QnA Matching for Stack Overflow
CS 6350 Summer 2018: Final Project Report

Siddhant Sahu, Ram Anand Vutukuru
`sxs173732, rxv162130`

July 30, 2018

1 Abstract

One non-trivial community that was constructed and which is growing rapidly on a simple idea of asking questions pertaining to wide variety of technological questions and seeking relevant answers is Stack Overflow. Stack Overflow has been the host for over 16 million questions by mid 2018. With such a large number of questions, more often than not, there would have been a similar question which would have been asked previously. With this project, we try to address that issue by identifying duplicate questions. We will be presented the data with questions posted using various tags such as python, java, javascript, etc. Our intention is to identify these repetitive questions based on these tags and reduce redundancy using a bag-of-words approach.

2 Related Work

One of the quickest ways to find a solution to a common question is most likely to be found in FAQ (Frequently Answered Questions) section of most organizations. FAQs are as nearly as perfect as it gets to answer open ended questions by multiple users online. The idea is to match an open-ended question to a pre-formatted FAQ. There is been a lot of research being done in this domain and there have been multiple approaches as well. To name a few, in *An Effective Similarity Measurement for FAQ Question Answering System*[5], the author addresses the problems by using a semantic knowledge base in order to improve its ability to match a particular question to its relevant answer. In *Ontology-supported FAQ processing and ranking techniques*[6], speaks about an enhanced ranking technique to show query relevant answers by accumulating FAQ information from heterogeneous sources and saves in on an ontological database. The irrelevant words are trimmed, and a partial or full word match is run in-order to get relevant results during the retrieval of FAQs.

3 Dataset

3.1 Raw Data

We use the StackSample[2] dataset, a dataset curated by Stack Overflow containing 10% of all Stack Overflow questions and answers on programming topics, and PostLinks data, obtained from StackExchange data archive. The raw dataset consists of four files:

1. `Questions.csv`: title, body, score, date, closed date (if applicable) and owner ID for each question

2. `Answers.csv`: body, score, date and owner ID for each answer.

3. `Tags.csv`: tags for each question

4. `Postlinks.xml`: Related posts for each post with link type (`duplicate` or `linked`)

3.2 Processed Data

The raw data has a lot of information that is not directly relevant to our main classifier module. The processed data has the following schema:

1. `orig-q.parquet`: raw text, answer IDs and date for each question

2. `dup-q.parquet`: raw text, answer IDs and date for each duplicate question

3.3 ETL Pipeline

Spark[1] is unable to parse multi-line csv files even after the added support in Spark 2.2. Thus, for our experiments, we created an ETL pipeline as follows to convert the raw data to processed data.

1. Parse csv files using pandas[3] (pandas has excellent support for parsing csv files, including multi-line csv files) and load it into sqlite database. Also parse the xml file in python, create a csv file and load it into the sqlite[4] database.

2. Since the process involves joins and the csv files are too big to fit in memory, we rely on Spark SQL to do the joins and save the final output to parquet files.

4 Methodology

4.1 Pre-processing

Duplicates are those questions that have the same answer as the original question. We consider these questions equivalent, semantically. In the table below is the distribution of answer classes against the number of duplicate questions linked for tag `python`. Almost 90% of the answers are linked to 4 or fewer questions. It is prohibitively expensive and doesn't yield as much benefit to train the algorithm for all the answer classes. Therefore, we set our threshold to a number between 10-15 answer classes while training the classifier.

Duplicates count	Number of answer classes
1-2	1653
3-4	198
5-10	121
11-20	34
21+	26

4.2 Feature Extraction

We use only the text column to determine duplicates. Limited on time, we choose only the feature extraction methods provided by Spark ML. The first step is to clean the body of the post – this means removing code blocks, html tags and urls/links. We use a bag-of-words model to determine the vector embedding for the words. We tokenize the text, remove stopwords and compute the tf-idf score for each word and limit the vocabulary to 500 words.

4.3 Modeling

We formulate the problem as a classification problem – with the task of classifying which answerID is the most likely answer for a given question. In simple terms, the input is a list of question-answer pairs (including duplicates) for a particular *tag*. For a new question, our model returns as output, a list of scores for each answer class in the training set.

We use the *One-vs-Rest* strategy to break down this multi-class classification problem into several binary classification problems. Spark's One-vs-Rest classifier does a great job but doesn't output probabilities of

classes. So, we write a custom one-vs-rest classifier with Logistic Regression as the underlying algorithm to tackle this problem.

With popular tags, like `javascript` or `java`, with a lot of questions, the number of answer classes can be as high as 200 or 250 even after the pre-processing step. This presents an imbalanced classification problem for each underlying classifier. For example, some answer classes will have a few positive examples when compared with the rest of the dataset (negative examples). One solution is to take a random subset of the negative examples so that the overall ratio of positive to negative examples is balanced. However, this approach involves writing a custom function to subsample negative examples and is also troublesome while combining outputs from all classifiers. Thus, we resorted to using the weights approach while training Logistic Regression models. We specified weights for both positive and negative examples in each class that effectively imposes a much larger penalty on misclassifying the positive class.

4.4   Evaluation

We use stratified random sampling to split our data into training (75%) and test sets (25%). Stratified sampling proportionately allocates 75% of the questions *per answer class* in the training set and the rest in the test set. Spark doesn't have an inbuilt method for stratified sampling so we use a different approach using window and rank function to accomplish this.

With our current choice of training strategy, i.e. One-vs-rest, using cross-validation and hyperparameter tuning was out of the question, mainly because the model was too expensive computationally.

We use average rank to evaluate our classifier. Average rank is a learning-to-rank evaluation metric that calculates the average rank of the correct answer class in the list of all answer classes. Essentially, the closer this metric is to 1, the better.

5   Experiments

We ran a few experiments – classifiers for the most popular tags on stackoverflow: `python`, `java` and `javascript`

Tag	Average Rank
java	2.27
javascript	2.87
python	2.1
android	3

6   Future Work

There is ample scope of improvement of our model. Given more time and computational power, we'd like to explore along the following areas.

- Use word2vec embeddings instead of tf-idf scores.

- Formulate it as a ranking problem, that can be solved using xgboost.

- Learn useful phrases (otherwise known as collocations) to see how they impact the overall rank.

- Use syntactic features by learning Part-of-speech tags in text. In addition, use several word level and sentence level features.

- Explore deep learning and ensemble models.

References

[1] Apache Spark `https://spark.apache.org/docs/latest/`

[2] Stacksample dataset `https://www.kaggle.com/stackoverflow/stacksample`

[3] Pandas `https://pandas.pydata.org/`

[4] SQLite `https://www.sqlite.org/index.html`

[5] Juan, Zhong Min. "An effective similarity measurement for faq question answering system." *Electrical and Control Engineering (ICECE), 2010 International Conference* on. IEEE, 2010.

[6] Yang, Sheng-Yuan, Fang-Chen Chuang, and Cheng-Seen Ho. "Ontology-supported FAQ processing and ranking techniques." *Journal of Intelligent Information Systems* 28.3 (2007): 233-251.