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 flask_sqlalchemy import SQLAlchemy
from sqlalchemy import create_engine # database connection
import sqlalchemy
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
print('Done importing all')
```

```
Done importing all
```

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 predic
t.

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 lo
wercase, 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
         return 0;    \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.
__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 [2]:

```
#Creating db file from csv
start = datetime.now()
disk_engine = create_engine('sqlite:///train.db')

start = dt.datetime.now()
chunksize = 100000
j = 0
index_start = 1
for df in pd.read_csv('Train.csv', names=['Id', 'Title', 'Body', 'Tags'], chunksize=chu
nksize, iterator=True, encoding='utf-8', ):
    df.index += index_start
    j+=1
    print('{} rows'.format(j*chunksize))
    df.to_sql('train_data_of_stackoverflow', disk_engine, if_exists='append')
    index_start = df.index[-1] + 1
    print("Time taken to run this cell :", datetime.now() - start)
```

```
100000 rows
Time taken to run this cell : 0:00:16.814435
200000 rows
Time taken to run this cell : 0:00:24.448987
300000 rows
Time taken to run this cell : 0:00:32.435646
400000 rows
Time taken to run this cell : 0:00:40.369646
500000 rows
Time taken to run this cell : 0:00:48.488472
600000 rows
Time taken to run this cell : 0:00:56.467731
700000 rows
Time taken to run this cell : 0:01:04.549541
800000 rows
Time taken to run this cell : 0:01:12.471692
900000 rows
Time taken to run this cell : 0:01:20.393455
1000000 rows
Time taken to run this cell : 0:01:28.205321
1100000 rows
Time taken to run this cell : 0:01:35.980612
1200000 rows
Time taken to run this cell : 0:01:43.631223
1300000 rows
Time taken to run this cell : 0:01:51.895985
1400000 rows
Time taken to run this cell : 0:01:59.021229
1500000 rows
Time taken to run this cell : 0:02:06.206323
1600000 rows
Time taken to run this cell : 0:02:13.053242
1700000 rows
Time taken to run this cell : 0:02:19.806309
1800000 rows
Time taken to run this cell : 0:02:26.529283
1900000 rows
Time taken to run this cell : 0:02:33.389793
2000000 rows
Time taken to run this cell : 0:02:40.286096
2100000 rows
Time taken to run this cell : 0:02:47.484527
2200000 rows
Time taken to run this cell : 0:02:54.515509
2300000 rows
Time taken to run this cell : 0:03:02.389808
2400000 rows
Time taken to run this cell : 0:03:09.348553
2500000 rows
Time taken to run this cell : 0:03:16.504127
2600000 rows
Time taken to run this cell : 0:03:23.473060
2700000 rows
Time taken to run this cell : 0:03:30.495699
2800000 rows
Time taken to run this cell : 0:03:37.421675
2900000 rows
Time taken to run this cell : 0:03:44.619511
3000000 rows
Time taken to run this cell : 0:03:51.736209
3100000 rows
```

```
Time taken to run this cell : 0:03:59.100239
3200000 rows
Time taken to run this cell : 0:04:06.150052
3300000 rows
Time taken to run this cell : 0:04:13.417716
3400000 rows
Time taken to run this cell : 0:04:20.678650
3500000 rows
Time taken to run this cell : 0:04:28.218265
3600000 rows
Time taken to run this cell : 0:04:35.977438
3700000 rows
Time taken to run this cell : 0:04:44.146600
3800000 rows
Time taken to run this cell : 0:04:51.271802
3900000 rows
Time taken to run this cell : 0:04:58.699664
4000000 rows
Time taken to run this cell : 0:05:05.817907
4100000 rows
Time taken to run this cell : 0:05:12.953025
4200000 rows
Time taken to run this cell : 0:05:20.199818
4300000 rows
Time taken to run this cell : 0:05:27.687480
4400000 rows
Time taken to run this cell : 0:05:34.998835
4500000 rows
Time taken to run this cell : 0:05:42.185017
4600000 rows
Time taken to run this cell : 0:05:49.634094
4700000 rows
Time taken to run this cell : 0:05:57.294978
4800000 rows
Time taken to run this cell : 0:06:05.639677
4900000 rows
Time taken to run this cell : 0:06:13.564874
5000000 rows
Time taken to run this cell : 0:06:21.073485
5100000 rows
Time taken to run this cell : 0:06:28.415967
5200000 rows
Time taken to run this cell : 0:06:36.274012
5300000 rows
Time taken to run this cell : 0:06:44.149100
5400000 rows
Time taken to run this cell : 0:06:51.557722
5500000 rows
Time taken to run this cell : 0:06:58.869096
5600000 rows
Time taken to run this cell : 0:07:06.003746
5700000 rows
Time taken to run this cell : 0:07:13.543783
5800000 rows
Time taken to run this cell : 0:07:20.853520
5900000 rows
Time taken to run this cell : 0:07:29.640555
6000000 rows
Time taken to run this cell : 0:07:36.477392
6100000 rows
Time taken to run this cell : 0:07:39.232582
```

3.1.2 Counting the number of rows

In [3]:

```
start = datetime.now()

#*************** Now we have a sqlite database, every time when we have to access it, j
ust use the 'connect' command*************

con = sqlite3.connect('train.db')
num_rows = pd.read_sql_query("""SELECT count(*) FROM train_data_of_stackoverflow""", co
n)
#Always remember to close the database


print("Number of rows in the database :","\n",num_rows['count(*)'].values[0])
con.close()
print("Time taken to count the number of rows :", datetime.now() - start)
```

```
Number of rows in the database :
 36205176
Time taken to count the number of rows : 0:00:07.908222
```

3.1.3 Checking for duplicates

In [4]:

```
#Learn SQL: https://www.w3schools.com/sql/default.asp
# if os.path.isfile('train.db'):
start = datetime.now()
con = sqlite3.connect('train.db')
df_no_dup = pd.read_sql('SELECT Title, Body, Tags, COUNT(*) as Count_duplicate_question
s FROM train_data_of_stackoverflow GROUP BY Title, Body, Tags', con)
con.close()
print("Time taken to run this cell :", datetime.now() - start)
```

```
---------------------------------------------------------------------------
-
OperationalError                          Traceback (most recent call las
t)
C:\anaconda\lib\site-packages\pandas\io\sql.py in execute(self, *args, **k
wargs)
   1594             else:
-> 1595                 cur.execute(*args)
   1596             return cur

OperationalError: database or disk is full

During handling of the above exception, another exception occurred:

DatabaseError                             Traceback (most recent call las
t)
<ipython-input-4-44509b173949> in <module>
      3 start = datetime.now()
      4 con = sqlite3.connect('train.db')
----> 5 df_no_dup = pd.read_sql('SELECT Title, Body, Tags, COUNT(*) as Cou
nt_duplicate_questions FROM train_data_of_stackoverflow GROUP BY Title, Bo
dy, Tags', con)
      6 con.close()
      7 print("Time taken to run this cell :", datetime.now() - start)

C:\anaconda\lib\site-packages\pandas\io\sql.py in read_sql(sql, con, index
_col, coerce_float, params, parse_dates, columns, chunksize)
    408             coerce_float=coerce_float,
    409             parse_dates=parse_dates,
--> 410             chunksize=chunksize,
    411         )
    412 

C:\anaconda\lib\site-packages\pandas\io\sql.py in read_query(self, sql, in
dex_col, coerce_float, params, parse_dates, chunksize)
   1643 
   1644         args = _convert_params(sql, params)
-> 1645         cursor = self.execute(*args)
   1646         columns = [col_desc[0] for col_desc in cursor.description]
   1647 

C:\anaconda\lib\site-packages\pandas\io\sql.py in execute(self, *args, **k
wargs)
   1608                 "Execution failed on sql '{sql}': {exc}".format(sq
l=args[0], exc=exc)
   1609             )
-> 1610             raise_with_traceback(ex)
   1611 
   1612     @staticmethod

C:\anaconda\lib\site-packages\pandas\compat\__init__.py in raise_with_trac
eback(exc, traceback)
     42     if traceback == Ellipsis:
     43         _, _, traceback = sys.exc_info()
---> 44     raise exc.with_traceback(traceback)
     45 
     46 

C:\anaconda\lib\site-packages\pandas\io\sql.py in execute(self, *args, **k
wargs)
   1593                 cur.execute(*args, **kwargs)
```

```
   1594             else:
-> 1595                 cur.execute(*args)
   1596             return cur
   1597         except Exception as exc:

DatabaseError: Execution failed on sql 'SELECT Title, Body, Tags, COUNT(*)
as Count_duplicate_questions FROM train_data_of_stackoverflow GROUP BY Tit
le, Body, Tags': database or disk is full
```

In [ ]:

```
df_no_dup.head()
# we can observe that there are duplicates
```

In [ ]:

```
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,"% )")


# From the 6 million  ,1.8 million are duplicates
```

In [ ]:

```
# number of times each question appeared in our database
df_no_dup.Count_duplicate_questions.value_counts()



# only 6 questions that are appear 5 times
# questions that appear 1 times are -> 2.6 millions   .
```

In [ ]:

```
df=df_no_dup
df.shape
```

In [ ]:

```
sd=[]
start = datetime.now()
for i in range(df_no_dup.shape[0]):
    f=df_no_dup["Tags"][i]# no of characters==0
    if f==None:# when no tag given just remove that datapoint
        df_no_dup=df_no_dup.drop(i,axis=0)     # remove this datapoint
    else:
        d=len(df_no_dup["Tags"][i].split(" "))
        sd.append(d)

print(datetime.now()-start)
```

In [ ]:

```
df_no_dup.shape
```

In [ ]:

```
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()
```

In [ ]:

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

Save the Non_duplicate questions in a new database

In [ ]:

```
#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)



#***********************************************train_no_dup.db is the new database ***
***********************************
```

In [ ]:

```
#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(" The Time taken to run this cell is :", datetime.now() - start)
# else:
#     print("Please download the train.db file from drive or run the above cells to gen
arate train.db file")
```

In [ ]:

```
tag_data.head()
#no_dup.head()
```

3.2 Analysis of Tags

3.2.1 Total number of unique tags

In [ ]:

```
#***************************************First we have to count (A tag appear how many
 times)  or  frequency of tags.
# this can be done by countvectorizer that can give us Tag_name : Frequency
# 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 [ ]:

```
print("Number of data points :", tag_dtm.shape[0])
print("Number of unique tags :", tag_dtm.shape[1])
# we have  42048 total  unique tags!
```

In [ ]:

```
#'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])
```

3.2.3 Number of times a tag appeared

In [ ]:

```
#    THIS IS THE REPRESENTATION OF THE DATAPOINTS WITH THEIR DIMENSIONS      (SPARCE M
ATRIX)

'''          TAG1     TAG2      TAG3      .    ..  ..     TAG42048
DP1       1          0               1                            0
DP2       0          0                1                            1
DP3        0         0                0                            1
.
.
DP4206307    0                  1                           1



for calculating how many times a single tag appeared, we have to count the number of on
e's in each column
'''



# https://stackoverflow.com/questions/15115765/how-to-access-sparse-matrix-elements
#Lets now store the document term matrix in a dictionary.


'''Each row in the array is one of your original documents (strings), each column is a
 feature (word),
and the element is the count for that particular word and document.
You can see that if you sum each column you'll get the correct number'''
freqs = tag_dtm.sum(axis=0).A1
result = dict(zip(tags, freqs))
```

In [ ]:

```
#***********************************Saving this dictionary  of tagsto csv files ******
****************************************

if not os.path.isfile('tag_counts_dict_dtm.csv'):
    with open('tag_counts_dict_dtm.csv', 'w') as csv_file:#   parameter is 'w' this mea
ns we are writing the file in the harddisk
        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()



# We are saving each and every thing to database or file, so that if our computer crash
es we can start from their-> where we left
```

In [ ]:

```
# ************************Sort the tags in DESC order, so that we can find the most f
requent tags*******************************



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

In [ ]:

```
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 [ ]:

```
# first 10k tags

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])#  :25 is the step sizes
```

Observations:

- Some Tags appear zero times,but its not much clear how many tags appear zero times, we have to zoom the plot.

In [ ]:

```
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])         # these are the step siz
es
```

In [ ]:

```
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])



# some tags are very huge in number  , some tags are very less in number.
```

Observations:

- Some Tags appear large number of times and some tags are appear very few times, so we can say micro average f1 is good matric for

  measuring performance.

In [ ]:

```
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")

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])
```

In [ ]:

```
# Store tags greater than 10K in one list
lst_tags_gt_10k = tag_df[tag_df.Counts>10000]
#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]
#Print the length of the list.
print ('{} Tags are used more than 100000 times'.format(len(lst_tags_gt_100k)))
```

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 [ ]:

```
#    THIS IS THE REPRESENTATION OF THE DATAPOINTS WITH THEIR DIMENSIONS       (SPARCE M
ATRIX)

'''         TAG1     TAG2      TAG3      .   ..  ..    TAG42048
DP1      1          0              1                            0
DP2      0          0               1                            1
DP3       0         0               0                            1
.
.
DP4206307   0                   1                            1



for calculating in one questions how many tags apear, just sum  the numer of ones in th
e single row.
'''

#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])
```

In [ ]:

```
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)))
```

In [ ]:

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

Observations:

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

3.2.5 Most Frequent Tags

In [ ]:

```
# Ploting word cloud
start = datetime.now()

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

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

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

3.2.6 The top 20 tags

In [ ]:

```
i=np.arange(20)
tag_df_sorted.head(20).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 [5]:

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

In [6]:

```
#*****************************************Some functions for databases***************
************************
#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()



#**********************************************Create a databse with the empty table**
**************************
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 [7]:

```
# 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'     # old database which has all the duplicates rows
write_db = 'Processed.db'        # new database which i make in this it has one table q
uestions_preprocessed
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 100000;")



  #*****************************We get the 100000 datapoints from the  train_no_dup.db d
atabase

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")  # rows are empty
 by the way
            print("Cleared All the rows")
print("Time taken to run this cell :", datetime.now() - start)



#*****************************Previously we created this table, now we checking if its e
mpty or not, if not emtpy delete al the rows*************************
```

```
Tables in the databse:
QuestionsProcessed
Cleared All the rows
Time taken to run this cell : 0:02:13.422156
```

In [8]:

```
import nltk
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Hp\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Out[8]:

```
True
```

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

In [9]:

```
#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: # reading one row

    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




    #*************************   We are inseting the updated preprocessed data to the n
ew table   'QuestionsProcessed'   ********************


    writer.execute("insert into QuestionsProcessed(question,code,tags,   words_pre,   w
ords_post,     is_code) values (?,?,?,?,?,?)",tup)
    if (questions_proccesed%100000==0):
        print("number of questions completed=",questions_proccesed)

no_dup_avg_len_pre=(len_pre*1.0)/questions_proccesed
no_dup_avg_len_post=(len_post*1.0)/questions_proccesed

print( "Avg. length of questions(Title+Body) before processing: %d"%no_dup_avg_len_pre)
print( "Avg. length of questions(Title+Body) after processing: %d"%no_dup_avg_len_post)
print ("Percent of questions containing code: %d"%((questions_with_code*100.0)/question
```

```
s_proccesed))

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

```
Avg. length of questions(Title+Body) before processing: 1175
Avg. length of questions(Title+Body) after processing: 326
Percent of questions containing code: 57
Time taken to run this cell : 0:05:20.314822
```

In [10]:

```
# 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 [11]:

```
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
==========================================================================
==========================
('c program dump entir hklm registri tree consol tri write simpl consol ap
p dump content hklm consol output look someth like much luck research help
would great appreci',)
--------------------------------------------------------------------------
--------------------------
('android gridview column make ui like net gridview column product name te
xtview product quantiti spinner price textview delet button button delet r
ow question best way control android sdk ui new android think gridview goo
d alreadi follow http www mkyong com android android gridview exampl tri u
se column spinner show text show littl spinner gridview',)
--------------------------------------------------------------------------
--------------------------
('import databas magento want import tecdoc databas magento without succes
s tecdoc msql format export csv xml problem import product keep schema dat
abas thank',)
--------------------------------------------------------------------------
--------------------------
('exampl libpcap libnet want captur ip packag one server forward packag an
oth server libnet exampl thank advanc',)
--------------------------------------------------------------------------
--------------------------
('getscript stylesheet jqueri titl say equival jqueri load stylesheet',)
--------------------------------------------------------------------------
--------------------------
('apach truncat static content tri set moinmoin offic wiki window server r
un apach origin thought everyth work fine except reason imag display prope
r turn static serv content get truncat charact tri figur error log show an
yth access log say file deliv either ok unchang dynam content seem display
ok untrunc django instal server also work normal might caus odd behaviour
curious bit math think point encod issu text file number charact miss equa
l number newlin charact truncat version file chang encod file seem help pu
t content onto one line come fine seem work around issu chang file would b
it crufti guess imag addendum see client use wireshark follow tcp stream f
unction main thing notic dynam content bgcwiki number newlin follow static
content alway even newlin particular png whitespac end content next get re
quest apach configur file moinmoin pretti standard',)
--------------------------------------------------------------------------
--------------------------
('googl map plot multipl marker array tri plot marker array use code pop c
urrent locat write array seem show besid copi part consol log convert arra
y json show show question plot latitud longitud marker map current method
seem go queri limit per second thank advanc',)
--------------------------------------------------------------------------
--------------------------
('perl event loop multipl block watcher tri figur event loop perl current
program someth like wait event block tri figur use ev anyev ae someth els
add anoth event watcher exampl want abl call tri someth everi second curre
nt stuck put event loop also would like add form interact program possibl
socket anoth watcher thank',)
--------------------------------------------------------------------------
--------------------------
('mvvmlight viewmodelloc regist dataservic question might look naiv unders
tand code viewmodelloc cs file see use dataservic get data wcf servic exam
pl assign mainviewmodel regist one viewmodel like let say anoth dataservic
dataservic exampl one use page viewmodel also someon help even give link r
ead code clue mean',)
--------------------------------------------------------------------------
--------------------------
```

In [12]:

```
#****************************From the Processed.db database select the table 'Questions
Processed'************************
#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 [13]:

```
preprocessed_data.head()
```

Out[13]:

	question	tags
0	user unabl access site connect vpn one user un...	networking vpn routing
1	c program dump entir hklm registri tree consol...	c# registry
2	android gridview column make ui like net gridv...	android gridview
3	import databas magento want import tecdoc data...	database magento import
4	exampl libpcap libnet want captur ip packag on...	linux libpcap libnet

In [14]:

```
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 : 99999
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 [15]:

```
# 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 [16]:

```
multilabel_y.shape# we have the total 18585 labels or tags.
```

Out[16]:

```
(99999, 18511)
```

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

In [17]:

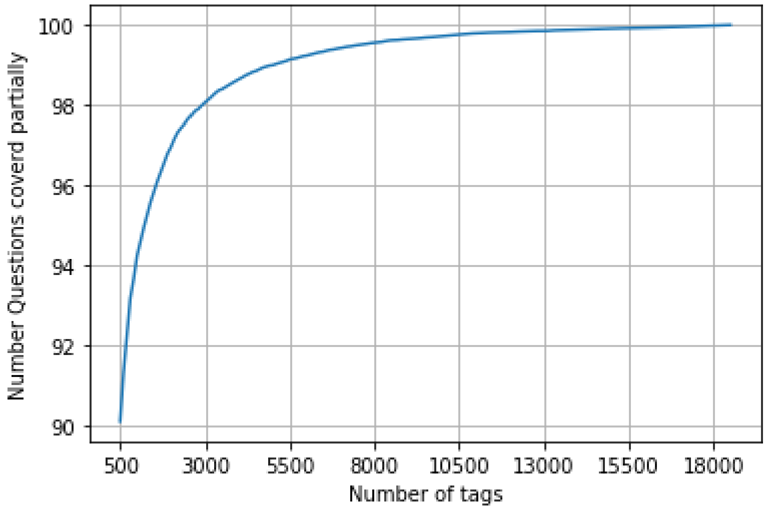
```
def tags_to_choose(n):
    t = multilabel_y.sum(axis=0).tolist()[0]# Frequency of the particular tag
count the columns in the binary vectorizer or bag of words
    #print(len(t))
    sorted_tags_i = sorted(range(len(t)), key=lambda i: t[i], reverse=True)# sort based
on the decending order of tags values (value is number of times it appear)
    #print(sorted_tags_i[:n])
    multilabel_yn=multilabel_y[:,sorted_tags_i[:n]]# questions with  the tags(that get
 in second step) or frequent tags
    #print('*******************************************************************')
    #print(multilabel_yn)
    return multilabel_yn

def questions_explained_fn(n):
    multilabel_yn = tags_to_choose(n)# tags output that i discussed
    x= multilabel_yn.sum(axis=1)#  how many tags a single quesition has !
    #print(x)
    return ((np.count_nonzero(x==0)))# that questions we not able to explain with the l
abels
```

In [18]:

```
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("Number Questions coverd partially")

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




```
with  5500 tags we are covering  99.138 % of questions
```

In [20]:

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

```
number of questions that are not covered : 862 out of  99999
(99999, 5500)
```

Out[20]:

```
(99999, 2)
```

In [21]:

```
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 : 18511
number of tags taken : 5500 ( 29.71206309761763 %)
```

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

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

In [22]:

```
# If we given with the time, we will do teh time split. because tags are changing with
 the time,, may be first asp.1 versoin we had, now today new version
# launched asp.2   . so time based splitting will work here,



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)
print(x_train.shape)
print(x_test.shape)
y_train = multilabel_yx[0:train_size,:]
y_test = multilabel_yx[train_size:total_size,:]
```

```
(79999, 2)
(20000, 2)
```

In [23]:

```
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 : (79999, 5500)
Number of data points in test data : (20000, 5500)
```

4.3 Featurizing data

In [24]:

```
start = datetime.now()
vectorizer = TfidfVectorizer(min_df=0.00009, max_features=50000, smooth_idf=True, norm=
"l2", \
                              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 : 0:01:03.359503
```

In [25]:

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

```
Dimensions of train data X: (79999, 50000) Y : (79999, 5500)
Dimensions of test data X: (20000, 50000) Y: (20000, 5500)
```

In [26]:

```
# 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[26]:

```
"\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"
```

4.5 Modeling with less data points (0.1M data points) and more weight to title and 500 tags only.

In [27]:

```
# Now we'll repeat all the code from the previous sections
# procedure
#1. Take less datapoints
#2. remove the questions and give the high weitage to the title, by just repeating it 3
times.  Also with this we can reduce the dimensions.
#3.If we see logically think, users have to write the title so much attractive or Title
have to cover the overall view of our error, so it can be useful.


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("Titlemoreweightw.db", sql_create_table)
```

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

In [28]:

```
# 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 = 'Titlemoreweightw.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 100000;")
        # 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 [29]:

```
#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)
```

```
Avg. length of questions(Title+Body) before processing: 1232
Avg. length of questions(Title+Body) after processing: 441
Percent of questions containing code: 57
Time taken to run this cell : 0:07:51.700574
```

In [30]:

```
# 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 [31]:

```
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 [32]:

```
#Taking 0.5 Million entries to a dataframe.
write_db = 'Titlemoreweightw.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 [33]:

```
preprocessed_data.shape
```

Out[33]:

```
(99999, 2)
```

In [34]:

```
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 : 99999
number of dimensions : 2
```

Converting string Tags to multilable output variables

In [35]:

```
vectorizer = CountVectorizer(binary='true')
multilabel_y = vectorizer.fit_transform(preprocessed_data['tags'])
```

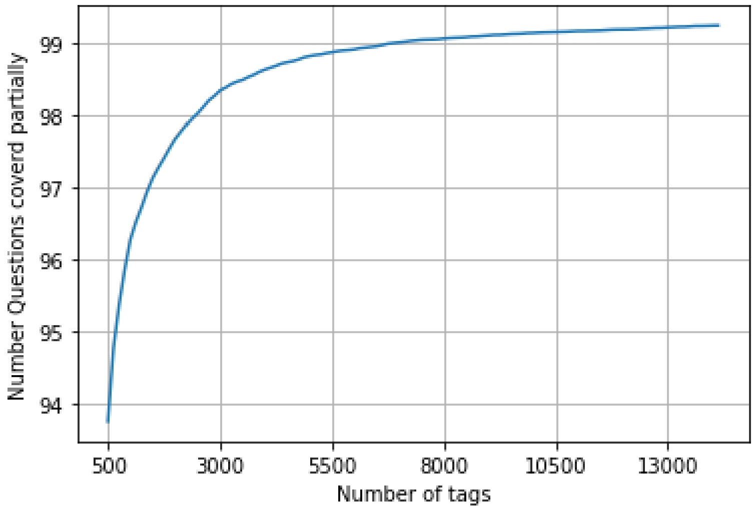
Selecting 500 Tags

In [36]:

```
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 [37]:

```
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  98.986 % of questions
with  500 tags we are covering  93.743 % of questions
```

In [38]:

```
# 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 : 6257 out of  99999
```

In [39]:

```
preprocessed_data.shape[0]
```

Out[39]:

```
99999
```

In [40]:

```
# If we given with the time, we will do teh time split. because tags are changing with
 the time,, may be first asp.1 versoin we had, now today new version
# launched asp.2   . so time based splitting will work here,


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)
print(x_train.shape)
print(x_test.shape)
y_train = multilabel_yx[0:train_size,:]
y_test = multilabel_yx[train_size:total_size,:]
```

```
(79999, 2)
(20000, 2)
```

In [41]:

```
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 : (79999, 500)
Number of data points in test data : (20000, 500)
```

4.5.2 Featurizing data with TfIdf vectorizer

In [42]:

```
start = datetime.now()
vectorizer = TfidfVectorizer(min_df=0.00009, max_features=10000, smooth_idf=True, norm=
"l2", 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 : 0:01:16.136466
```

In [43]:

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

```
Dimensions of train data X: (79999, 10000) Y : (79999, 500)
Dimensions of test data X: (20000, 10000) Y: (20000, 500)
```

4.5.3 Applying Logistic Regression with OneVsRest Classifier

In [44]:

```
import warnings
warnings.filterwarnings("ignore")
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 : 0.1937
Hamming loss  0.0035708
Micro-average quality numbers
Precision: 0.7346, Recall: 0.3800, F1-measure: 0.5009
Macro-average quality numbers
Precision: 0.5558, Recall: 0.2813, F1-measure: 0.3510
              precision    recall  f1-score   support

           0       0.80      0.47      0.59      1805
           1       0.86      0.53      0.65      1186
           2       0.87      0.55      0.68       484
           3       0.82      0.46      0.59      1323
           4       0.87      0.60      0.71       739
           5       0.87      0.48      0.62      1023
           6       0.77      0.39      0.52      1421
           7       0.95      0.62      0.75      1450
           8       0.98      0.82      0.89      1368
           9       0.68      0.45      0.54       914
          10       0.80      0.41      0.55       186
          11       0.77      0.49      0.60       553
          12       0.78      0.40      0.53       644
          13       0.52      0.19      0.28       424
          14       0.70      0.39      0.50        36
          15       0.59      0.37      0.45       352
          16       0.64      0.23      0.34       437
          17       0.76      0.46      0.57       435
          18       0.68      0.56      0.61       153
          19       0.98      0.60      0.75       727
          20       0.63      0.19      0.30       488
          21       0.85      0.62      0.72       272
          22       0.92      0.58      0.71       530
          23       0.95      0.54      0.69       618
          24       0.96      0.55      0.70       614
          25       0.68      0.29      0.40       231
          26       0.53      0.33      0.41       588
          27       0.58      0.40      0.47      1224
          28       0.71      0.45      0.55       165
          29       0.62      0.54      0.58       231
          30       0.72      0.28      0.40       190
          31       0.82      0.59      0.69       296
          32       0.69      0.34      0.46       274
          33       0.56      0.38      0.45       292
          34       0.73      0.27      0.40       190
          35       0.86      0.44      0.59        99
          36       0.88      0.59      0.71       357
          37       0.69      0.38      0.49       870
          38       0.81      0.47      0.60       135
          39       1.00      0.35      0.52        17
          40       0.53      0.08      0.14        99
          41       0.67      0.29      0.40       176
          42       0.29      0.05      0.09       236
          43       0.88      0.32      0.47        22
          44       0.53      0.19      0.28       106
          45       0.56      0.13      0.22       178
          46       0.43      0.24      0.30       241
          47       0.64      0.17      0.27       217
          48       0.64      0.49      0.55       223
          49       0.67      0.07      0.13        54
          50       0.62      0.35      0.44        92
          51       0.86      0.59      0.70       203
          52       0.71      0.47      0.57       116
```

53	0.81	0.49	0.61	72
54	0.38	0.20	0.26	15
55	0.25	0.02	0.03	60
56	0.90	0.79	0.84	216
57	0.35	0.08	0.13	74
58	0.35	0.13	0.19	139
59	0.71	0.45	0.55	91
60	0.48	0.10	0.17	156
61	0.42	0.33	0.37	76
62	0.52	0.18	0.27	89
63	0.48	0.17	0.25	173
64	0.53	0.28	0.36	227
65	0.45	0.11	0.18	383
66	0.65	0.22	0.32	148
67	0.56	0.40	0.46	189
68	0.75	0.35	0.48	169
69	0.14	0.06	0.08	50
70	0.68	0.26	0.38	145
71	0.42	0.26	0.32	31
72	0.93	0.72	0.81	141
73	0.88	0.43	0.58	246
74	0.54	0.30	0.39	210
75	0.70	0.10	0.18	159
76	0.49	0.21	0.30	108
77	0.94	0.78	0.86	65
78	0.97	0.70	0.81	145
79	0.91	0.71	0.79	41
80	0.73	0.57	0.64	129
81	0.89	0.53	0.66	76
82	0.63	0.45	0.53	124
83	0.41	0.13	0.20	69
84	0.44	0.16	0.24	91
85	0.49	0.42	0.46	66
86	0.21	0.08	0.12	100
87	0.43	0.26	0.33	38
88	0.73	0.45	0.56	98
89	0.52	0.39	0.45	38
90	0.97	0.68	0.80	154
91	0.88	0.65	0.75	152
92	0.00	0.00	0.00	13
93	0.00	0.00	0.00	47
94	0.80	0.27	0.41	44
95	0.78	0.29	0.43	200
96	0.40	0.24	0.30	25
97	0.61	0.28	0.39	39
98	0.58	0.43	0.49	51
99	0.35	0.26	0.30	43
100	0.33	0.11	0.16	211
101	0.57	0.22	0.32	18
102	0.67	0.50	0.57	32
103	0.77	0.42	0.54	24
104	0.80	0.29	0.42	14
105	0.70	0.48	0.57	96
106	1.00	0.41	0.58	32
107	0.60	0.38	0.46	80
108	0.74	0.19	0.31	160
109	0.39	0.07	0.12	123
110	0.37	0.05	0.09	202
111	0.56	0.46	0.51	39
112	0.35	0.07	0.11	123
113	0.71	0.53	0.60	55

114	0.45	0.13	0.20	98
115	0.35	0.16	0.22	50
116	0.84	0.54	0.65	275
117	0.40	0.04	0.07	101
118	0.67	0.12	0.20	50
119	0.57	0.20	0.29	41
120	0.62	0.27	0.37	98
121	0.44	0.13	0.21	30
122	0.83	0.33	0.47	73
123	0.91	0.79	0.85	121
124	0.55	0.38	0.45	29
125	0.92	0.21	0.34	57
126	0.50	0.15	0.23	48
127	0.90	0.75	0.82	24
128	0.48	0.25	0.33	48
129	0.75	0.19	0.30	48
130	0.89	0.51	0.65	99
131	0.50	0.38	0.43	29
132	0.45	0.08	0.14	60
133	0.71	0.74	0.73	89
134	0.36	0.04	0.08	113
135	0.38	0.13	0.19	70
136	0.38	0.07	0.12	68
137	0.94	0.55	0.70	146
138	0.79	0.33	0.47	66
139	0.38	0.06	0.11	49
140	0.89	0.47	0.62	51
141	0.56	0.33	0.42	27
142	0.20	0.04	0.06	54
143	0.50	0.10	0.16	21
144	0.40	0.14	0.21	43
145	0.95	0.41	0.57	49
146	0.64	0.54	0.58	137
147	0.84	0.47	0.61	91
148	0.48	0.34	0.40	29
149	0.95	0.62	0.75	88
150	0.70	0.10	0.18	67
151	0.70	0.41	0.52	46
152	0.59	0.33	0.42	187
153	0.81	0.42	0.55	60
154	0.83	0.38	0.52	40
155	0.38	0.04	0.08	67
156	0.33	0.11	0.16	46
157	0.64	0.30	0.41	23
158	0.68	0.50	0.57	54
159	0.46	0.37	0.41	87
160	0.70	0.21	0.33	66
161	0.88	0.54	0.67	69
162	0.41	0.15	0.22	78
163	0.98	0.82	0.89	50
164	0.39	0.11	0.18	115
165	0.65	0.18	0.29	71
166	0.12	0.01	0.02	81
167	0.40	0.52	0.45	52
168	0.62	0.36	0.46	22
169	0.00	0.00	0.00	292
170	0.32	0.40	0.35	45
171	0.31	0.03	0.06	146
172	0.00	0.00	0.00	5
173	0.53	0.30	0.38	66
174	0.30	0.14	0.19	21

175	0.50	0.08	0.13	26
176	0.42	0.09	0.15	86
177	0.43	0.17	0.24	18
178	0.12	0.04	0.06	27
179	0.00	0.00	0.00	0
180	1.00	0.71	0.83	7
181	1.00	0.53	0.69	34
182	0.73	0.63	0.68	35
183	0.68	0.51	0.58	51
184	0.89	0.63	0.74	38
185	0.20	0.05	0.08	39
186	0.50	0.08	0.13	13
187	0.60	0.34	0.44	35
188	0.31	0.11	0.17	44
189	0.50	0.11	0.18	46
190	0.69	0.17	0.28	52
191	0.48	0.11	0.18	88
192	0.25	0.02	0.04	41
193	0.96	0.53	0.69	88
194	0.50	0.04	0.07	51
195	0.55	0.20	0.30	127
196	0.00	0.00	0.00	60
197	1.00	0.17	0.29	18
198	0.33	0.03	0.05	36
199	0.19	0.04	0.06	85
200	0.50	0.19	0.27	48
201	0.45	0.29	0.36	17
202	0.40	0.22	0.29	27
203	0.65	0.18	0.29	60
204	0.82	0.50	0.62	105
205	0.64	0.50	0.56	50
206	0.55	0.27	0.36	45
207	0.40	0.32	0.35	19
208	0.57	0.27	0.37	73
209	0.00	0.00	0.00	51
210	0.80	0.20	0.32	20
211	0.00	0.00	0.00	47
212	0.00	0.00	0.00	44
213	0.63	0.35	0.45	34
214	0.72	0.49	0.58	106
215	0.79	0.44	0.57	59
216	0.33	0.10	0.16	87
217	0.80	0.26	0.39	31
218	0.74	0.61	0.67	46
219	0.60	0.11	0.19	27
220	0.27	0.08	0.12	39
221	0.75	0.38	0.51	55
222	0.67	0.12	0.20	34
223	0.67	0.36	0.47	11
224	0.35	0.12	0.18	51
225	0.18	0.07	0.10	46
226	0.50	0.09	0.15	47
227	0.25	0.07	0.11	14
228	0.83	0.24	0.37	21
229	0.62	0.07	0.13	67
230	0.00	0.00	0.00	229
231	0.67	0.11	0.19	54
232	0.77	0.10	0.18	98
233	0.92	0.43	0.59	53
234	0.57	0.22	0.32	36
235	0.68	0.47	0.56	53

236	0.51	0.34	0.41	68
237	0.31	0.13	0.19	38
238	0.46	0.11	0.17	102
239	0.33	0.33	0.33	6
240	0.00	0.00	0.00	5
241	0.50	0.33	0.40	3
242	0.50	0.13	0.21	68
243	0.50	0.43	0.46	91
244	0.92	0.73	0.81	30
245	0.79	0.22	0.34	50
246	1.00	0.25	0.40	4
247	0.65	0.27	0.38	41
248	0.64	0.21	0.32	98
249	0.00	0.00	0.00	0
250	1.00	1.00	1.00	1
251	1.00	0.19	0.32	26
252	0.66	0.29	0.40	66
253	0.79	0.66	0.72	67
254	0.00	0.00	0.00	32
255	0.00	0.00	0.00	2
256	0.60	0.09	0.16	32
257	1.00	0.50	0.67	4
258	0.75	0.08	0.14	39
259	0.85	0.45	0.59	73
260	1.00	0.60	0.75	55
261	0.50	0.33	0.40	12
262	0.44	0.27	0.33	41
263	0.71	0.36	0.48	14
264	0.69	0.16	0.26	56
265	0.86	0.23	0.37	77
266	0.00	0.00	0.00	13
267	0.45	0.31	0.37	16
268	0.00	0.00	0.00	34
269	0.00	0.00	0.00	45
270	1.00	0.07	0.13	43
271	0.44	0.29	0.35	56
272	0.60	0.27	0.37	11
273	0.00	0.00	0.00	42
274	0.85	0.63	0.72	35
275	0.44	0.07	0.12	59
276	0.29	0.10	0.15	49
277	0.63	0.66	0.64	44
278	0.56	0.11	0.18	46
279	0.00	0.00	0.00	7
280	0.88	0.66	0.75	58
281	0.67	0.35	0.46	46
282	0.36	0.40	0.38	10
283	0.58	0.33	0.42	21
284	0.25	0.04	0.07	47
285	0.57	0.17	0.27	23
286	0.92	0.69	0.79	48
287	0.58	0.60	0.59	35
288	0.15	0.02	0.04	81
289	0.73	0.47	0.57	47
290	0.73	0.71	0.72	93
291	0.10	0.02	0.03	61
292	0.70	0.61	0.65	23
293	0.83	0.50	0.62	10
294	0.50	0.03	0.06	30
295	0.00	0.00	0.00	24
296	0.00	0.00	0.00	54

297	0.56	0.65	0.60	34
298	0.37	0.33	0.35	69
299	0.87	0.75	0.80	44
300	0.71	0.38	0.50	13
301	0.88	0.54	0.67	68
302	0.00	0.00	0.00	33
303	0.62	0.44	0.52	18
304	0.20	0.08	0.11	13
305	0.75	0.34	0.47	53
306	0.73	0.21	0.33	75
307	0.88	0.53	0.66	55
308	0.95	0.61	0.74	61
309	0.80	0.41	0.54	90
310	0.56	0.09	0.15	58
311	0.89	0.84	0.86	19
312	0.67	0.06	0.11	34
313	0.40	0.31	0.35	13
314	0.20	0.25	0.22	4
315	0.44	0.10	0.16	41
316	0.81	0.41	0.54	54
317	0.86	0.24	0.38	25
318	0.20	0.25	0.22	4
319	0.40	0.07	0.12	29
320	0.62	0.22	0.32	37
321	1.00	0.17	0.29	6
322	0.14	0.05	0.07	22
323	0.25	0.05	0.09	19
324	0.20	0.25	0.22	4
325	0.54	0.39	0.45	18
326	0.75	0.43	0.55	21
327	0.00	0.00	0.00	26
328	0.72	0.47	0.57	49
329	0.61	0.54	0.58	35
330	1.00	0.05	0.10	19
331	0.60	0.20	0.30	15
332	0.00	0.00	0.00	10
333	0.74	0.53	0.62	38
334	0.14	0.11	0.12	9
335	0.60	0.06	0.10	53
336	1.00	0.56	0.72	32
337	0.33	0.04	0.07	24
338	1.00	0.67	0.80	3
339	0.00	0.00	0.00	1
340	0.00	0.00	0.00	0
341	0.71	0.45	0.56	11
342	0.68	0.47	0.56	40
343	0.00	0.00	0.00	30
344	0.40	0.08	0.14	24
345	0.50	0.04	0.08	23
346	0.61	0.28	0.38	69
347	0.20	0.06	0.09	18
348	0.17	0.03	0.05	65
349	0.47	0.23	0.31	78
350	1.00	0.08	0.15	12
351	0.50	0.08	0.13	13
352	0.40	0.11	0.17	18
353	1.00	0.63	0.77	46
354	0.82	0.57	0.68	40
355	0.00	0.00	0.00	19
356	0.67	0.08	0.14	26
357	0.53	0.23	0.32	39

358	1.00	0.17	0.29	12
359	0.60	0.19	0.29	16
360	0.70	0.29	0.41	24
361	0.33	0.11	0.16	57
362	0.84	0.80	0.82	20
363	0.83	0.06	0.11	84
364	0.73	0.65	0.69	54
365	0.44	0.12	0.19	33
366	0.67	0.13	0.22	30
367	1.00	0.07	0.12	30
368	0.20	0.05	0.08	19
369	0.00	0.00	0.00	19
370	1.00	0.03	0.06	32
371	0.62	0.42	0.50	12
372	0.50	0.07	0.12	15
373	0.12	0.07	0.09	15
374	0.92	0.65	0.76	17
375	1.00	0.66	0.79	41
376	0.94	0.55	0.70	29
377	0.00	0.00	0.00	28
378	0.50	0.16	0.24	19
379	0.40	0.06	0.11	31
380	0.67	0.14	0.23	29
381	0.29	0.08	0.13	49
382	0.00	0.00	0.00	8
383	0.29	0.08	0.13	24
384	0.50	0.35	0.41	20
385	0.00	0.00	0.00	15
386	0.81	0.57	0.67	37
387	0.00	0.00	0.00	22
388	1.00	0.04	0.07	27
389	0.50	0.38	0.43	29
390	0.00	0.00	0.00	20
391	0.72	0.54	0.62	39
392	0.50	0.10	0.17	10
393	0.38	0.14	0.21	42
394	0.67	0.09	0.15	46
395	0.10	0.10	0.10	10
396	0.75	0.08	0.14	39
397	0.00	0.00	0.00	43
398	0.71	0.30	0.42	50
399	1.00	0.57	0.73	7
400	0.25	0.06	0.10	17
401	1.00	0.17	0.29	6
402	0.00	0.00	0.00	26
403	1.00	0.10	0.18	10
404	0.71	0.36	0.48	14
405	0.00	0.00	0.00	14
406	0.82	0.41	0.55	22
407	0.62	0.17	0.26	60
408	0.39	0.17	0.24	40
409	0.00	0.00	0.00	31
410	0.38	0.33	0.35	9
411	0.42	0.26	0.32	19
412	0.67	0.53	0.59	19
413	0.50	0.20	0.29	5
414	0.33	0.08	0.13	12
415	1.00	0.66	0.79	29
416	0.67	0.06	0.11	33
417	0.33	0.03	0.06	33
418	0.40	0.17	0.24	12

419	0.36	0.10	0.15	42
420	0.50	0.58	0.54	12
421	0.33	0.18	0.24	98
422	0.33	0.12	0.18	8
423	0.00	0.00	0.00	7
424	0.75	0.46	0.57	13
425	0.33	0.08	0.12	13
426	0.33	0.10	0.15	20
427	0.25	0.05	0.09	58
428	0.67	1.00	0.80	2
429	0.38	0.30	0.33	27
430	0.50	0.37	0.42	38
431	0.56	0.23	0.32	40
432	1.00	0.05	0.09	43
433	0.96	0.57	0.72	42
434	0.64	0.29	0.40	24
435	0.33	0.03	0.06	31
436	0.40	0.33	0.36	30
437	0.25	0.06	0.10	16
438	0.67	0.45	0.54	22
439	1.00	1.00	1.00	1
440	0.17	0.11	0.13	19
441	0.67	0.22	0.33	9
442	0.33	0.11	0.17	100
443	0.83	0.36	0.50	28
444	0.75	0.60	0.67	20
445	0.45	0.45	0.45	29
446	0.00	0.00	0.00	21
447	0.80	0.20	0.32	20
448	0.88	0.55	0.68	38
449	0.00	0.00	0.00	22
450	0.61	0.52	0.56	21
451	0.00	0.00	0.00	13
452	0.00	0.00	0.00	24
453	0.55	0.12	0.20	48
454	0.47	0.11	0.17	75
455	0.00	0.00	0.00	18
456	0.00	0.00	0.00	3
457	0.55	0.46	0.50	13
458	0.50	0.15	0.24	13
459	0.27	0.25	0.26	24
460	0.62	0.28	0.38	36
461	0.64	0.50	0.56	18
462	0.50	0.23	0.31	31
463	0.67	0.07	0.13	28
464	0.00	0.00	0.00	7
465	0.89	0.30	0.44	27
466	1.00	0.83	0.91	12
467	0.67	0.14	0.24	14
468	0.00	0.00	0.00	6
469	0.27	0.18	0.21	17
470	0.30	0.17	0.21	18
471	0.67	0.07	0.12	29
472	0.00	0.00	0.00	2
473	0.38	0.09	0.14	34
474	0.00	0.00	0.00	8
475	0.25	0.25	0.25	4
476	0.69	0.50	0.58	22
477	0.50	0.67	0.57	6
478	0.33	0.24	0.28	17
479	0.00	0.00	0.00	23

```
         480       0.86      0.33      0.48        18
         481       0.83      0.45      0.59        11
         482       1.00      0.29      0.44        35
         483       0.59      0.62      0.60        21
         484       0.86      0.64      0.73        28
         485       0.62      0.36      0.45        14
         486       0.90      0.82      0.86        11
         487       1.00      0.13      0.24        15
         488       0.58      0.18      0.28        38
         489       0.08      0.01      0.02        75
         490       0.97      0.57      0.72        51
         491       1.00      0.68      0.81        19
         492       0.50      0.19      0.28        21
         493       0.67      0.12      0.21        16
         494       1.00      0.83      0.91         6
         495       0.40      0.18      0.25        22
         496       0.68      0.35      0.46        37
         497       0.29      0.20      0.24        20
         498       0.70      0.58      0.64        24
         499       0.00      0.00      0.00        17

   micro avg       0.73      0.38      0.50     47151
   macro avg       0.56      0.28      0.35     47151
weighted avg       0.68      0.38      0.47     47151
 samples avg       0.51      0.37      0.40     47151

Time taken to run this cell : 0:02:47.896331
```

In [45]:

```
# For saving the weights or results after run applying model
#joblib.dump(classifier, 'lr_with_more_title_weight.pkl')
```

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)

4.5.2 Featurizing data with BOW vectorizer

In [46]:

```
start = datetime.now()
vectorizer =  CountVectorizer(min_df=0.00009, max_features=10000, ngram_range=(1,4))
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 : 0:02:01.153712
```

In [47]:

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

```
Dimensions of train data X: (79999, 10000) Y : (79999, 500)
Dimensions of test data X: (20000, 10000) Y: (20000, 500)
```

Hyperparameter tuning:

In [49]:

```
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
#from sklearn.grid_search import GridSearchCV"
from sklearn.linear_model import LogisticRegression
from tqdm import tqdm

from sklearn.model_selection import learning_curve, GridSearchCV
```

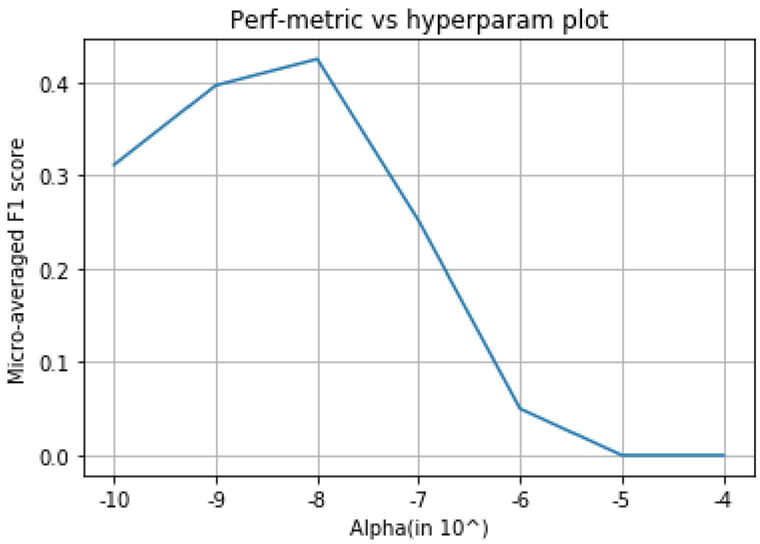
In [50]:

```
alpha =[10**-5,10**-4,10**-3,10**-2,10**-1,5,10]
perf_metric = []
for i in tqdm(alpha):

    clf = OneVsRestClassifier(SGDClassifier(loss='log', alpha=i, penalty='l1', random_s
tate=42))
    clf.fit(x_train_multilabel, y_train)
    predictions = clf.predict (x_test_multilabel)
    perf_metric.append(f1_score(y_test, predictions, average='micro'))

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

```
100%|██████████████████████████████████████████████████████████████████
██████████| 7/7 [3:24:36<00:00, 1753.72s/it]
```

In [51]:

```
# plot the perf metric for each hyperparam(alpha)
fig, ax = plt.subplots()
ax.plot(perf_metric)
xlabel = list(range(-11, -3))
ax.set_xticklabels(xlabel)
plt.title("Perf-metric vs hyperparam plot")
plt.xlabel("Alpha(in 10^)")
plt.ylabel("Micro-averaged F1 score")
plt.grid()
plt.show()
```



Training the model with best hyperparameter

In [52]:

```
start = datetime.now()
# fetching the best alpha
best_alpha = alpha[np.argmax(perf_metric)]
print('Best hyperparam(alpha) : ',best_alpha)

# train the LR model with the best alpha
classifier = OneVsRestClassifier(SGDClassifier(loss='log', alpha=best_alpha, penalty='l
1',  random_state=42), n_jobs=-1)
classifier.fit(x_train_multilabel, y_train)
predictions = classifier.predict (x_test_multilabel)

# print the various performance metrices
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("\nMicro-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("\nMacro-average quality numbers -")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall,
f1))

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

```
Best hyperparam(alpha) :  0.001
Accuracy : 0.1366
Hamming loss : 0.0044873

Micro-average quality numbers -
Precision: 0.5368, Recall: 0.3521, F1-measure: 0.4253

Macro-average quality numbers -
Precision: 0.3906, Recall: 0.2561, F1-measure: 0.2861
```

	precision	recall	f1-score	support
0	0.77	0.44	0.56	1805
1	0.83	0.47	0.60	1186
2	0.74	0.56	0.64	484
3	0.81	0.43	0.56	1323
4	0.86	0.59	0.70	739
5	0.88	0.47	0.62	1023
6	0.67	0.41	0.51	1421
7	0.84	0.64	0.73	1450
8	0.92	0.57	0.71	1368
9	0.55	0.41	0.47	914
10	0.59	0.46	0.52	186
11	0.73	0.50	0.59	553
12	0.73	0.39	0.51	644
13	0.46	0.16	0.24	424
14	0.51	0.56	0.53	36
15	0.43	0.43	0.43	352
16	0.49	0.27	0.34	437
17	0.62	0.42	0.50	435
18	0.60	0.42	0.50	153
19	0.89	0.61	0.73	727
20	0.46	0.18	0.25	488
21	0.71	0.37	0.49	272
22	0.77	0.67	0.71	530
23	0.89	0.55	0.68	618
24	0.89	0.54	0.67	614
25	0.56	0.23	0.33	231
26	0.60	0.25	0.35	588
27	0.13	0.21	0.16	1224
28	0.68	0.41	0.51	165
29	0.39	0.60	0.47	231
30	0.56	0.26	0.36	190
31	0.72	0.70	0.71	296
32	0.59	0.35	0.44	274
33	0.50	0.36	0.42	292
34	0.66	0.34	0.45	190
35	0.58	0.25	0.35	99
36	0.86	0.55	0.67	357
37	0.11	0.12	0.12	870
38	0.78	0.45	0.57	135
39	0.60	0.53	0.56	17
40	0.55	0.06	0.11	99
41	0.63	0.29	0.40	176
42	0.16	0.11	0.13	236
43	0.56	0.41	0.47	22
44	0.59	0.23	0.33	106
45	0.15	0.10	0.12	178
46	0.31	0.27	0.29	241
47	0.51	0.18	0.27	217

48	0.58	0.48	0.52	223
49	0.50	0.04	0.07	54
50	0.52	0.33	0.40	92
51	0.76	0.50	0.60	203
52	0.34	0.47	0.39	116
53	0.62	0.54	0.58	72
54	0.25	0.27	0.26	15
55	0.00	0.00	0.00	60
56	0.83	0.88	0.85	216
57	0.21	0.12	0.15	74
58	0.23	0.18	0.20	139
59	0.53	0.44	0.48	91
60	0.43	0.12	0.19	156
61	0.44	0.21	0.29	76
62	0.33	0.25	0.28	89
63	0.41	0.15	0.22	173
64	0.38	0.44	0.41	227
65	0.30	0.19	0.23	383
66	0.46	0.18	0.25	148
67	0.50	0.30	0.38	189
68	0.62	0.20	0.30	169
69	0.14	0.18	0.16	50
70	0.60	0.27	0.37	145
71	0.23	0.39	0.29	31
72	0.84	0.82	0.83	141
73	0.66	0.49	0.56	246
74	0.43	0.29	0.35	210
75	0.85	0.07	0.13	159
76	0.45	0.30	0.36	108
77	0.41	0.75	0.53	65
78	0.64	0.77	0.70	145
79	0.74	0.71	0.72	41
80	0.51	0.75	0.61	129
81	0.68	0.61	0.64	76
82	0.48	0.48	0.48	124
83	0.21	0.23	0.22	69
84	0.26	0.21	0.23	91
85	0.40	0.56	0.47	66
86	0.22	0.21	0.22	100
87	0.29	0.05	0.09	38
88	0.53	0.52	0.53	98
89	0.35	0.18	0.24	38
90	0.87	0.74	0.80	154
91	0.82	0.65	0.73	152
92	0.00	0.00	0.00	13
93	0.00	0.00	0.00	47
94	0.81	0.39	0.52	44
95	0.69	0.40	0.51	200
96	0.31	0.16	0.21	25
97	0.42	0.21	0.28	39
98	0.31	0.37	0.34	51
99	0.21	0.19	0.20	43
100	0.17	0.09	0.12	211
101	0.31	0.28	0.29	18
102	0.55	0.34	0.42	32
103	0.71	0.50	0.59	24
104	0.26	0.36	0.30	14
105	0.50	0.28	0.36	96
106	0.36	0.47	0.41	32
107	0.57	0.50	0.53	80
108	0.08	0.01	0.01	160

109	0.24	0.07	0.11	123
110	0.16	0.01	0.03	202
111	0.55	0.56	0.56	39
112	0.19	0.07	0.11	123
113	0.61	0.62	0.61	55
114	0.25	0.19	0.22	98
115	0.32	0.26	0.29	50
116	0.80	0.48	0.60	275
117	0.00	0.00	0.00	101
118	0.40	0.12	0.18	50
119	0.20	0.24	0.22	41
120	0.44	0.38	0.41	98
121	0.31	0.13	0.19	30
122	0.73	0.33	0.45	73
123	0.84	0.81	0.82	121
124	0.35	0.21	0.26	29
125	1.00	0.09	0.16	57
126	0.21	0.15	0.17	48
127	0.50	0.62	0.56	24
128	0.68	0.27	0.39	48
129	0.50	0.25	0.33	48
130	0.82	0.45	0.58	99
131	0.34	0.38	0.36	29
132	0.15	0.08	0.11	60
133	0.56	0.75	0.64	89
134	0.08	0.02	0.03	113
135	0.23	0.24	0.24	70
136	0.21	0.06	0.09	68
137	0.84	0.60	0.70	146
138	0.43	0.48	0.45	66
139	0.24	0.24	0.24	49
140	0.66	0.65	0.65	51
141	0.75	0.22	0.34	27
142	0.21	0.07	0.11	54
143	0.33	0.14	0.20	21
144	0.29	0.28	0.29	43
145	0.89	0.35	0.50	49
146	0.56	0.36	0.44	137
147	0.66	0.41	0.50	91
148	0.27	0.24	0.25	29
149	0.86	0.49	0.62	88
150	0.04	0.03	0.04	67
151	0.76	0.35	0.48	46
152	0.45	0.26	0.33	187
153	0.73	0.45	0.56	60
154	0.31	0.45	0.36	40
155	0.29	0.06	0.10	67
156	0.21	0.30	0.25	46
157	0.33	0.04	0.08	23
158	0.55	0.59	0.57	54
159	0.34	0.22	0.27	87
160	0.59	0.24	0.34	66
161	0.62	0.45	0.52	69
162	0.29	0.19	0.23	78
163	0.55	0.84	0.67	50
164	0.47	0.06	0.11	115
165	0.39	0.13	0.19	71
166	0.07	0.02	0.04	81
167	0.37	0.44	0.40	52
168	0.47	0.36	0.41	22
169	0.00	0.00	0.00	292

170	0.32	0.56	0.41	45
171	0.14	0.02	0.04	146
172	0.00	0.00	0.00	5
173	0.50	0.23	0.31	66
174	0.04	0.05	0.05	21
175	0.27	0.15	0.20	26
176	0.38	0.10	0.16	86
177	0.50	0.11	0.18	18
178	0.09	0.07	0.08	27
179	0.00	0.00	0.00	0
180	0.06	0.29	0.10	7
181	0.76	0.56	0.64	34
182	0.67	0.57	0.62	35
183	0.51	0.57	0.54	51
184	0.71	0.58	0.64	38
185	0.08	0.03	0.04	39
186	0.00	0.00	0.00	13
187	0.50	0.17	0.26	35
188	0.12	0.16	0.14	44
189	0.16	0.13	0.14	46
190	0.21	0.08	0.11	52
191	0.33	0.19	0.24	88
192	0.11	0.02	0.04	41
193	0.94	0.57	0.71	88
194	0.43	0.06	0.10	51
195	0.47	0.13	0.21	127
196	0.17	0.13	0.15	60
197	0.00	0.00	0.00	18
198	0.10	0.03	0.04	36
199	0.40	0.02	0.04	85
200	0.41	0.27	0.33	48
201	0.44	0.47	0.46	17
202	0.24	0.22	0.23	27
203	0.45	0.22	0.29	60
204	0.46	0.38	0.42	105
205	0.69	0.62	0.65	50
206	0.51	0.42	0.46	45
207	0.17	0.37	0.24	19
208	0.54	0.29	0.38	73
209	0.12	0.02	0.03	51
210	0.00	0.00	0.00	20
211	0.00	0.00	0.00	47
212	0.20	0.05	0.07	44
213	0.37	0.21	0.26	34
214	0.67	0.53	0.59	106
215	0.60	0.10	0.17	59
216	0.09	0.08	0.09	87
217	0.33	0.06	0.11	31
218	0.66	0.67	0.67	46
219	0.20	0.15	0.17	27
220	0.32	0.15	0.21	39
221	0.41	0.13	0.19	55
222	0.67	0.06	0.11	34
223	0.11	0.64	0.19	11
224	0.10	0.14	0.12	51
225	0.09	0.13	0.10	46
226	0.17	0.09	0.11	47
227	0.06	0.07	0.07	14
228	0.33	0.10	0.15	21
229	0.31	0.07	0.12	67
230	0.00	0.00	0.00	229

231	0.30	0.11	0.16	54
232	0.50	0.03	0.06	98
233	0.91	0.40	0.55	53
234	0.62	0.28	0.38	36
235	0.70	0.40	0.51	53
236	0.41	0.37	0.39	68
237	0.08	0.21	0.12	38
238	0.07	0.07	0.07	102
239	0.07	0.33	0.11	6
240	0.15	0.40	0.22	5
241	0.00	0.00	0.00	3
242	0.03	0.03	0.03	68
243	0.38	0.32	0.35	91
244	0.96	0.77	0.85	30
245	0.50	0.10	0.17	50
246	0.00	0.00	0.00	4
247	0.42	0.37	0.39	41
248	0.60	0.33	0.42	98
249	0.00	0.00	0.00	0
250	1.00	1.00	1.00	1
251	0.67	0.15	0.25	26
252	0.56	0.29	0.38	66
253	0.77	0.61	0.68	67
254	0.12	0.06	0.08	32
255	0.00	0.00	0.00	2
256	0.25	0.06	0.10	32
257	0.25	0.25	0.25	4
258	0.14	0.03	0.04	39
259	0.80	0.44	0.57	73
260	0.90	0.64	0.74	55
261	0.20	0.58	0.30	12
262	0.14	0.15	0.14	41
263	0.50	0.07	0.12	14
264	0.55	0.11	0.18	56
265	0.73	0.29	0.41	77
266	0.00	0.00	0.00	13
267	0.33	0.25	0.29	16
268	0.00	0.00	0.00	34
269	0.00	0.00	0.00	45
270	0.06	0.02	0.03	43
271	0.31	0.30	0.31	56
272	0.60	0.27	0.37	11
273	0.03	0.02	0.03	42
274	0.75	0.51	0.61	35
275	0.18	0.15	0.17	59
276	0.06	0.06	0.06	49
277	0.63	0.61	0.62	44
278	0.16	0.07	0.09	46
279	0.06	0.14	0.08	7
280	0.83	0.52	0.64	58
281	0.35	0.20	0.25	46
282	0.42	0.50	0.45	10
283	0.55	0.29	0.37	21
284	0.13	0.13	0.13	47
285	0.47	0.30	0.37	23
286	0.80	0.75	0.77	48
287	0.24	0.17	0.20	35
288	0.00	0.00	0.00	81
289	0.63	0.36	0.46	47
290	0.76	0.74	0.75	93
291	0.00	0.00	0.00	61

292	0.48	0.52	0.50	23
293	0.71	0.50	0.59	10
294	0.25	0.03	0.06	30
295	0.00	0.00	0.00	24
296	0.00	0.00	0.00	54
297	0.35	0.24	0.28	34
298	0.26	0.22	0.24	69
299	0.79	0.68	0.73	44
300	0.60	0.23	0.33	13
301	0.80	0.47	0.59	68
302	0.00	0.00	0.00	33
303	0.75	0.33	0.46	18
304	0.14	0.08	0.10	13
305	0.52	0.25	0.33	53
306	0.41	0.21	0.28	75
307	0.77	0.49	0.60	55
308	0.86	0.51	0.64	61
309	0.70	0.37	0.48	90
310	0.00	0.00	0.00	58
311	0.83	0.79	0.81	19
312	0.44	0.12	0.19	34
313	0.21	0.46	0.29	13
314	0.14	0.25	0.18	4
315	0.20	0.02	0.04	41
316	0.72	0.48	0.58	54
317	0.33	0.04	0.07	25
318	0.14	0.25	0.18	4
319	0.20	0.10	0.14	29
320	0.80	0.22	0.34	37
321	1.00	0.50	0.67	6
322	0.17	0.23	0.20	22
323	0.29	0.11	0.15	19
324	0.14	0.25	0.18	4
325	0.86	0.33	0.48	18
326	0.69	0.43	0.53	21
327	0.00	0.00	0.00	26
328	0.70	0.43	0.53	49
329	0.40	0.51	0.45	35
330	0.00	0.00	0.00	19
331	1.00	0.07	0.12	15
332	0.00	0.00	0.00	10
333	0.68	0.55	0.61	38
334	0.08	0.11	0.10	9
335	0.83	0.09	0.17	53
336	0.83	0.47	0.60	32
337	0.05	0.12	0.07	24
338	0.17	0.33	0.22	3
339	0.00	0.00	0.00	1
340	0.00	0.00	0.00	0
341	0.00	0.00	0.00	11
342	0.54	0.55	0.54	40
343	0.21	0.10	0.14	30
344	0.25	0.08	0.12	24
345	0.17	0.30	0.22	23
346	0.38	0.22	0.28	69
347	0.03	0.06	0.04	18
348	0.04	0.03	0.03	65
349	0.43	0.38	0.41	78
350	1.00	0.08	0.15	12
351	0.14	0.08	0.10	13
352	0.38	0.28	0.32	18

353	1.00	0.54	0.70	46
354	0.62	0.40	0.48	40
355	0.00	0.00	0.00	19
356	0.00	0.00	0.00	26
357	0.38	0.08	0.13	39
358	1.00	0.17	0.29	12
359	0.00	0.00	0.00	16
360	0.22	0.08	0.12	24
361	0.22	0.11	0.14	57
362	0.67	0.90	0.77	20
363	0.00	0.00	0.00	84
364	0.60	0.46	0.52	54
365	0.30	0.09	0.14	33
366	0.03	0.03	0.03	30
367	0.00	0.00	0.00	30
368	0.17	0.05	0.08	19
369	0.00	0.00	0.00	19
370	0.33	0.03	0.06	32
371	0.40	0.67	0.50	12
372	0.00	0.00	0.00	15
373	0.03	0.07	0.05	15
374	0.83	0.59	0.69	17
375	0.83	0.61	0.70	41
376	0.86	0.41	0.56	29
377	0.00	0.00	0.00	28
378	0.50	0.21	0.30	19
379	0.06	0.03	0.04	31
380	0.00	0.00	0.00	29
381	0.13	0.18	0.15	49
382	0.00	0.00	0.00	8
383	0.38	0.12	0.19	24
384	0.33	0.20	0.25	20
385	0.38	0.20	0.26	15
386	0.64	0.49	0.55	37
387	0.00	0.00	0.00	22
388	0.00	0.00	0.00	27
389	0.16	0.17	0.16	29
390	0.17	0.10	0.12	20
391	0.54	0.33	0.41	39
392	0.00	0.00	0.00	10
393	1.00	0.05	0.09	42
394	0.11	0.04	0.06	46
395	0.20	0.30	0.24	10
396	1.00	0.05	0.10	39
397	0.00	0.00	0.00	43
398	0.30	0.20	0.24	50
399	0.33	0.29	0.31	7
400	0.00	0.00	0.00	17
401	1.00	0.17	0.29	6
402	0.00	0.00	0.00	26
403	0.04	0.10	0.06	10
404	0.62	0.36	0.45	14
405	0.11	0.07	0.09	14
406	0.80	0.36	0.50	22
407	0.39	0.12	0.18	60
408	0.15	0.10	0.12	40
409	0.00	0.00	0.00	31
410	0.25	0.22	0.24	9
411	0.62	0.26	0.37	19
412	0.69	0.58	0.63	19
413	1.00	0.20	0.33	5

414	0.17	0.08	0.11	12
415	0.86	0.62	0.72	29
416	0.13	0.15	0.14	33
417	0.00	0.00	0.00	33
418	0.08	0.08	0.08	12
419	0.05	0.02	0.03	42
420	0.33	0.42	0.37	12
421	0.00	0.00	0.00	98
422	0.00	0.00	0.00	8
423	0.75	0.43	0.55	7
424	0.50	0.38	0.43	13
425	0.03	0.08	0.04	13
426	0.00	0.00	0.00	20
427	0.00	0.00	0.00	58
428	0.67	1.00	0.80	2
429	0.29	0.26	0.27	27
430	0.49	0.50	0.49	38
431	0.39	0.30	0.34	40
432	0.00	0.00	0.00	43
433	0.96	0.52	0.68	42
434	0.50	0.33	0.40	24
435	0.25	0.03	0.06	31
436	0.33	0.30	0.32	30
437	0.00	0.00	0.00	16
438	0.56	0.45	0.50	22
439	0.00	0.00	0.00	1
440	0.06	0.05	0.06	19
441	0.25	0.22	0.24	9
442	0.00	0.00	0.00	100
443	0.50	0.32	0.39	28
444	0.60	0.60	0.60	20
445	0.44	0.41	0.43	29
446	0.17	0.05	0.07	21
447	0.00	0.00	0.00	20
448	0.85	0.29	0.43	38
449	0.00	0.00	0.00	22
450	0.56	0.48	0.51	21
451	0.00	0.00	0.00	13
452	0.00	0.00	0.00	24
453	0.40	0.04	0.08	48
454	0.00	0.00	0.00	75
455	0.00	0.00	0.00	18
456	0.00	0.00	0.00	3
457	0.32	0.46	0.37	13
458	0.00	0.00	0.00	13
459	0.06	0.04	0.05	24
460	0.27	0.17	0.21	36
461	0.20	0.11	0.14	18
462	0.50	0.03	0.06	31
463	0.00	0.00	0.00	28
464	0.25	0.14	0.18	7
465	0.80	0.30	0.43	27
466	0.00	0.00	0.00	12
467	0.00	0.00	0.00	14
468	0.00	0.00	0.00	6
469	0.25	0.18	0.21	17
470	0.15	0.22	0.18	18
471	0.02	0.03	0.03	29
472	0.00	0.00	0.00	2
473	0.33	0.03	0.05	34
474	0.00	0.00	0.00	8

```
         475       1.00      0.25      0.40         4
         476       0.22      0.09      0.13        22
         477       0.20      0.50      0.29         6
         478       0.26      0.29      0.28        17
         479       0.20      0.04      0.07        23
         480       1.00      0.06      0.11        18
         481       0.67      0.18      0.29        11
         482       0.93      0.37      0.53        35
         483       0.61      0.52      0.56        21
         484       0.83      0.54      0.65        28
         485       0.33      0.29      0.31        14
         486       0.91      0.91      0.91        11
         487       1.00      0.20      0.33        15
         488       0.00      0.00      0.00        38
         489       0.00      0.00      0.00        75
         490       1.00      0.12      0.21        51
         491       1.00      0.58      0.73        19
         492       0.45      0.24      0.31        21
         493       0.00      0.00      0.00        16
         494       0.44      0.67      0.53         6
         495       0.11      0.09      0.10        22
         496       0.49      0.49      0.49        37
         497       0.17      0.10      0.12        20
         498       0.56      0.42      0.48        24
         499       0.00      0.00      0.00        17

   micro avg       0.54      0.35      0.43     47151
   macro avg       0.39      0.26      0.29     47151
weighted avg       0.55      0.35      0.41     47151
 samples avg       0.42      0.34      0.35     47151

Time taken to run this cell : 0:03:22.300054
```

Task 3: Apply OneVsRestClassifier with Linear-SVM

Hyperparameter Tuning

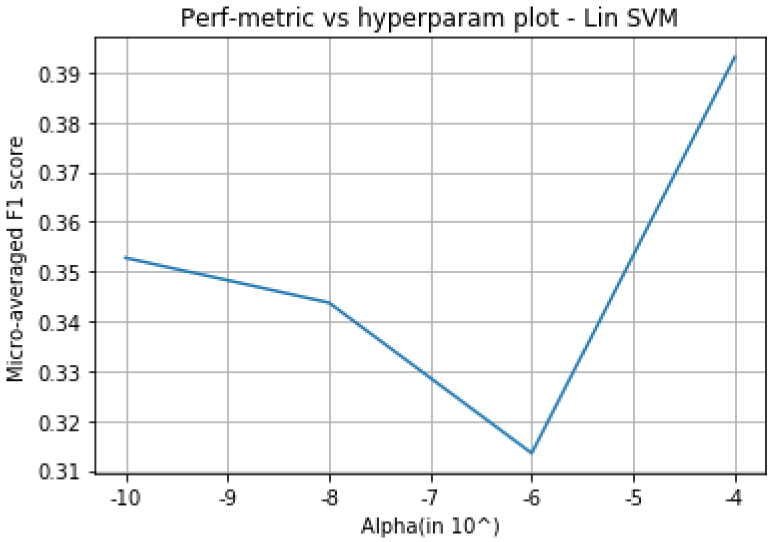
In [53]:

```
from tqdm import tqdm
start = datetime.now()
alpha = [10 ** x for x in range(-10, -3, 2)]
perf_metric = []
for i in tqdm(alpha):
    clf = OneVsRestClassifier(SGDClassifier(loss='hinge', alpha=i, penalty='l1', random
_state=42), n_jobs=-1)
    clf.fit(x_train_multilabel, y_train)
    predictions = clf.predict (x_test_multilabel)
    # append the micro-f1 score for the particular alpha trained classifier
    perf_metric.append(f1_score(y_test, predictions, average='micro'))
print("Time taken to run this cell :", datetime.now() - start)
```

```
100%|██████████████████████████████████████████████
█████████| 4/4 [29:51<00:00, 447.80s/it]

Time taken to run this cell : 0:29:51.230015
```

In [54]:

```
# plot the perf metric for each hyperparam(alpha)
fig, ax = plt.subplots()
ax.plot(perf_metric)
xlabel = list(range(-11, -3))
ax.set_xticklabels(xlabel)
plt.title("Perf-metric vs hyperparam plot - Lin SVM")
plt.xlabel("Alpha(in 10^)")
plt.ylabel("Micro-averaged F1 score")
plt.grid()
plt.show()
```



In [55]:

```
start = datetime.now()
# fetching the best alpha
best_alpha = alpha[np.argmax(perf_metric)]
print('Best hyperparam(alpha) : ',best_alpha)

# train the Lin SVM model with the best alpha
classifier = OneVsRestClassifier(SGDClassifier(loss='hinge', alpha=best_alpha, penalty=
'l1',  random_state=42), n_jobs=-1)
classifier.fit(x_train_multilabel, y_train)
predictions = classifier.predict (x_test_multilabel)

# print the various performance metrices
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("\nMicro-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("\nMacro-average quality numbers -")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall,
f1))

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

```
Best hyperparam(alpha) :  0.0001
Accuracy : 0.0877
Hamming loss : 0.006592

Micro-average quality numbers -
Precision: 0.3473, Recall: 0.4527, F1-measure: 0.3931

Macro-average quality numbers -
Precision: 0.2390, Recall: 0.3426, F1-measure: 0.2656
```

	precision	recall	f1-score	support
0	0.59	0.53	0.56	1805
1	0.64	0.59	0.62	1186
2	0.52	0.70	0.60	484
3	0.60	0.51	0.55	1323
4	0.56	0.65	0.60	739
5	0.65	0.50	0.56	1023
6	0.54	0.47	0.50	1421
7	0.75	0.72	0.74	1450
8	0.83	0.80	0.82	1368
9	0.50	0.51	0.51	914
10	0.25	0.51	0.33	186
11	0.57	0.53	0.54	553
12	0.57	0.52	0.54	644
13	0.34	0.33	0.34	424
14	0.11	0.47	0.18	36
15	0.34	0.44	0.38	352
16	0.30	0.38	0.34	437
17	0.42	0.51	0.46	435
18	0.44	0.65	0.53	153
19	0.78	0.70	0.74	727
20	0.32	0.44	0.37	488
21	0.48	0.71	0.58	272
22	0.58	0.71	0.64	530
23	0.80	0.63	0.71	618
24	0.81	0.63	0.70	614
25	0.22	0.44	0.30	231
26	0.30	0.66	0.41	588
27	0.24	0.45	0.31	1224
28	0.54	0.60	0.57	165
29	0.32	0.61	0.42	231
30	0.20	0.42	0.27	190
31	0.63	0.66	0.64	296
32	0.33	0.36	0.35	274
33	0.29	0.42	0.34	292
34	0.31	0.35	0.33	190
35	0.47	0.58	0.52	99
36	0.56	0.65	0.60	357
37	0.25	0.39	0.30	870
38	0.50	0.56	0.53	135
39	0.12	0.47	0.19	17
40	0.17	0.17	0.17	99
41	0.25	0.40	0.31	176
42	0.14	0.19	0.16	236
43	0.09	0.32	0.14	22
44	0.21	0.29	0.25	106
45	0.19	0.24	0.21	178
46	0.18	0.31	0.23	241
47	0.26	0.28	0.27	217

48	0.44	0.54	0.48	223
49	0.09	0.11	0.10	54
50	0.29	0.50	0.37	92
51	0.65	0.51	0.57	203
52	0.29	0.51	0.37	116
53	0.19	0.56	0.29	72
54	0.04	0.20	0.07	15
55	0.17	0.15	0.16	60
56	0.71	0.86	0.78	216
57	0.28	0.19	0.23	74
58	0.12	0.06	0.09	139
59	0.37	0.58	0.45	91
60	0.15	0.19	0.17	156
61	0.31	0.37	0.34	76
62	0.13	0.21	0.16	89
63	0.15	0.21	0.17	173
64	0.37	0.51	0.43	227
65	0.26	0.31	0.28	383
66	0.18	0.32	0.24	148
67	0.44	0.53	0.48	189
68	0.29	0.37	0.33	169
69	0.06	0.22	0.10	50
70	0.24	0.42	0.31	145
71	0.14	0.26	0.18	31
72	0.57	0.82	0.67	141
73	0.39	0.59	0.47	246
74	0.35	0.37	0.36	210
75	0.26	0.20	0.23	159
76	0.19	0.29	0.23	108
77	0.59	0.78	0.67	65
78	0.63	0.72	0.67	145
79	0.53	0.76	0.63	41
80	0.46	0.74	0.56	129
81	0.28	0.63	0.38	76
82	0.28	0.50	0.36	124
83	0.08	0.19	0.11	69
84	0.14	0.21	0.17	91
85	0.20	0.53	0.29	66
86	0.15	0.23	0.18	100
87	0.22	0.34	0.27	38
88	0.34	0.28	0.31	98
89	0.29	0.58	0.38	38
90	0.55	0.73	0.63	154
91	0.49	0.72	0.58	152
92	0.00	0.00	0.00	13
93	0.02	0.02	0.02	47
94	0.14	0.45	0.21	44
95	0.30	0.47	0.36	200
96	0.21	0.28	0.24	25
97	0.22	0.36	0.27	39
98	0.23	0.45	0.31	51
99	0.13	0.26	0.17	43
100	0.26	0.22	0.24	211
101	0.14	0.33	0.20	18
102	0.37	0.56	0.44	32
103	0.09	0.58	0.15	24
104	0.03	0.21	0.05	14
105	0.42	0.43	0.42	96
106	0.41	0.53	0.47	32
107	0.46	0.34	0.39	80
108	0.42	0.34	0.37	160

109	0.14	0.11	0.12	123
110	0.13	0.22	0.16	202
111	0.34	0.56	0.43	39
112	0.23	0.26	0.24	123
113	0.39	0.51	0.44	55
114	0.16	0.14	0.15	98
115	0.12	0.30	0.18	50
116	0.34	0.57	0.43	275
117	0.08	0.11	0.09	101
118	0.14	0.24	0.18	50
119	0.19	0.27	0.22	41
120	0.33	0.32	0.32	98
121	0.07	0.20	0.10	30
122	0.29	0.49	0.36	73
123	0.49	0.83	0.62	121
124	0.26	0.55	0.36	29
125	0.43	0.37	0.40	57
126	0.14	0.12	0.13	48
127	0.19	0.79	0.30	24
128	0.20	0.23	0.21	48
129	0.24	0.27	0.25	48
130	0.39	0.62	0.48	99
131	0.09	0.41	0.14	29
132	0.08	0.13	0.10	60
133	0.53	0.62	0.57	89
134	0.07	0.11	0.09	113
135	0.12	0.33	0.18	70
136	0.11	0.22	0.15	68
137	0.59	0.62	0.61	146
138	0.30	0.47	0.36	66
139	0.07	0.22	0.11	49
140	0.20	0.59	0.30	51
141	0.26	0.41	0.32	27
142	0.07	0.13	0.09	54
143	0.05	0.14	0.07	21
144	0.15	0.44	0.22	43
145	0.45	0.37	0.40	49
146	0.36	0.49	0.41	137
147	0.37	0.56	0.45	91
148	0.16	0.41	0.24	29
149	0.50	0.62	0.56	88
150	0.11	0.16	0.13	67
151	0.40	0.41	0.40	46
152	0.24	0.32	0.27	187
153	0.27	0.43	0.33	60
154	0.42	0.40	0.41	40
155	0.17	0.06	0.09	67
156	0.18	0.33	0.23	46
157	0.17	0.43	0.25	23
158	0.41	0.59	0.48	54
159	0.22	0.23	0.22	87
160	0.37	0.36	0.37	66
161	0.42	0.58	0.48	69
162	0.15	0.36	0.21	78
163	0.65	0.88	0.75	50
164	0.15	0.23	0.18	115
165	0.39	0.15	0.22	71
166	0.05	0.07	0.06	81
167	0.18	0.50	0.26	52
168	0.28	0.59	0.38	22
169	0.50	0.01	0.01	292

170	0.20	0.58	0.30	45
171	0.08	0.03	0.04	146
172	0.00	0.00	0.00	5
173	0.14	0.05	0.07	66
174	0.08	0.29	0.13	21
175	0.10	0.23	0.14	26
176	0.16	0.14	0.15	86
177	0.09	0.17	0.12	18
178	0.04	0.11	0.06	27
179	0.00	0.00	0.00	0
180	0.11	0.57	0.19	7
181	0.54	0.62	0.58	34
182	0.46	0.66	0.54	35
183	0.35	0.57	0.43	51
184	0.41	0.68	0.51	38
185	0.00	0.00	0.00	39
186	0.25	0.38	0.30	13
187	0.34	0.40	0.37	35
188	0.09	0.18	0.12	44
189	0.12	0.24	0.16	46
190	0.17	0.23	0.20	52
191	0.17	0.23	0.20	88
192	0.04	0.07	0.05	41
193	0.86	0.70	0.78	88
194	0.05	0.10	0.07	51
195	0.28	0.32	0.30	127
196	0.07	0.12	0.09	60
197	0.11	0.22	0.15	18
198	0.04	0.06	0.04	36
199	0.09	0.16	0.11	85
200	0.22	0.31	0.26	48
201	0.17	0.71	0.28	17
202	0.21	0.30	0.24	27
203	0.17	0.45	0.25	60
204	0.44	0.51	0.48	105
205	0.42	0.44	0.43	50
206	0.21	0.33	0.25	45
207	0.28	0.58	0.37	19
208	0.22	0.38	0.28	73
209	0.21	0.12	0.15	51
210	0.15	0.25	0.19	20
211	0.08	0.09	0.08	47
212	0.06	0.05	0.05	44
213	0.32	0.35	0.34	34
214	0.46	0.55	0.50	106
215	0.31	0.47	0.37	59
216	0.12	0.22	0.15	87
217	0.23	0.29	0.25	31
218	0.35	0.72	0.47	46
219	0.04	0.22	0.06	27
220	0.09	0.10	0.10	39
221	0.24	0.35	0.29	55
222	0.40	0.18	0.24	34
223	0.16	0.64	0.26	11
224	0.11	0.14	0.12	51
225	0.07	0.11	0.08	46
226	0.11	0.23	0.15	47
227	0.06	0.14	0.09	14
228	0.12	0.24	0.16	21
229	0.15	0.25	0.18	67
230	0.00	0.00	0.00	229

231	0.09	0.15	0.11	54
232	0.36	0.16	0.22	98
233	0.63	0.45	0.53	53
234	0.19	0.33	0.24	36
235	0.25	0.51	0.33	53
236	0.28	0.40	0.33	68
237	0.05	0.21	0.09	38
238	0.14	0.23	0.17	102
239	0.07	0.33	0.12	6
240	0.03	0.20	0.06	5
241	0.15	0.67	0.25	3
242	0.16	0.16	0.16	68
243	0.38	0.38	0.38	91
244	0.35	0.83	0.49	30
245	0.21	0.32	0.26	50
246	0.06	0.25	0.10	4
247	0.25	0.44	0.32	41
248	0.29	0.26	0.27	98
249	0.00	0.00	0.00	0
250	0.10	1.00	0.18	1
251	0.12	0.27	0.16	26
252	0.42	0.27	0.33	66
253	0.44	0.70	0.54	67
254	0.02	0.06	0.03	32
255	0.00	0.00	0.00	2
256	0.07	0.16	0.10	32
257	0.03	0.50	0.05	4
258	0.04	0.08	0.05	39
259	0.51	0.49	0.50	73
260	0.71	0.55	0.62	55
261	0.24	0.67	0.36	12
262	0.11	0.24	0.15	41
263	0.25	0.29	0.27	14
264	0.15	0.20	0.17	56
265	0.37	0.38	0.37	77
266	0.00	0.00	0.00	13
267	0.27	0.44	0.33	16
268	0.02	0.03	0.03	34
269	0.04	0.02	0.03	45
270	0.06	0.12	0.08	43
271	0.27	0.46	0.34	56
272	0.14	0.27	0.19	11
273	0.10	0.05	0.06	42
274	0.69	0.57	0.62	35
275	0.04	0.03	0.04	59
276	0.07	0.18	0.11	49
277	0.61	0.64	0.62	44
278	0.11	0.11	0.11	46
279	0.00	0.00	0.00	7
280	0.55	0.66	0.60	58
281	0.48	0.26	0.34	46
282	0.19	0.50	0.27	10
283	0.30	0.33	0.32	21
284	0.07	0.11	0.09	47
285	0.15	0.26	0.19	23
286	0.49	0.77	0.60	48
287	0.39	0.51	0.44	35
288	0.05	0.04	0.04	81
289	0.37	0.53	0.44	47
290	0.62	0.83	0.71	93
291	0.18	0.21	0.19	61

292	0.26	0.70	0.38	23
293	0.23	0.50	0.31	10
294	0.22	0.07	0.10	30
295	0.05	0.08	0.06	24
296	0.08	0.07	0.08	54
297	0.23	0.62	0.34	34
298	0.21	0.39	0.28	69
299	0.63	0.86	0.73	44
300	0.47	0.54	0.50	13
301	0.52	0.56	0.54	68
302	0.02	0.06	0.04	33
303	0.28	0.39	0.33	18
304	0.07	0.38	0.11	13
305	0.19	0.28	0.23	53
306	0.22	0.33	0.26	75
307	0.45	0.62	0.52	55
308	0.79	0.67	0.73	61
309	0.46	0.49	0.47	90
310	0.53	0.16	0.24	58
311	0.28	0.84	0.42	19
312	0.16	0.24	0.19	34
313	0.16	0.38	0.22	13
314	0.11	0.50	0.18	4
315	0.06	0.07	0.07	41
316	0.43	0.52	0.47	54
317	0.18	0.24	0.20	25
318	0.13	0.50	0.21	4
319	0.04	0.14	0.06	29
320	0.10	0.16	0.13	37
321	0.33	0.50	0.40	6
322	0.22	0.50	0.30	22
323	0.08	0.11	0.09	19
324	0.12	0.50	0.20	4
325	0.29	0.56	0.38	18
326	0.35	0.57	0.44	21
327	0.07	0.12	0.09	26
328	0.28	0.55	0.37	49
329	0.43	0.51	0.47	35
330	0.00	0.00	0.00	19
331	0.17	0.20	0.18	15
332	0.08	0.20	0.12	10
333	0.51	0.50	0.51	38
334	0.06	0.22	0.09	9
335	0.39	0.21	0.27	53
336	0.68	0.66	0.67	32
337	0.05	0.08	0.06	24
338	0.03	0.33	0.05	3
339	0.00	0.00	0.00	1
340	0.00	0.00	0.00	0
341	0.12	0.27	0.17	11
342	0.27	0.72	0.39	40
343	0.12	0.07	0.09	30
344	0.11	0.04	0.06	24
345	0.08	0.35	0.13	23
346	0.29	0.25	0.27	69
347	0.10	0.22	0.14	18
348	0.11	0.20	0.14	65
349	0.33	0.27	0.30	78
350	0.00	0.00	0.00	12
351	0.10	0.46	0.16	13
352	0.00	0.00	0.00	18

353	0.76	0.76	0.76	46
354	0.24	0.47	0.32	40
355	0.05	0.11	0.07	19
356	0.08	0.08	0.08	26
357	0.23	0.23	0.23	39
358	0.15	0.17	0.16	12
359	0.00	0.00	0.00	16
360	0.08	0.08	0.08	24
361	0.20	0.18	0.19	57
362	0.55	0.85	0.67	20
363	0.36	0.11	0.17	84
364	0.44	0.67	0.53	54
365	0.15	0.21	0.18	33
366	0.05	0.10	0.07	30
367	0.08	0.03	0.05	30
368	0.00	0.00	0.00	19
369	0.00	0.00	0.00	19
370	0.05	0.06	0.06	32
371	0.15	0.67	0.25	12
372	0.04	0.07	0.05	15
373	0.05	0.13	0.07	15
374	0.61	0.65	0.63	17
375	0.52	0.83	0.64	41
376	0.28	0.62	0.39	29
377	0.04	0.04	0.04	28
378	0.09	0.16	0.12	19
379	0.16	0.16	0.16	31
380	0.18	0.14	0.16	29
381	0.11	0.22	0.14	49
382	0.03	0.12	0.05	8
383	0.08	0.29	0.13	24
384	0.11	0.45	0.17	20
385	0.06	0.13	0.09	15
386	0.52	0.59	0.56	37
387	0.03	0.09	0.05	22
388	0.00	0.00	0.00	27
389	0.20	0.34	0.25	29
390	0.06	0.10	0.07	20
391	0.33	0.56	0.42	39
392	0.06	0.10	0.07	10
393	0.27	0.31	0.29	42
394	0.12	0.13	0.12	46
395	0.03	0.10	0.05	10
396	0.00	0.00	0.00	39
397	0.00	0.00	0.00	43
398	0.27	0.24	0.25	50
399	0.12	0.71	0.21	7
400	0.02	0.06	0.03	17
401	0.08	0.33	0.13	6
402	0.00	0.00	0.00	26
403	0.02	0.10	0.03	10
404	0.17	0.29	0.22	14
405	0.07	0.14	0.10	14
406	0.50	0.55	0.52	22
407	0.22	0.23	0.22	60
408	0.26	0.28	0.27	40
409	0.06	0.13	0.08	31
410	0.17	0.33	0.22	9
411	0.15	0.32	0.20	19
412	0.41	0.63	0.50	19
413	0.17	0.20	0.18	5

414	0.00	0.08	0.01	12
415	0.47	0.59	0.52	29
416	0.04	0.09	0.06	33
417	0.10	0.21	0.13	33
418	0.04	0.08	0.05	12
419	0.11	0.14	0.12	42
420	0.12	0.08	0.10	12
421	0.29	0.32	0.30	98
422	0.06	0.12	0.08	8
423	0.08	0.14	0.10	7
424	0.25	0.62	0.36	13
425	0.08	0.15	0.11	13
426	0.13	0.30	0.18	20
427	0.07	0.12	0.09	58
428	0.25	1.00	0.40	2
429	0.28	0.41	0.33	27
430	0.35	0.39	0.37	38
431	0.22	0.38	0.28	40
432	0.03	0.12	0.05	43
433	0.60	0.67	0.63	42
434	0.21	0.21	0.21	24
435	0.12	0.23	0.16	31
436	0.17	0.17	0.17	30
437	0.06	0.12	0.08	16
438	0.31	0.41	0.35	22
439	0.00	0.00	0.00	1
440	0.09	0.11	0.10	19
441	0.03	0.22	0.05	9
442	0.30	0.20	0.24	100
443	0.44	0.57	0.50	28
444	0.34	0.75	0.47	20
445	0.41	0.62	0.49	29
446	0.04	0.10	0.06	21
447	0.24	0.45	0.32	20
448	0.78	0.55	0.65	38
449	0.05	0.09	0.06	22
450	0.45	0.43	0.44	21
451	0.07	0.23	0.11	13
452	0.03	0.08	0.05	24
453	0.20	0.10	0.14	48
454	0.28	0.32	0.30	75
455	0.14	0.22	0.17	18
456	0.04	0.33	0.07	3
457	0.19	0.46	0.27	13
458	0.00	0.00	0.00	13
459	0.18	0.33	0.24	24
460	0.20	0.36	0.25	36
461	0.31	0.61	0.42	18
462	0.18	0.32	0.23	31
463	0.22	0.14	0.17	28
464	0.00	0.00	0.00	7
465	0.26	0.33	0.30	27
466	0.41	0.75	0.53	12
467	0.12	0.21	0.15	14
468	0.00	0.00	0.00	6
469	0.08	0.12	0.10	17
470	0.11	0.28	0.16	18
471	0.14	0.10	0.12	29
472	0.00	0.00	0.00	2
473	0.35	0.18	0.24	34
474	0.00	0.00	0.00	8

```
         475       0.10      0.50      0.17         4
         476       0.28      0.50      0.35        22
         477       0.09      0.67      0.15         6
         478       0.20      0.24      0.22        17
         479       0.25      0.04      0.07        23
         480       0.18      0.28      0.22        18
         481       0.04      0.18      0.06        11
         482       0.32      0.34      0.33        35
         483       0.31      0.57      0.40        21
         484       0.47      0.61      0.53        28
         485       0.11      0.21      0.15        14
         486       0.42      0.73      0.53        11
         487       0.21      0.33      0.26        15
         488       0.16      0.21      0.18        38
         489       0.15      0.16      0.15        75
         490       0.89      0.49      0.63        51
         491       0.74      0.74      0.74        19
         492       0.25      0.24      0.24        21
         493       0.07      0.25      0.11        16
         494       0.31      0.83      0.45         6
         495       0.05      0.05      0.05        22
         496       0.27      0.49      0.35        37
         497       0.03      0.05      0.04        20
         498       0.55      0.50      0.52        24
         499       0.04      0.12      0.06        17

   micro avg       0.35      0.45      0.39     47151
   macro avg       0.24      0.34      0.27     47151
weighted avg       0.40      0.45      0.41     47151
 samples avg       0.39      0.44      0.37     47151

Time taken to run this cell : 0:05:57.704523
```

In [56]:

```
from prettytable import PrettyTable
tb = PrettyTable()
tb.field_names= ("Vectorizer",                        "Model",
"  Micro Averaged F1 Score")
tb.add_row(["                tf-idf",                  "Logistic Regression with OVR c
lassifier",0.5005])
tb.add_row(["                Bow",                  "Logistic Regression with OVR clas
sifier",     0.498])
tb.add_row(["                Bow",                   "SGD classifier(Logistic loss) wi
th OVR classifier with parameter tuning",0.4995])
tb.add_row(["                Bow",                    "SGD classifier(Hinge loss)  wit
h OVR classifier with parameter tuning", 0.3931])
print(tb.get_string(titles = "KNN - Observations"))
```

```
+------------------------+----------------------------------------------
-------------------------+--------------------------+
|       Vectorizer       |                                     Model
|   Micro Averaged F1 Score |
+------------------------+----------------------------------------------
-------------------------+--------------------------+
|                 tf-idf |                Logistic Regression with OVR cl
assifier                 |            0.5005          |
|                    Bow |                Logistic Regression with OVR cl
assifier                 |            0.498           |
|                    Bow | SGD classifier(Logistic loss) with OVR classifi
er with parameter tuning |            0.4995          |
|                    Bow |  SGD classifier(Hinge loss)  with OVR classifie
r with parameter tuning  |            0.3931          |
+------------------------+----------------------------------------------
-------------------------+--------------------------+
```

Step by Step Procedure

- Get the Data from csv file and load into the sqlite database.
- Remove the duplicates rows and load the data in a new database.
- Analysis on tags and save the dictionary(Frequecny of each tag) into csv file.
- Text preprocessing and save the preprocessed text in a new database.
- Now we have 42k tags, now we will reduce the unnecessary tags and use only the most frequent 5500 tags that covered 99.08% questions.
- Now we have many rows, high dimensions with 5500 tags, even if we apply a simple logistic regression with one vs rest classifier it'll take above24 hours with my low ram.
- Now i Took a 0.1 million datapoint From Non_duplicate_Rows_table and again did all the steps ->
    - Text Preprocessing and gave high weitage to title by repeating it 3 times.
- Took a first 500 frequent tags that cover the 90% of questions.
- Now apply a logistic regression with tfidf vectorizer.
- Now at last i applied 2 modles logistic regression and linear svm One vs rest classifier with hyperparameter tuning on BOW vectorizer.
- Compare all models