What Defines a Good Stack Overflow Answer Post, An Automated Post Rater

1. Abstract

Major Q&A sites for professional and enthusiast programmers, like Stack Overflow, have gained phenomenal popularity during recent years. Programmers are relying on these sites to seek inspiration, find solutions and help others. However, many of the question owners do not select their most preferred answer. Therefore, it would be quite inconvenient for other viewers to quickly identify useful information, especially for some unpopular questions with zero vote. In this project, we want to address this issue by building an automated agent that can identify "good" answers. A feature extractor is first used to preprocess the data. Then, we apply logistic regression and neural network to train our model. With our first implementation, we can get 65%~70% prediction accuracy when measured against our metric[1].

2. Introduction

As there are many desired qualities that define a good answer. We build our own feature extractor to gauge our data. The input to our algorithms is the raw dataset obtained from the Stack Overflow online public archive. We used SQL to query related questions, answers, and user information from it. Then we preprocess the data to obtain relevant features, such as comment count, post length, user reputation, user profile views, total user upvotes, total user down votes, the number of code words used in the description, the length of code blocks, hyperlinks(reference), and edit, etc.

Other than the domain knowledge related features mentioned above, we also considered the overall style and fluency of an answer. We trained a unigram and a bigram language model using exemplary text that matches the overall writing style of a good answer post. An n-gram sequence model is a function that, given n consecutive words, provides a cost based on the negative log likelihood that the n-th word appears just after the first n-1 words. The cost will always be positive, and lower costs indicate better fluency.

Then, the extracted features are feed into a logistic regression model and a neural network model. The prediction accuracy is then defined as the percentage of correctly rated posts measured by whether the post is marked as "accepted" by the question owner.

4. Dataset and Features

We scrape all the posts generated by users on Stack Overflow during the period of 2018.04.01~2018.05.01 as our training data, posts generated by users on 2018.03.01 as validation data, and posts generated by users on 2018.06.01 as test data. In total, we have 14504 training data points, 600 validation data points, and 486 testing data points. Since each different feature have relatively different scales (e.g. feature "reputation" can go as high as ~100000, while feature "edit" would only be either 0 or 1), we need to normalize all features to make them relatively comparable. For the logistic regression, we subtracted all features by their mean and then divided them by their standard deviation. For the training on neural network, we used *sklearn.preprocessing.StandardScaler* to perform standardization on our data to speed up the training process and improve numerical stability. Here is one of the training data points we used during training (without normalization):

unigramCost	bigramCost	parsed_CommentCount	parsed_BodyLength	parsed_UserReputation	parsed_UserViews	parsed_UserUpVotes	parsed_UserDownVotes	InlineCode	BlockCode	BlockCodeLine	Hyperlink	Edit	Label
8.756180 04	12.10848 39	0	769	8366	1001	1578	8	7	0	0	1	0	0

Example Training Data Point

Description of each feature:
unigramCost: Text fluency measured by unigram.
bigramCost: Text fluency measured by bigram.
parsed_CommentCount: how many comments are followed to this answer.

[1] Our definition of a "good" answer: The answer marked as "accepted" by the person who asked the question.

parsed_BodyLength: how many words the answer has.
parsed_UserViews: how many people has viewed this user's profile.
parsed_UserUpVotes: how many upvotes this user has earned for all of his previous answers.
parsed_UserDownVotes: how many downvotes the user has earned for all of his previous answers.
InlineCode: how many times code is used within text description.
BlockCode: how many code blocks are used to illustrate solution.
BlockCodeLine: how many lines of code are used in all code blocks.
Hyperlink: how many reference hyperlinks this answer contains.
Edit: whether this answer has been edited.

5. Methods

Because the automated rater rates each question as "will be accepted" and "will not be accepted", we need a classification algorithm to train the model.

The most fundamental algorithm for binary classification is Logistic Regression, so we started with it to have our first trial. Logistic regression uses a sigmoid function to convert a continuous linear regression algorithm into a discrete model. It predicts the probability of $P(y = 1|x; \theta)$. The complete hypothesis expression of it is:

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

by assuming that all training examples are generated independently, the log likelihood function can be obtained as:

$$l(\theta) = \sum_{i=1}^{n} y^{(i)} log h\left(x^{(i)}\right) + (1 - y^{(i)}) \log\left(1 - h\left(x^{(i)}\right)\right)$$

Then gradient decent and newton's method can usually be used to maximize this log likelihood.

Based on the logistic regression model, we attempted to analyze which features are important to the model through wrapper feature selection. We used the backward search approach, which starts with the full set of features and removes the least significant feature(s) at each iteration. At each iteration, the model is trained for each possible elimination of features and evaluated by the prediction accuracy in the validation data. The feature subset with the best model performance is retained. This process is repeated until a good balance between feature variety and model performance is achieved.

As a foray to improve the performance of our model, we also used neural network as the training algorithm. A neural network is formed by layers of neurons. Each neuron consists a weight vector and an activation function. A neuron takes a vector input, compute the weighted average and transform the result according to its activation function. A layer is formed by stacking multiple neurons. And, lastly, a neural network is formed by stacking multiple layers. Usually, a neuron in layer l only talks to the neurons in layer $l \mp 1$. Forward propagation and backward propagation are used to train a neural network.

Forward Propagation:
Given input x, we define $a^{[0]} = x$. Then for layer $l = 1, \dots, N$, where N is the number of layers of the network, we have:

$$z^{[l]} = W^{[l]} z^{[l-1]} + b^{[l]}; \quad a^{[l]} = g\left(z^{[l]}\right); \quad y = a^{[N]}$$

Note: $g(z)$ is the activation function.

Backward Propagation:
To develop a general approach for calculating the gradient of loss function with respect to $W^{[l]}, b^{[l]}, l = 1, \dots N$. We need to define:

$$\delta^{[l]} = \nabla_{z^{[l]}} L(y', y)$$

Then, the three-step "recipe" for computing the gradients with respect to $W^{[l]}, b^{[l]}, l = 1, \dots N$ is the following:

1. For output layer N, we have:
$$\delta^{[N]} = \nabla_{z^{[N]}} L(y', y)$$
2. For $l = N - 1, N - 2, \dots, 1$, we have:
$$\delta^{[l]} = \left(W^{[l+1]T} \delta^{[l+1]}\right) \cdot g'^{\left(z^{[l]}\right)}$$
3. We can compute the gradients for layer l as:
$$\nabla_{W^{[l]}} J(W, b) = \delta^{[l]} a^{[l-1]T}; \quad \nabla_{b^{[l]}} J(W, b) = \delta^{[l]}$$

6. Results and Discussion

Logistic Regression results:

By experimenting with multiple different step sizes, we selected the step size as 1 to get a relatively quick convergence speed. Batch gradient decent is used since the training data set used in the current stage is not too large.

The metrics we used to measure the performance of the automated rater are the confusion matrix, precision, recall, accuracy and F1 score:

Confusion Matrix	Truth, Good	Truth, Bad
Predict, Good	228	115
Predict, Bad	60	83

$\text{Precision} = 66\%, \quad Recall = 79\%$

$Accuracy = 64\%$

$F1\ Score = 0.72$

Feature Selection results:

Preliminary correlation analysis shows that most of the features has a weak positive correlation with the label, with correlation coefficient ranging from 0.013 to 0.179. The correlation is relatively stronger for comment count, user reputation, use votes, and body length.

unigramCost	bigramCost	CommentCount	Body Length	UserReputation	UserViews	User Up Votes	User Down Votes	Inline Code	Block Code	Block Code Line	Hyperlink	Edit
0.013	0.011	0.179	0.131	0.144	0.119	0.146	0.130	0.119	0.114	0.101	0.068	0.058

Feature Selection, Correlation Analysis

Backward search results indicate that the edit, block code (number and lines), and text fluency (unigram cost and bigram cost) features may be of less significance to the model. We will further evaluate whether we will include these features in our model. On the other hand, user features (reputation, up and down votes, and views), comment count, inline code, body length, and hyperlink are more significant in terms of predicting the label.

Iteration	0	1	2	3	4	5	6	7	8	9
Removed feature	unigramCost	Edit	BlockCode BlockCodeLine	bigram Cost	User Up Votes	User Views	Hyperlink	UserDownVotes	Body Length	CommentCount UserReputation InlineCode
Accuracy	0.658	0.660	0.663	0.671	0.669	0.663	0.660	0.652	0.644	0.593

Feature Selection, Backward Search

Neural Network results:

There are 6 hyperparameters in our implementation of neural network: hidden_layer_size, activation, max_iteration, regularization, solver, and tolerance. We choose these hyperparameters based on the validation set prediction results. To avoid over-fitting, the general approach we used to choose these hyperparameters is: Choose the simplest thing that works reasonably well. Here is the list of hyperparameters we used:

Hidden_layer_size	Activation	Max_iteration	Regularization	Solver	Tolerance
(10, 10, 10)	logistic	1000	1e-04	adam	1e-06

Hyperparameter of Neural Network

Here is the confusion matrix of the output from neural network:

Confusion Matrix	Truth, Good	Truth, Bad
Predict, Good	222	99
Predict, Bad	66	99

$\text{Precision} = 69.16\%, \quad Recall = 77.08\%$

$Accuracy = 66.05\%$

$F1\ Score = 0.7291$

7. Future Work

The way we extract features from the dataset limits the neural network's capability of discovering new features from the raw data. We need to come up with a new feature extractor that can preserve the semantic information of raw text in our data while still making the data compatible with the neural network.

8. Contributions

Yanpei Tian: Build the language model to measure "style and fluency" in a post; implement the neural network algorithm.
Yanhao Jiang: Queried data from public archive, feature exploration, logistic regression model implementation.
Chunyue Wei: Calculate features related to post format (e.g. code, hyperlink); conduct feature selection.

9. Reference

[1] *Text Reconstruction*, 8 Nov. 2019, https://stanford.cs221.github.io/autumn2019/assignments/reconstruct/index.html.
[2] "Query Stack Overflow." *Stack Exchange Data Explorer*, https://data.stackexchange.com/stackoverflow/query/new.
[3] "Learn." *Scikit*, https://scikit-learn.org/stable/.
[4] "Machine Learning." *CS229*, http://cs229.stanford.edu/syllabus.html.