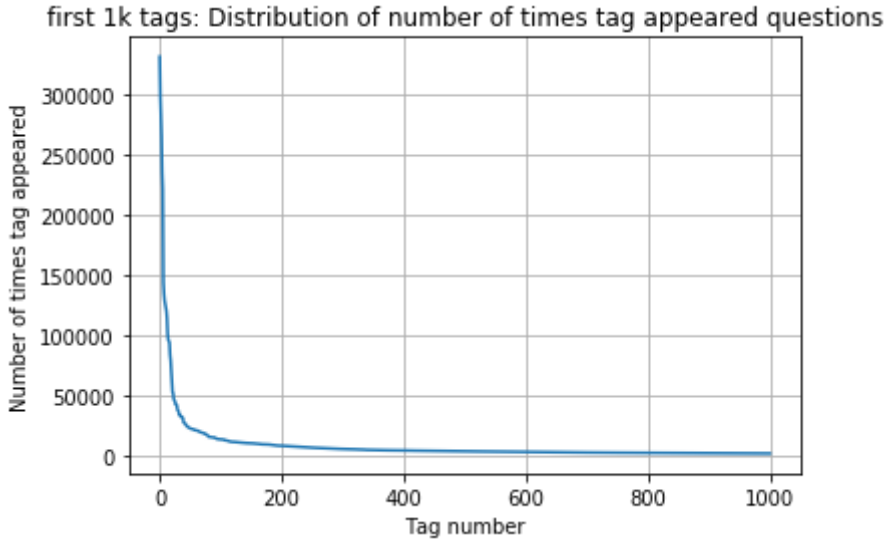
```
plt.plot(tag_counts[0:10000])
plt.title('first 10k tags: Distribution of number of times tag appeared questions')
plt.grid()
plt.xlabel("Tag number")
plt.ylabel("Number of times tag appeared")
plt.show()
print(len(tag_counts[0:10000:25]), tag_counts[0:10000:25])
```




```
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]:

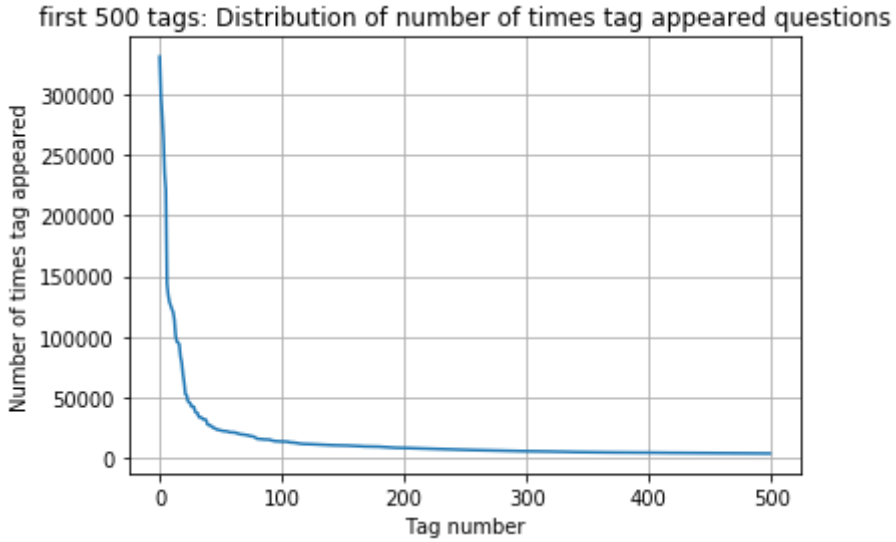
```
plt.plot(tag_counts[0:1000])
plt.title('first 1k tags: Distribution of number of times tag appeared questions')
plt.grid()
plt.xlabel("Tag number")
plt.ylabel("Number of times tag appeared")
plt.show()
print(len(tag_counts[0:1000:5]), tag_counts[0:1000:5])
```




```
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]:

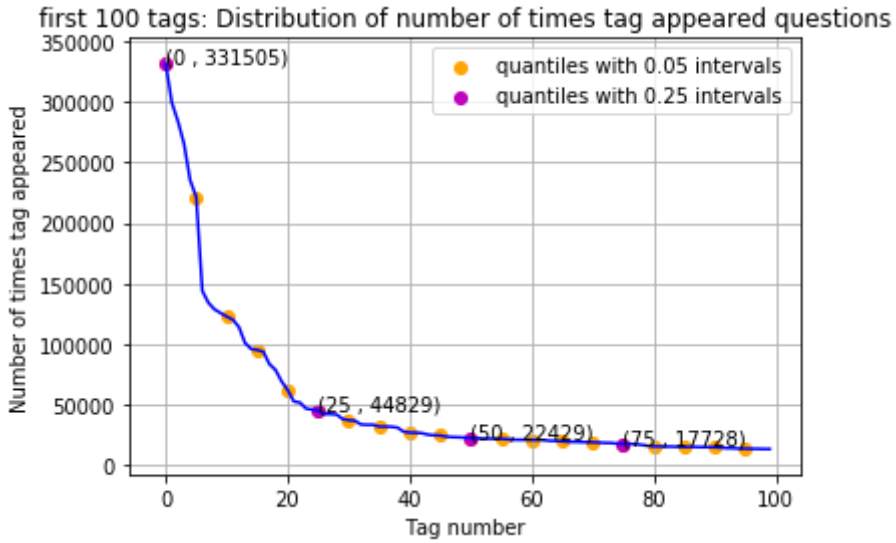
```
plt.plot(tag_counts[0:500])
plt.title('first 500 tags: Distribution of number of times tag appeared questions')
plt.grid()
plt.xlabel("Tag number")
plt.ylabel("Number of times tag appeared")
plt.show()
print(len(tag_counts[0:500:5]), tag_counts[0:500:5])
```




```
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.

3.2.4 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 list of lists into single list, we will get [[3], [4], [2], [2], [3]] and w
e are converting this to [3, 4, 2, 2, 3]
tag_quest_count=[int(j) for i in tag_quest_count for j in i]
print ('We have total {} datapoints.'.format(len(tag_quest_count)))

print(tag_quest_count[:5])
```

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

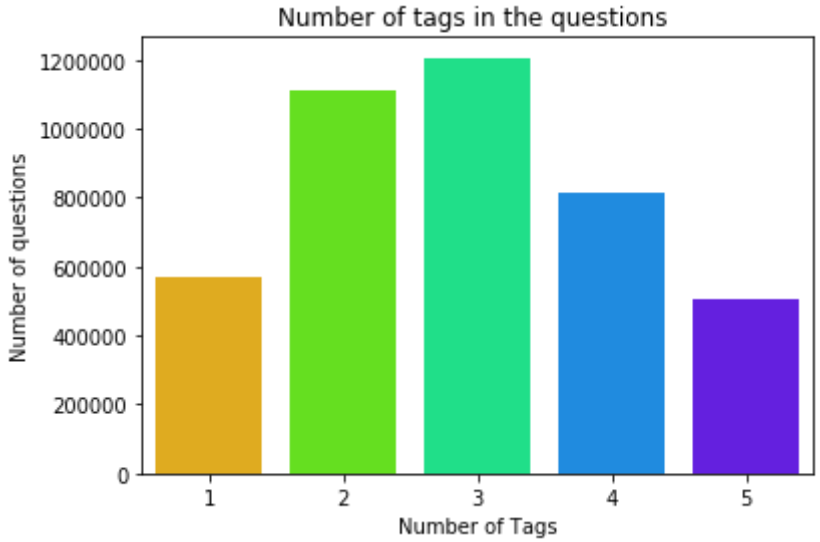
In [0]:

```
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_ques
t_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()
```



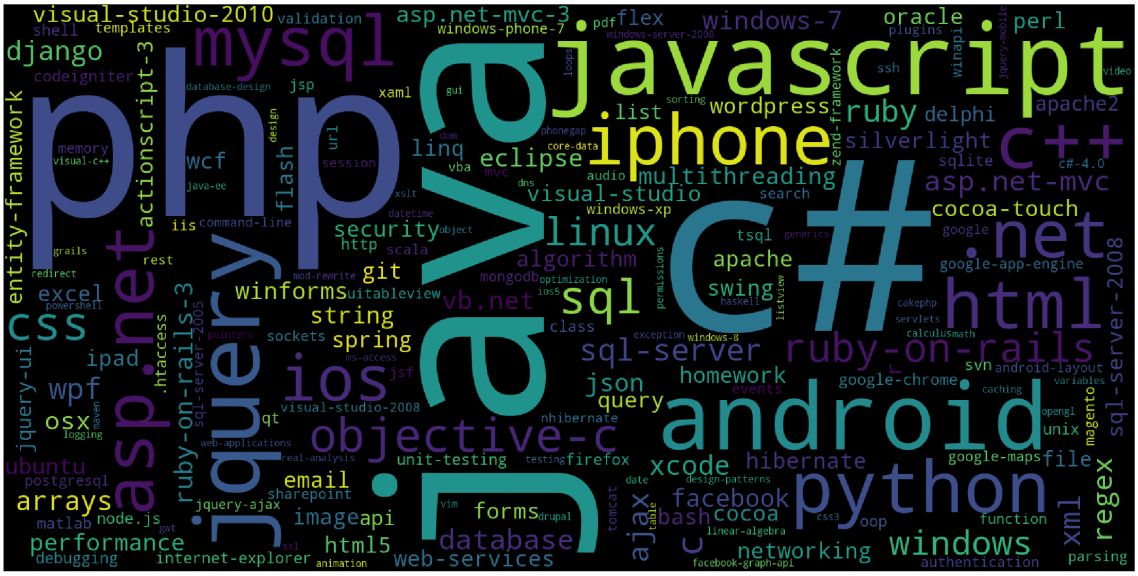
Observations:

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

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



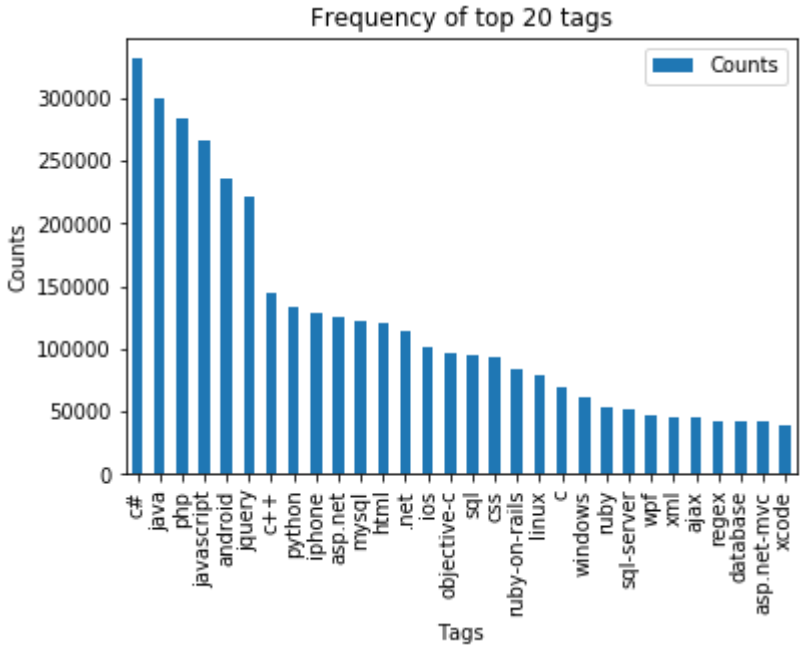
Time taken to run this cell : 0:00:05.470788

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 30 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.

3.3 Cleaning and preprocessing of Questions

3.3.1 Preprocessing

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

In [2]:

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

In [3]:

```
#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 intege
r);"""
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() LI
MIT 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
```

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 (l
en(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_p
ost,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)/question
s_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 ze
ro mani relationship entiti ef object model look like use fluent api objec
t composit pk defin batch id batch detail id use fluent api object composi
t pk defin batch detail id compani id map exist databas tpt basic idea sub
mittedtransact zero mani submittedsplittransact associ navig realli need o
ne way submittedtransact submittedsplittransact need dbcontext class onmod
elcr overrid map class lazi load occur submittedtransact submittedsplittra
nsact help would much appreci edit taken advic made follow chang dbcontext
class ad follow onmodelcr overrid must miss someth get follow except throw
n submittedtransact key batch id batch detail id zero one mani submittedsp
littransact key batch detail id compani id rather assum convent creat rela
tionship 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 li
st possibl candid solut list possibl coordin matrix wan na choos one candi
d compar possibl candid element equal wan na delet coordin call function s
kasuj look like ni knowledg haskel cant see what wrong',)
--------------------------------------------------------------------------
--------------------------
('step plan move one isp anoth one work busi plan switch isp realli soon n
eed chang lot inform dns wan wan wifi question guy help mayb peopl plan co
rrect chang current isp new one first dns know receiv new ip isp major cha
ng need take consider exchang server owa vpn two site link wireless connec
t km away citrix server vmware exchang domain control link place import se
rver crucial step inform need know avoid downtim busi regard ndavid',)
--------------------------------------------------------------------------
--------------------------
('use ef migrat creat databas googl migrat tutori af first run applic crea
t databas ef enabl migrat way creat databas migrat rune applic tri',)
--------------------------------------------------------------------------
--------------------------
('magento unit test problem magento site recent look way check integr mage
nto site given point unit test jump one method would assum would big job w
rite whole lot test check everyth site work anyon involv unit test magento
advis follow possibl test whole site custom modul nis exampl test would am
az given site heavili link databas would nbe possibl fulli test site witho
ut disturb databas better way automaticlli check integr magento site say i
ntegr 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 man
i type router mine exampl work block bonjour servic need find ip devic tri
connect applic specif port determin process run best approach accomplish t
ask 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 wa
```

```
y updat slide present automat data mysql databas websit',)
--------------------------------------------------------------------------------
-------------------------
```

In [6]:

```
#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 QuestionsPr
ocessed""", conn_r)
conn_r.commit()
conn_r.close()
```

In [7]:

```
preprocessed_data.head()
```

Out[7]:

	question	tags
0	chang cpu soni vaio pcg grx tri everywher find...	cpu motherboard sony-vaio replacement disassembly
1	display size grayscal qimag qt abl display ima...	c++ qt qt4
2	datagrid selecteditem set back null eventtocom...	mvvm silverlight-4.0
3	filter string collect base listview item resol...	c# winforms string listview collections
4	disabl home button without use type keyguard c...	android android-layout android-manifest androi...

In [8]:

```
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
```

4. Machine Learning Models

4.1 Converting tags for multilabel problems

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

In [9]:

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

In [40]:

```
print(multilabel_y.shape)
```

```
(999999, 35422)
```

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

In [2]:

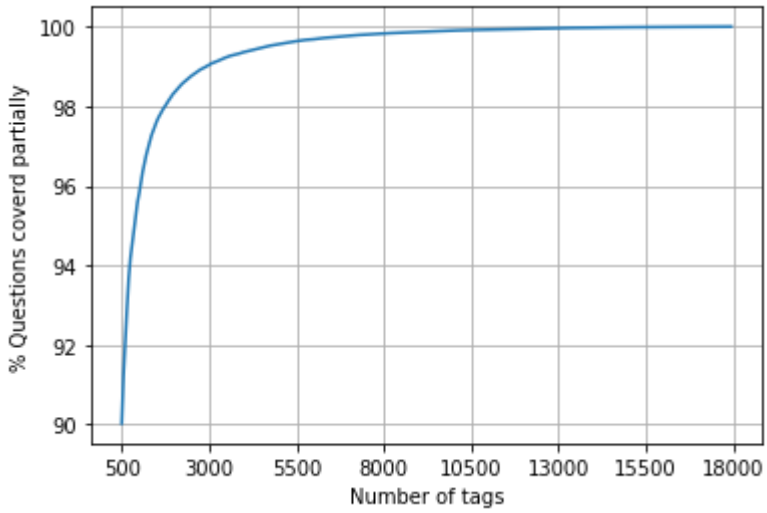
```
def tags_to_choose(n):
    t = multilabel_y.sum(axis=0).tolist()[0]
    sorted_tags_i = sorted(range(len(t)), key=lambda i: t[i], reverse=True)
    multilabel_yn=multilabel_y[:,sorted_tags_i[:n]]
    return multilabel_yn

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

In [17]:

```
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 [19]:

```
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("% Questions coverd partially")
plt.grid()
plt.show()
# you can choose any number of tags based on your computing power, minimum is 500(it co
vers 90% of the tags)
print("With ",5500,"tags we are covering ",questions_explained[50],"% of questions")
```




```
with  5500 tags we are covering  99.035 % of questions
```

In [24]:

```
multilabel_yx = tags_to_choose(5500)
print("number of questions that are not covered :", questions_explained_fn(5500),"out o
f ", total_qs)
```

```
number of questions that are not covered : 9645 out of  999999
```

In [25]:

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

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

In [36]:

```
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 [37]:

```
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)
```

4.3 Featurizing data

In [ ]:

```
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)
```

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-classificati
on/
#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[0]:

```
"\nfrom skmultilearn.adapt import MLkNN\nclassifier = MLkNN(k=21)\n\n# tra
in\nclassifier.fit(x_train_multilabel, y_train)\n\n# predict\npredictions
= classifier.predict(x_test_multilabel)\nprint(accuracy_score(y_test,predi
ctions))\nprint(metrics.f1_score(y_test, predictions, average = 'macro'))
\nprint(metrics.f1_score(y_test, predictions, average = 'micro'))\nprint(m
etrics.hamming_loss(y_test,predictions))\n\n"
```

4.4 Applying Logistic Regression with OneVsRest Classifier

In [0]:

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

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

```
accuracy : 0.081965
macro f1 score : 0.0963020140154
micro f1 scoore : 0.374270748817
hamming loss : 0.00041225090909090907
Precision recall report :
              precision    recall  f1-score   support

           0       0.62      0.23      0.33     15760
           1       0.79      0.43      0.56     14039
           2       0.82      0.55      0.66     13446
           3       0.76      0.42      0.54     12730
           4       0.94      0.76      0.84     11229
           5       0.85      0.64      0.73     10561
           6       0.70      0.30      0.42      6958
           7       0.87      0.61      0.72      6309
           8       0.70      0.40      0.50      6032
           9       0.78      0.43      0.55      6020
          10       0.86      0.62      0.72      5707
          11       0.52      0.17      0.25      5723
          12       0.55      0.10      0.16      5521
          13       0.59      0.25      0.35      4722
          14       0.61      0.22      0.32      4468
          15       0.79      0.52      0.63      4536
          16       0.58      0.27      0.37      4545
          17       0.80      0.53      0.64      4069
          18       0.61      0.24      0.35      3638
          19       0.57      0.18      0.27      3218
          20       0.33      0.06      0.10      3000
          21       0.73      0.34      0.46      2585
          22       0.59      0.29      0.38      2439
          23       0.88      0.61      0.72      2199
          24       0.64      0.39      0.48      2157
          25       0.67      0.39      0.49      2123
          26       0.86      0.65      0.74      1948
          27       0.35      0.07      0.12      2027
          28       0.59      0.29      0.39      2013
          29       0.61      0.20      0.30      1801
          30       0.48      0.24      0.32      1728
          31       0.94      0.75      0.84      1725
          32       0.60      0.26      0.36      1581
          33       0.49      0.14      0.22      1533
          34       0.81      0.33      0.47      1565
          35       0.75      0.62      0.68      1568
          36       0.76      0.50      0.60      1542
          37       0.74      0.50      0.59      1536
          38       0.37      0.12      0.19      1524
          39       0.40      0.12      0.19      1345
          40       0.65      0.38      0.48      1292
          41       0.41      0.11      0.17      1264
          42       0.69      0.25      0.37      1265
          43       0.59      0.29      0.38      1171
          44       0.41      0.15      0.22      1173
          45       0.38      0.10      0.16      1137
          46       0.62      0.12      0.20      1125
          47       0.26      0.07      0.11      1116
          48       0.44      0.15      0.22      1042
          49       0.40      0.02      0.03      1096
          50       0.63      0.38      0.48      1031
          51       0.47      0.14      0.22      1033
          52       0.87      0.68      0.76      1042
          53       0.32      0.09      0.14      1027
```

```
        5483       0.00      0.00      0.00         6
        5484       0.00      0.00      0.00         9
        5485       0.00      0.00      0.00         8
        5486       0.00      0.00      0.00         8
        5487       0.00      0.00      0.00         9
        5488       0.00      0.00      0.00         7
        5489       0.00      0.00      0.00        10
        5490       0.00      0.00      0.00        12
        5491       0.00      0.00      0.00         6
        5492       0.00      0.00      0.00         8
        5493       0.00      0.00      0.00        13
        5494       0.00      0.00      0.00         6
        5495       0.00      0.00      0.00        10
        5496       0.00      0.00      0.00         7
        5497       0.00      0.00      0.00         9
        5498       0.00      0.00      0.00         6
        5499       0.00      0.00      0.00        13

avg / total       0.53      0.26      0.33    530065
```

In [0]:

```
from sklearn.externals import joblib
joblib.dump(classifier, 'lr_with_equal_weight.pkl')
```

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

In [0]:

```
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 intege
r);"""
create_database_table("Titlemoreweight.db", sql_create_table)
```

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

In [5]:

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

read_db = 'train_no_dup.db'
write_db = 'Titlemoreweight.db'
train_datasize = 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 RANDOM() L
IMIT 500001;")

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

```
Tables in the databse:
QuestionsProcessed
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. 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]:

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

    is_code = 0

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

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

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

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

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

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

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

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

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

    #Removing all single letter and and stopwords from question exceptt for the letter
 'c'
    question=' '.join(str(stemmer.stem(j)) for j in words if j not in stop_words and (l
en(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_p
ost,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)/question
```

```
s_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
Avg. length of questions(Title+Body) before processing: 1239
Avg. length of questions(Title+Body) after processing: 424
Percent of questions containing code: 57
Time taken to run this cell : 0:23:12.329039
```

In [0]:

```
# 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 [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
================================================================================
=========================
('dynam datagrid bind silverlight dynam datagrid bind silverlight dynam da
tagrid bind silverlight bind datagrid dynam code wrote code debug code blo
ck seem bind correct grid come column form come grid column although neces
sari bind nthank repli advance..',)
--------------------------------------------------------------------------------
-------------------------
('java.lang.noclassdeffounderror javax servlet jsp tagext taglibraryvalid
java.lang.noclassdeffounderror javax servlet jsp tagext taglibraryvalid ja
va.lang.noclassdeffounderror javax servlet jsp tagext taglibraryvalid foll
ow guid link instal jstl got follow error tri launch jsp page java.lang.no
classdeffounderror javax servlet jsp tagext taglibraryvalid taglib declar
instal jstl 1.1 tomcat webapp tri project work also tri version 1.2 jstl s
till messag caus solv',)
--------------------------------------------------------------------------------
-------------------------
('java.sql.sqlexcept microsoft odbc driver manag invalid descriptor index
java.sql.sqlexcept microsoft odbc driver manag invalid descriptor index ja
va.sql.sqlexcept microsoft odbc driver manag invalid descriptor index use
follow code display caus solv',)
--------------------------------------------------------------------------------
-------------------------
('better way updat feed fb php sdk better way updat feed fb php sdk better
way updat feed fb php sdk novic facebook api read mani tutori still confus
ed.i find post feed api method like correct second way use curl someth lik
e way better',)
--------------------------------------------------------------------------------
-------------------------
('btnadd click event open two window record ad btnadd click event open two
window record ad btnadd click event open two window record ad open window
search.aspx use code hav add button search.aspx nwhen insert record btnadd
click event open anoth window nafter insert record close window',)
--------------------------------------------------------------------------------
-------------------------
('sql inject issu prevent correct form submiss php sql inject issu prevent
correct form submiss php sql inject issu prevent correct form submiss php
check everyth think make sure input field safe type sql inject good news s
afe bad news one tag mess form submiss place even touch life figur exact h
tml use templat file forgiv okay entir php script get execut see data post
none forum field post problem use someth titl field none data get post cur
rent use print post see submit noth work flawless statement though also me
ntion script work flawless local machin use host come across problem state
list input test mess',)
--------------------------------------------------------------------------------
-------------------------
('countabl subaddit lebesgu measur countabl subaddit lebesgu measur counta
bl subaddit lebesgu measur let lbrace rbrace sequenc set sigma -algebra ma
thcal want show left bigcup right leq sum left right countabl addit measur
defin set sigma algebra mathcal think use monoton properti somewher proof
start appreci littl help nthank ad han answer make follow addit construct
given han answer clear bigcup bigcup cap emptyset neq left bigcup right le
ft bigcup right sum left right also construct subset monoton left right le
q left right final would sum leq sum result follow',)
--------------------------------------------------------------------------------
-------------------------
('hql equival sql queri hql equival sql queri hql equival sql queri hql qu
eri replac name class properti name error occur hql error',)
--------------------------------------------------------------------------------
-------------------------
```

```
('undefin symbol architectur i386 objc class skpsmtpmessag referenc error
undefin symbol architectur i386 objc class skpsmtpmessag referenc error un
defin symbol architectur i386 objc class skpsmtpmessag referenc error impo
rt framework send email applic background import framework i.e skpsmtpmess
ag somebodi suggest get error collect2 ld return exit status import framew
ork correct sorc taken framework follow mfmailcomposeviewcontrol question
lock field updat answer drag drop folder project click copi nthat',)
--------------------------------------------------------------------------
-------------------------
```

Saving Preprocessed data to a Database

In [ ]:

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

In [44]:

```
preprocessed_data.head()
```

Out[44]:

	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 [11]:

```
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 : 500000
number of dimensions : 2
```

Converting string Tags to multilable output variables

In [12]:

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

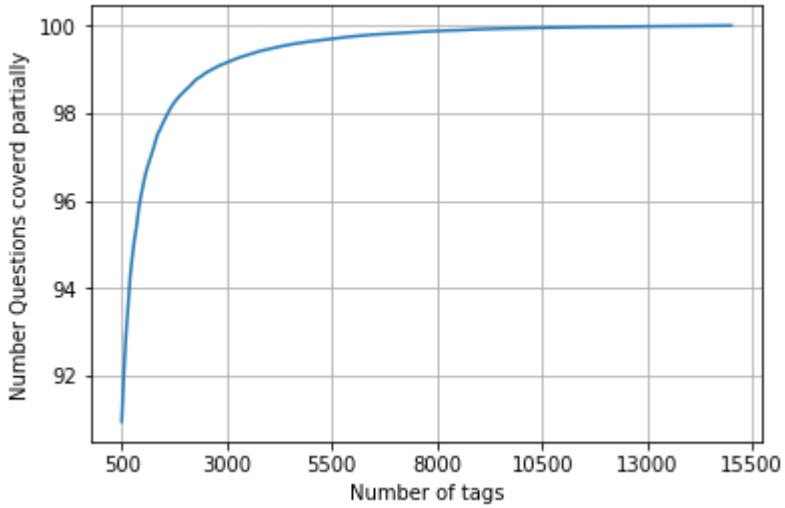
Selecting 500 Tags

In [15]:

```
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 [16]:

```
fig, ax = plt.subplots()
ax.plot(questions_explained)
xlabel = list(500+np.array(range(-50,450,50))*50)
ax.set_xticklabels(xlabel)
plt.xlabel("Number of tags")
plt.ylabel("Number Questions coverd partially")
plt.grid()
plt.show()
# you can choose any number of tags based on your computing power, minimun is 500(it co
vers 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  99.157 % of questions
with  500 tags we are covering  90.956 % of questions
```

In [17]:

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

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

4.5.2 Splitting preprocessed_data into train & test

In [20]:

```
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],:]
```

In [21]:

```
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)
```

4.5.3 Featurizing data with Tfidf vectorizer

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 : 0:03:52.522389
```

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: (400000, 94927) Y : (400000, 500)
Diamensions of test data X: (100000, 94927) Y: (100000, 500)
```

4.5.4 Applying Logistic Regression with OneVsRest Classifier

In [0]:

```
start = datetime.now()
classifier = OneVsRestClassifier(SGDClassifier(loss='log', alpha=0.00001, penalty='l1'
), n_jobs=-1)
classifier.fit(x_train_multilabel, y_train)
predictions = classifier.predict (x_test_multilabel)


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


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

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

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

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

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

```
Accuracy : 0.23623
Hamming loss  0.00278088
Micro-average quality numbers
Precision: 0.7216, Recall: 0.3256, F1-measure: 0.4488
Macro-average quality numbers
Precision: 0.5473, Recall: 0.2572, F1-measure: 0.3339
             precision    recall  f1-score   support

          0       0.94      0.64      0.76      5519
          1       0.69      0.26      0.38      8190
          2       0.81      0.37      0.51      6529
          3       0.81      0.43      0.56      3231
          4       0.81      0.40      0.54      6430
          5       0.82      0.33      0.47      2879
          6       0.87      0.50      0.63      5086
          7       0.87      0.54      0.67      4533
          8       0.60      0.13      0.22      3000
          9       0.81      0.53      0.64      2765
         10       0.59      0.17      0.26      3051
         11       0.70      0.33      0.45      3009
         12       0.64      0.24      0.35      2630
         13       0.71      0.23      0.35      1426
         14       0.90      0.53      0.67      2548
         15       0.66      0.18      0.28      2371
         16       0.65      0.23      0.34       873
         17       0.89      0.61      0.72      2151
         18       0.62      0.23      0.33      2204
         19       0.71      0.40      0.51       831
         20       0.77      0.41      0.53      1860
         21       0.27      0.07      0.11      2023
         22       0.49      0.23      0.31      1513
         23       0.91      0.49      0.64      1207
         24       0.56      0.29      0.38       506
         25       0.68      0.30      0.42       425
         26       0.65      0.40      0.49       793
         27       0.60      0.32      0.42      1291
         28       0.75      0.36      0.48      1208
         29       0.42      0.09      0.15       406
         30       0.75      0.18      0.29       504
         31       0.29      0.10      0.14       732
         32       0.59      0.24      0.35       441
         33       0.56      0.18      0.27      1645
         34       0.71      0.25      0.37      1058
         35       0.83      0.54      0.66       946
         36       0.69      0.21      0.32       644
         37       0.96      0.68      0.79       136
         38       0.64      0.37      0.47       570
         39       0.85      0.29      0.43       766
         40       0.62      0.28      0.38      1132
         41       0.46      0.19      0.27       174
         42       0.81      0.51      0.63       210
         43       0.80      0.41      0.54       433
         44       0.66      0.50      0.57       626
         45       0.75      0.32      0.45       852
         46       0.75      0.42      0.54       534
         47       0.34      0.14      0.20       350
         48       0.74      0.51      0.60       496
         49       0.79      0.62      0.70       785
         50       0.16      0.04      0.06       475
         51       0.33      0.10      0.15       305
         52       0.50      0.04      0.07       251
```

480	0.82	0.50	0.62	100
481	0.47	0.17	0.26	103
482	0.47	0.23	0.31	74
483	0.85	0.57	0.68	105
484	0.25	0.02	0.04	83
485	0.17	0.01	0.02	82
486	0.36	0.11	0.17	71
487	0.43	0.18	0.26	120
488	0.33	0.02	0.04	105
489	0.72	0.30	0.42	87
490	1.00	0.81	0.90	32
491	0.00	0.00	0.00	69
492	0.00	0.00	0.00	49
493	0.00	0.00	0.00	117
494	0.52	0.18	0.27	61
495	0.98	0.65	0.78	344
496	0.36	0.19	0.25	52
497	0.60	0.18	0.28	137
498	0.33	0.04	0.07	98
499	0.65	0.16	0.26	79
avg / total	0.67	0.33	0.43	173812

Time taken to run this cell : 0:10:14.264591

In [0]:

```
joblib.dump(classifier, 'lr_with_more_title_weight.pkl')
```

Out[0]:

['lr_with_more_title_weight.pkl']

In [0]:

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


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

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

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

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

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

```
Accuracy : 0.25108
Hamming loss  0.00270302
Micro-average quality numbers
Precision: 0.7172, Recall: 0.3672, F1-measure: 0.4858
Macro-average quality numbers
Precision: 0.5570, Recall: 0.2950, F1-measure: 0.3710
             precision    recall  f1-score   support

           0       0.94      0.72      0.82      5519
           1       0.70      0.34      0.45      8190
           2       0.80      0.42      0.55      6529
           3       0.82      0.49      0.61      3231
           4       0.80      0.44      0.57      6430
           5       0.82      0.38      0.52      2879
           6       0.86      0.53      0.66      5086
           7       0.87      0.58      0.70      4533
           8       0.60      0.13      0.22      3000
           9       0.82      0.57      0.67      2765
          10       0.60      0.20      0.30      3051
          11       0.68      0.38      0.49      3009
          12       0.62      0.29      0.40      2630
          13       0.73      0.30      0.43      1426
          14       0.89      0.57      0.70      2548
          15       0.65      0.23      0.34      2371
          16       0.65      0.25      0.37       873
          17       0.89      0.63      0.74      2151
          18       0.60      0.25      0.35      2204
          19       0.71      0.41      0.52       831
          20       0.76      0.47      0.58      1860
          21       0.29      0.09      0.14      2023
          22       0.52      0.24      0.33      1513
          23       0.89      0.55      0.68      1207
          24       0.56      0.28      0.38       506
          25       0.69      0.34      0.45       425
          26       0.65      0.43      0.52       793
          27       0.62      0.38      0.47      1291
          28       0.74      0.39      0.51      1208
          29       0.46      0.10      0.17       406
          30       0.76      0.21      0.33       504
          31       0.26      0.08      0.12       732
          32       0.60      0.29      0.39       441
          33       0.60      0.27      0.38      1645
          34       0.69      0.26      0.38      1058
          35       0.83      0.58      0.68       946
          36       0.65      0.24      0.35       644
          37       0.98      0.65      0.78       136
          38       0.62      0.38      0.47       570
          39       0.84      0.31      0.45       766
          40       0.59      0.35      0.44      1132
          41       0.47      0.18      0.26       174
          42       0.76      0.49      0.59       210
          43       0.75      0.42      0.54       433
          44       0.66      0.52      0.58       626
          45       0.71      0.36      0.47       852
          46       0.77      0.45      0.57       534
          47       0.37      0.15      0.22       350
          48       0.75      0.52      0.62       496
          49       0.78      0.64      0.71       785
          50       0.21      0.06      0.09       475
          51       0.37      0.13      0.19       305
          52       0.42      0.03      0.06       251
```

```
          480       0.79      0.50      0.61       100
          481       0.51      0.28      0.36       103
          482       0.40      0.22      0.28        74
          483       0.78      0.63      0.69       105
          484       0.20      0.02      0.04        83
          485       0.20      0.02      0.04        82
          486       0.48      0.15      0.23        71
          487       0.45      0.21      0.29       120
          488       0.50      0.06      0.10       105
          489       0.73      0.37      0.49        87
          490       1.00      0.81      0.90        32
          491       0.33      0.03      0.05        69
          492       0.33      0.02      0.04        49
          493       0.11      0.02      0.03       117
          494       0.52      0.23      0.32        61
          495       0.95      0.79      0.87       344
          496       0.32      0.13      0.19        52
          497       0.59      0.28      0.38       137
          498       0.31      0.10      0.15        98
          499       0.48      0.20      0.29        79

avg / total       0.67      0.37      0.46    173812

Time taken to run this cell : 1:09:41.236859
```

5. Assignments

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

Data preparation

In [2]:

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

In [3]:

```
#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 intege
r);"""
create_database_table("Processed.db", sql_create_table)
```

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

In [4]:

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

In [5]:

```
preprocessed_data.head()
```

Out[5]:

	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 [6]:

```
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 : 500000
number of dimensions : 2
```

Considering 0.25 million data points due to computational constraints

In [7]:

```
preprocessed_data=preprocessed_data.iloc[:250000,:]
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 : 250000
number of dimensions : 2
```

In [8]:

```
preprocessed_data.head(2)
```

Out[8]:

	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

Converting string Tags to multilable output variables

In [9]:

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

Selecting 500 Tags

In [10]:

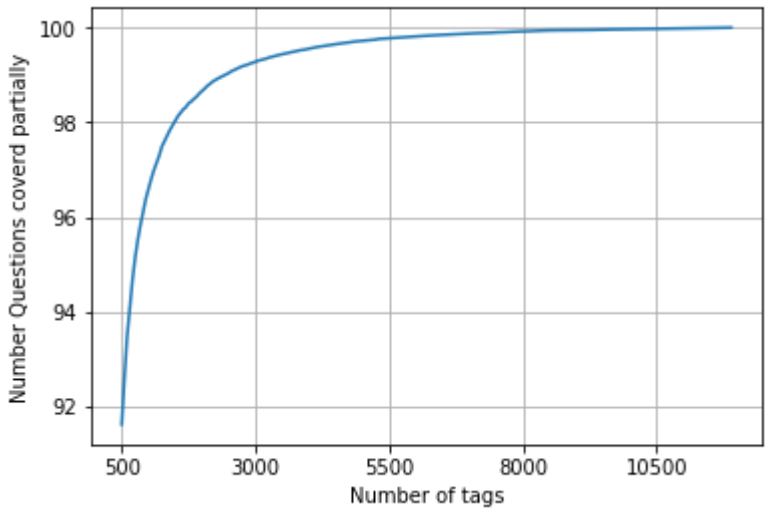
```
def tags_to_choose(n):
    t = multilabel_y.sum(axis=0).tolist()[0]
    sorted_tags_i = sorted(range(len(t)), key=lambda i: t[i], reverse=True)
    multilabel_yn=multilabel_y[:,sorted_tags_i[:n]]
    return multilabel_yn

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

In [11]:

```
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 [12]:

```
fig, ax = plt.subplots()
ax.plot(questions_explained)
xlabel = list(500+np.array(range(-50,450,50))*50)
ax.set_xticklabels(xlabel)
plt.xlabel("Number of tags")
plt.ylabel("Number Questions coverd partially")
plt.grid()
plt.show()
# you can choose any number of tags based on your computing power, minimun is 500(it co
vers 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  99.28 % of questions
with  500 tags we are covering  91.621 % of questions
```

In [13]:

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

```
number of questions that are not covered : 20948 out of  250000
```

Splitting preprocessed_data into train & test(80:20)

In [14]:

```
total=preprocessed_data.shape[0]
train=int(0.8*total)
x_train=preprocessed_data.head(train)
x_test=preprocessed_data.tail(total - train)

y_train = multilabel_yx[0:train,:]
y_test = multilabel_yx[train:total,:]
```

In [15]:

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

```
Number of data points in train data : (200000, 500)
Number of data points in test data : (50000, 500)
```

5.1 Vectorizing 'question' column using BOW and n_grams=4

In [19]:

```
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer(min_df=0.00009, max_features=25000,\
                             tokenizer = lambda x: x.split(),  ngram_range=(1,4)) #cons
idered 25k as max_features due to compuational constraints

x_train_multilabel = vectorizer.fit_transform(x_train['question']) # fit has to happen
 only on train data
x_test_multilabel = vectorizer.transform(x_test['question'])
```

In [20]:

```
print("Dimensions of train data :",x_train_multilabel.shape, "Y :",y_train.shape)
print("Dimensions of test data :",x_test_multilabel.shape,"Y:",y_test.shape)
```

```
Dimensions of train data : (200000, 25000) Y : (200000, 500)
Dimensions of test data : (50000, 25000) Y: (50000, 500)
```

In [21]:

```
#Saving the 2D arrays as npz files

from scipy import sparse
import numpy as np

sparse.save_npz("x_train_multilabel.npz", x_train_multilabel)
sparse.save_npz("x_test_multilabel.npz", x_test_multilabel)

np.save('y_train', y_train)
np.save('y_test', y_test)
```

In [16]:

```
#loading the saved files

from scipy import sparse
import numpy as np
x_train_multilabel= sparse.load_npz("x_train_multilabel.npz")
x_test_multilabel = sparse.load_npz("x_test_multilabel.npz")
#y_train = np.load('y_train.npy',allow_pickle=True)
#y_test = np.load('y_test.npy',allow_pickle=True)
```

5.2 Logistic Regression with hyperparameter tuning using Grid search

5.2.1 Hyperparameter tuning to find best alpha

In [22]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import f1_score

alpha = {'estimator__alpha': [10**-6, 10**-5, 10**-3, 10**-2]}

clf=OneVsRestClassifier(SGDClassifier(loss='log', penalty='l1')) #https://stackoverflow.com/questions/12632992/gridsearch-for-an-estimator-inside-a-onevsrestclassifier

gs1=GridSearchCV(estimator=clf, param_grid=alpha, cv=3, scoring='f1_micro', n_jobs=-1)

model1=gs1.fit(x_train_multilabel,y_train)
```

5.2.2 2D plot

In [23]:

```
df1=pd.DataFrame(gs1.cv_results_)
df1.head(2)
df1.to_csv(r'LR_cv.csv')
```

In [25]:

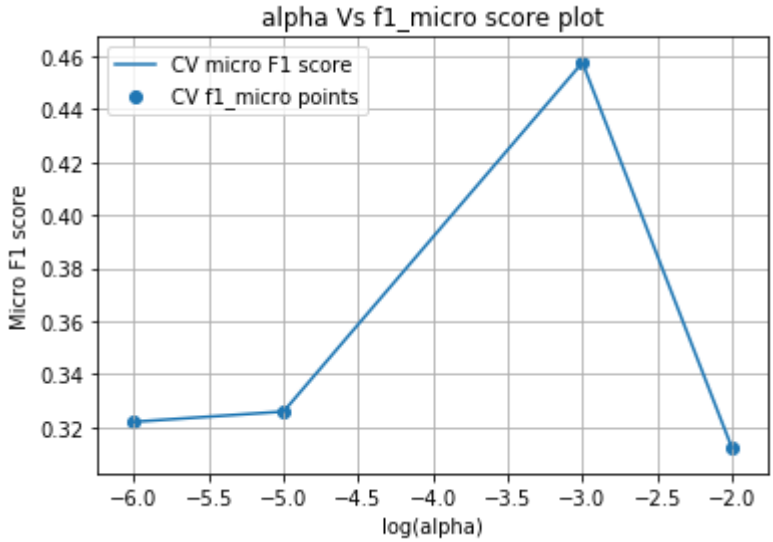
```
df1.head(2)
```

Out[25]:

	mean_fit_time	mean_score_time	mean_test_score	param_estimator__alpha	
0	6386.28904	8.251518	0.322091	1e-06	{'esti 1e-0
1	5909.25959	8.068128	0.325969	1e-05	{'esti 1e-0

In [30]:

```
#plotting
x=alpha.get('estimator__alpha')
y=df1['mean_test_score'].tolist()

plt.plot(np.log10(x), y, label='CV micro F1 score')

plt.scatter(np.log10(x), y, label='CV f1_micro points')
plt.legend()
plt.xlabel("log(alpha)")
plt.ylabel("Micro F1 score")
plt.title("alpha Vs f1_micro score plot")
plt.grid()
plt.show()
```



In [31]:

```
print(gs1.best_estimator_)
```

```
OneVsRestClassifier(estimator=SGDClassifier(alpha=0.001, average=False,
                                            class_weight=None,
                                            early_stopping=False, epsilon=
0.1,
                                            eta0=0.0, fit_intercept=True,
                                            l1_ratio=0.15,
                                            learning_rate='optimal', loss
='log',
                                            max_iter=1000, n_iter_no_chang
e=5,
                                            n_jobs=None, penalty='l1',
                                            power_t=0.5, random_state=Non
e,
                                            shuffle=True, tol=0.001,
                                            validation_fraction=0.1, verbo
se=0,
                                            warm_start=False),
                    n_jobs=None)
```

- From the above cell, we can conclude that alpha=0.001 is the best alpha yielding a high F1score

In [38]:

```
print("F1 score for the best alpha=0.001 =",gs1.best_score_)
```

```
F1 score for the best alpha=0.001 = 0.45750056628976976
```

5.2.3 Applying Best Alpha on train & test data

In [39]:

```
clf1 = OneVsRestClassifier(SGDClassifier(loss='log', alpha =0.001, penalty='l1'), n_job
s=-1)
clf1.fit(x_train_multilabel, y_train)
predictions = clf1.predict(x_test_multilabel)


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


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

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

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

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

print (metrics.classification_report(y_test, predictions))
```

```
Accuracy : 0.17186
Hamming loss  0.0034038
Micro-average quality numbers
Precision: 0.5261, Recall: 0.3203, F1-measure: 0.3981
Macro-average quality numbers
Precision: 0.3591, Recall: 0.2491, F1-measure: 0.2736
              precision    recall  f1-score   support

           0       0.81      0.53      0.64      2220
           1       0.52      0.10      0.17      3473
           2       0.80      0.29      0.43      3976
           3       0.82      0.61      0.70      2437
           4       0.74      0.38      0.50      2054
           5       0.67      0.54      0.60      2580
           6       0.82      0.49      0.61      1475
           7       0.63      0.19      0.29      1493
           8       0.68      0.56      0.61       957
           9       0.66      0.35      0.46      1781
          10       0.79      0.37      0.50      1568
          11       0.57      0.28      0.38      1477
          12       0.68      0.43      0.53       306
          13       0.34      0.28      0.31       916
          14       0.68      0.11      0.20      1161
          15       0.67      0.53      0.59       811
          16       0.68      0.60      0.64      1200
          17       0.78      0.49      0.60       686
          18       0.71      0.58      0.64       236
          19       0.72      0.54      0.62       680
          20       0.40      0.50      0.44      2233
          21       0.49      0.36      0.42      2785
          22       0.62      0.51      0.56       409
          23       0.47      0.29      0.36       533
          24       0.47      0.23      0.31       860
          25       0.45      0.38      0.41       391
          26       0.58      0.13      0.22       889
          27       0.52      0.54      0.53      1280
          28       0.20      0.10      0.14       739
          29       0.60      0.46      0.52       337
          30       0.05      0.07      0.06        54
          31       0.50      0.50      0.50       325
          32       0.27      0.27      0.27       274
          33       0.29      0.20      0.24       398
          34       0.61      0.29      0.39       305
          35       0.61      0.38      0.47       221
          36       0.66      0.47      0.55       486
          37       0.55      0.31      0.40       583
          38       0.27      0.34      0.30       340
          39       0.54      0.36      0.43        80
          40       0.72      0.53      0.61       243
          41       0.74      0.55      0.63       420
          42       0.68      0.40      0.50       685
          43       0.53      0.36      0.43       262
          44       0.61      0.61      0.61       283
          45       0.40      0.33      0.36       301
          46       0.48      0.20      0.28       507
          47       0.46      0.58      0.51        85
          48       0.21      0.11      0.14       280
          49       0.19      0.23      0.21       160
          50       0.28      0.17      0.21       419
          51       0.33      0.30      0.31       446
          52       0.00      0.00      0.00       242
```

480	0.44	0.07	0.13	54
481	0.64	0.37	0.47	68
482	0.45	0.36	0.40	14
483	0.00	0.00	0.00	31
484	0.62	0.11	0.19	45
485	0.00	0.00	0.00	49
486	0.00	0.00	0.00	25
487	0.00	0.00	0.00	50
488	0.80	0.36	0.50	11
489	0.39	0.16	0.23	67
490	0.29	0.19	0.23	42
491	0.53	0.21	0.31	42
492	0.00	0.00	0.00	19
493	0.61	0.60	0.61	78
494	0.45	0.28	0.34	18
495	0.05	0.03	0.04	31
496	0.00	0.00	0.00	161
497	0.00	0.00	0.00	35
498	0.00	0.00	0.00	34
499	0.33	0.10	0.15	10
micro avg	0.53	0.32	0.40	87885
macro avg	0.36	0.25	0.27	87885
weighted avg	0.52	0.32	0.38	87885
samples avg	0.36	0.30	0.30	87885

5.3 Linear SVM with hyperparameter tuning using Grid search

5.3.1 Hyperparameter tuning to find best alpha

In [17]:

```
start = datetime.now()

from sklearn.model_selection import GridSearchCV
from sklearn.metrics import f1_score

alpha = {'estimator__alpha': [10**-6, 10**-5, 10**-3, 10**-2]}

clf2=OneVsRestClassifier(SGDClassifier(loss='hinge', penalty='l1'))

gs2=GridSearchCV(estimator=clf2, param_grid=alpha, cv=3, scoring='f1_micro', n_jobs=-1)

model2=gs2.fit(x_train_multilabel,y_train)

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

```
Time taken to run this cell : 1:46:52.945907
```

5.3.2 2D plot

In [18]:

```
df2=pd.DataFrame(gs2.cv_results_)
df2.head(2)
```

Out[18]:

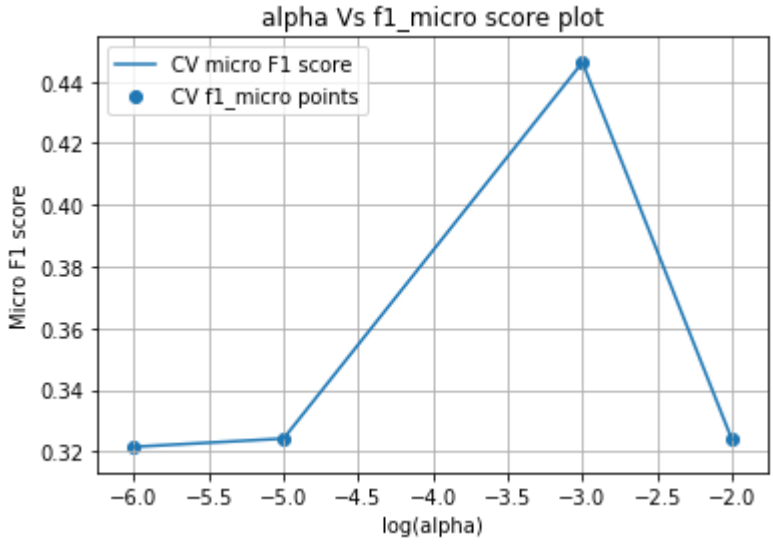
	mean_fit_time	mean_score_time	mean_test_score	param_estimator__alpha	
0	4946.956881	7.388797	0.321565	1e-06	{'esti 1e-0
1	4756.267300	6.367725	0.324241	1e-05	{'esti 1e-0

In [19]:

```
df2.to_csv(r'svm_cv.csv')
```

In [20]:

```
#plotting
x=alpha.get('estimator__alpha')
y=df2['mean_test_score'].tolist()

plt.plot(np.log10(x), y, label='CV micro F1 score')

plt.scatter(np.log10(x), y, label='CV f1_micro points')
plt.legend()
plt.xlabel("log(alpha)")
plt.ylabel("Micro F1 score")
plt.title("alpha Vs f1_micro score plot")
plt.grid()
plt.show()
```



In [21]:

```
print(gs2.best_estimator_)
```

```
OneVsRestClassifier(estimator=SGDClassifier(alpha=0.001, average=False,
                                            class_weight=None,
                                            early_stopping=False, epsilon=
0.1,
                                            eta0=0.0, fit_intercept=True,
                                            l1_ratio=0.15,
                                            learning_rate='optimal',
                                            loss='hinge', max_iter=1000,
                                            n_iter_no_change=5, n_jobs=Non
e,
                                            penalty='l1', power_t=0.5,
                                            random_state=None, shuffle=Tru
e,
                                            tol=0.001, validation_fraction
=0.1,
                                            verbose=0, warm_start=False),
                    n_jobs=None)
```

- From the above cell, we can conclude that alpha=0.001 is the best alpha yielding a high F1score

In [22]:

```
print("F1 score for the best alpha=0.001 =",gs2.best_score_)
```

```
F1 score for the best alpha=0.001 = 0.4461735251642685
```

5.3.3 Applying Best Alpha on train & test data

In [23]:

```
start = datetime.now()
clf3 = OneVsRestClassifier(SGDClassifier(loss='hinge', alpha= 0.001 , penalty='l1'), n_
jobs=-1)
clf3.fit(x_train_multilabel, y_train)
predictions = clf3.predict(x_test_multilabel)


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


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

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

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

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

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

```
Accuracy : 0.17434
Hamming loss  0.00336084
Micro-average quality numbers
Precision: 0.5401, Recall: 0.2963, F1-measure: 0.3826
Macro-average quality numbers
Precision: 0.2889, Recall: 0.2270, F1-measure: 0.2306
              precision    recall  f1-score   support

           0       0.87      0.59      0.70      2220
           1       0.68      0.03      0.05      3473
           2       0.80      0.27      0.40      3976
           3       0.81      0.64      0.71      2437
           4       0.74      0.32      0.45      2054
           5       0.68      0.54      0.60      2580
           6       0.76      0.56      0.65      1475
           7       0.61      0.20      0.31      1493
           8       0.67      0.52      0.58       957
           9       0.66      0.36      0.47      1781
          10       0.72      0.42      0.53      1568
          11       0.65      0.18      0.28      1477
          12       0.52      0.45      0.48       306
          13       0.00      0.00      0.00       916
          14       0.49      0.16      0.24      1161
          15       0.67      0.54      0.60       811
          16       0.63      0.62      0.62      1200
          17       0.64      0.65      0.64       686
          18       0.77      0.63      0.69       236
          19       0.82      0.43      0.56       680
          20       0.38      0.68      0.48      2233
          21       0.46      0.34      0.40      2785
          22       0.50      0.60      0.54       409
          23       0.54      0.15      0.23       533
          24       0.59      0.20      0.30       860
          25       0.73      0.32      0.44       391
          26       0.00      0.00      0.00       889
          27       0.00      0.00      0.00      1280
          28       0.11      0.00      0.00       739
          29       0.31      0.42      0.36       337
          30       0.00      0.00      0.00        54
          31       0.36      0.47      0.41       325
          32       0.37      0.27      0.31       274
          33       0.28      0.06      0.10       398
          34       0.70      0.31      0.43       305
          35       0.51      0.39      0.44       221
          36       0.60      0.42      0.49       486
          37       0.55      0.27      0.36       583
          38       0.59      0.26      0.37       340
          39       0.46      0.49      0.48        80
          40       0.66      0.48      0.55       243
          41       0.78      0.61      0.69       420
          42       0.81      0.33      0.47       685
          43       0.53      0.34      0.41       262
          44       0.64      0.59      0.62       283
          45       0.46      0.31      0.37       301
          46       0.41      0.18      0.25       507
          47       0.53      0.60      0.56        85
          48       0.00      0.00      0.00       280
          49       0.00      0.00      0.00       160
          50       0.00      0.00      0.00       419
          51       0.00      0.00      0.00       446
          52       0.00      0.00      0.00       242
```

```
           480       0.00      0.00      0.00        54
           481       0.42      0.38      0.40        68
           482       1.00      0.14      0.25        14
           483       0.20      0.06      0.10        31
           484       0.00      0.00      0.00        45
           485       0.00      0.00      0.00        49
           486       0.00      0.00      0.00        25
           487       0.00      0.00      0.00        50
           488       0.50      0.55      0.52        11
           489       0.13      0.21      0.16        67
           490       0.00      0.00      0.00        42
           491       0.26      0.31      0.28        42
           492       0.50      0.42      0.46        19
           493       0.00      0.00      0.00        78
           494       0.00      0.00      0.00        18
           495       0.00      0.00      0.00        31
           496       0.03      0.01      0.02       161
           497       0.00      0.00      0.00        35
           498       0.00      0.00      0.00        34
           499       0.38      0.50      0.43        10

   micro avg       0.54      0.30      0.38     87885
   macro avg       0.29      0.23      0.23     87885
weighted avg       0.46      0.30      0.33     87885
 samples avg       0.35      0.28      0.29     87885

Time taken to run this cell : 0:03:37.745036
```

6. Summary table of model performances

In [28]:

```
#Ref: http://zetcode.com/python/prettytable/
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Vectorizer","Models","Accuracy","Hamming-Loss","Precision","Recall",
"Micro F1 score"]
x.add_row(["TFIDF(upto 3-grams)", "OneVsRest-SGD Classifier(log)", 0.251, 0.0027,0.72,
0.33, 0.4488])
x.add_row(["BOW(upto 4-grams)", "OneVsRest-SGD Classifier(log)", 0.17, 0.0034, 0.53,0.3
2,0.40])
x.add_row(["BOW(upto 4-grams)", "OneVsRest-SGD Classifier(hinge)", 0.17, 0.0033, 0.54,
0.30,0.38])
print(x)
```

```
+---------------------+---------------------------------+----------+------
--------+-----------+--------+----------------+
|      Vectorizer     |              Models             | Accuracy | Hammi
ng-Loss | Precision | Recall | Micro F1 score |
+---------------------+---------------------------------+----------+------
--------+-----------+--------+----------------+
| TFIDF(upto 3-grams) |  OneVsRest-SGD Classifier(log)  |  0.251   |    0.
0027    |    0.72   |  0.33  |     0.4488     |
|  BOW(upto 4-grams)  |  OneVsRest-SGD Classifier(log)  |   0.17   |    0.
0034    |    0.53   |  0.32  |      0.4       |
|  BOW(upto 4-grams)  | OneVsRest-SGD Classifier(hinge) |   0.17   |    0.
0033    |    0.54   |  0.3   |      0.38      |
+---------------------+---------------------------------+----------+------
--------+-----------+--------+----------------+
```

- It can be observed from the above table that Logistic regression performed better marginally compared to linear SVM when BOW vectorization was used. The results could have been better if all 500K or 4.2 million data points were considered for training & testing

Procedure followed to solve this case study:

1. The stackoverflow tag predictor task was mapped to a ML problem as a "Multi-label classification" where there were close to 42k labels/tags. The performance metric chosen was micro F1 score.
2. In the EDA phase, duplicates in the data was checked and if present were eliminated. Then the tags were analysed for their uniqueness in the data, frequency of occurence,# of tags per question, most frequent tags using wordcloud & the top 30 tags just to get a jist of tag names that were occuring a lot.
3. Then came the data preprocessing stage where the text in title & body combined together were preprocessed to remove special characters, stopwords & html tags. All the characters were converted to lower case & stemming was done using snowball stemmer.
4. A technique called "Partial coverage" was implemented to reduce the enormous number of tags which was around 42K to a small number say 500 where those 500 tags were present in most of the data rows i.e. covering roughly about 90% of the data. By doing so, the computational power required to run the models reduced drastically.
5. Then the total data was split into train & test in 80:20 ratio. I considered 0.25 million datapoints before splitting keeping in the mind the computational constraints.
6. Then the text was featurized using BOW with max features as 25k.
7. The best value of alpha was found by hyperparameter tuning of the Onevsrestclassifier with SGDclassifier(loss=log) using grid search.
8. The best alpha was then used to predict the labels for test data and hence parameters like accuracy,hamming loss & f1 score was observed.
9. Steps 7 & 8 were repeated with a subtle change in the loss function of SGDclassifier where loos=hinge indicated linear SVM.
10. Performance metrics for all models and their respective featurizaions were summarized in a pretty table.