Shoot the Long Waiting Pain — Recommendation Pipeline for Stack Overflow

Chen Yiqiu, Gao Jin, Li Xinlin, Qu Mingyu, Tan Sijie[1]

https://github.com/Benjaminlxl/BT5153-Recommendation-System-for-Stack-Overflow

1. Introduction

Which Q&A website is programmers' top preference when they seek help? Most of the programmers would answer: "stack overflow!" Stack Overflow (SO), under the network of Stack Exchange, has 14 million users and 11 million visits per day. There are 21 million questions posted and 31 million answers so far. Among the answers received, the users can select the one he or she thinks most constructive to be accepted. By asking good questions and providing useful answers, the users can earn reputation points and receive badges for their contribution.

As one is posting his or her question, he or she may wonder how long it might take to get a solid answer? In the project, we wish to predict the waiting time to receive a solid answer[2] by identifying potential influential features. By digging into the post content, we also wish to launch a recommendation system that provides the most relevant posts to one certain question posted to improve user experience by shortening their waiting time.

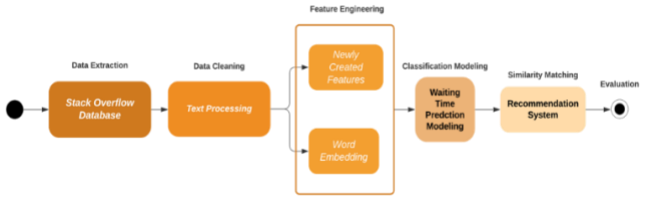

Figure 1. Flow chart of the whole report

2. Related Works

As the popularity of Q&A forums rises, lots of researchers have studied problems related to our tasks. In the paper 'Exploiting User Feedback to Learn to Rank Answers in Q&A Forums' (Daniel et al., 2013), author Daniel Hasan Dalip proposed a learning-to-rank approach for ranking answers in Q&A forums. He and his fellow researchers found out that review and user features are the most important features in the Q&A forums, and text features are useful for assessing the quality of new answers. This paper provides us insights for feature engineering. Regarding the second task, community-based question-to-question similarity and question retrieval has been studied for a long time. In the early stage, Xue applied a translation model to detect duplicate questions. (Xue et al., 2008). The recent work includes tree kernel with neural networks (Romeo et al., 2016) and encoder-decoder architectures with shallow lexical matching and mismatching (Zhang and Wu 2018.)

In this paper, we attempt to implement several multiple traditional classification models and Neural Networks to predict the waiting time of getting an accepted answer. Followed by further exploring the content of questions, we attempt to propose the recommendation model to capture the semantic similarity which can help us to find the most relevant historical posts to one certain question posted by users.

3. Data Collection

3.1 Data Source

Stack Overflow provides an open Database API and allows all users to extract the data by customizing the SQL query based on their own requests. We explored the Entity-Relationship Diagram (ERD) and identified three key

[1] Chen Yiqiu (A0218914B), Gao Jin (A0218905B), Li Xinlin (A0150639A), Qu Mingyu (A0218907X), Tan Sijie (A0162550M)

[2] We take the answer accepted by the question owner (i.e., accepted answer) as solid answer in the project.

tables which are relevant for our analytic purpose as shown in Figure 1 below.

- *Question* table consists of information of question posts like titles with the corresponding accepted answer ID which is the key to link to *Answer* table.
- *User* table consists of the information of the question owner like *reputation* etc.
- *Answer* table consists of the details of the answer for the questions which ID is *ParentId*.

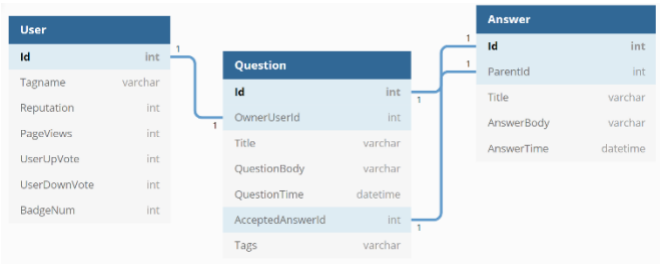

Figure 2. Stack Overflow databases

We have extracted a raw dataset from the Stack Exchange Data Explorer, a tool for executing arbitrary SQL queries against data from Stack Overflow Database, using the custom query. This raw dataset consists of total 190234 questions which are relevant to *Python* and *Data Science* (post tags contain "python" "numpy" "scikit-learn" etc.) with the accepted answers in the past 2 years covered from Jan 2019 to Jan 2021.

3.2 Variable Description

The table below summarizes all the variables extracted from *User* table, *Question* table, and *Answer* table. As shown in the table, we extracted 14 major variables in total.

Variable	Description	Data Type
Id	The unique identifier for each post	Int
Title	Title of the question	String
QuestionBody	Content of the question	String
AnswerBody	Content of the answer to the question	String
QuestionTime	The time when the post is created	Datetime
AnswerTime	The time when the accepted answer is created	Datetime
AnswerTimeDiff	Time needed to get the accepted answer	Int
Tags	Words or Phrases that describe the topic of the question	String
UserId	The unique identifier for each user	Int
UserReputation	A rough measurement of how much the community trusts the user	Int
UserPageViews	Number of times the profile is viewed	Int
UserUpvotes	How many upvotes the user has cast	Int
UserDownvotes	How many downvotes the user has cast	Int
BadgeNum	How many 'Question'-related badges the user achieves	Int

Note: *AnswerTimeDiff* is defined as the time difference between *QuestionTime* and *AnswerTime* and created by ourselves in the custom query.

Table 1. Variable description

3.3 Exploratory Data Analysis

Exploratory data analysis was conducted to gain some insights into our target variable and the feature *Tag*, which may affect how long the question would get a response.

How We Bin the Target Variable

We observe that there are around 7k+ questions related to *Python* and *Data Science* that could be answered each month, and many of the users can get their accepted answer within 30 mins. However, we also observe that there are still quite some questions needed to wait for quite a long time to get their accepted answer, which might negatively affect the user's experience on Stack Overflow. The potential reasons could be that the question is hardly understandable, or the tags the user chose are a relatively new topic so others could not address the questions immediately, etc.
The median waiting time of the data is 26 minutes. Take the user habit and experience factors into consideration, 30-minute was determined as the borderline of the class, so we decided to divide the time ranges into 2 categories.

Dive into Tag-related Features

In our scraped datasets, for each row there is a *Tag* column containing all the tags for that 1 question. We think tags generanlly represent question topics so it might be useful to predict the waiting time range. We use Regular Expression to separate the tags, count the

frequency of all tags in the whole dataset. There are 10596 tags in total. The figure shows the top 10 tag counts, which are popular Python-related topics in each of our daily life's study and coding.

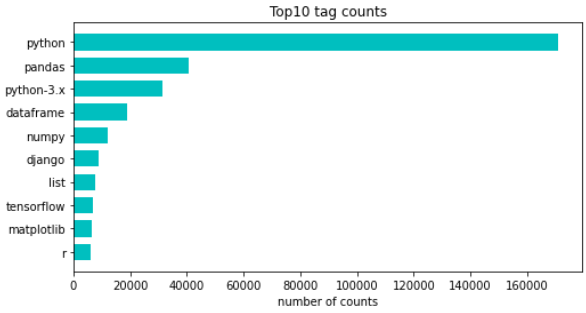

Figure 3. Top 10 tag counts

On the other hand, some tags just appear fewer than 10 times in our nearly 200K questions dataset: "django-oauth", "ibm-cloud-functions", "receiver" etc. These tags are not frequently discuessed by coders before, therefore not subscribed by lots of users, so these questions may need longer time to be answered.

Then we also want to know whether questions under certain tags are answered faster. The pie charts show 2 cases for illustration. "Pandas" questions are mostly answered within 30 mins (blue), however "TensorFlow" & "Amazon Web Services" (AWS not in the pie chart) questions are mostly above 30 mins (orange). It indicates that if a question has some tags which are related to difficult topic, even if the tag is a top 10 frequent tag like TensorFlow, then users might still need to wait for a long time to receive satisfied answers.

Figure 4. Pandas & Tensorflow time range

4. Data Processing & Feature Engineering

4.1 Text Pre-processing

Prior to the model implementation, text cleaning needs to be conducted and is divided into the following steps: (1) remove all HTML tags (e.g.,); (2) remove punctuations; (3) change remaining words into lowercases (4) remove stop words (Python NLTK package); (5) join words back to sentences. As the result, we cleaned original texts into 3 new columns "clean_title", "clean_question" and "clean_answer".

4.2 Feature Engineering

Feature engineering involves leveraging data mining techniques to extract new features from raw data along with the use of domain knowledge (online Q&A forum here). It is useful to improve the performance of machine learning algorithms and is often considered applied machine learning.

Self-defined Features

In this section, features that may influence the waiting time of getting an accepted answer are generated from raw data.

Variable	Description
Code-Related	
code_include	Whether contain code or not
question_code_len	Code Lenth
question_code_num	Number of code segments
Tag-Related	
tag_numbers	Number of tags
tag_score	Average Frequency of the tags used
tag_class0_numbers	Number of tags fallen in class 0
tag_class1_numbers	Number of tags fallen in class 1
tags	100 dummy columns for top 100 tags
Text-Related	
title_char_length	Title length
question_char_length	Question body length
title_word_counts	Number of words in the title
question_word_counts	Number of words in the question body
title_in_question_format	Whether the title is in question format
weekday	Dummy: whether weekdays

Table 2. Self-defined features

Here are explanations of some variables in the table:

- ***Code-Related:***
  code_include is a dummy variable. Some Stack Overflow questions have chunks of raw codes (e.g., <code></code>). It is not regarded as the length of

texts, since codes have fewer textual meanings than regular English sentences; but if a user posts some codes, it can facilitate others to better answer the questions. We also calculated the length of characters & the length of words of these codes.

- ***Tag-Related:***
  tag_score: in the previous EDA part, we have calculated the frequencies of all 10K tags. Since a user usually adds several tags to one question, we want to calculate a sufficient statistic to represent the overall frequency. Therefore, the average frequency is calculated as summing up all of them, then dividing it by the number of tags.
  tag_class0 or 1_numbers: If a question contains several tags which usually take >= 30mins, then this particular question is more likely to be >= 30mins. we count the number of tags that falling in a class-specific tag set, which is a set of tags that captures the topic characteristic of posts with the different waiting time. There are two class-specific tag sets: class 0 tag set (waiting time < 30 mins) and class 1 tag set (>= 30 mins). The tag set is the union of frequency tag set and unique tag set. The frequency tag set is a top-100-frequent-tag list for < 30 mins and >= 30 mins respectively. (5 common tags like "python" "numpy" are removed since their frequencies are too high) The unique tag set is a tag list that removes their intersection containing common tags.
- ***Text-Related:***
  title_word_counts etc.: The text length varies a lot among different questions. So we built related features to count the number of words.
  title_in_question_format: If a person posts his/her questions in a interrogative sentence, we think that it might catch more attention. We built this feature to indentify whether those titles start with "how" "what" "why".
  weekday: In our initial analysis we found that there are more questions posted on weekdays, so there might be different patterns on weekdays & weekends.

Word Embedding

In this section, we transform the user input questions ('clean_question') to numerical vectors by 3 methods. The words will be represented as an N-dimensional vector.

- TF-IDF

TF-IDF stands for Term Frequency – Inverse Document Frequency. It is one of the most important techniques used for information retrieval to represent how important a specific word or phrase is to a given document. The TF-IDF value increases in proportion to the number of times a word appears in the document but is often offset by the frequency of the word in the corpus, which helps to adjust concerning the fact that some words appear more frequently in general.

- Bag-of-words (BOW)

BOW can be done by assigning each word a unique number. Then any document we see can be encoded as a fixed-length vector with the length of the vocabulary of known words. The value in each position in the vector could be filled with a count or frequency of each word in the encoded document.

- Word2Vec

Word2Vec is one of the most popular representations of document vocabulary using a shallow neural network. It is capable of capturing the context of a word in a document, semantic and syntactic similarity, relation with other words.

5. Modelling

5.1 Waiting Time Prediction

Waiting time is an important factor affecting user experience on the website. In this section, the main objective is to apply classification models to predict whether the users could get the satisfied answers within 30 mins based on their question input with other features.

5.1.1 Representation Learning

In machine learning, representation learning is a set of techniques that allows a system to automatically discover the representations needed for feature detection or classification from raw data.

From the section feature engineering, all the features were selected can be summarized as 3 types of features as below (more details are shown in Table 3).

- Text Embedding

The result of the matrix that transformed from user question text input by using TF-IDF, BOW and Word2Vec.

- Profile Features

The initial features that were retrieved from the database that describe the user profile like reputation, number of badges earned, etc.

- Extracted Features

All the features that were created in the section of feature engineering section like the length of the user question input, number of tags and whether the question time is on weekdays, etc.

Type of Features	Variable
Text Embedding	'clean_question'
Profile Features	'UserReputation' 'UserPageViews' UserUpVote' 'UserDownVotes' 'BadgeNum'
Extracted Features	'code_include' Top 100 Tags Dummies Columns 'Tilte_in_question_format' 'Weekday' 'Tag_class0_numbers' 'Tag_class1_numbers' 'Title_word_counts' 'Question_word_counts' 'tag_numbers' 'Tag_Score' 'Question_code_num' 'Question_code_len'

Table 3. Variables input to models

Based on these 3 types of features, overall Representation Learning strategy process can be described in 5 steps:

1. In the beginning, we only use Profile Features which is from the raw data to train the model by using Logistic Regression and take it as our baseline model.

2. Base on this LR model, add in Extracted Features into the baseline model to see if the performance has been improved.

3. Introduced other advanced models like XGBoost/LightGBM and apply on both Profile Features and Extracted Features to compare the prediction performance.

4. Add in Text Embedding Features by using TF-IDF and BOW to apply on Bayes and LightGBM models, and check if the prediction performance has been improved.

5. Add in Text Embedding Features by using Word2Vec to apply on LightGBM and Neural Network models, and check if the prediction performance has been improved.

5.1.2 Model Development

As mentioned, since the object is to predict if the waiting time is beyond 30 mins, it is a binary classification. Various classic machine learning models were applied in this case. Following classifiers were trained and tuned with their respective parameters in this project:

• Naive Bayes

Naive Bayes model is easy to build and useful for large datasets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods. Hence it is mostly used in text classification.

• Logistic Regression

Logistic regression is a classic machine learning algorithm that utilizes a sigmoid function and works best on binary classification problems, in this project we use this model to build up as a baseline model.

• XGBoost

XGBoost is an optimized Gradient Boosting algorithm through parallel processing, tree-pruning, handling missing values and regularization to avoid overfitting/bias. It's suitable for categorical-intensive data with ideal generalization ability.

• LightGBM

LightGBM uses histogram-based algorithms, which bucket continuous feature (attribute) values into discrete bins. This speeds up training and reduces memory usage. It is capable of performing equally well with large datasets with a significant reduction in training time as compared to XGBoost.

• Neural Network

In the past few years, deep learning models have significantly expanded the capability of the natural language processing, and nearly all the model is based on the representation learning method. Neural networks are suitable models with nonlinear data with a large number of inputs, which makes them a compatible solution for natural language processing. It also could capture the

massive interactions between heterogeneous types of features.

5.1.3 Experiment

Multiple rounds of experiments for various combinations of models with different types of features summarized in Representation Learning were conducted. In this case, we take ***f1 score*** as our main metric to measure the prediction performance for all the models.

The prediction performance on the test dataset result (f1 score) is shown in the table below. From the results, we can find that the LightGBM with Text Embedding methods achieved the best performance (0.71) among all of the 6 models which is quite ideal in real business.

Model	Naive Bayes	Logistic Regression	Random Forest	XGBoost	LightGBM	Neural Network
Profile Only	0.38	0.36	0.56	0.49	0.51	-
Profile + Extracted	0.61	0.69	0.69	0.70	0.70	-
TF - IDF	0.60	0.62	0.69	0.69	**0.71**	-
BOW	0.60	0.62	0.70	0.69	**0.71**	-
Word2Vec	0.64	0.69	0.69	0.70	0.70	0.66

Table 4. Model comparisons on F1 score

5.1.4 Machine Learning Interpretability

Feature Importance

Feature importance generated by the model (Figure 5)indicated the count-based importance (numbers of splits are counted), in which *Quenstion_Char_Length* ranks first and then followed by *Tag_Score* and *UserReputation*.

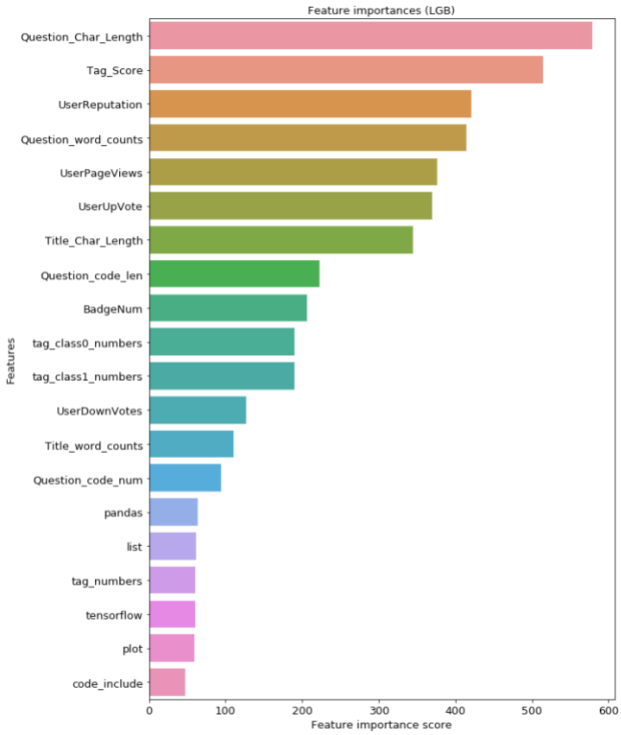

Figure 5. Feature importance

Permutation feature importance measures the importance of a feature by calculating the increase in the model's prediction error after permuting the feature. A feature is important if shuffling its values increases the model error, because in this case the model relied on the feature for the prediction.

Weight	Feature
0.0968 ± 0.0031	tag_class1_numbers
0.0456 ± 0.0039	Question_Char_Length
0.0268 ± 0.0018	tag_class0_numbers
0.0055 ± 0.0010	Tag_Score
0.0051 ± 0.0011	pandas
0.0039 ± 0.0009	list
0.0039 ± 0.0005	tensorflow
0.0035 ± 0.0005	code_include
0.0026 ± 0.0014	Question_word_counts
0.0025 ± 0.0007	plot
0.0019 ± 0.0007	dataframe
0.0019 ± 0.0005	string
0.0017 ± 0.0006	opencv
0.0015 ± 0.0005	oop
0.0015 ± 0.0012	UserReputation
0.0013 ± 0.0002	pyspark
0.0012 ± 0.0008	UserPageViews
0.0012 ± 0.0006	dictionary
0.0011 ± 0.0006	pyqt
0.0011 ± 0.0004	scikit-learn
… 98 more …	

Figure 6. Permutation feature importance

The most important feature based on permutation is *tag_class1_numbers*. It is reasonable as according to Figure 7 below, the respective tag sets for two waiting time classes are quite different.

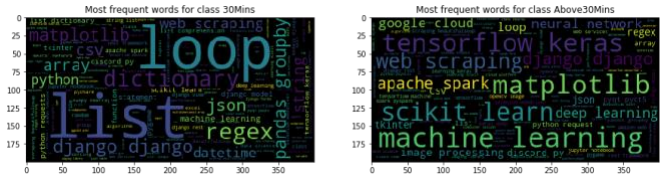

Figure 7. Word cloud of frequent tags for 2 classes

Partial Dependence Plot

We use partial dependence plots (PDP) to show the marginal effect of the important features have on the predicted outcome of LightGBM. The marginal effect of "number of tags in class1" on the probability of getting satisfied answers exceeding 30mins shows a generally increasing trend as the number of class1 tags increases. That is, if the question has more tags in class 1, it is more likely to get a satisfactory answer for more than 30min. Other PDP are listed in Appendix.

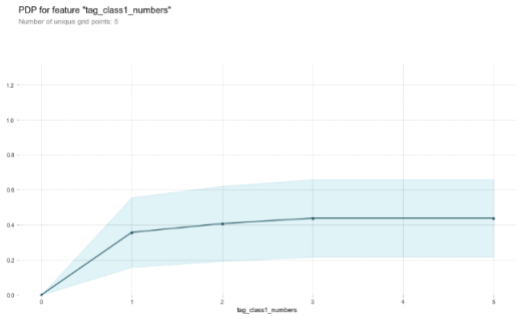

Figure 8. Partial dependence plot for tag_class1_numbers

LIME

In addition to global importance, we use LIME to interpret the local importance of the features and show how they influence the expected answer time.

Case 1: 0 percentile of exceeding 30min

In this case, the probability of getting an accepted answer within 30 minutes is 0.99. Having more easy-to-be-answered tags and less complex tags, like apache-spark and scipy, will decrease the expected time of getting a satisfied answer.

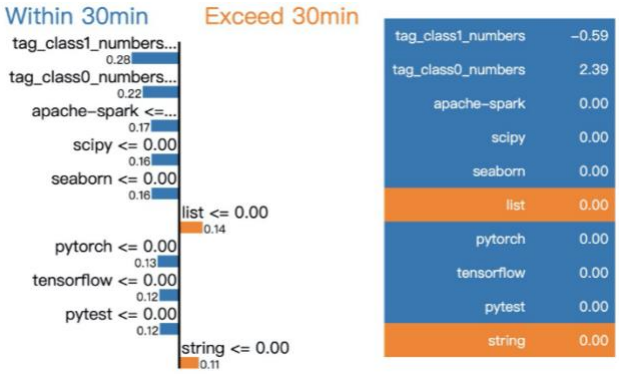

Figure 9. LIME - 0 percentile

Case 2: 99 percentiles of exceeding 30min

In this case, the probability of getting an accepted answer within 30 minutes is 0.04. It's caused by a too lengthy question and related to difficult topics including PyTorch and TensorFlow.

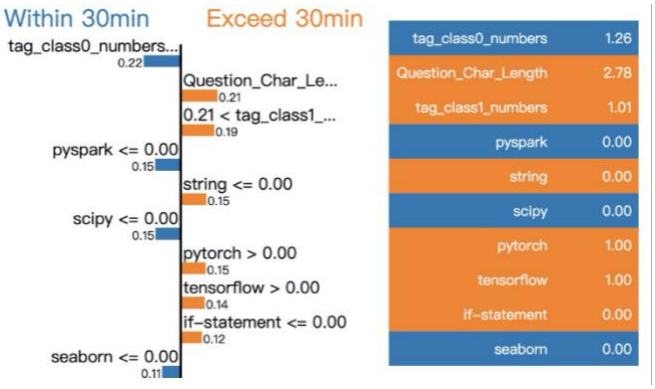

Figure 10. LIME - 99 percentile

5.2 Recommendation System Building

If the previous prediction model shows that a user will wait a quite long time which is beyond 30 mins for the accepted answer, we could take proactive actions to recommend historical similar questions with the answers to users for reference.

The core idea in task here is to detect semantic similarity in questions posted in the forum. If a similar question is posted by a user, the system can identify the related question and then promote these solved questions with an accepted answer to the user.

5.2.1 Methodology

Figure 11. Steps for Recommendationn System

The main idea of the recommendation system is to use title and tag to filter based on cosine similarity at first, then recommend the questions with the highest cosine similarity with the input question.

(1) Text Cleansing

Given an input question and history question, similar to the cleansing in the previous part of the report, we firstly clean the title and question body by removing stopwords, HTML part, digits, punctuation, transforming all words to lowercase and finally lemmatization. For tags of input questions and history questions, we extract the words from the tags.

(2) Filter based on SimTitle and SimTag

(2.1) Calculation of similarity score between title:

In this step, we fit a TF-IDF vectorizer using cleaned title data from history questions. After we have the fitted vectorizer, we transform the title of history questions and the input title into a dense matrix. Finally, we calculate the cosine similarity between the input title and each title from the raw dataset to get a similarity score for the title as SimTitle.

(2.2) Calculation of similarity score between tag:

Similarly, we have a TF-IDF vectorizer fitted on the tags for each question in the raw dataset. Then we apply the transform function on the input and all the tags in the raw dataset. After we get the dense TF-IDF matrix, we then get the cosine similarity score between each row of tags in the dataset and the input as SimTag.

(2.3) Filter by combinations of title and tags

Based on the similarity score of both title and tags, we then calculate a similarity score with the formula below:

$$SimScore(Ip, Hp) = \alpha * SimTitle(Ip, Hp) + (1 - \alpha) * SimTag(Ip, Hp)$$

Ip states input Question, *Hp* states History question in the dataset.

In this project, we set α to be 0.9 which means that we put more emphasis on the title similarity and the filter threshold score to be 0.2. If SimScore is more than 0.2, we then add the ID of the questions to the candidate for final recommendation.

(3) Recommendation based on Question Body Similarity Score

After we get filtered data, we then proceed to the final step of the recommendation system. Like the first 2 steps, we have a TF-IDF vectorizer fitted on the cleaned Question Body of history questions in the raw data set and transform the cleaned input question body and clean question body in the dataset into a dense matrix. Then we calculate the cosine similarity scores between the input question body and each clean question body of the history question. Finally, we rank the history questions by cosine similarity and output top 5 questions with the highest cosine similarity with the input question.

5.2.2 Results

After we implement the recommendation system, we tried a sample question from the raw dataset and checked what our recommendation system will output. The input question can be found at https://stackoverflow.com/questions/53992768, and the screenshot of the input question is in the appendix. We decide to recommend questions, and their website and cosine similarity is shown below:

Output Question	Cosine Similarity
https://stackoverflow.com/questions/59559519/how-to-drop-duplicates-in-dataframe-ignoring-punctuations	0.555
https://stackoverflow.com/questions/54066612/boolean-subset-with-named-index	0.546
https://stackoverflow.com/questions/59641251/drop-rows-if-any-of-multiple-columns-have-duplicates-rows-in-pandas	0.543
https://stackoverflow.com/questions/61289172/pandas-drop-subset-of-dataframe	0.542
https://stackoverflow.com/questions/58134427/can-i-use-pd-drop-in-method-chaining-to-drop-specific-rows	0.542

Table 5. Recommendation of 5 similar questions

We can see that the input question asks about 'the difference between the function duplicated() and the function drop_duplicates()'. The top three recommended questions are related to drop_duplicated and the last 2 questions are related to droping specific rows. While the recommended questions do not have the exactly same meaning as the input question, their contents are similar to

the input question and their answers could be helpful to solve the input questions.

6. Business Insights

- ***Encourage users to answer questions related to long-waiting tags related to deep learning:*** We noticed that tag-related features play quite important roles in predicting long-waiting questions, and most long-waiting time tags are related to deep learning. To encourage users to answer questions under tags like Spark and TensorFlow by giving them badges and rewards would potentially decrease the average waiting time.
- ***Long question body and the existence of code chunks would imply higher waiting time even though questions have popular tags like pandas:*** As shown the case in the LIME analysis, we noticed that when the question body is super long, even though questions are under short-waiting time tags like pandas, the effect of length of question body is strong enough to classify the questions into long-waiting class.
- ***The average frequency of tags matters rather than the number of tags:*** We noticed that the average frequency of tags (which is *Tag_Score*) shows a higher feature importance rank in predicting than the number of tags a question related does. It means that business people at StackOverflow should manage and monitor tags in an individual way if they want to improve user experience in terms of receiving answers.
- ***Questions that post by users with high reputations should be focused on:*** We noticed that the user reputation of who posts questions ranked high in feature importance. The higher the reputation, the longer time the answer would be accepted. It is possibly due to that highly reputed users tend to post hard or deep questions in a certain field. We should focus more on these cases, not only because of their long waiting time but also due to the potentially high value of these questions.
- ***A Three-filtering-step recommendation system is recommended:*** Based on insights and focuses mentioned above, we recommend business people try the three-filtering-steps recommendation system we built. The filtering hierarchically (Tag - Title - Question body) is tailored to the characteristics of Q&A platform Stack Overflow.

7. Limitation & Future Improvement

- ***Further feature engineering related to tag subscribers required:*** Based on results from previous related research papers, we found features that capture user activation tend to have higher prediction power. To improve our model performance, we may extract data related to active subscribers in each tag and create certain metrics as input features.
- ***Provide a better evaluation metric for the recommendation system:*** We did not implement metrics to evaluate the performance of our recommendation system because we do not have the computation power to collect enough recommendation samples. It can be done by inputting a sample of questions and getting recommendation questions for each sample question, and then identifying whether recommended questions are related to the input questions by humans and finally calculating recall and precision to evaluate performance.
- ***Try multiple distance measures in the calculation of similarity for the recommendation system:*** Different distance measures such as Euclidean distance, Jaccard distance, or self-defined distance could be tried to quantify the similarity.

8. Conclusion

In this study, we explored potential features that may affect the waiting time of accepted answers for one post and built the waiting time prediction model. We showed that the waiting time prediction can be handled as a binary classification task. Based on actual data extracted from the Stack Overflow database, we found tag-related and text-related features play important roles in predictive power in the optimal model LightGBM with word vectorization. We also conducted machine learning interpretability analysis to have a deep dive into how features affect prediction outcomes in both the global and local settings.

Based on the outcome of prediction models, we then proposed the recommendation model, which basic idea is built on cosine similarity of input text, to recommend similar posts to those questions that are predicted to have longer than 30mins waiting time. In the whole project we built an integrated pipeline from data extraction to time prediction, then to post recommendations, which is of real business value for managing user posting experience on Q&A platform like Stack Overflow.

References

"What is reputation? How do I earn (and lose) it?". Stack Overflow. Archived from the original on 9 June 2013. Retrieved 14 August 2010.

Daniel Hasan Dalip, Marcos André Gonçalves, Marco Cristo, and Pavel Calado. 2013. Exploiting user feedback to learn to rank answers in q&a forums: a case study with stack overflow. In Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval (SIGIR '13). Association for Computing Machinery, New York, NY, USA, 543–552. DOI:https://doi.org/10.1145/2484028.2484072

Xiaobing Xue, Jiwoon Jeon, and W. Bruce Croft. 2008. Retrieval models for question and answer archives. In Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2008), pages 475–482.

Salvatore Romeo, Giovanni Da San Martino, Alberto Barron-Cede ´ no, Alessandro Moschitti, Yonatan Be- ˜ linkov, Wei-Ning Hsu, Yu Zhang, Mitra Mohtarami, and James Glass. 2016. Neural attention for learning to rank questions in community question answering. In Proceedings of the 26th International Conference on Computational Linguistics (COLING 2016), pages 1734–1745.

Minghua Zhang and Yunfang Wu. 2018. An unsupervised model with attention autoencoders for question retrieval. In Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI 2018), pages 4978–4986.

Jie Wan, FengYu Zhao, Ya Liu. 2019. Feature Integration Answer Recommendation Strategy For Stack Overflow System.

Appendix

Cont'd for 5.1.4 Machine Leaning Interpretability:

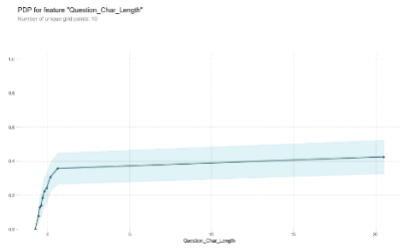

Figure 12. PDP - "Question_Char_Length"

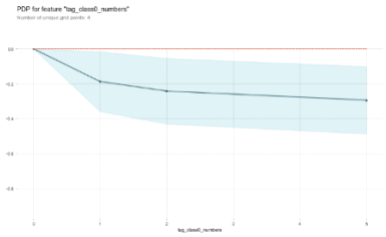

Figure 13. PDP - "Tag_Class0_Numbers"

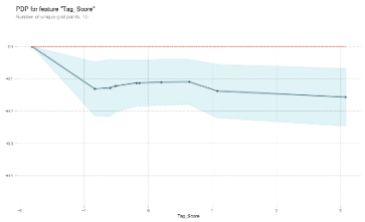

Figure 14. PDP - "Tag_Score"

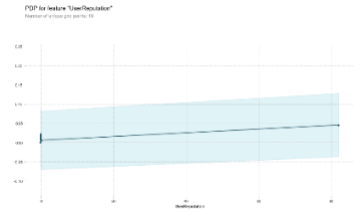

Figure 15. PDP - "UserReputation"

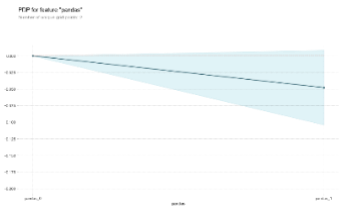

Figure 16. PDP - "Pandas"

Cont'd for 5.2 Recommendation System Building - Case for Evaluation

Input Question:

Figure 17. Example – Question Asked

Recommended Questions:

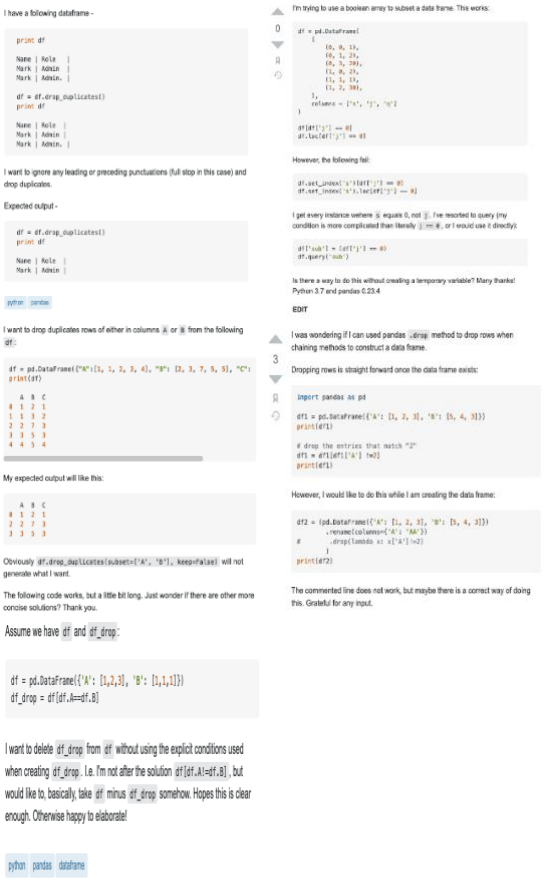

Figure 18. Example - The 5 similar questions