My Blog post: https://medium.com/analytics-vidhya/stack-overflow-question-tagger-96a045c1ca4c

It won't be wrong to assert that every developer/engineer/student has used the website Stack Overflow more than once in their journey. Widely considered as one of the largest and more trusted websites for developers to learn and share their knowledge, it presently hosts more than 10,000,000 questions. In this post, I try to predict the question tag based on the question text asked on the website. The most common tags on the website include Java, JavaScript, C#, PHP, Android amongst others.

Proper prediction of the tags is important to ensure that the questions are suggested to users having substantial experience in answers related to the topic suggested.

Reference links for project idea-

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

Notes-

1. I have used SQLite 3 with Pandas for reading the data as the amount of data is too large for Pandas to handle it. SQLite 3 is smoothly integrated with Pandas.

2. If using Colab Free, your RAM and disk space may reach maximum limits.

3. Code credit for removal of HTML tags:

https://stackoverflow.com/questions/3075130/what-is-the-difference-between-and-regular-expressions

4. A few SQL functions are written. One for connecting SQLite server with out database. One for creating table using SQL commands from our connection,1 for checking if table exists within the connection and printing it. One for both connecting the database and creating table and printing it.

5. Create Table query is passed as a parameter to the create_database_table function. The table contains question text, code text, tags etc. as mentioned.

```
#http://www.sqlitetutorial.net/sqlite-python/create-tables/
```

6. Using SQL cursor I read row wise no_dup_train sqlite table (created before). The questions with code in their body have the code removed. The titles are encoded in

UTF-8. Title and question is combined. Characters other than alphanumeric are removed.

7. Features are added to the database table like code, words_pre, words_post, len_pre etc. for each row iteration.

8. A few rows from the sqlite table QuestionsProcessed is printed using reader

9. 1 Million entries from the table (just the questions and tags) is read into a pd dataframe.

10. Tags are converted into one-hot-encoded vectors (using binary Count Vectorizer)

11. Percentage of questions covered by tags is measured. And the top 5500 tags are chosen.

12. Functions called tags to choose returns the tags data points with multilabels as top n tags (sorted)

13. The function is applied on our y data points. Now it contains 5500 tags as labels. Shape= (no of data points, 5000)

14. Train_test split of 80:20 is done. X=questions and y= tags (one-hot-encoded)

14. Tf-idf vectorizer with modified min_df and max_features is used on x_train and x_test.

15. Modified Multi-label KNN is applied for multi-labels. But it returned memory error. https://www.analyticsvidhya.com/blog/2017/08/introduction-to-multi-label-classification/

16. Hence, a OnevsRest Logistic Regression classifier was used. https://prakhartechviz.blogspot.com/2019/02/multi-label-classification-python.html#:~:text=One%2DVs%2DRest%20is%20one,time%20and%20leaving%20rest%20out.

Basically, a onevsRest for multilabel classification is equivalent to binary relevance.

17. First, I used SGD classifier (OnevsRest) with log-loss on the data. This gave a reasonably good metrics like accuracy, hamming loss, micro-f1 score, macro-f1 score etc. Next, I used LogisticRegression function with 'l1 regularization' (due to data sparsity). This slightly improved the metrics.

18. The model is saved. The same procedure is next repeated giving thrice Iight to the question body and one to the title. 0.5 million data points are used and 500 top labels. This again improved the performance.

19. Future improvements-

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)

20. Conclusion: Since our data is text primarily, it is high-dimensional. Linear models like Log-Reg and Linear SVMs work best in such case. Thus, more complex models are avoided. Moreover, because OnevsRest classifier basically trains n number of models (n=no. of classes), complex models would be extremely computationally expensive.