Stack_overflow_tagging

March 17, 2020

[0]:
```
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import sqlite3
import csv
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from wordcloud import WordCloud
import re
import os
from sqlalchemy import create_engine # database connection
import datetime as dt
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem.snowball import SnowballStemmer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.multiclass import OneVsRestClassifier
from sklearn.linear_model import SGDClassifier
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 datetime import datetime
```

[0]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id
=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redire
ct_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&response_type=code&scope=email%20http
s%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.c
om%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.reado
nly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly

Enter your authorization code:
··········
```

```
Mounted at /content/drive
```

Load the Processed Data

[0]:
```
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
```

[0]:
```
#Taking 0.5 Million entries to a dataframe.
write_db = '/content/drive/My Drive/data (1)/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 * FROM␣
 ↪QuestionsProcessed LIMIT 100000""", conn_r)
conn_r.commit()
conn_r.close()
```

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

[0]:

	question	…	is_code
0	chang cpu soni vaio pcg grx tri everywher find…	…	0
1	display size grayscal qimag qt abl display ima…	…	1
2	datagrid selecteditem set back null eventtocom…	…	1
3	filter string collect base listview item resol…	…	1
4	disabl home button without use type keyguard c…	…	1

[5 rows x 6 columns]

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

```
number of data points in sample : 100000
number of dimensions : 6
```

Machine Learning Models

[0]:
```
# Converting tags for multilabel problems
# binary='true' will give a binary vectorizer
```

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

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

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

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

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

[0]:
```
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 : 9960 out of  100000
```

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

```
Number of tags in sample : 18688
number of tags taken : 500 ( 2.675513698630137 %)
```

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

[0]:
```
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,:]
```

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

Featurizing data

[0]:
```
start = datetime.now()
vectorizer = CountVectorizer(min_df=0.00009, max_features=20000,  \
                             tokenizer = lambda x: x.split(), 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 : 0:01:00.144274
```

[0]:
```
print("Dimensions of train data X:",x_train_multilabel.shape, "Y :",y_train.
 ↪shape)
print("Dimensions of test data X:",x_test_multilabel.shape,"Y:",y_test.shape)
```

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

Applying Logistic Regression with OneVsRest Classifier on Bag of Words

[0]:
```
# Applying Logistic Regression with OneVsRest Classifier
from sklearn.model_selection import GridSearchCV
param={'estimator__alpha': [10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1]}
classifier = OneVsRestClassifier(SGDClassifier(loss='log', penalty='l1'))
gsv = GridSearchCV(estimator = classifier, param_grid=param, cv=3, verbose=0,␣
 ↪scoring='f1_micro',n_jobs=-1)
gsv.fit(x_train_multilabel, y_train)
best_alpha = gsv.best_estimator_.get_params()['estimator__alpha']
print('value of alpha after hyperparameter tuning : ',best_alpha)
print('-----------------------------------------------------------')
```

```
value of alpha after hyperparameter tuning :  0.0001
-----------------------------------------------------------
```

[0]:
```
start = datetime.now()
#best_alpha = gsv.best_estimator_.get_params()['estimator__alpha']
classifier = OneVsRestClassifier(SGDClassifier(loss='log', alpha=best_alpha,␣
 ↪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 : 0.1433
Hamming loss  0.0038797
Micro-average quality numbers
Precision: 0.4593, Recall: 0.4123, F1-measure: 0.4345
Macro-average quality numbers
Precision: 0.3478, Recall: 0.3329, F1-measure: 0.3265
Time taken to run this cell : 0:07:46.844164
```

Applying Linear SVM with OneVsRest Classifier on Bag of Words

[0]:
```
param={'estimator__alpha': [10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1]}
classifier = OneVsRestClassifier(SGDClassifier(loss='hinge', penalty='l1'))
gsv = GridSearchCV(estimator = classifier, param_grid=param, cv=3, verbose=0,␣
 ↪scoring='f1_micro',n_jobs=-1)
gsv.fit(x_train_multilabel, y_train)

best_alpha = gsv.best_estimator_.get_params()['estimator__alpha']
print('value of alpha after hyperparameter tuning : ',best_alpha)
print('-----------------------------------------------------------------')
```

```
value of alpha after hyperparameter tuning :  0.0001
-----------------------------------------------------------------
```

[0]:
```
start = datetime.now()
#best_alpha = gsv.best_estimator_.get_params()['estimator__alpha']
classifier = OneVsRestClassifier(SGDClassifier(loss='hinge', alpha=best_alpha,␣
 ↪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 : 0.1365
Hamming loss  0.0040282
Micro-average quality numbers
Precision: 0.4381, Recall: 0.4043, F1-measure: 0.4205
Macro-average quality numbers
Precision: 0.3162, Recall: 0.3260, F1-measure: 0.3112
Time taken to run this cell : 0:06:28.549833
```

[0]:
```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Sr.No","MODEL","FEATURIZATION","ALPHA",'MICRO_F1_SCORE']
x.add_row(["1", 'OneVsRest+SGD=Logistic Regression', "Bag-of-words",0.0001,0.
 ↪4345])
x.add_row(["2", 'OneVsRest+SGD=Linear SVM', "Bag-of-words",0.0001,0.4205])
print(x)
```

```
+-------+-----------------------------------+--------------+--------+----------
------+
| Sr.No |               MODEL               | FEATURIZATION | ALPHA  |
MICRO_F1_SCORE |
+-------+-----------------------------------+--------------+--------+----------
```

```
------+
|   1   | OneVsRest+SGD=Logistic Regression |  Bag-of-words | 0.0001 |
0.4345     |
|   2   |      OneVsRest+SGD=Linear SVM     |  Bag-of-words | 0.0001 |
0.4205     |
+-------+-----------------------------------+---------------+--------+----------
------+
```