Predicting Good Stack Overflow Answers

Kai Ye, Hu Zongqi, Li Haoyi, Dong Yujie

Data and code available at https://github.com/nbzhyekai/group10

Abstract

Stack Overflow is an online question and answer community for both professional and enthusiast programmers, and question owners are able to choose one answer as the accepted answer so it brings convenience for other visitors who search for the same question later on. However, one of the pain points is that question owners often do not accept any answer. In this project, we collected one-year data of questions and respective answers in the 'Python' topic from Stack Overflow. We analyzed how different features about questions, answers and users can determine whether an answer is accepted. Based on these features, we built a prediction model to automatically select the best answer among all answers submitted for the question, without having the question owner to manually choose. This will also further improve the online user experience for all visitors by auto-identifying good answers for every question they search for.

1. Background

Specialized programmer forums, Stack Overflow (SO) for example, have been actively operating in recent years. Such platforms provide developers with a community to share programming tips, discuss technical challenges and also allow programming newbies to find solutions to their problems. Besides SO, there are over 20 international popular programming forums including Reddit, Codecall and JQuery (VironIT, 2019). These platforms compete against each other in absorbing more excellent programmer users and developing to a more monopolistic online community to occupy as much market share. In order to attract more users, these platforms have sought to improve user experience by enhancing the content quality and search efficiency.

To cope with such a problem, SO has incorporated several features to help visitors identify good answers, including allowing readers to upvote or downvote answer posts. The most reliable indicator for a good answer is that the answer post is accepted by the question post owner. However, it is often the case that question askers have not chosen any answer to accept. This leads to our motivation to build an auto tagging model which tags an answer post as a good answer post as long as the post meets certain criteria.

Several works have been done in the relevant field. Gantayat et. al. (2015) found a positive relationship between voting and answer acceptance. Calefato et. al. (2015) predicted successful SO answers using affective states, such as answer and comment sentiment, and answer presentation including length, upper case ratio and URL count. Omondiagbe, Licorish and MacDonell (2019) predicted the acceptability of Java and JavaScript answers based on time lag, URL count, comment count, reputation, text polarity etc. These studies mainly focus on the linguistic aspect and structure of answer posts and few attention has been paid to the technical aspect of the answer.

The aim of our project is to analyze the features of accepted SO answers to Python questions and implement a prediction model to automatically identify successful answer posts.

2. Dataset

2.1 Data Collection

We collect all posts generated by users with the tag "Python" on SO, where post creation date is from 01/01/2019 to 31/12/2019. In terms of the collection process, we directly query data of our interest and retrieve them from Stack Exchange Data Explorer using SQL (Stack Exchange Data Explorer, 2020). We use two tables of the SO database, namely Posts and Users and their schema are shown below in Figure 1.

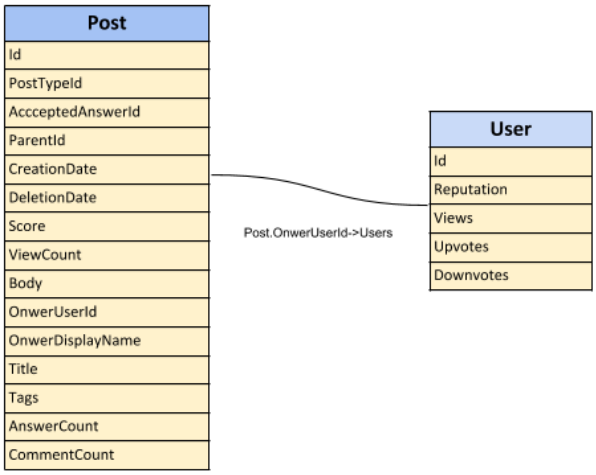


Figure 1. SO database 'Post' and 'User' data schema

The Posts dataset contains features information of both the question posted and answers related to each question. The "AcceptedAnswerId " is the point to the answer post Id accepted by the question post owner. Besides that, the Users dataset has features information about registered users who are in the SO community. Two tables can be joined on Post.OwnerUserId and User.Id.

2.2 Data Description

The Posts table stores all the question and answer posts, and the Users table stores the profile of the user. The variables of Posts and Users datasets are shown below in Table 1 and Table 2.

Table 1. Description of Posts Table

Variable Name	Variable Type	Description
Id	integer	The unique identifier for each post (Applicable for both question and answer post)
PostTypeId	integer	The identifier to determine whether a post is an answer or question. 1 refers to question posts and 2 refers to answer posts
Acccepted AnswerId	integer	The unique identifier for the answer post, selected as "accepted" by the question owner corresponding to the question post (Only applicable for questions)
ParentId	integer	The unique identifier for the question, which the answer is responded to (Only applicable for answers)
CreationDate	datetime	Date when a post is created (Applicable for both question and answers)
DeletionDate	datetime	Date when a post is deleted (Applicable for both question and answers)
Score	integer	Difference between upvotes and downvotes received by a post (Applicable for both question and answers)
ViewCount	integer	Number of views for a post (Applicable for both question and answers)
Body	string	Content of a post (Applicable for both question and answers)
OnwerUser Id	integer	The unique identifier for a user (Applicable for both question and answers)
OnwerDisp layName	string	The name of a user (Applicable for both question and answers)
Title	string	Title of the question (Only applicable for questions)
Tags	string	Tags labelled to a post (Only applicable for questions)
AnswerCou nt	integer	Number of answers responded to the specific question (Only applicable for questions)
CommentC ount	integer	Number of comments posted for a post (Applicable for both question and answers)

Table 2. Description of Users Table

Variable Name	Variabl e Type	Description
Id	integer	The unique identifier for a user
Reputation	integer	The reputation of the user
Views	integer	Number of total views for all posts of a user
Upvotes	integer	Number of total upvotes received for all posts of a user

Based on the Posts datasets, we have extracted question posts and answer posts separately. Upon extraction, the question posts datasets contain 287,905 rows of records, while answer posts datasets contain 305,910 rows of

records. In terms of Users datasets, it contains another 199,360 rows.

2.3 Basic Data Cleaning

We have then joined 3 datasets and renamed some columns. We have also excluded question posts without an "accepted answer" tag to ensure all data are labelled and removed rows where responses are in anonymity. After basic data cleaning, we have in total 180,186 rows and 14 columns. For train/test split, we will use question posts with creation date during the period of 01/01/2019 to 31/10/2019 as training data and with creation date during the period of 01/11/2019 to 31/12/2019 as test data.

For textual data, we have also cleaned them up by removing new lines, removing tags, removing inline codes and block codes from the answer text. For inline codes and block codes, respective new features will be created during the feature engineering.

3. Data Exploration & Feature Engineering

There are 6 aspects of features that we have worked on: language, technical, non-textual, user, bag-of-words representation, and FastText embedding. We have also performed exploratory analysis for both textual and non-textual features.

3.1 Language

For the language aspect, we have extracted the length of the answer by counting the number of characters in the answer body. We have also extracted the readability with the help of textstat package (The Python Package Index, 2020). Since we are analysing answer posts for python related questions, we would expect good answers to be rather technical and thus have lower readability. We initially planned to apply the BERT model to vectorize texts and represent the contextual relations of words between question and answer body (Horev, 2018). But considering BERT is super time consuming, later we adopted Bag of words(BoW) and fast text instead to vectorize text and conduct word embedding.

We then explored the difference of average length of answer and answer text readability between accepted answer and non-accepted answers in Figure 2. It showed that accepted answers have longer answer length compared to non-accepted ones in our dataset. In terms of answer readability, interestingly, non-accepted answers tend to have slightly higher readability scores than accepted ones.

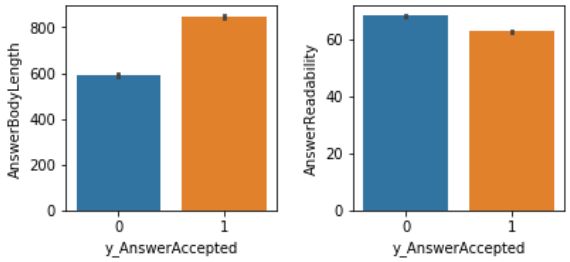


Figure 2. Bar chart of average answer body length and answer readability between accepted and non-accepted answers

3.2 Technical

For technical aspects, we have extracted the number of inline code, code blocks, lines of block code, as well as reference hyperlinks an answer contains. We would expect that the more codes or hyperlink references used, the more likely the answer is to be a good answer, as this indicates the responder input more technical effort in the answer.

Additionally, we have performed similar exploratory analysis for each of the technical features in Figure 3. We found out that in general as compared to non-accepted answers, accepted answers tend to have more code blocks, inline codes and hyperlinks provided.

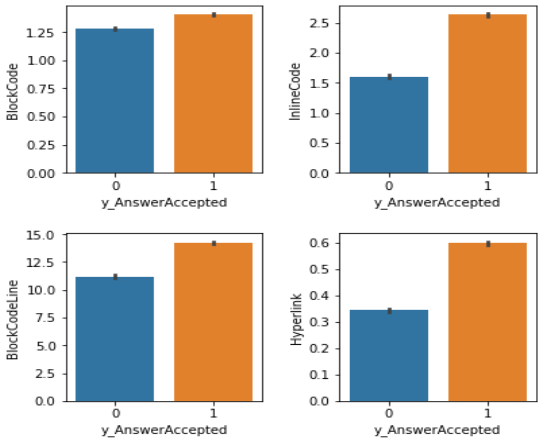


Figure 3. Bar charts of averages of respective technical features between accepted and non-accepted answers

3.3 Non-textual

For non-textual aspects, we have extracted the time lag between the question post time and the answer post time and created the response time (in second) feature. Together with other original non-textual features like AnswerCount, AnswerScore and AnswerCommentCount, we explored the relationship between respective features against whether an answer is accepted or not in Figure 4.

From the exploratory analysis, we found out that accepted answers on average tend to have higher answer score ratings, more comments regarding the answer post, less response time to respond to a question post and lower answer counts with regards to the question post.

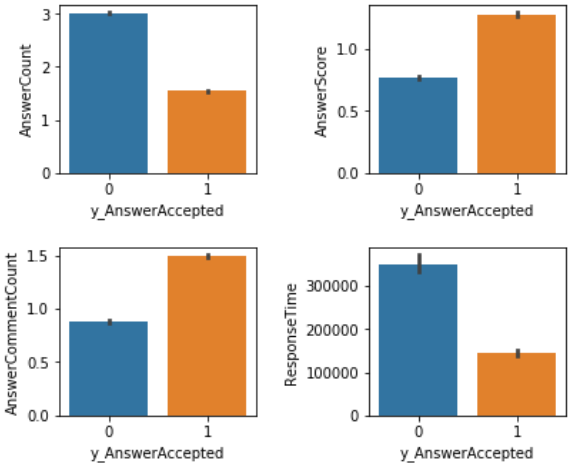


Figure 4. Bar charts of averages of respective non-textual features between accepted and non-accepted answers

3.4 User

For the user aspect, we have extracted a few attributes from the responder's user profile that can indicate whether the user had been a good responder. Generally, if the user with the higher reputation, more answer view, more upvotes, and less downvotes, would be expected to be a better response.

From the exploratory analysis of user features in Figure 5, it showed the same patterns of which accepted answers have significant higher user reputations, user views, upvotes and downvotes as compared to the non-accepted answers.

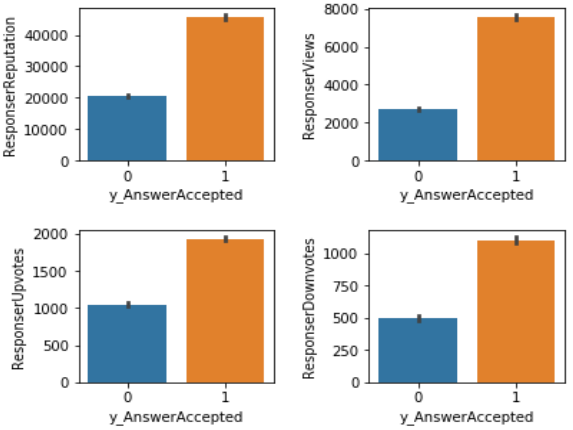


Figure 5. Bar charts of averages of respective user features between accepted and non-accepted answers

3.5 Textual Data Exploratory Analysis

Before textual data exploration on answer body text, we have removed the stopwords from the text data and performed stemming ro reduce inflected word to its base or root form. We then plotted the word cloud for both accepted answer and non-accepted answer after removing top 100 frequent words in each group, as some of top frequent words like 'use', 'python' and 'one' carry little meaning in this context and the results are shown as Figure 6 and Figure 7.

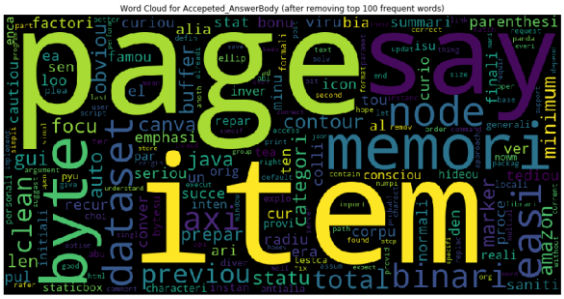


Figure 6. Word Cloud of accepted answer body text

Figure 7. Word Cloud of non-accepted answer body text

3.6 Bag-of-words Representation

For the Bag-of-words (BoW) Representation aspect, we have transformed the text data of each answer into a 'bag of words'. This vectorizes sentences or paragraphs of the answer body by representing the contextual text with words with measurable features. We have extracted the term frequency of all distinct words and visualized the word cloud image of terms in accepted answers to get a quick glimpse of the trend. Answers with higher or lower frequency of some words would be expected to be more likely accepted.

3.7 FastText Embedding

For the FastText Embedding aspect, we have extracted new vectorized features of both answer and question body by this word embedding algorithm. The text data and

contextual relations of answer and question bodys have been mapped into numerical vectors to achieve a higher text classification accuracy. The shorter distance between an answer's embedding vectors and the defined accepted one's, the more likely it would be accepted.

4. Model

4.1 Evaluation metrics

In this project, precision is chosen to be the evaluation metrics for model results comparison instead of accuracy. This is because we would like our model to have fewer false positives, which could give users wrong information. Thus, precision, which is the ratio of true positive (case of accepted answer) out of all positive predictions made, is the better choice of evaluation metrics.

4.2 Model selection

In term of model selection, we have considered below choices for this binary classification case

- Logistics Regression
- Decision Tree Classifier
- Random Forest
- XGBoost

Other conventional models like KNN and SVM are not chosen due to long training time, which hinders the feasibility of model tuning.

4.3 Hyperparameter Tuning

We have also performed hyperparameter tuning on each of the model choices to optimize the precision score of the respective model. Randomized search is the method used to conduct hyperparameter tuning as it is more computationally efficient than gird search. The lists of parameters for each model are as below in Table 3.

Table 3. List of parameter for tuning in each model

Model	List of Parameters
Logistics Regression	c: LogUniform (1e-2, 10) penalty: (L1, L2)
Decision Tree	criterion: (gini, entropy) splitter: (best, random) max_depth: 2~10 min_samples_split: 2~10
Random Forest	max_features: (auto, sqrt) max_depth: 5~60 with increment min_samples_split: (2, 5, 8, 10) min_samples_leaf: (1, 2, 4, 6, 8)
XGBoost	learning_rate: (0.01, 0.03, 0.1) max_depth: (4, 5, 6) gamma: (0, 0.01, 0.03, 1)

4.4 Baseline Model

For each model, we have trained it to obtain optimized hyperparameter settings with the training dataset, and compute the precision score on the test dataset for results comparison. In addition, we further run our model with different combinations of word embedding and feature selections. Both bag of words and FastText embedding method are tried out in all model choices respectively. In terms of feature selection (FS), we have utilized the random forest method to select features only above a certain significant feature importance threshold. Below Table 4.1 and 4.2 are the precision score results on the test dataset for various model choices and different combinations.

Table 4.1. Baseline model results comparison - part 1

Model	Without Embedding	BoW	FastText
Logistics Regression	0.790	0.658	0.783
Decision Tree	0.951	0.967	0.967
Random Forest	0.888	0.840	0.840
XGBoost	0.900	0.900	0.900

Table 4.2. Baseline model results comparison - part 2

Model	BoW with FS	FastText With FS
Logistics Regression	0.658	0.658
Decision Tree	0.984	0.988
Random Forest	0.891	0.880
XGBoost	0.900	0.900

4.5 Preliminary Results

Based on the baseline model results comparison, Decision Tree Classifier has performed the best on the test dataset and with the addition of Bag of Words or FastText Embedding techniques, the result improved from the

baseline model. Besides that, by conducting feature selection, it further improved the results on Decision Tree Classifier as the best-performing model.

The feature importance of the best performing model has been plotted as shown in Figure 8.

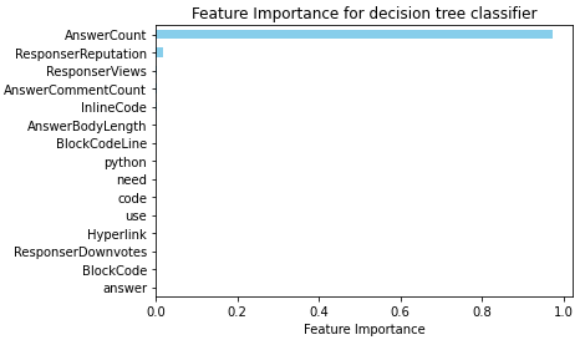


Figure 8. Global feature importance of best-performing baseline model

We found out that the feature 'AnswerCount' has the most significant feature importance than any other features. This is due to imbalance distribution between accepted and non-accepted answers in the situation when answer count equals 1, which is further validated in the frequency distribution bar chart against AnswerCount for both accepted and non-accepted answers as Figure 9.

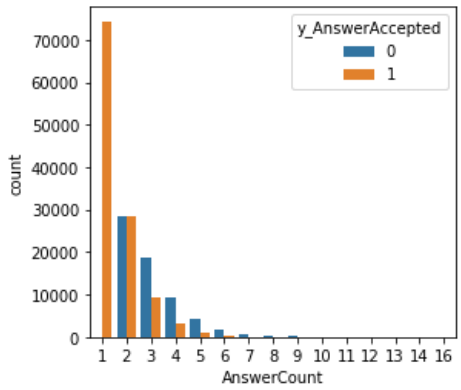


Figure 9. Frequency distribution bar chart against AnswerCount

This is also due to the limitation of our dataset, in which the question owner can only tag accepted labels to a specific answer. Since our dataset is all labeled with whether accepted or non-accepted, all answers will be labeled as accepted in the situation when only 1 answer is submitted to a question. To overcome this limitation and make our prediction model more feasible, we decided on two options where the first one is to remove data where AnswerCount = 1, and the other one is to total remove AnswerCount as a feature in the model.

4.6 Improved Model & Results

For option one, after removing AnswerCount = 1, we run the same set of models with the same parameter tuning settings. Additionally, we have tried different limits on the number of words used in the model based on the frequency of each word. Different limits are removing top 100 words and retaining top 600 words, removing top 100 words and retaining top 1000 words, and removing top 100 words and retaining top 3000 words. This is because some of the top frequent words like 'use' or 'python' or 'one' are less meaningful in this context of identifying good answers. The results of some combination of settings are as shown in Table 5 and 6.

Table 5. Results comparison after removing AnswerCount = 1 with no words limit

Model	BoW	FastText	BoW with FS	FastText With FS
Logistics Regression	0.540	0.681	0.540	0.547
Decision Tree	0.685	0.682	0.810	0.644
Random Forest	0.763	0.741	0.685	0.691
XGBoost	0.654	0.652	0.650	0.652

Table 6. Results comparison after removing AnswerCount = 1with words limit between top 100 and top 1000

Model	BoW	FastText	BoW with FS	FastText With FS
Logistics Regression	0.540	0.681	0.540	0.547
Decision Tree	0.670	0.682	0.634	0.644
Random Forest	0.816	0.741	0.700	0.691
XGBoost	0.652	0.652	0.655	0.652

After comparing all the results among all combinations of model, feature selection and words limits, Random Forest model with Bag of Words representation and without feature selection in the setting of words limit between top 100 and top 1000 has the best precision of 0.816. The confusion matrix is shown below in Table 7. Although the model has a high precision, its recall is very low (0.038). This means that our model is unlikely to misclassify a bad answer as good, but it can potentially miss many good answers.

Table 7. Confusion matrix of the best-performing model after removing AnswerCount = 1 with words limit between top 100 and top 1000

	Actual 1	Actual 0
Predict 1	395	89
Predict 0	9949	15018

The feature importance of the best performing model has been plotted as shown in Figure 10.

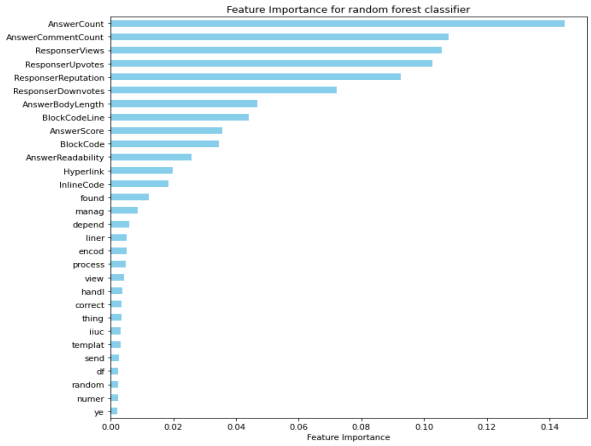


Figure 10. Global feature importance of best-performing model after removing AnswerCount = 1 with words limit between top 100 and top 1000

As shown in the figure, AnswerCount still has the highest feature importance. Since in real situations, we cannot guarantee that we have waited a long enough time for all answers to be posted, we tried option two to completely remove the feature 'AnswerCount' and run the same set of models with the same parameter tuning settings. The results of some combination of settings are as shown in Table 8 and Table 9.

Table 8. Results comparison after removing AnswerCount with words limit top 1000

Model	BoW	FastText	BoW with FS	FastText With FS
Logistics Regression	0.659	0.701	0.659	0.659
Decision Tree	0.732	0.719	0.726	0.717
Random Forest	0.728	0.714	0.730	0.720
XGBoost	0.725	0.725	0.729	0.724

Table 9. Results comparison after removing AnswerCount with words limit between top 100 and top 1000

Model	BoW	FastText	BoW with FS	FastText With FS
Logistics Regression	0.659	0.701	0.659	0.659
Decision Tree	0.723	0.720	0.723	0.720
Random Forest	0.729	0.714	0.731	0.717
XGBoost	0.725	0.725	0.729	0.724

After comparing all the results among all combinations of model, feature selection and words limits, Random Forest model with Bag of Words representation and without feature selection in the setting of words limit between top 100 and top 1000 has the best precision of 0.731. The confusion matrix is shown below in Table 10. The model has a recall of 0.863 which is better than the previous model. Thus, we choose the Random Forest model after removing AnswerCount with a word limit between top 100 and top 1000 as the best prediction model in our project.

Table 10. Confusion matrix of the best-performing model after removing AnswerCount with words limit between top 100 and top 1000

	Actual 1	Actual 0
Predict 1	25178	9275
Predict 0	3992	5832

The feature importance of the best performing model has been plotted as shown in Figure 11.

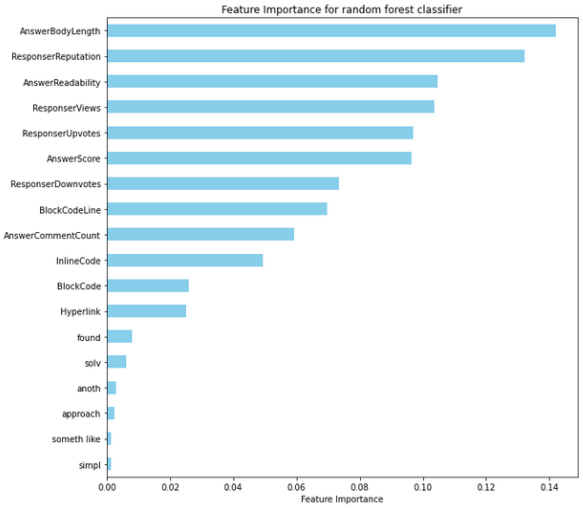


Figure 11. Global feature importance of best-performing model after removing AnswerCount with words limit between top 100 and top 1000

4.7 Model Interpretation and Evaluation

Apart from the global importance, Lime was used to examine the local importance and understand the rationale behind the model's prediction on each observation.

Case 1: True positive

For this case, our model correctly identifies a good answer. The reason behind its prediction as shown in Figure 12 is because the length of the answer is large, the answer gets a positive answer score and it receives some comments from others. Although the answer does not have hyperlinks or many block codes, it is still a good answer.

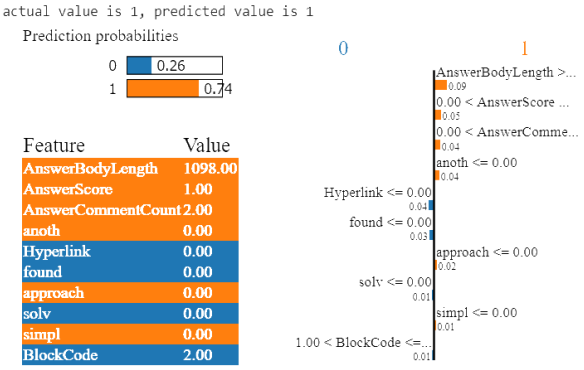


Figure 12. Local feature importance of true positive case

Case 2: True negative

For this case, our model correctly identifies a bad answer. it predicts 73% of chance that the answer is a bad answer. The rationale behind the prediction as shown in Figure 13 is because the answer gets an answer score of 0, it receives no comments and it does not include any hyperlinks or inline code.

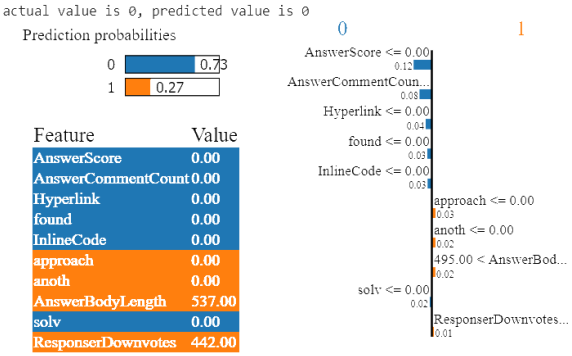


Figure 13. Local feature importance of true negative case

Case 3: False positive 1

It is more useful to look at those cases, where our model makes mistakes in classifying a bad answer as good. For this case, our model is very confident that the answer is a good answer as shown in Figure 14, however this answer is not accepted by the question owner.

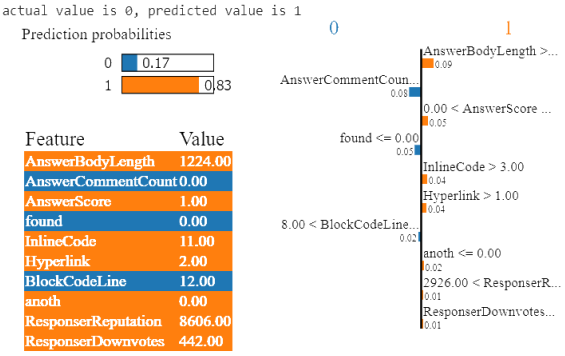


Figure 14. Local feature importance of false positive case 1

To understand the reason for the model prediction, we find the original post on Stack Overflow[1]. The question is on why do two functions give different results (Appendix 1).

The posts on the below are the accepted answer and the test answer shown in Figure 15 and 16 respectively. It can be seen that the test answer also addresses the question – it also includes some codes and provides hyperlinks and is clearly written. The reason for the test answer not being accepted could be that it is not posted as fast as the user accepted answer. The user accepted answer was posted at 18:09 on Oct 8, 2019, and the test answer was posted 15

[1] Original question post on Stack Overflow for false positive case 1. https://stackoverflow.com/questions/58291713/why-do-dict1-items-dict2-items-and-dict1-viewitems-dict2-viewitems

minutes later. However, the test answer is still a good answer in nature.

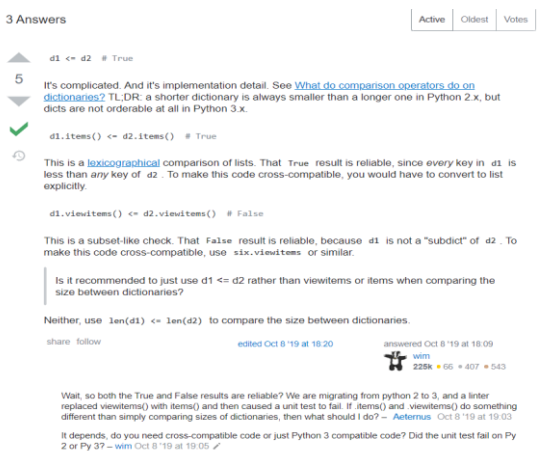


Figure 15. Actual accepted answer for false positive case 1

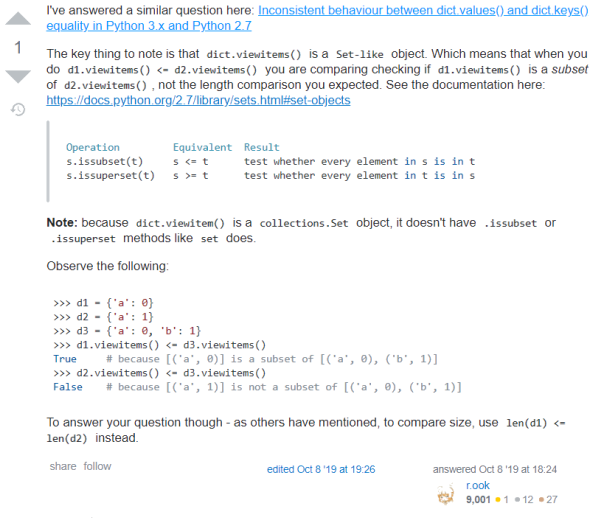


Figure 16. Predicted accepted answer for false positive case 1

Case 4: False positive 2

One more case where the model misclassifies a bad answer as good is inspected. Looking at the Lime interpretation, the model predicts a 0.88 chance that the answer post is a good answer because the answer is elaborate, it receives a positive answer score and it is posted by a user with a good reputation as shown in Figure 17.

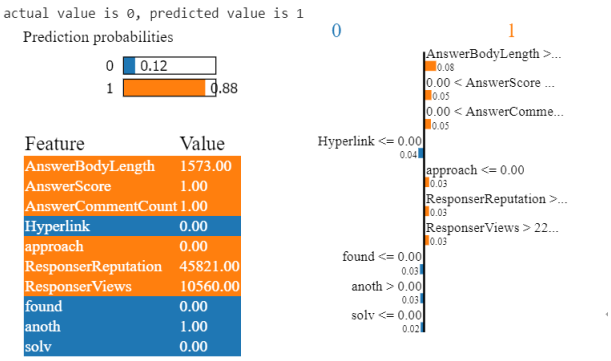


Figure 17. Local feature importance of false positive case 2

As before, we find the original post on Stack Overflow[2]. The question is on how to use column names for arguments of apply function (Appendix 2).

We also find the user accepted answer and the test answer shown in Figure 18 and 19 respectively. We can see that the test answer looks much better than the user accepted answer. The accepted answer is actually posted by the question owner himself. The question owner posted the question and he figured out the answer later on, so he posted his solution and marked his answer as accepted. But the test answer is also a good answer in nature. Therefore, although our model makes some false positives, most of them are good answers in nature. It is just these answers are not labeled as accepted answers for some reason.

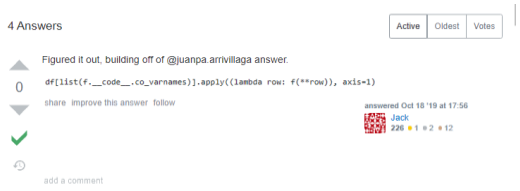


Figure 18. Actual accepted answer for false positive case 2

[2] Original question post on Stack Overflow for false positive case 2. https://stackoverflow.com/questions/58455054/python-pandas-apply-function-using-column-names-as-named-arguments

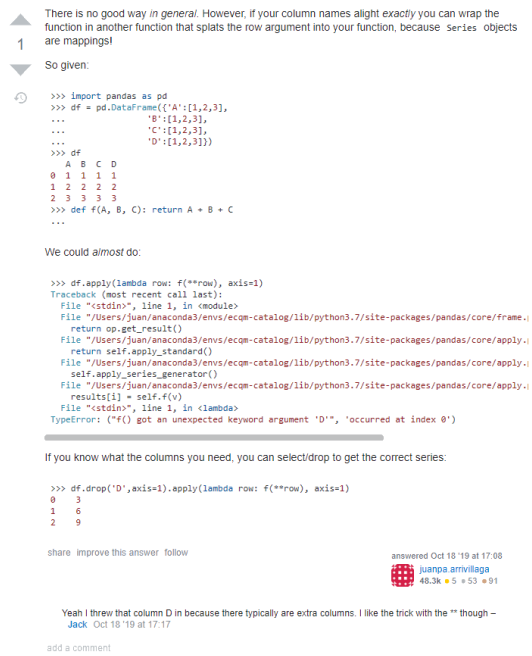


Figure 19. Predicted accepted answer for false positive case 2

4.8 Insights

Based on the feature importance of the best performing model, we can see that non-textual features have a larger weight than textual features in terms of predicting whether an answer is going to be accepted. Features like number of comments an answer received, some reputation aspects of the user who post the answer and number of code blocks an answer has will have major significance in whether this particular answer is going to be accepted. And question owners may not pay much attention to the textbody of an answer, if the code can solve your problem in our context. Besides that, Bag of Words performs better in this dataset.

Moreover, in terms of accuracy of predicting accepted answers, although we cannot correctly predict every accepted answer in our dataset, generally predicted good answers are genuinely reasonable answers. As Stack Overflow only allows one answer to be tagged as accepted, other good answers may not be accepted by the question owner in certain circumstances. Additionally, our prediction model will be largely used in question posts where none of submitted answers is accepted, thus being able to identify one good answer among possible various good answers is sufficient to help users.

On the other side, some intangible aspects like quality of code provided in the answer cannot be perfectly captured in our dataset and feature engineering, this might be one of the limitations our models have in terms of predicting an answer is accepted or not.

5. Conclusion and future work

The project developed binary classifications models that can predict whether stack-overflow python answer posts will be accepted based on features from several different aspects with decent precision scores. The trained models developed can be applied to automatically identify successful answers if question post owners do not choose accepted answers for a long time. This project can be further improved by attempting more time-consuming models such as Support Vector Machine, Deep Neural Networks or even stacking models. This project could also be adjusted to suit different use cases, e.g. the model can help to evaluate answer post quality if only answer content related features are used.

6. References

25 Best Active Forums for Programmers: VironIT. (2019, December 4). Retrieved from https://vironit.com/best-active-forums-for-programmers/

N. Gantayat, P. Dhoolia, R. Padhye, S. Mani and V. S. Sinha, "*The Synergy between Voting and Acceptance of Answers on StackOverflow - Or the Lack Thereof*," 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories, Florence, 2015, pp. 406-409

F. Calefato, F. Lanubile, M. Marasciulo, and N. Novielli, *"Mining Successful Answers in Stack Overflow"* 2015 12th Working Conference on Mining Software Repositories, Mar 2015, pp. 430-433

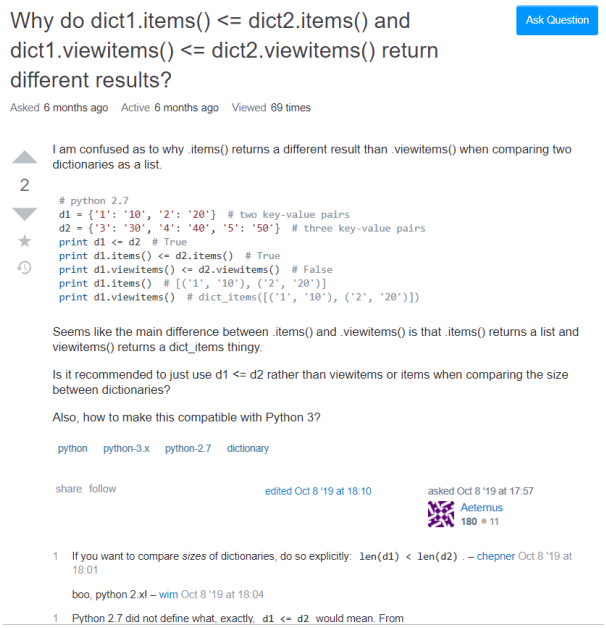
Omondiagbe, Osayande P., Sherlock A. Licorish, and Stephen G. MacDonell. *"Features that Predict the Acceptability of Java and JavaScript Answers on Stack Overflow."* Proceedings of the Evaluation and Assessment on Software Engineering. 2019. 101-110.

Stack Exchange Data Explorer, *"Query Stack Overflow."* Stack Exchange, 10 Feb,2020. Retrieved from https://data.stackexchange.com/stackoverflow/query/new

The Python Package Index. textstat 0.6.0. Retrieved February 10, 2020, from https://pypi.org/project/textstat/

Horev, R. (2018, November 17). *BERT Explained: State of the art language model for NLP*. Retrieved from https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270

7. Appendix



Why do dict1.items() <= dict2.items() and dict1.viewitems() <= dict2.viewitems() return different results?

Ask Question

Asked 6 months ago Active 6 months ago Viewed 69 times

I am confused as to why .items() returns a different result than .viewitems() when comparing two dictionaries as a list.

2

```
# python 2.7
d1 = {'1': '10', '2': '20'}  # two key-value pairs
d2 = {'3': '30', '4': '40', '5': '50'}  # three key-value pairs
print d1 <= d2  # True
print d1.items() <= d2.items()  # True
print d1.viewitems() <= d2.viewitems()  # False
print d1.items()  # [('1', '10'), ('2', '20')]
print d1.viewitems()  # dict_items([('1', '10'), ('2', '20')])
```

Seems like the main difference between .items() and .viewitems() is that .items() returns a list and viewitems() returns a dict_items thingy.

Is it recommended to just use d1 <= d2 rather than viewitems or items when comparing the size between dictionaries?

Also, how to make this compatible with Python 3?

python python-3.x python-2.7 dictionary

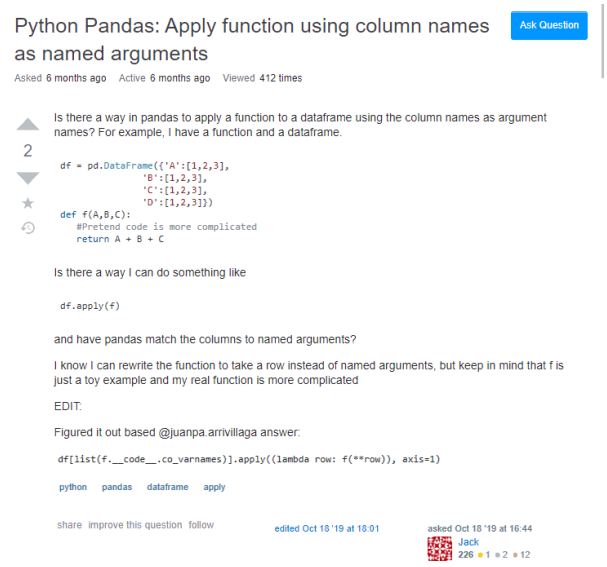
share follow edited Oct 8 '19 at 18:10 asked Oct 8 '19 at 17:57 Aeternus 180 • 11

1 If you want to compare *sizes* of dictionaries, do so explicitly: `len(d1) < len(d2)` . – chepner Oct 8 '19 at 18:01

boo, python 2.xl – wim Oct 8 '19 at 18:04

1 Python 2.7 did not define what, exactly, `d1 <= d2` would mean. From

Appendix 1. Question post on Stack Overflow for false positive case 1



Python Pandas: Apply function using column names as named arguments

Ask Question

Asked 6 months ago Active 6 months ago Viewed 412 times

Is there a way in pandas to apply a function to a dataframe using the column names as argument names? For example, I have a function and a dataframe.

2

```
df = pd.DataFrame({'A':[1,2,3],
                   'B':[1,2,3],
                   'C':[1,2,3],
                   'D':[1,2,3]})
def f(A,B,C):
    #Pretend code is more complicated
    return A + B + C
```

Is there a way I can do something like

```
df.apply(f)
```

and have pandas match the columns to named arguments?

I know I can rewrite the function to take a row instead of named arguments, but keep in mind that f is just a toy example and my real function is more complicated

EDIT:

Figured it out based @juanpa.arrivillaga answer:

```
df[list(f.__code__.co_varnames)].apply((lambda row: f(**row)), axis=1)
```

python pandas dataframe apply

share improve this question follow edited Oct 18 '19 at 18:01 asked Oct 18 '19 at 16:44 Jack 226 • 1 • 2 • 12

Appendix 2. Question post on Stack Overflow for false positive case 2