

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

▼ Stack Overflow: Tag Prediction

1. Business Problem

1.1 Description

Description

Stack Overflow is the largest, most trusted online community for developers to learn, share their p careers.

Stack Overflow is something which every programmer use one way or another. Each month, over 5 to learn, share their knowledge, and build their careers. It features questions and answers on a wic The website serves as a platform for users to ask and answer questions, and, through membershi

and answers up or down and edit questions and answers in a fashion similar to a wiki or Digg. As 4,000,000 registered users, and it exceeded 10,000,000 questions in late August 2015. Based on the eight most discussed topics on the site are: Java, JavaScript, C#, PHP, Android, jQuery, Python and

Problem Statement

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

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

1.2 Source / useful links

Data Source : <https://www.kaggle.com/c/facebook-recruiting-iii-keyword-extraction/data>

Youtube : <https://youtu.be/nNDqbUhtIRg>

Research paper : <https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/taggi>

Research paper : <https://dl.acm.org/citation.cfm?id=2660970&dl=ACM&coll=DL>

1.3 Real World / Business Objectives and Constraints

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

2. Machine Learning problem

2.1 Data

2.1.1 Data Overview

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

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

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

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

Size of Train.csv - 6.75GB

Size of Test.csv - 2GB

Number of rows in Train.csv = 6034195

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

Data Field Explanation

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

Id - Unique identifier for each question

Title - The question's title

Body - The body of the question

Tags - The tags associated with the question in a space-separated format (all lowercase,



2.1.2 Example Data point

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

Body :

```
#include<
iostream>\n
#include<
stdlib.h>\n\n
using namespace std;\n\n
int main()\n
{\n
    int n,a[n],x,c,u[n],m[n],e[n][4];\n
    cout<<"Enter the number of variables";\n          cin>>n;\n\n
    cout<<"Enter the Lower, and Upper Limits of the variables";\n
    for(int y=1; y<n+1; y++)\n
    {\n
        cin>>m[y];\n
```

```

        cin>>u[y];\n
    }\n
    for(x=1; x<n+1; x++)\n
    {\n
        a[x] = (m[x] + u[x])/2;\n
    }\n
    c=(n*4)-4;\n
    for(int a1=1; a1<n+1; a1++)\n
    {\n\n
        e[a1][0] = m[a1];\n
        e[a1][1] = m[a1]+1;\n
        e[a1][2] = u[a1]-1;\n
        e[a1][3] = u[a1];\n
    }\n
    for(int i=1; i<n+1; i++)\n
    {\n
        for(int l=1; l<=i; l++)\n
        {\n
            if(l!=1)\n
            {\n
                cout<<a[l]<<"\\t";\n
            }\n
        }\n
        for(int j=0; j<4; j++)\n
        {\n
            cout<<e[i][j];\n
            for(int k=0; k<n-(i+1); k++)\n
            {\n
                cout<<a[k]<<"\\t";\n
            }\n
            cout<<"\\n";\n
        }\n
    }\n\n
    system("PAUSE");\n
    return 0;    \n
}\n

```

\n\n

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

<pre><code>

```

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

```

```

50          2          50\n
50          99         50\n
50          100        50\n
50          50         1\n
50          50         2\n
50          50         99\n
50          50         100\n
</code></pre>\n\n
<p>if the no of inputs is 3 and their ranges are\n
1,100\n
1,100\n
1,100\n
(could be varied too)</p>\n\n
<p>The output is not coming,can anyone correct the code or tell me what's wrong?</p>\n'

```

Tags : 'c++ c'

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

2.2.1 Type of Machine Learning Problem

It is a multi-label classification problem

Multi-label Classification: Multilabel classification assigns to each sample a set of target labels. 1 of a data-point that are not mutually exclusive, such as topics that are relevant for a document. A c 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 score reaches its best value at 1 and worst score at 0. The relative contribution of precision and recall to 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. imbalance.

'Macro f1 score':

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

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

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>

3. Exploratory Data Analysis

3.1 Data Loading and Cleaning

3.1.1 Using Pandas with SQLite to Load the data

```
#Creating db file from csv
#Learn SQL: https://www.w3schools.com/sql/default.asp
if not os.path.isfile('train.db'):
    start = datetime.now()
    disk_engine = create_engine('sqlite:///train.db')
    start = dt.datetime.now()
    chunksize = 180000
    j = 0
    index_start = 1
    for df in pd.read_csv('Train.csv', names=['Id', 'Title', 'Body', 'Tags'], chunksize=ch
        df.index += index_start
        j+=1
        print('{ } rows'.format(j*chunksize))
        df.to_sql('data', disk_engine, if_exists='append')
        index_start = df.index[-1] + 1
    print("Time taken to run this cell :", datetime.now() - start)
```

3.1.2 Counting the number of rows

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

3.1.3 Checking for duplicates

#Learn SQL: <https://www.w3schools.com/sql/default.asp>

```
if os.path.isfile('train.db'):
    start = datetime.now()
    con = sqlite3.connect('train.db')
    df_no_dup = pd.read_sql_query('SELECT Title, Body, Tags, COUNT(*) as cnt_dup FROM data')
    con.close()
    print("Time taken to run this cell :", datetime.now() - start)
else:
    print("Please download the train.db file from drive or run the first to generate train
```

```
df_no_dup.head()
```

```
# we can observe that there are duplicates
```

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

```
# number of times each question appeared in our database
```

```
df_no_dup.cnt_dup.value_counts()
```

```
start = datetime.now()
```

```
df_no_dup["tag_count"] = df_no_dup["Tags"].str.count(' ') + 1
```

```
# adding a new feature number of tags per question
```

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

```
df_no_dup.head()
```

```
# distribution of number of tags per question
```

```
df_no_dup.tag_count.value_counts()
```

```
#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)
```

```
#This method seems more appropriate to work with this much data.
```

```
#creating the connection with database file.
```

```
if os.path.isfile('train_no_dup.db'):
```

```
    start = datetime.now()
```

```
    con = sqlite3.connect('train_no_dup.db')
```

```
    tag_data = pd.read_sql_query("""SELECT Tags FROM no_dup_train""", con)
```

```
    #Always remember to close the database
```

```
    con.close()
```

```
# Let's now drop unwanted column.
```

```
tag_data.drop(tag_data.index[0], inplace=True)
```

```
#Printing first 5 columns from our data frame
```

```
tag_data.head()
```

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

```
else:
```

```
    print("Please download the train.db file from drive or run the above cells to generate
```

3.2 Analysis of Tags

3.2.1 Total number of unique tags

```
# 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'])

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

#'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

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

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

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

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()

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])

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])

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])

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 wi
# quantiles with 0.25 difference
plt.scatter(x=list(range(0,100,25)), y=tag_counts[0:100:25], c='m', label = "quantiles wit

for x,y in zip(list(range(0,100,25)), tag_counts[0:100:25]):
    plt.annotate(s="({} , {})".format(x,y), xy=(x,y), xytext=(x-0.05, y+500))

plt.title('first 100 tags: Distribution of number of times tag appeared questions')
plt.grid()
plt.xlabel("Tag number")
plt.ylabel("Number of times tag appeared")
plt.legend()
plt.show()
print(len(tag_counts[0:100:5]), tag_counts[0:100:5])

# Store tags greater than 10K in one list
lst_tags_gt_10k = tag_df[tag_df.Counts>10000].Tags
#Print the length of the list
print ('{} Tags are used more than 10000 times'.format(len(lst_tags_gt_10k)))
# Store tags greater than 100K in one list
lst_tags_gt_100k = tag_df[tag_df.Counts>100000].Tags
#Print the length of the list.
print ('{} Tags are used more than 100000 times'.format(len(lst_tags_gt_100k)))

```

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 frequently than others, Micro-averaged F1-score is the appropriate m

3.2.4 Tags Per Question

```
#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 we a
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])

print( "Maximum number of tags per question: %d"%max(tag_quest_count))
print( "Minimum number of tags per question: %d"%min(tag_quest_count))
print( "Avg. number of tags per question: %f"% ((sum(tag_quest_count)*1.0)/len(tag_quest_c

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

```
# Plotting 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

3.2.6 The top 20 tags

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

Observations:

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

3.3 Cleaning and preprocessing of Questions

3.3.1 Preprocessing

1. Sample 1M data points
2. Separate out code-snippets from Body
3. Remove Special 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

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

<http://www.sqlitetutorial.net/sqlite-python/create-tables/>

```
def create_connection(db_file):
    """ create a database connection to the SQLite database
        specified by db_file
    :param db_file: database file
    :return: Connection object or None
    """
    try:
        conn = sqlite3.connect(db_file)
        return conn
    except Error as e:
        print(e)

    return None

def create_table(conn, create_table_sql):
    """ create a table from the create_table_sql statement
    :param conn: Connection object
    :param create_table_sql: a CREATE TABLE statement
    :return:
    """
    try:
        c = conn.cursor()
        c.execute(create_table_sql)
    except Error as e:
        print(e)

def checkTableExists(dbcon):
    cursr = dbcon.cursor()
    str = "select name from sqlite_master where type='table'"
    table_names = cursr.execute(str)
    print("Tables in the databse:")
    tables = table_names.fetchall()
    print(tables[0][0])
    return(len(tables))

def create_database_table(database, query):
    conn = create_connection(database)
    if conn is not None:
        create_table(conn, query)
        checkTableExists(conn)
    else:
        print("Error! cannot create the database connection.")
    conn.close()
```

```
sql_create_table = """CREATE TABLE IF NOT EXISTS QuestionsProcessed (question text NOT NULL,
create_database_table("Processed.db", sql_create_table)
```



Tables in the databse:
QuestionsProcessed

```
# http://www.sqlitetutorial.net/sqlite-delete/
# https://stackoverflow.com/questions/2279706/select-random-row-from-a-sqlite-table
start = datetime.now()
read_db = 'train_no_dup.db'
```

```

write_db = 'Processed.db'
if os.path.isfile(read_db):
    conn_r = create_connection(read_db)
    if conn_r is not None:
        reader = conn_r.cursor()
        reader.execute("SELECT Title, Body, Tags From no_dup_train ORDER BY RANDOM() LIMIT

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

```

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

<http://www.bernzilla.com/2008/05/13/selecting-a-random-row-from-an-sqlite-table/>

```

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

    is_code = 0

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

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

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

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

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

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

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

```

```

len_post+=len(question)
tup = (question,code,tags,x,len(question),is_code)
questions_proccesed += 1
writer.execute("insert into QuestionsProcessed(question,code,tags,words_pre,words_post
if (questions_proccesed%100000==0):
    print("number of questions completed=",questions_proccesed)

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

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

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

# 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()

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()

#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 QuestionsProce
    conn_r.commit()
    conn_r.close()

preprocessed_data.head()

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

```

4. Machine Learning Models

4.1 Converting tags for multilabel problems

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

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

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

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

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

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)*10

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

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

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

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

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

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

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

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

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

4.3 Featurizing data

```
start = datetime.now()
vectorizer = TfidfVectorizer(min_df=0.00009, max_features=200000, smooth_idf=True, norm="l1",
                             tokenizer = lambda x: x.split(), sublinear_tf=False, ngram_range=(1,1))
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)

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)

# https://www.analyticsvidhya.com/blog/2017/08/introduction-to-multi-label-classification/
# https://stats.stackexchange.com/questions/117796/scikit-multi-label-classification
# classifier = LabelPowerset(GaussianNB())
"""
from skmultilearn.adapt import MLkNN
classifier = MLkNN(k=21)

# train
classifier.fit(x_train_multilabel, y_train)

# predict
predictions = classifier.predict(x_test_multilabel)
print(accuracy_score(y_test,predictions))
print(metrics.f1_score(y_test, predictions, average = 'macro'))
print(metrics.f1_score(y_test, predictions, average = 'micro'))
print(metrics.hamming_loss(y_test,predictions))

"""

# we are getting memory error because the multilearn package
# is trying to convert the data into dense matrix
# -----
#MemoryError                                Traceback (most recent call last)
```



```
#<ipython-input-170-f0e7c7f3e0be> in <module>()
#----> classifier.fit(x_train_multilabel, y_train)
```

4.4 Applying Logistic Regression with OneVsRest Classifier

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

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

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

4.5 Modeling with less data points (0.5M data points) and more wei

```
sql_create_table = """CREATE TABLE IF NOT EXISTS QuestionsProcessed (question text NOT NUL
create_database_table("Titlmoreweight.db", sql_create_table)
```



Tables in the database:
QuestionsProcessed

```
# 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 = 'Titlmoreweight.db'
train_datasize = 400000
if os.path.isfile(read_db):
    conn_r = create_connection(read_db)
    if conn_r is not None:
        reader =conn_r.cursor()
        # for selecting first 0.5M rows
        reader.execute("SELECT Title, Body, Tags From no_dup_train LIMIT 500001;")
        # for selecting random points
        #reader.execute("SELECT Title, Body, Tags From no_dup_train ORDER BY RANDOM() LIM

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 database:
QuestionsProcessed
Cleared All the rows

4.5.1 Preprocessing of questions

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

```
<li> Remove stop words (Except 'C') </li>
<li> Remove HTML Tags </li>
<li> Convert all the characters into small letters </li>
<li> Use SnowballStemmer to stem the words </li>
```

<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_processed = 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 (len(

len_post+=len(question)
tup = (question,code,tags,x,len(question),is_code)
questions_proccesed += 1
writer.execute("insert into QuestionsProcessed(question,code,tags,words_pre,words_post
if (questions_proccesed%100000==0):
    print("number of questions completed=",questions_proccesed)

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

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

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

```



```

number of questions completed= 100000
number of questions completed= 200000
number of questions completed= 300000
number of questions completed= 400000
Avg. length of questions(Title+Body) before processing: 1239
Avg. length of questions(Title+Body) after processing: 424
Percent of questions containing code: 57
Time taken to run this cell : 0:18:51.052368

```

```

# 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 __

```

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)

```

```

        print("number of data points in sample :", preprocessed_data.shape[0])
conn_r.commit()
conn_r.close()

```



Questions after preprocessed

```

=====
('java.sql.sqlexcept microsoft odbc driver manag invalid descriptor index java.sql.sq
-----
('better way updat feed fb php sdk better way updat feed fb php sdk better way updat
-----
('btnadd click event open two window record ad btnadd click event open two window rec
-----
('sql inject issu prevent correct form submiss php sql inject issu prevent correct fo
-----
('countabl subaddit lebesgu measur countabl subaddit lebesgu measur countabl subaddit
-----
('hql equival sql queri hql equival sql queri hql equival sql queri hql queri replac
-----
('undefin symbol architectur i386 objc class skpsmtpmessag referenc error undefin sym
-----
('java.lang.nosuchmethoderror javax.servlet.servletcontext.geteffectivesessiontrackin
-----
('obtain updat locat use gps servic obtain updat locat use gps servic obtain updat lo
-----

```

__ Saving Preprocessed data to a Database __

```

#Taking 0.5 Million entries to a dataframe.
write_db = 'Titlmoreweight.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 QuestionsProce
conn_r.commit()
conn_r.close()

```

```
preprocessed_data.head()
```



	question	tags
0	java.lang.noclassdeffoundererror javax servlet j...	jsp jstl
1	java.sql.sqlexcept microsoft odbc driver manag...	java jdbc
2	better way updat feed fb php sdk better way up...	facebook api facebook-php-sdk
3	btnadd click event open two window record ad b...	javascript asp.net web
4	sql inject issu prevent correct form submiss p...	php forms

```

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 : 499998
number of dimensions : 2

```

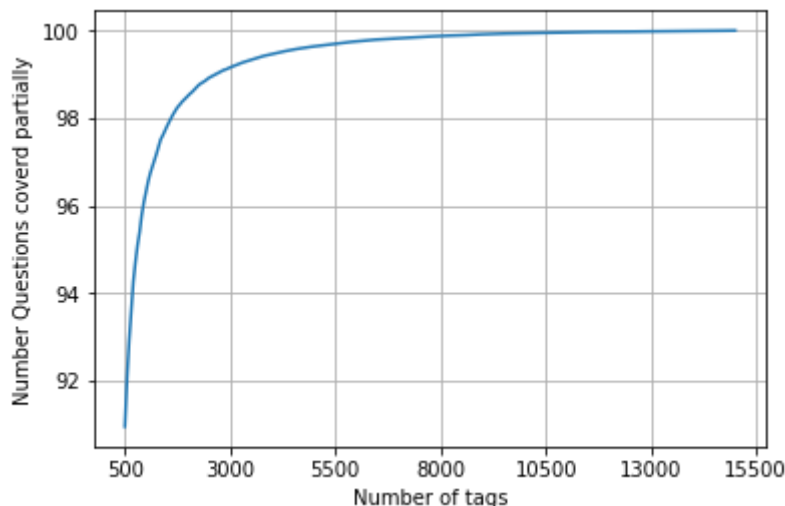
__ Converting string Tags to multilable output variables __

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

__ Selecting 500 Tags __

```
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))

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 covered partially")
plt.grid()
plt.show()
# you can choose any number of tags based on your computing power, minimum is 500(it cover
print("with ",5500,"tags we are covering ",questions_explained[50],"% of questions")
print("with ",500,"tags we are covering ",questions_explained[0],"% of questions")
```



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

```
# 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 ",
```



```
number of questions that are not covered : 45221 out of 499998
```

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

```
y_train = multilabel_yx[0:train_datasize,:]
```

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

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

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



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

Number of data points in test data : (99998, 500)

4.5.2 Featurizing data with Tfidf vectorizer

```
start = datetime.now()
```

```
vectorizer = TfidfVectorizer(min_df=0.00009, max_features=200000, smooth_idf=True, norm="l1",  
                             tokenizer = lambda x: x.split(), sublinear_tf=False, ngram_range=(1, 1))
```

```
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)
```

```
from sklearn.feature_extraction.text import CountVectorizer
```

```
start = datetime.now()
```

```
vectorizer = CountVectorizer(min_df=0.00009, max_features=20000, tokenizer = lambda x: x.split())
```

```
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 : 0:01:42.011543

```
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: (400000, 20000) Y : (400000, 500)

Dimensions of test data X: (99998, 20000) Y: (99998, 500)

```
x_train_multilabel=x_train_multilabel[0:160000]
```

```
y_train=y_train[0:160000]
```

```
x_test_multilabel=x_test_multilabel[0:40000]
```

```
y_test=y_test[0:40000]
```

```
x_train_multilabel=pd.DataFrame.sparse.from_spmatrix(x_train_multilabel)
```

```
y_train=pd.DataFrame.sparse.from_spmatrix(y_train)
```

```
x_test_multilabel=pd.DataFrame.sparse.from_spmatrix(x_test_multilabel)
```

```
y_test=pd.DataFrame.sparse.from_spmatrix(y_test)
```

```
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: (160000, 20000) Y : (160000, 500)

Dimensions of test data X: (40000, 20000) Y: (40000, 500)

```
print(type(x_train_multilabel))
print(type(y_train))
print(type(x_test_multilabel))
print(type(y_test))
```

```
<class 'scipy.sparse.csr.csr_matrix'>
<class 'scipy.sparse.csr.csr_matrix'>
<class 'scipy.sparse.csr.csr_matrix'>
<class 'scipy.sparse.csr.csr_matrix'>
```

4.5.3 Applying Logistic Regression with OneVsRest Classifier

```
start = datetime.now()
classifier = OneVsRestClassifier(SGDClassifier(loss='log', alpha=0.00001, penalty='l1'))
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 : 0.069875

Hamming loss 0.00880855

Micro-average quality numbers

Precision: 0.1881, Recall: 0.4859, F1-measure: 0.2712

Macro-average quality numbers

Precision: 0.1347, Recall: 0.4231, F1-measure: 0.1956

	precision	recall	f1-score	support
0	0.67	0.80	0.73	1936
1	0.37	0.40	0.38	3047
2	0.47	0.52	0.49	2805
3	0.49	0.66	0.56	1644
4	0.51	0.57	0.53	2441
5	0.35	0.46	0.39	1171
6	0.51	0.62	0.56	1942
7	0.46	0.64	0.53	1343
8	0.23	0.28	0.25	1427
9	0.42	0.64	0.51	587
10	0.24	0.33	0.28	888
11	0.40	0.54	0.46	1197
12	0.31	0.42	0.36	964
13	0.26	0.46	0.33	696
14	0.52	0.68	0.59	1102
15	0.28	0.41	0.33	832
16	0.18	0.36	0.24	393
17	0.43	0.70	0.53	790
18	0.23	0.41	0.30	836
19	0.17	0.45	0.24	337
20	0.31	0.53	0.39	522
21	0.15	0.24	0.19	865
22	0.22	0.39	0.28	614
23	0.39	0.70	0.50	313
24	0.17	0.36	0.23	194
25	0.14	0.51	0.23	195
26	0.25	0.50	0.33	344
27	0.29	0.50	0.37	525
28	0.32	0.57	0.41	432
29	0.10	0.34	0.15	151
30	0.12	0.32	0.18	201
31	0.10	0.23	0.14	307
32	0.16	0.47	0.24	130
33	0.26	0.49	0.34	634
34	0.17	0.38	0.23	320
35	0.33	0.67	0.44	297
36	0.14	0.43	0.21	195
37	0.32	0.72	0.44	69
38	0.27	0.57	0.37	332
39	0.19	0.49	0.27	276
40	0.23	0.51	0.31	421
41	0.04	0.27	0.07	77
42	0.20	0.70	0.31	73
43	0.28	0.61	0.38	178
44	0.27	0.57	0.37	367
45	0.20	0.51	0.29	319
46	0.24	0.58	0.34	261
47	0.09	0.35	0.15	121
48	0.16	0.48	0.24	172
49	0.42	0.76	0.54	315
50	0.07	0.23	0.11	122
51	0.06	0.26	0.10	187
52	0.06	0.19	0.09	63

53	0.21	0.46	0.29	271
54	0.15	0.34	0.21	334
55	0.03	0.16	0.04	58
56	0.15	0.39	0.22	313
57	0.12	0.29	0.17	325
58	0.22	0.47	0.30	270
59	0.43	0.82	0.56	267
60	0.07	0.22	0.11	158
61	0.13	0.51	0.21	67
62	0.46	0.85	0.60	396
63	0.34	0.73	0.46	158
64	0.10	0.43	0.16	46
65	0.06	0.25	0.09	121
66	0.11	0.30	0.16	221
67	0.21	0.67	0.32	105
68	0.33	0.66	0.44	222
69	0.17	0.62	0.27	129
70	0.08	0.32	0.13	135
71	0.11	0.38	0.18	125
72	0.24	0.62	0.34	172
73	0.14	0.88	0.25	26
74	0.10	0.55	0.16	42
75	0.40	0.82	0.54	50
76	0.06	0.17	0.09	158
77	0.08	0.45	0.13	74
78	0.17	0.55	0.26	147
79	0.14	0.37	0.21	187
80	0.12	0.45	0.19	128
81	0.08	0.26	0.13	173
82	0.06	0.22	0.09	103
83	0.20	0.58	0.30	168
84	0.13	0.45	0.20	76
85	0.15	0.35	0.21	212
86	0.09	0.27	0.13	184
87	0.08	0.38	0.13	74
88	0.07	0.50	0.13	38
89	0.06	0.27	0.10	59
90	0.19	0.39	0.26	194
91	0.13	0.48	0.20	164
92	0.26	0.66	0.37	194
93	0.17	0.64	0.27	120
94	0.04	0.16	0.06	49
95	0.08	0.33	0.13	86
96	0.13	0.29	0.18	277
97	0.18	0.53	0.27	116
98	0.20	0.73	0.32	59
99	0.10	0.36	0.16	95
100	0.09	0.26	0.14	178
101	0.46	0.76	0.58	136
102	0.56	0.81	0.67	295
103	0.07	0.29	0.11	70
104	0.05	0.17	0.08	113
105	0.22	0.59	0.32	145
106	0.09	0.29	0.14	118
107	0.24	0.64	0.35	146
108	0.18	0.47	0.26	152
109	0.16	0.55	0.25	114
110	0.04	0.16	0.07	91
111	0.04	0.31	0.07	54
112	0.11	0.57	0.18	60
113	0.10	0.43	0.16	72
114	0.06	0.30	0.10	82

115	0.14	0.47	0.22	159
116	0.23	0.60	0.33	154
117	0.13	0.37	0.20	142
118	0.08	0.38	0.13	50
119	0.15	0.45	0.23	168
120	0.03	0.13	0.05	92
121	0.30	0.68	0.41	229
122	0.06	0.28	0.10	60
123	0.02	0.08	0.03	95
124	0.36	0.79	0.50	96
125	0.07	0.38	0.11	56
126	0.25	0.68	0.36	157
127	0.02	0.20	0.04	50
128	0.06	0.25	0.09	95
129	0.13	0.36	0.19	42
130	0.07	0.43	0.12	44
131	0.05	0.21	0.08	76
132	0.15	0.45	0.23	137
133	0.41	0.76	0.54	232
134	0.31	0.84	0.46	49
135	0.05	0.28	0.08	40
136	0.05	0.46	0.10	59
137	0.16	0.40	0.22	126
138	0.07	0.28	0.11	116
139	0.11	0.40	0.17	73
140	0.22	0.63	0.33	114
141	0.06	0.21	0.09	84
142	0.19	0.56	0.28	119
143	0.58	0.85	0.69	194
144	0.08	0.47	0.14	66
145	0.15	0.55	0.24	100
146	0.17	0.47	0.25	167
147	0.09	0.47	0.15	36
148	0.07	0.28	0.11	78
149	0.27	0.65	0.38	187
150	0.07	0.28	0.11	156
151	0.09	0.78	0.15	18
152	0.31	0.71	0.43	150
153	0.09	0.42	0.15	38
154	0.09	0.30	0.14	118
155	0.05	0.19	0.08	95
156	0.24	0.82	0.37	66
157	0.09	0.27	0.14	86
158	0.18	0.67	0.29	33
159	0.06	0.35	0.10	49
160	0.06	0.28	0.11	83
161	0.17	0.66	0.27	76
162	0.05	0.23	0.09	91
163	0.18	0.49	0.26	121
164	0.16	0.51	0.24	95
165	0.14	0.44	0.22	122
166	0.12	0.55	0.19	53
167	0.11	0.42	0.17	97
168	0.13	0.41	0.19	135
169	0.12	0.33	0.18	105
170	0.02	0.17	0.03	29
171	0.12	0.49	0.19	43
172	0.09	0.51	0.16	83
173	0.16	0.45	0.23	130
174	0.05	0.26	0.09	66
175	0.39	0.93	0.55	56

176	0.39	0.77	0.52	159
177	0.26	0.67	0.38	130
178	0.20	0.67	0.31	115
179	0.03	0.23	0.06	60
180	0.08	0.61	0.13	51
181	0.03	0.28	0.05	46
182	0.04	0.24	0.07	51
183	0.18	0.54	0.27	165
184	0.05	0.23	0.09	97
185	0.03	0.19	0.06	32
186	0.07	0.27	0.11	79
187	0.20	0.72	0.32	87
188	0.08	0.34	0.12	80
189	0.16	0.57	0.25	51
190	0.23	0.65	0.34	80
191	0.05	0.27	0.08	83
192	0.18	0.61	0.27	103
193	0.05	0.40	0.09	20
194	0.19	0.62	0.29	89
195	0.04	0.17	0.06	77
196	0.03	0.16	0.06	91
197	0.12	0.31	0.17	112
198	0.11	0.29	0.16	118
199	0.04	0.25	0.07	53
200	0.34	0.79	0.48	86
201	0.13	0.43	0.20	129
202	0.35	0.67	0.46	158
203	0.09	0.24	0.13	111
204	0.09	0.32	0.14	93
205	0.04	0.26	0.08	34
206	0.01	0.04	0.01	50
207	0.10	0.45	0.17	31
208	0.22	0.63	0.32	138
209	0.09	0.30	0.14	106
210	0.15	0.44	0.22	126
211	0.06	0.41	0.10	51
212	0.05	0.28	0.08	32
213	0.05	0.43	0.09	14
214	0.10	0.60	0.17	40
215	0.10	0.51	0.16	57
216	0.20	0.42	0.27	186
217	0.04	0.35	0.07	20
218	0.43	0.89	0.58	100
219	0.19	0.51	0.27	96
220	0.35	0.75	0.48	109
221	0.36	0.79	0.50	129
222	0.12	0.55	0.20	64
223	0.03	0.14	0.05	49
224	0.17	0.59	0.26	85
225	0.10	0.59	0.18	69
226	0.11	0.34	0.17	82
227	0.17	0.51	0.25	69
228	0.04	0.19	0.07	97
229	0.03	0.20	0.06	60
230	0.04	0.25	0.07	77
231	0.10	0.50	0.16	22
232	0.05	0.27	0.09	59
233	0.13	0.56	0.22	77
234	0.09	0.32	0.14	37
235	0.12	0.47	0.19	30
236	0.27	0.77	0.40	39
237	0.07	0.26	0.10	101

238	0.03	0.20	0.05	50
239	0.01	0.05	0.01	21
240	0.08	0.30	0.13	33
241	0.05	0.25	0.08	88
242	0.07	0.42	0.12	36
243	0.03	0.31	0.06	29
244	0.22	0.49	0.30	112
245	0.06	0.30	0.10	44
246	0.37	0.79	0.50	62
247	0.04	0.31	0.07	26
248	0.02	0.12	0.04	51
249	0.02	0.13	0.03	31
250	0.04	0.35	0.08	23
251	0.12	0.47	0.19	73
252	0.03	0.13	0.05	53
253	0.10	0.43	0.16	44
254	0.05	0.26	0.09	50
255	0.07	0.25	0.11	158
256	0.06	0.24	0.09	83
257	0.07	0.45	0.12	40
258	0.29	0.54	0.38	98
259	0.10	0.40	0.16	63
260	0.14	0.41	0.21	95
261	0.10	0.35	0.16	85
262	0.07	0.38	0.11	42
263	0.22	0.41	0.29	162
264	0.05	0.39	0.09	41
265	0.07	0.41	0.12	34
266	0.04	0.33	0.07	12
267	0.03	0.29	0.06	21
268	0.19	0.59	0.28	92
269	0.05	0.38	0.08	21
270	0.15	0.63	0.24	62
271	0.16	0.61	0.26	77
272	0.10	0.37	0.16	89
273	0.45	0.85	0.59	119
274	0.23	0.84	0.36	37
275	0.04	0.23	0.07	40
276	0.02	0.12	0.03	51
277	0.21	0.63	0.31	54
278	0.16	0.68	0.26	57
279	0.37	0.80	0.50	89
280	0.08	0.38	0.13	29
281	0.27	0.64	0.38	81
282	0.22	0.66	0.32	56
283	0.10	0.33	0.16	137
284	0.13	0.52	0.20	29
285	0.18	0.65	0.28	103
286	0.30	0.77	0.43	100
287	0.13	0.57	0.22	51
288	0.02	0.14	0.04	43
289	0.03	0.28	0.05	32
290	0.03	0.24	0.06	50
291	0.10	0.43	0.16	58
292	0.06	0.36	0.11	33
293	0.13	0.62	0.21	50
294	0.08	0.50	0.14	68
295	0.05	0.24	0.08	37
296	0.01	0.17	0.03	12
297	0.04	0.26	0.07	47
298	0.08	0.53	0.14	17

299	0.17	0.73	0.27	22
300	0.04	0.25	0.08	60
301	0.01	0.08	0.02	52
302	0.00	0.05	0.01	21
303	0.21	0.61	0.31	90
304	0.32	0.83	0.46	88
305	0.37	0.81	0.51	64
306	0.24	0.51	0.33	101
307	0.04	0.18	0.07	45
308	0.00	0.03	0.01	40
309	0.04	0.22	0.06	41
310	0.13	0.50	0.21	68
311	0.08	0.45	0.13	20
312	0.03	0.20	0.06	20
313	0.03	0.27	0.05	30
314	0.14	0.48	0.21	81
315	0.19	0.64	0.29	59
316	0.06	0.33	0.10	24
317	0.06	0.24	0.09	78
318	0.16	0.48	0.24	60
319	0.17	0.57	0.26	46
320	0.07	0.41	0.12	61
321	0.17	0.60	0.26	48
322	0.14	0.55	0.22	58
323	0.25	0.55	0.35	155
324	0.04	0.30	0.08	37
325	0.03	0.24	0.06	33
326	0.04	0.29	0.07	52
327	0.03	0.14	0.04	59
328	0.28	0.79	0.41	71
329	0.13	0.48	0.20	66
330	0.08	0.30	0.13	50
331	0.03	0.28	0.06	32
332	0.01	0.20	0.02	10
333	0.04	0.14	0.06	81
334	0.04	0.23	0.07	30
335	0.03	0.14	0.05	36
336	0.11	0.62	0.19	21
337	0.02	0.23	0.03	26
338	0.17	0.55	0.26	53
339	0.01	0.10	0.02	29
340	0.23	0.62	0.33	55
341	0.06	0.33	0.10	12
342	0.06	0.37	0.10	19
343	0.08	0.50	0.14	20
344	0.03	0.14	0.05	21
345	0.12	0.71	0.21	21
346	0.01	0.20	0.03	25
347	0.12	0.45	0.19	47
348	0.01	0.08	0.01	25
349	0.18	0.79	0.30	34
350	0.04	0.50	0.08	8
351	0.05	0.50	0.09	10
352	0.22	0.56	0.31	61
353	0.03	0.30	0.06	10
354	0.04	0.22	0.07	58
355	0.09	0.54	0.16	59
356	0.23	0.50	0.32	145
357	0.24	0.57	0.33	135
358	0.09	0.36	0.14	36
359	0.26	0.67	0.38	67
360	0.07	0.34	0.11	62

361	0.42	0.86	0.57	76
362	0.01	0.12	0.02	24
363	0.33	0.81	0.46	64
364	0.02	0.17	0.04	18
365	0.04	0.30	0.07	27
366	0.28	0.85	0.42	27
367	0.06	0.26	0.10	68
368	0.01	0.08	0.01	13
369	0.02	0.25	0.04	24
370	0.01	0.09	0.01	22
371	0.03	0.19	0.05	36
372	0.22	0.43	0.29	30
373	0.19	0.64	0.29	74
374	0.06	0.40	0.10	30
375	0.02	0.15	0.03	20
376	0.00	0.00	0.00	15
377	0.19	0.42	0.26	79
378	0.01	0.09	0.02	35
379	0.00	0.00	0.00	7
380	0.12	0.51	0.19	57
381	0.04	0.20	0.06	59
382	0.03	0.20	0.05	41
383	0.13	0.48	0.21	69
384	0.03	0.21	0.06	42
385	0.08	0.33	0.12	21
386	0.02	0.12	0.03	24
387	0.05	0.29	0.09	34
388	0.06	0.38	0.10	39
389	0.02	0.13	0.03	45
390	0.35	0.68	0.46	75
391	0.06	0.34	0.10	32
392	0.07	0.62	0.12	8
393	0.01	0.07	0.02	59
394	0.04	0.21	0.07	38
395	0.09	0.39	0.15	61
396	0.07	0.34	0.12	65
397	0.27	0.47	0.34	190
398	0.16	0.83	0.27	36
399	0.02	0.22	0.04	23
400	0.06	0.33	0.10	27
401	0.07	0.44	0.11	36
402	0.11	0.42	0.17	52
403	0.03	0.45	0.06	11
404	0.15	0.83	0.26	6
405	0.07	0.33	0.11	49
406	0.06	0.68	0.11	19
407	0.06	0.27	0.10	11
408	0.11	0.52	0.18	46
409	0.05	0.25	0.08	16
410	0.00	0.00	0.00	5
411	0.08	0.29	0.13	68
412	0.07	0.31	0.12	32
413	0.05	0.18	0.08	61
414	0.02	0.19	0.03	21
415	0.07	0.44	0.12	18
416	0.08	0.64	0.15	25
417	0.01	0.12	0.02	16
418	0.04	0.27	0.06	26
419	0.12	0.50	0.19	50
420	0.01	0.14	0.02	7
421	0.05	0.32	0.08	28

422	0.07	0.41	0.12	32
423	0.05	0.36	0.08	28
424	0.22	0.89	0.35	9
425	0.00	0.00	0.00	6
426	0.02	0.18	0.04	33
427	0.37	0.80	0.50	64
428	0.21	0.86	0.34	14
429	0.42	0.73	0.53	11
430	0.12	0.43	0.19	61
431	0.05	0.24	0.08	49
432	0.04	0.43	0.08	7
433	0.04	0.36	0.08	11
434	0.24	0.67	0.36	15
435	0.07	0.50	0.13	16
436	0.13	0.45	0.20	42
437	0.02	0.10	0.04	49
438	0.35	0.78	0.48	49
439	0.08	0.27	0.12	55
440	0.05	0.22	0.08	45
441	0.03	0.21	0.05	24
442	0.04	0.32	0.07	31
443	0.06	0.24	0.10	25
444	0.04	0.28	0.07	25
445	0.07	0.41	0.13	22
446	0.05	0.31	0.08	16
447	0.01	0.33	0.03	9
448	0.26	0.53	0.35	43
449	0.12	0.43	0.19	44
450	0.09	0.46	0.16	37
451	0.08	0.37	0.13	30
452	0.02	0.23	0.04	13
453	0.01	0.12	0.03	25
454	0.17	0.67	0.27	39
455	0.13	0.47	0.21	98
456	0.02	1.00	0.05	2
457	0.16	0.54	0.24	54
458	0.02	0.30	0.04	10
459	0.03	0.24	0.05	21
460	0.02	0.14	0.03	29
461	0.12	0.50	0.20	28
462	0.02	0.11	0.04	27
463	0.06	0.40	0.11	40
464	0.14	0.37	0.21	49
465	0.06	0.44	0.10	32
466	0.16	0.38	0.22	90
467	0.17	0.67	0.27	49
468	0.10	0.41	0.16	59
469	0.03	0.14	0.05	35
470	0.27	0.83	0.41	42
471	0.10	0.45	0.17	42
472	0.17	0.36	0.23	64
473	0.08	0.41	0.14	22
474	0.05	0.24	0.08	37
475	0.02	0.13	0.03	30
476	0.20	0.66	0.31	29
477	0.30	0.66	0.41	67
478	0.20	0.94	0.33	16
479	0.03	0.16	0.05	37
480	0.19	0.70	0.29	46
481	0.17	0.43	0.24	56
482	0.04	0.25	0.07	28
483	0.34	0.76	0.47	51

484	0.07	0.28	0.11	32
485	0.02	0.11	0.03	37
486	0.07	0.32	0.12	41
487	0.09	0.46	0.14	24
488	0.01	0.08	0.02	26
489	0.10	0.39	0.16	36
490	0.40	0.83	0.54	29
491	0.03	0.24	0.06	37
492	0.00	0.00	0.00	16
493	0.02	0.12	0.04	41
494	0.14	0.35	0.20	26
495	0.15	0.67	0.25	21
496	0.03	0.17	0.05	12
497	0.12	0.46	0.18	39
498	0.05	0.29	0.08	31
499	0.05	0.31	0.09	45
micro avg	0.19	0.49	0.27	67448
macro avg	0.13	0.42	0.20	67448
weighted avg	0.27	0.49	0.33	67448
samples avg	0.29	0.45	0.30	67448

Time taken to run this cell : 1:10:54.946941

SGD with hinge loss

```

start = datetime.now()
classifier = OneVsRestClassifier(SGDClassifier(loss='hinge', alpha=0.00001, penalty='l1'))
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 : 0.069

Hamming loss 0.00882585

Micro-average quality numbers

Precision: 0.1871, Recall: 0.4836, F1-measure: 0.2699

Macro-average quality numbers

Precision: 0.1325, Recall: 0.4194, F1-measure: 0.1928

	precision	recall	f1-score	support
0	0.64	0.79	0.71	1936
1	0.36	0.39	0.37	3047
2	0.47	0.51	0.49	2805
3	0.45	0.66	0.53	1644
4	0.51	0.55	0.53	2441
5	0.34	0.47	0.39	1171
6	0.50	0.62	0.56	1942
7	0.49	0.66	0.56	1343
8	0.22	0.25	0.23	1427
9	0.43	0.64	0.51	587
10	0.23	0.30	0.26	888
11	0.40	0.53	0.46	1197
12	0.29	0.42	0.34	964
13	0.27	0.46	0.34	696
14	0.53	0.69	0.60	1102
15	0.30	0.46	0.37	832
16	0.19	0.37	0.25	393
17	0.44	0.70	0.54	790
18	0.23	0.40	0.29	836
19	0.16	0.43	0.23	337
20	0.33	0.57	0.41	522
21	0.16	0.28	0.20	865
22	0.22	0.39	0.28	614
23	0.37	0.66	0.47	313
24	0.16	0.35	0.22	194
25	0.13	0.50	0.21	195
26	0.25	0.50	0.33	344
27	0.28	0.54	0.37	525
28	0.31	0.55	0.40	432
29	0.08	0.28	0.13	151
30	0.12	0.29	0.17	201
31	0.09	0.22	0.13	307
32	0.12	0.38	0.18	130
33	0.25	0.51	0.34	634
34	0.17	0.39	0.24	320
35	0.33	0.67	0.44	297
36	0.17	0.46	0.24	195
37	0.33	0.78	0.47	69
38	0.20	0.49	0.28	332
39	0.18	0.46	0.26	276
40	0.22	0.50	0.30	421
41	0.04	0.29	0.07	77
42	0.19	0.67	0.30	73
43	0.26	0.58	0.36	178
44	0.26	0.57	0.36	367
45	0.21	0.48	0.29	319
46	0.26	0.61	0.37	261
47	0.08	0.30	0.13	121
48	0.16	0.52	0.24	172
49	0.40	0.71	0.51	315
50	0.07	0.27	0.11	122
51	0.07	0.32	0.12	187
52	0.06	0.21	0.09	63

53	0.22	0.47	0.30	271
54	0.17	0.36	0.23	334
55	0.03	0.16	0.05	58
56	0.16	0.41	0.22	313
57	0.13	0.30	0.18	325
58	0.23	0.46	0.31	270
59	0.45	0.80	0.58	267
60	0.09	0.26	0.13	158
61	0.13	0.46	0.20	67
62	0.49	0.86	0.63	396
63	0.36	0.75	0.48	158
64	0.09	0.41	0.15	46
65	0.08	0.25	0.12	121
66	0.11	0.33	0.17	221
67	0.22	0.66	0.33	105
68	0.39	0.68	0.50	222
69	0.21	0.60	0.31	129
70	0.07	0.30	0.12	135
71	0.10	0.34	0.15	125
72	0.26	0.61	0.36	172
73	0.15	0.85	0.25	26
74	0.12	0.50	0.19	42
75	0.25	0.80	0.38	50
76	0.05	0.15	0.07	158
77	0.07	0.41	0.12	74
78	0.17	0.56	0.27	147
79	0.15	0.38	0.22	187
80	0.14	0.46	0.22	128
81	0.09	0.25	0.13	173
82	0.05	0.22	0.08	103
83	0.20	0.65	0.30	168
84	0.13	0.39	0.20	76
85	0.14	0.40	0.21	212
86	0.08	0.26	0.12	184
87	0.08	0.41	0.13	74
88	0.08	0.42	0.14	38
89	0.06	0.25	0.09	59
90	0.18	0.44	0.25	194
91	0.12	0.38	0.18	164
92	0.24	0.62	0.34	194
93	0.19	0.63	0.29	120
94	0.04	0.18	0.06	49
95	0.06	0.26	0.10	86
96	0.13	0.29	0.18	277
97	0.18	0.58	0.28	116
98	0.21	0.76	0.32	59
99	0.14	0.46	0.22	95
100	0.10	0.27	0.15	178
101	0.41	0.73	0.52	136
102	0.56	0.81	0.66	295
103	0.07	0.29	0.11	70
104	0.06	0.19	0.09	113
105	0.18	0.59	0.28	145
106	0.10	0.26	0.15	118
107	0.28	0.62	0.39	146
108	0.18	0.55	0.27	152
109	0.20	0.61	0.30	114
110	0.04	0.18	0.06	91
111	0.03	0.20	0.05	54
112	0.11	0.53	0.19	60
113	0.10	0.36	0.16	72
114	0.06	0.34	0.10	82

115	0.14	0.45	0.21	159
116	0.23	0.56	0.33	154
117	0.11	0.30	0.16	142
118	0.07	0.38	0.12	50
119	0.14	0.43	0.21	168
120	0.04	0.18	0.07	92
121	0.32	0.66	0.43	229
122	0.06	0.32	0.10	60
123	0.02	0.08	0.03	95
124	0.39	0.77	0.52	96
125	0.06	0.29	0.09	56
126	0.28	0.73	0.40	157
127	0.02	0.22	0.04	50
128	0.05	0.23	0.08	95
129	0.08	0.36	0.13	42
130	0.07	0.43	0.12	44
131	0.03	0.14	0.05	76
132	0.17	0.46	0.25	137
133	0.47	0.72	0.57	232
134	0.26	0.90	0.41	49
135	0.03	0.20	0.05	40
136	0.06	0.47	0.11	59
137	0.15	0.40	0.22	126
138	0.08	0.31	0.12	116
139	0.12	0.45	0.19	73
140	0.21	0.57	0.31	114
141	0.06	0.20	0.09	84
142	0.21	0.56	0.31	119
143	0.59	0.87	0.70	194
144	0.09	0.45	0.15	66
145	0.13	0.51	0.20	100
146	0.19	0.51	0.27	167
147	0.08	0.56	0.15	36
148	0.07	0.31	0.11	78
149	0.27	0.63	0.37	187
150	0.08	0.29	0.12	156
151	0.07	0.72	0.13	18
152	0.26	0.66	0.38	150
153	0.11	0.45	0.18	38
154	0.08	0.22	0.11	118
155	0.04	0.18	0.07	95
156	0.19	0.80	0.31	66
157	0.06	0.19	0.09	86
158	0.15	0.64	0.24	33
159	0.05	0.33	0.09	49
160	0.08	0.36	0.13	83
161	0.14	0.61	0.22	76
162	0.06	0.20	0.09	91
163	0.21	0.51	0.30	121
164	0.16	0.57	0.25	95
165	0.16	0.43	0.23	122
166	0.13	0.58	0.22	53
167	0.08	0.35	0.13	97
168	0.11	0.37	0.17	135
169	0.10	0.34	0.16	105
170	0.02	0.24	0.04	29
171	0.11	0.47	0.17	43
172	0.09	0.48	0.16	83
173	0.16	0.51	0.24	130
174	0.08	0.41	0.13	66
175	0.37	0.93	0.53	56

176	0.38	0.75	0.50	159
177	0.28	0.69	0.40	130
178	0.22	0.66	0.33	115
179	0.04	0.25	0.07	60
180	0.10	0.57	0.16	51
181	0.02	0.22	0.04	46
182	0.02	0.12	0.03	51
183	0.21	0.52	0.30	165
184	0.04	0.23	0.07	97
185	0.03	0.16	0.05	32
186	0.08	0.33	0.13	79
187	0.24	0.71	0.36	87
188	0.05	0.24	0.09	80
189	0.22	0.63	0.33	51
190	0.23	0.66	0.34	80
191	0.05	0.25	0.09	83
192	0.17	0.56	0.27	103
193	0.06	0.50	0.11	20
194	0.20	0.64	0.30	89
195	0.04	0.19	0.07	77
196	0.03	0.16	0.05	91
197	0.10	0.33	0.16	112
198	0.09	0.26	0.14	118
199	0.06	0.26	0.10	53
200	0.33	0.80	0.46	86
201	0.14	0.31	0.19	129
202	0.28	0.66	0.40	158
203	0.10	0.31	0.15	111
204	0.11	0.34	0.16	93
205	0.05	0.26	0.08	34
206	0.02	0.08	0.03	50
207	0.07	0.39	0.12	31
208	0.25	0.62	0.35	138
209	0.11	0.29	0.16	106
210	0.14	0.44	0.21	126
211	0.07	0.45	0.12	51
212	0.03	0.19	0.05	32
213	0.05	0.50	0.08	14
214	0.11	0.57	0.18	40
215	0.07	0.49	0.12	57
216	0.17	0.39	0.24	186
217	0.02	0.30	0.04	20
218	0.38	0.86	0.53	100
219	0.18	0.52	0.26	96
220	0.30	0.75	0.43	109
221	0.37	0.78	0.50	129
222	0.15	0.59	0.24	64
223	0.04	0.20	0.06	49
224	0.18	0.58	0.27	85
225	0.08	0.41	0.13	69
226	0.10	0.35	0.16	82
227	0.13	0.48	0.21	69
228	0.02	0.09	0.04	97
229	0.03	0.17	0.05	60
230	0.04	0.27	0.08	77
231	0.06	0.41	0.10	22
232	0.05	0.20	0.08	59
233	0.11	0.45	0.18	77
234	0.10	0.43	0.16	37
235	0.10	0.47	0.16	30
236	0.34	0.74	0.46	39
237	0.07	0.24	0.10	101

238	0.03	0.26	0.06	50
239	0.01	0.05	0.01	21
240	0.11	0.39	0.17	33
241	0.05	0.26	0.09	88
242	0.10	0.53	0.17	36
243	0.03	0.24	0.05	29
244	0.24	0.54	0.33	112
245	0.05	0.20	0.08	44
246	0.29	0.79	0.43	62
247	0.04	0.38	0.07	26
248	0.02	0.14	0.04	51
249	0.01	0.03	0.01	31
250	0.06	0.48	0.11	23
251	0.10	0.41	0.16	73
252	0.04	0.15	0.06	53
253	0.08	0.39	0.13	44
254	0.04	0.24	0.07	50
255	0.08	0.27	0.12	158
256	0.06	0.23	0.09	83
257	0.06	0.38	0.10	40
258	0.23	0.55	0.33	98
259	0.09	0.32	0.14	63
260	0.11	0.35	0.17	95
261	0.09	0.32	0.14	85
262	0.10	0.48	0.17	42
263	0.23	0.43	0.30	162
264	0.05	0.34	0.08	41
265	0.08	0.47	0.14	34
266	0.01	0.08	0.02	12
267	0.04	0.29	0.07	21
268	0.15	0.50	0.24	92
269	0.03	0.29	0.05	21
270	0.15	0.61	0.24	62
271	0.10	0.58	0.17	77
272	0.09	0.33	0.14	89
273	0.43	0.85	0.57	119
274	0.16	0.68	0.25	37
275	0.05	0.23	0.08	40
276	0.03	0.18	0.06	51
277	0.17	0.69	0.27	54
278	0.14	0.65	0.23	57
279	0.25	0.83	0.38	89
280	0.07	0.41	0.12	29
281	0.31	0.64	0.42	81
282	0.20	0.64	0.30	56
283	0.13	0.42	0.20	137
284	0.14	0.59	0.22	29
285	0.21	0.63	0.31	103
286	0.32	0.83	0.47	100
287	0.19	0.61	0.29	51
288	0.03	0.23	0.05	43
289	0.02	0.16	0.04	32
290	0.03	0.22	0.05	50
291	0.11	0.45	0.17	58
292	0.05	0.30	0.08	33
293	0.16	0.74	0.26	50
294	0.09	0.43	0.14	68
295	0.10	0.38	0.16	37
296	0.02	0.33	0.03	12
297	0.04	0.21	0.07	47
298	0.10	0.65	0.17	17

299	0.17	0.73	0.28	22
300	0.05	0.20	0.08	60
301	0.01	0.06	0.01	52
302	0.02	0.14	0.03	21
303	0.21	0.68	0.32	90
304	0.34	0.81	0.48	88
305	0.36	0.77	0.49	64
306	0.22	0.49	0.30	101
307	0.04	0.20	0.07	45
308	0.00	0.03	0.01	40
309	0.04	0.27	0.06	41
310	0.18	0.56	0.27	68
311	0.08	0.50	0.14	20
312	0.04	0.25	0.07	20
313	0.01	0.10	0.02	30
314	0.16	0.49	0.24	81
315	0.12	0.47	0.20	59
316	0.06	0.29	0.10	24
317	0.06	0.26	0.09	78
318	0.17	0.48	0.25	60
319	0.20	0.59	0.30	46
320	0.09	0.48	0.15	61
321	0.16	0.60	0.26	48
322	0.14	0.60	0.23	58
323	0.27	0.55	0.36	155
324	0.05	0.30	0.08	37
325	0.04	0.27	0.07	33
326	0.04	0.27	0.06	52
327	0.01	0.07	0.02	59
328	0.34	0.83	0.48	71
329	0.12	0.45	0.19	66
330	0.09	0.32	0.14	50
331	0.05	0.28	0.08	32
332	0.01	0.20	0.03	10
333	0.05	0.20	0.08	81
334	0.06	0.30	0.10	30
335	0.07	0.33	0.12	36
336	0.12	0.57	0.19	21
337	0.02	0.19	0.03	26
338	0.16	0.47	0.23	53
339	0.00	0.07	0.01	29
340	0.18	0.56	0.27	55
341	0.05	0.42	0.10	12
342	0.10	0.58	0.17	19
343	0.07	0.45	0.12	20
344	0.10	0.48	0.17	21
345	0.12	0.57	0.20	21
346	0.01	0.12	0.02	25
347	0.14	0.55	0.22	47
348	0.02	0.28	0.04	25
349	0.20	0.76	0.32	34
350	0.07	0.75	0.12	8
351	0.05	0.40	0.08	10
352	0.21	0.61	0.31	61
353	0.04	0.50	0.07	10
354	0.04	0.26	0.08	58
355	0.11	0.49	0.19	59
356	0.23	0.57	0.33	145
357	0.24	0.51	0.33	135
358	0.11	0.53	0.18	36
359	0.22	0.66	0.33	67
360	0.07	0.29	0.11	62

361	0.38	0.87	0.53	76
362	0.03	0.25	0.05	24
363	0.34	0.81	0.47	64
364	0.06	0.33	0.10	18
365	0.05	0.33	0.08	27
366	0.15	0.78	0.26	27
367	0.07	0.28	0.12	68
368	0.02	0.23	0.03	13
369	0.01	0.12	0.02	24
370	0.01	0.09	0.02	22
371	0.03	0.22	0.05	36
372	0.20	0.37	0.26	30
373	0.21	0.62	0.32	74
374	0.05	0.33	0.09	30
375	0.01	0.15	0.03	20
376	0.00	0.00	0.00	15
377	0.15	0.46	0.22	79
378	0.01	0.06	0.01	35
379	0.00	0.00	0.00	7
380	0.15	0.42	0.22	57
381	0.03	0.15	0.04	59
382	0.04	0.24	0.07	41
383	0.15	0.52	0.23	69
384	0.01	0.05	0.01	42
385	0.05	0.19	0.08	21
386	0.02	0.12	0.04	24
387	0.05	0.29	0.09	34
388	0.07	0.38	0.11	39
389	0.02	0.18	0.04	45
390	0.35	0.69	0.47	75
391	0.06	0.31	0.10	32
392	0.08	0.62	0.14	8
393	0.01	0.03	0.01	59
394	0.03	0.16	0.05	38
395	0.08	0.41	0.13	61
396	0.06	0.26	0.09	65
397	0.26	0.58	0.36	190
398	0.15	0.75	0.26	36
399	0.01	0.13	0.02	23
400	0.05	0.33	0.09	27
401	0.05	0.33	0.09	36
402	0.08	0.33	0.12	52
403	0.02	0.36	0.04	11
404	0.12	0.83	0.22	6
405	0.07	0.27	0.10	49
406	0.06	0.58	0.11	19
407	0.04	0.27	0.07	11
408	0.13	0.63	0.21	46
409	0.01	0.06	0.02	16
410	0.00	0.00	0.00	5
411	0.08	0.28	0.12	68
412	0.05	0.28	0.09	32
413	0.05	0.16	0.07	61
414	0.02	0.14	0.03	21
415	0.04	0.33	0.07	18
416	0.08	0.56	0.14	25
417	0.01	0.06	0.01	16
418	0.03	0.31	0.06	26
419	0.12	0.48	0.19	50
420	0.03	0.29	0.05	7
421	0.04	0.29	0.07	28

422	0.06	0.44	0.11	32
423	0.06	0.43	0.11	28
424	0.25	0.89	0.39	9
425	0.00	0.00	0.00	6
426	0.03	0.21	0.05	33
427	0.34	0.81	0.48	64
428	0.16	0.79	0.27	14
429	0.36	0.73	0.48	11
430	0.12	0.46	0.19	61
431	0.08	0.39	0.13	49
432	0.00	0.00	0.00	7
433	0.06	0.55	0.10	11
434	0.15	0.60	0.24	15
435	0.05	0.38	0.09	16
436	0.10	0.43	0.16	42
437	0.05	0.18	0.08	49
438	0.35	0.76	0.47	49
439	0.07	0.31	0.11	55
440	0.04	0.16	0.06	45
441	0.03	0.21	0.06	24
442	0.03	0.19	0.05	31
443	0.06	0.28	0.11	25
444	0.04	0.36	0.08	25
445	0.07	0.45	0.12	22
446	0.05	0.38	0.10	16
447	0.02	0.44	0.04	9
448	0.33	0.60	0.43	43
449	0.12	0.39	0.18	44
450	0.09	0.38	0.15	37
451	0.07	0.33	0.11	30
452	0.03	0.23	0.05	13
453	0.02	0.16	0.03	25
454	0.14	0.62	0.22	39
455	0.10	0.38	0.16	98
456	0.02	0.50	0.03	2
457	0.10	0.39	0.16	54
458	0.01	0.20	0.02	10
459	0.04	0.24	0.06	21
460	0.01	0.07	0.02	29
461	0.14	0.54	0.23	28
462	0.03	0.15	0.05	27
463	0.06	0.42	0.10	40
464	0.16	0.43	0.23	49
465	0.09	0.47	0.15	32
466	0.18	0.47	0.26	90
467	0.18	0.59	0.28	49
468	0.10	0.41	0.16	59
469	0.01	0.03	0.01	35
470	0.28	0.81	0.42	42
471	0.11	0.38	0.17	42
472	0.17	0.45	0.25	64
473	0.07	0.41	0.12	22
474	0.04	0.19	0.06	37
475	0.03	0.17	0.05	30
476	0.20	0.69	0.31	29
477	0.27	0.70	0.39	67
478	0.17	0.88	0.29	16
479	0.04	0.19	0.07	37
480	0.23	0.67	0.34	46
481	0.13	0.34	0.19	56
482	0.05	0.39	0.09	28
483	0.32	0.76	0.45	51

484	0.04	0.22	0.06	32
485	0.02	0.11	0.03	37
486	0.05	0.29	0.09	41
487	0.08	0.46	0.14	24
488	0.01	0.12	0.03	26
489	0.10	0.42	0.16	36
490	0.27	0.83	0.40	29
491	0.04	0.24	0.07	37
492	0.00	0.06	0.01	16
493	0.01	0.07	0.02	41
494	0.08	0.35	0.13	26
495	0.13	0.67	0.22	21
496	0.08	0.33	0.13	12
497	0.10	0.41	0.16	39
498	0.04	0.23	0.07	31
499	0.06	0.36	0.10	45

micro avg	0.19	0.48	0.27	67448
macro avg	0.13	0.42	0.19	67448
weighted avg	0.27	0.48	0.33	67448
samples avg	0.29	0.45	0.30	67448

Time taken to run this cell : 1:07:51.576493

```
from prettytable import PrettyTable
```

```
x = PrettyTable()
```

```
x.field_names = ["Model", "Accuracy", "Hamming loss", "micro-f1", "Macro-f1"]
```

```
x.add_row(["Logistic regression", 0.069875, 0.008, 0.2712, 0.1956])
```

```
x.add_row(["Linear svm", 0.069, 0.008, 0.2699, 0.1928])
```

```
print(x)
```

```

┌───+-----+-----+-----+-----+-----+
|      Model      | Accuracy | Hamming loss | micro-f1 | Macro-f1 |
├───+-----+-----+-----+-----+-----+
| Logistic regression | 0.069875 | 0.008      | 0.2712  | 0.1956  |
| Linear svm        | 0.069    | 0.008      | 0.2699  | 0.1928  |
└───+-----+-----+-----+-----+-----+

```

