Stack Overflow Tag Prediction

PG Project
Monsoon 2019
Guided by: Dr. Radhika Mamidi

Team Members:

Yaswanth Koravi (2018202011)

Aishwarya Shivachandra (2018202005)

Outline

Problem Statement

Provided a bunch of questions, by using the text in the title and description, suggest the tags based on the content of the question posted on Stack Overflow.

This is a Multi-label Classification problem.

What is Stack Overflow Tag prediction ?

Stack Overflow is the largest, most trusted online community for developers to learn, share their programming knowledge, and build their careers. It features questions and answers on a wide range of topics in computer programming. The website serves as a platform for users to ask and answer questions, and, through membership and active participation, to vote questions and answers up or down and edit questions and answers.

A question in Stack Overflow contains three segments Title, Description and Tags. Stack Overflow predicts the tags using the text present in title and description.

Motivation

Stack Overflow features questions and answers on a wide range of technical topics. The website serves as a platform for users to ask and answer questions, to vote questions and answers up or down and edit questions and answers. Based on the type of tags assigned to questions, the most discussed topics on the site are: C#, Java, PHP, JavaScript, Android, jQuery,etc.

Stack Overflow detects that a particular user has answered a lot of questions related to a particular subject in the past and thus recognizes that he is an expert in the subject. So it raises a notification to the right people, such that they are able to answer the questions related to that subject. Stack Overflow can predict the tags using the text in title and description.

This is extremely business critical. Thus our model is most suitable for such question and answer forum related websites so that whenever a question is posted, there is high chance that it will be answered quickly if we predict the tags more accurately and the appropriate subject experts are notified for each question.

Data

Data contains 4 fields:

- Id - Unique identifier for each question
- Title - The question’s title
- Body - The body of the question
- Tags - The tags associated with the question

Size of Train.csv - 6.75GB

Number of rows in Train.csv = 60,34,195

Data Sample Point

ID: "27"

TITLE: "Is my understanding of MAKE correct?"

BODY: "<p>Taking the GNU make application as an example, when I create a makefile it is likened to a Visual Studio project file in that it contains the list of files and other items needed to compile an application. The makefile contains the path to the compiler as well as the path to all the files that could possibly be used in the compiling operation. The MAKE application reads the makefile and combines the sources into 1 file which the compiler then reads and converts into the relevant assembly/machine code.</p>

<p>The #include directive is a directive that tells the make file to:""include the contents of this file into this point of the larger file that you're making to feed to the compiler then resume with the current file"".</p><p>The MAKE application also tells the compiler where to ""emit"" the result of the compile operation once the compiling is completed. In this regard a Visual Studio .csproj file is technically a XML based makefile for a Microsoft specific MAKE application.</p><p>Is this understanding of makefiles correct?</p>"

TAGS: "visual-studio makefile gnu"

Performance Metrics

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.

If we want to predict any of the tags we should have a high precision value i.e, we have to be really sure that the predicted tag belongs to the given question. Also, we want to have a high Recal rate, which means If the tag actually supposed to be present, we want it to be present most number of times. So mertic considered is f1 score.

Exploratory Data Analysis- Tags analysis

- Minimum number of tags per question: 1
- Maximum number of tags per question: 5
- Avg. number of tags per question: 2.887779
- Questions with 3 tags are maximum

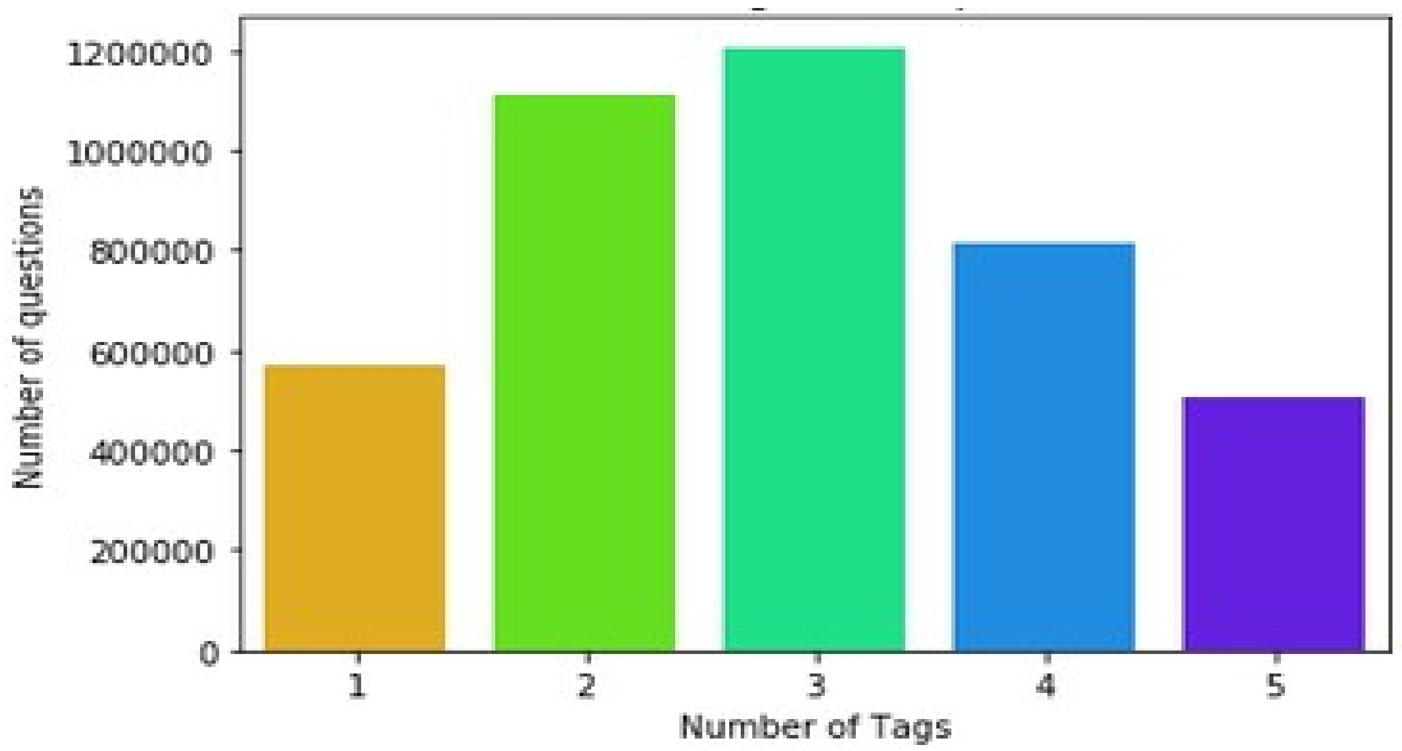
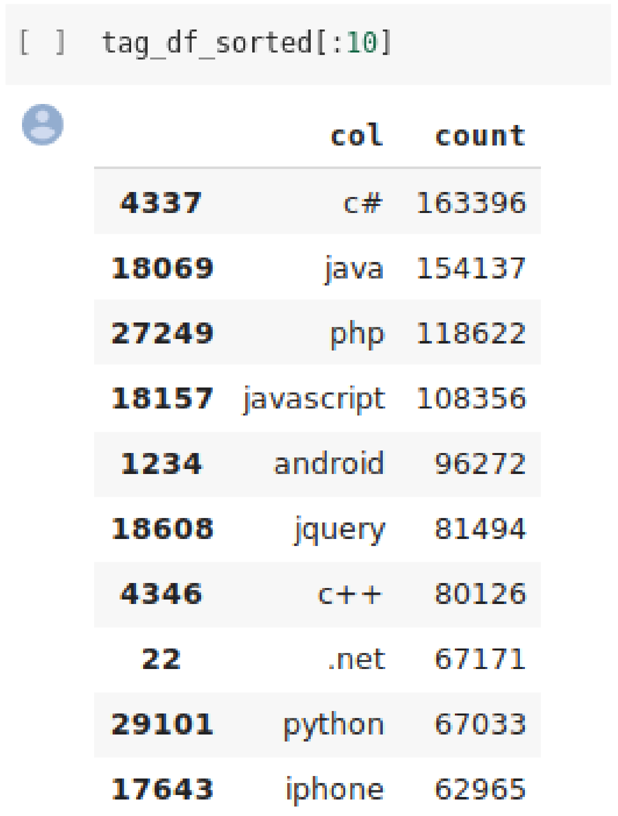


Fig1 :Question wise tag count

Dataset - EDA

Most frequent tags



```
[ ]  tag_df_sorted[:10]
```

	col	count
4337	c#	163396
18069	java	154137
27249	php	118622
18157	javascript	108356
1234	android	96272
18608	jquery	81494
4346	c++	80126
22	.net	67171
29101	python	67033
17643	iphone	62965

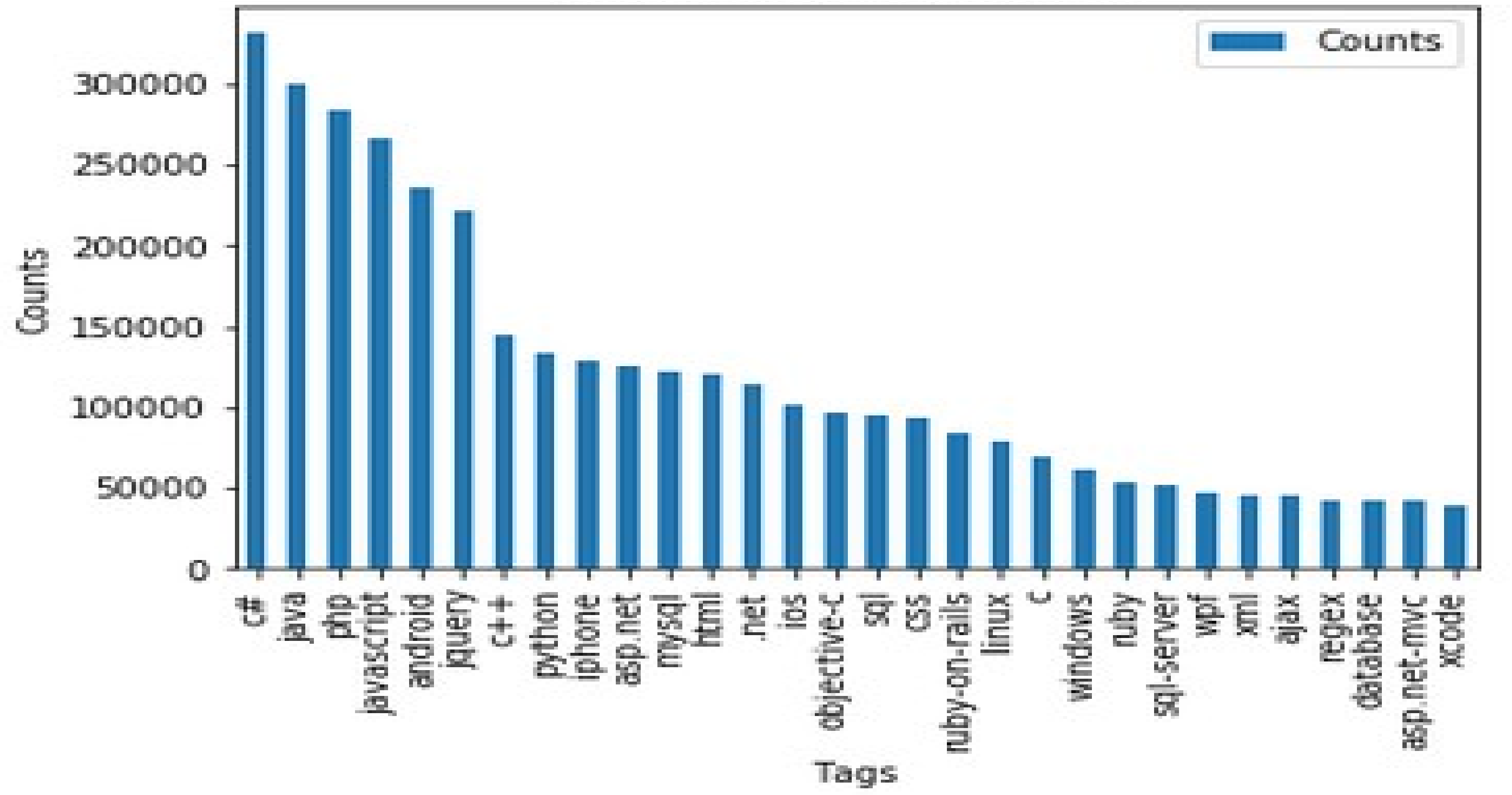


Fig2 :Frequency of tags

Dataset - EDA

- With 5500 tags 99.041% of questions are covered.
- With 500 tags 90.013% of questions are covered.

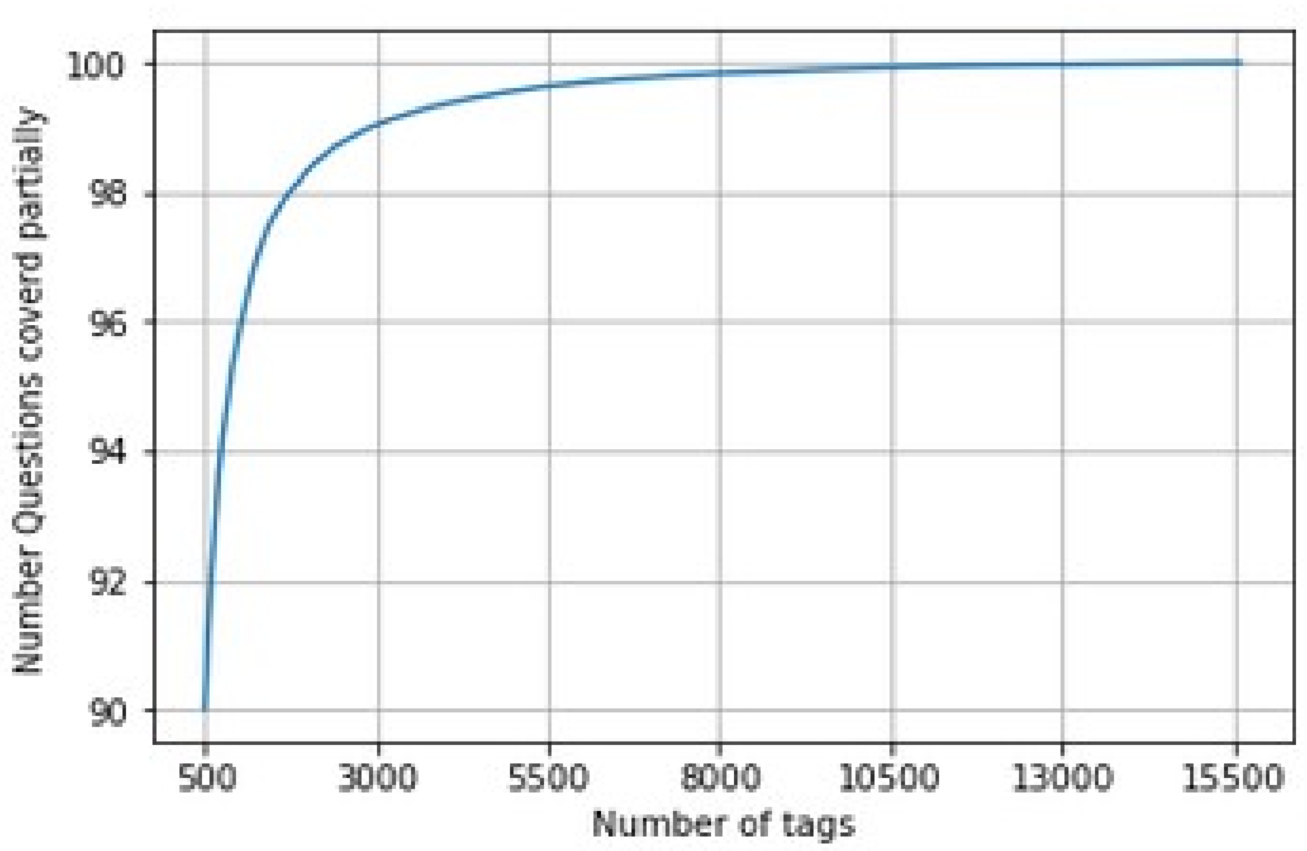


Fig5 : Number Questions covered per number of tags

Data cleaning

We cleaned the data by removing those text from the dataset that do not contribute to tag prediction:

- Remove stop words.
- Convert all the characters into small letters.
- Use SnowballStemmer to stem the words.
- Remove code snippets from the body.
- Concatenate title and body into single column.

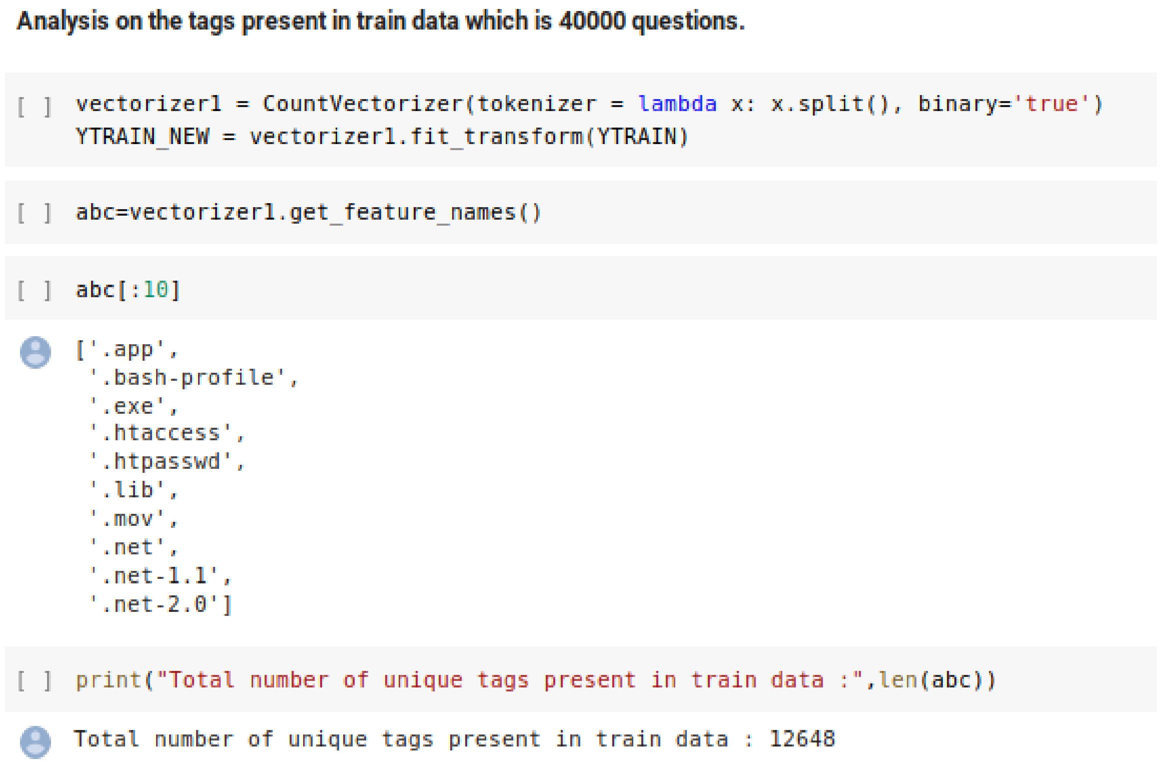
Text Pre-processing

Due to constraint on the computing power of our system, we have taken a a subset of data from Train.csv and sampled only 50,000 questions.

Then, split the given dataset using 80:20 train-validation split. Now number of rows in train data are 40,000.

Converted the text present in train data using TFIDF vectorizer with max features set as 60000 so that the maximum columns for the converted row will be 60000 .We also considered ngram range from 1-3. This technique is used to convert the text data present in train and test to numeric data.

Total number of unique tags present in train data: 12648



Analysis on the tags present in train data which is 40000 questions.

```
[ ] vectorizer1 = CountVectorizer(tokenizer = lambda x: x.split(), binary='true')
    YTRAIN_NEW = vectorizer1.fit_transform(YTRAIN)
```

```
[ ] abc=vectorizer1.get_feature_names()
```

```
[ ] abc[:10]
```

```
['.app',
 '.bash-profile',
 '.exe',
 '.htaccess',
 '.htpasswd',
 '.lib',
 '.mov',
 '.net',
 '.net-1.1',
 '.net-2.0']
```

```
[ ] print("Total number of unique tags present in train data :",len(abc))
```

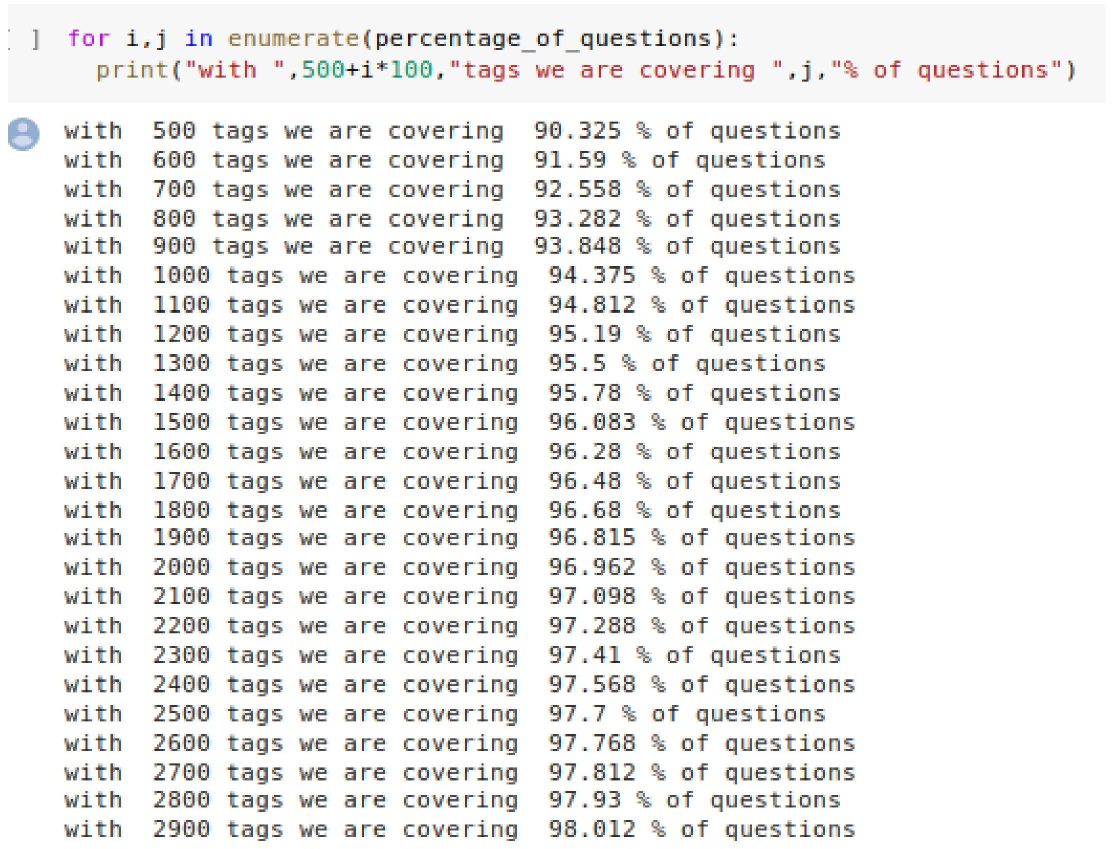
```
Total number of unique tags present in train data : 12648
```

Text Pre-processing

If we consider all the tags ppresent in train data then we have to train nearly 12580 models on complete which is not computationally efficient, so we are compromising on the number tags based on the partial coverage of the questions.

Observation: By choosing top 500 tags we can cover 90% of the questions in train data.

Using **bag of words** technique and vocabulary as the top 500 tags choosen, we converted tags columns into sparse matrix which is suitable for modeling.



```
for i,j in enumerate(percentage_of_questions):
  print("with ",500+i*100,"tags we are covering ",j,"% of questions")
```

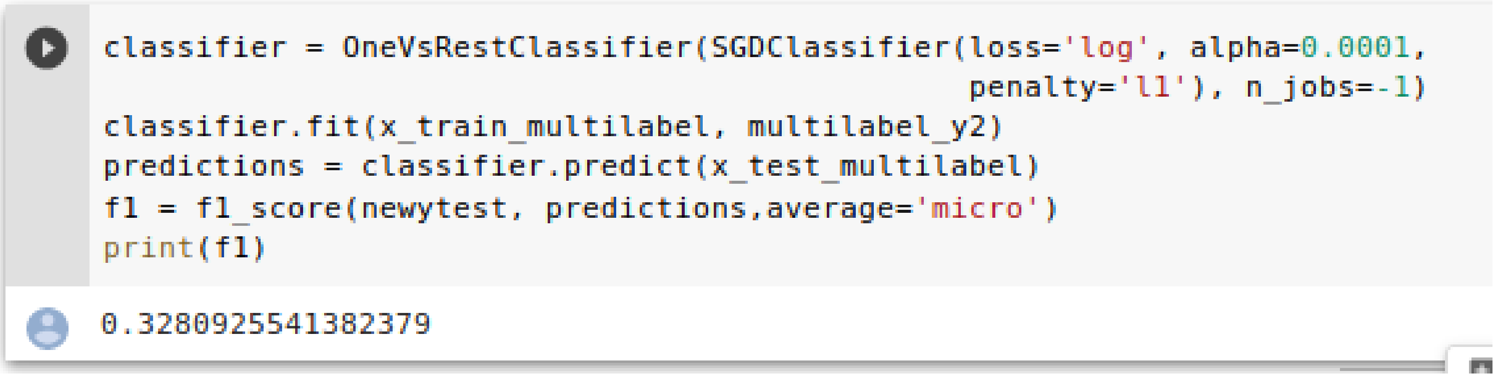
```
with  500 tags we are covering  90.325 % of questions
with  600 tags we are covering  91.59 % of questions
with  700 tags we are covering  92.558 % of questions
with  800 tags we are covering  93.282 % of questions
with  900 tags we are covering  93.848 % of questions
with  1000 tags we are covering  94.375 % of questions
with  1100 tags we are covering  94.812 % of questions
with  1200 tags we are covering  95.19 % of questions
with  1300 tags we are covering  95.5 % of questions
with  1400 tags we are covering  95.78 % of questions
with  1500 tags we are covering  96.083 % of questions
with  1600 tags we are covering  96.28 % of questions
with  1700 tags we are covering  96.48 % of questions
with  1800 tags we are covering  96.68 % of questions
with  1900 tags we are covering  96.815 % of questions
with  2000 tags we are covering  96.962 % of questions
with  2100 tags we are covering  97.098 % of questions
with  2200 tags we are covering  97.288 % of questions
with  2300 tags we are covering  97.41 % of questions
with  2400 tags we are covering  97.568 % of questions
with  2500 tags we are covering  97.7 % of questions
with  2600 tags we are covering  97.768 % of questions
with  2700 tags we are covering  97.812 % of questions
with  2800 tags we are covering  97.93 % of questions
with  2900 tags we are covering  98.012 % of questions
```

Model implementation

By using those 500 tags as the vocabulary for the training and,

by using simple logistic model with SGDClassifier using one vs rest technique,

we got an F1 score of 0.32

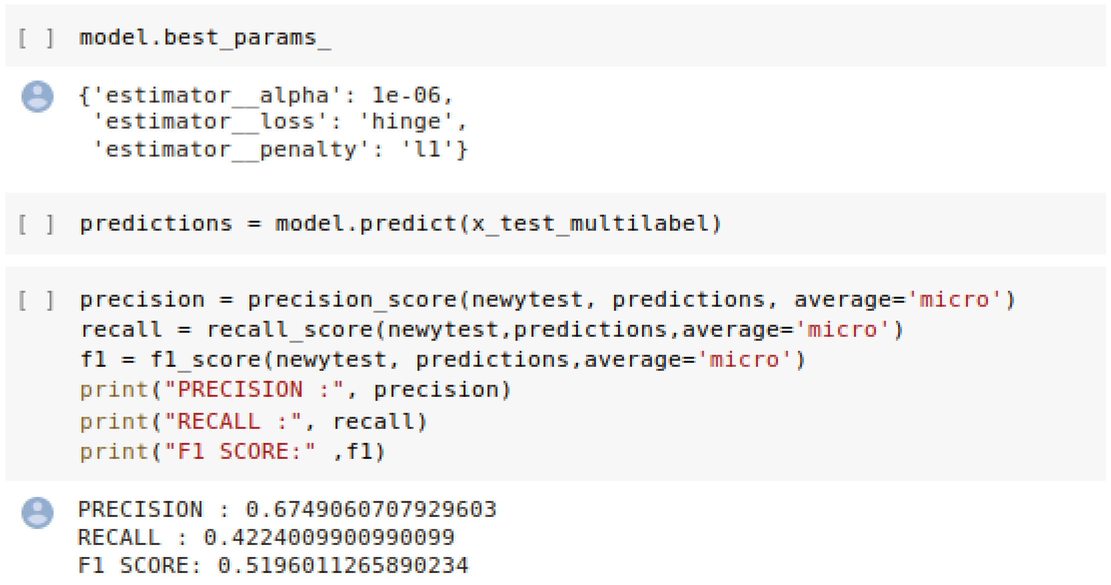


```
classifier = OneVsRestClassifier(SGDClassifier(loss='log', alpha=0.0001,
                                               penalty='l1'), n_jobs=-1)
classifier.fit(x_train_multilabel, multilabel_y2)
predictions = classifier.predict(x_test_multilabel)
f1 = f1_score(newytest, predictions,average='micro')
print(f1)
```

```
0.3280925541382379
```

Hyper-parameter Tuning

With hyper parameter tuning on alpha and loss functions, the **F1 score is further improved to 0.51**



```
[ ] model.best_params_
```

```
{'estimator__alpha': 1e-06,
 'estimator__loss': 'hinge',
 'estimator__penalty': 'l1'}
```

```
[ ] predictions = model.predict(x_test_multilabel)
```

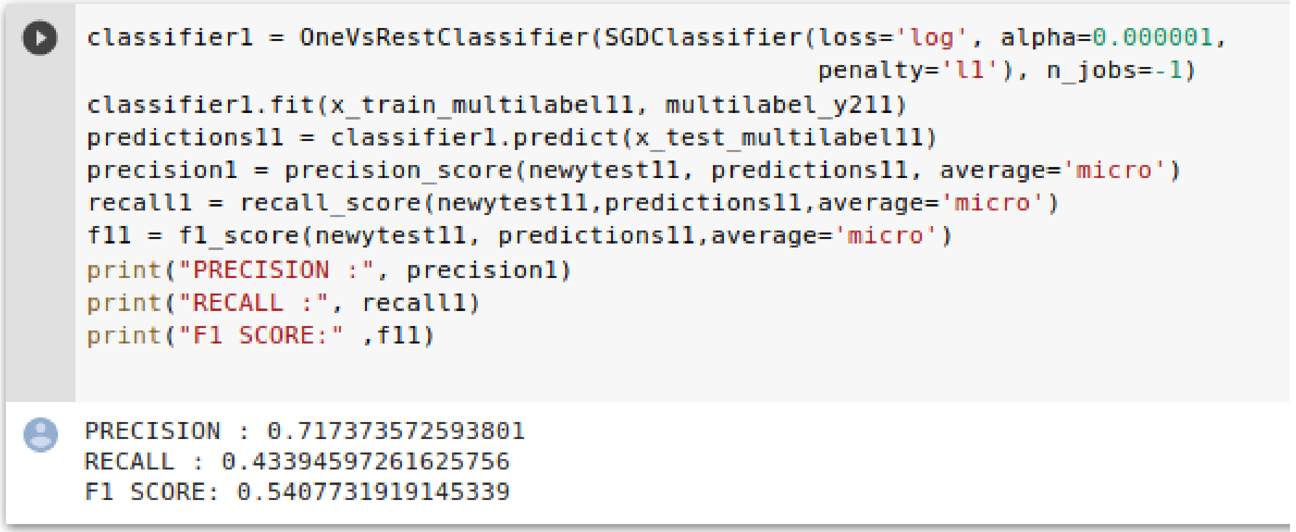
```
[ ] precision = precision_score(newytest, predictions, average='micro')
    recall = recall_score(newytest,predictions,average='micro')
    f1 = f1_score(newytest, predictions,average='micro')
    print("PRECISION :", precision)
    print("RECALL :", recall)
    print("F1 SCORE:" ,f1)
```

```
PRECISION : 0.6749060707929603
RECALL : 0.4224009900990099
F1 SCORE: 0.5196011265890234
```

Feature Engineering

Considering the fact that title has more important than body, we gave more weightage to title.

Thus, on hyper parameter tuning for both loss function and alpha and with input as (3*title +description) the **F1 score got improved to 0.54**



```
classifier1 = OneVsRestClassifier(SGDClassifier(loss='log', alpha=0.000001,
                                                penalty='l1'), n_jobs=-1)
classifier1.fit(x_train_multilabel11, multilabel_y211)
predictions11 = classifier1.predict(x_test_multilabel11)
precision1 = precision_score(newytest11, predictions11, average='micro')
recall1 = recall_score(newytest11,predictions11,average='micro')
f11 = f1_score(newytest11, predictions11,average='micro')
print("PRECISION :", precision1)
print("RECALL :", recall1)
print("F1 SCORE:" ,f11)
```

```
PRECISION : 0.717373572593801
RECALL : 0.43394597261625756
F1 SCORE: 0.5407731919145339
```

Result and Conclusion

After tuning hyper-parameters ’alpha’ and loss functions, and thus obtaining highly improved F1 score, we train our model for predicting tags based on Stack Overflow questions.

Using this trained model, we developed a GUI that:

- inputs question title and description
- outputs one or more tags which are most relevant to the question

GUI sample

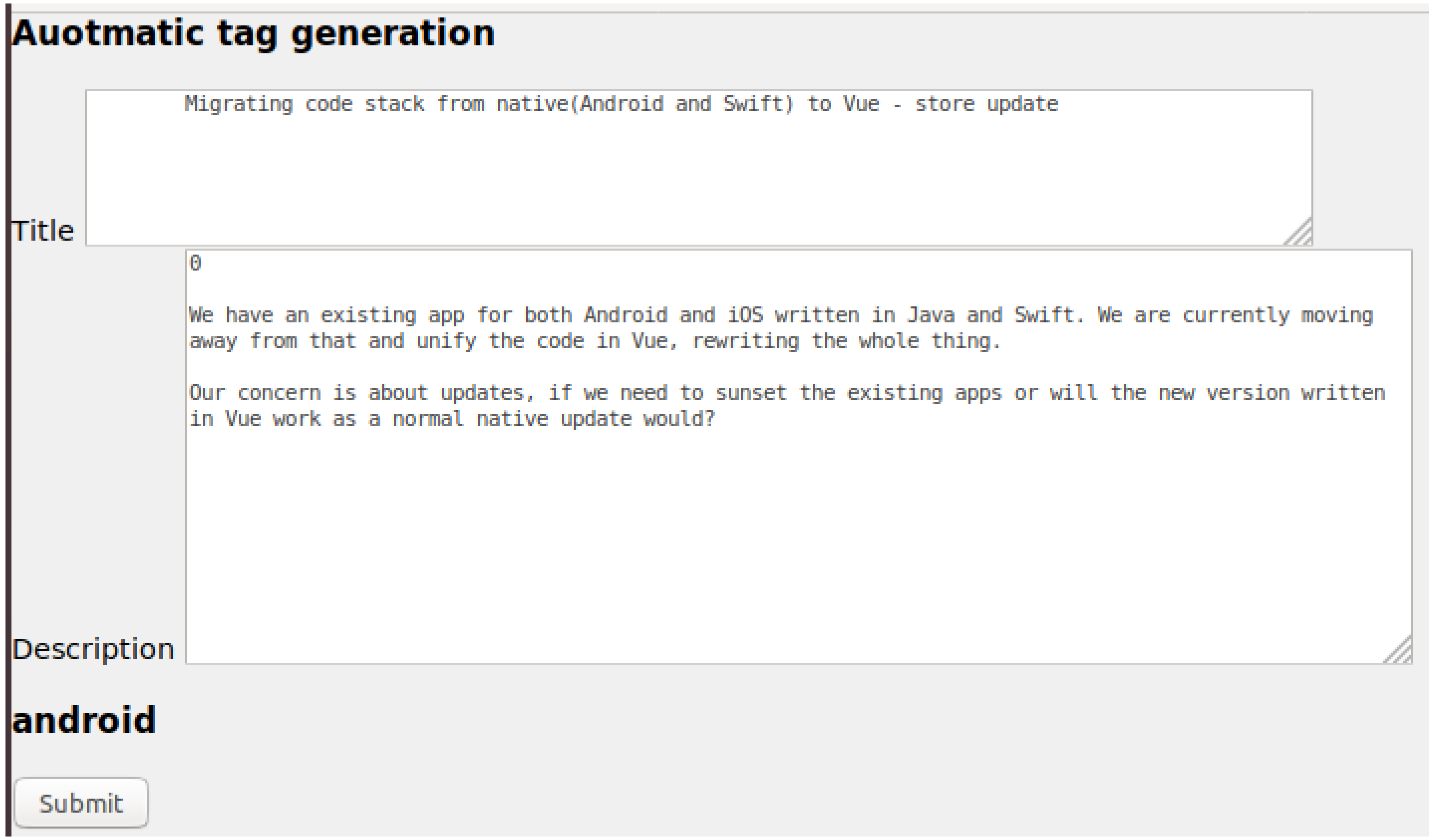
Title: Migrating code stack from native (Android and Swift) to Vue - store update

Description: We have an existing app for both Android and iOS written in Java and Swift. We are currently moving away from that and unify the code in Vue, rewriting the whole thing.

Our concern is about updates, if we need to sunset the existing apps or will the new version

written in Vue work as a normal native update would?

Predicted Tag: android



Auotmatic tag generation

Title: Migrating code stack from native(Android and Swift) to Vue - store update

Description:

0

We have an existing app for both Android and iOS written in Java and Swift. We are currently moving away from that and unify the code in Vue, rewriting the whole thing.

Our concern is about updates, if we need to sunset the existing apps or will the new version written in Vue work as a normal native update would?

android

Thank You.