Duplicate Detection Strategy for Question Answering Sites

KEVIN NANHEKHAN, Delft University of Technology, Netherlands

In this manuscript, we discuss the strategy proposed to detect duplicate questions in community question-answering sites like Quora. An initial solution was used compromised of exact string matching with Data cleaning. To improve this solution, following what existing literature has described, various enhancements have been made such as the use of Tree-based models, Feature Engineering, Neural Networks and Hyperparameter tuning. From the results, we see that the XGBoost classifier performs the best together with a combination of different extracted features (namely basic, fuzzy and especially distance). Through the results, some insights can be derived on the importance of each kind of feature and the various combinations that help the models learn semantics better for determining on questions being duplicates.

CCS Concepts: • **Information systems → Near-duplicate and plagiarism detection**; *Content analysis and feature selection.*

Additional Key Words and Phrases: feature engineering, information retrieval evaluation

ACM Reference Format:
Kevin Nanhekhan. 2023. Duplicate Detection Strategy for Question Answering Sites. In *IN4325-Q3-23: Information Retrieval, TU Deflt*. ACM, New York, NY, USA, 4 pages.

1 INTRODUCTION

Question answering sites, like Quora or StackExchange, offer users access to information based on the wisdom of crowds. How users express their information needs can differ; this translates into duplicates, but non-exactly matching, questions. As stated by Sharma et al. (2019)[6], *"if treated independently, duplicate questions may prevent a user from seeing a high-quality response that already exists and responders are unlikely to answer the same question twice."*.

To address this issue, we have designed and developed a duplicate detection strategy to accurately be able to determine whether two questions are duplicates. The initial strategy is to start with a simple exact string-matching solution and some data cleaning and pre-processing to make duplicate questions be formatted more similarly. These involved: *substituting words, symbols and contractions; setting text to lowercase; removal of leading and trailing spaces, hyperlinks and non-word characters; performing lemmatization.* During testing the accuracy score was the main indication of a solution's performance. It should be noted that the Quora dataset that was used, is imbalanced with only **36.9%** being duplicate questions. This means that simply classifying questions as non-duplicates can already lead to an accuracy above 0.5. Therefore the F1-score was also calculated to see how well the solution performs in terms of precision and recall rate. The initial solution has an accuracy score of **0.63290** and an extremely low F1-score of **0.00993**. Learning semantics between similar questions is an important part of the solution to improve upon. This can be done by using *Tree-based Models, Feature engineering, Hyperparameter tuning and Neural Networks.* The last one was not implemented for the classification process but more so used more in the Feature Engineering part of the duplicate detection strategy.

2 ENHANCING THE SOLUTION

As to not reinvent the wheel, a look has been taken at several papers[1, 3, 6] on which models perform well and what the commonly used features are. From here, we only considered *Random Forest Classifier* and the *XGBoost Classifier* with the following parameters, used during testing, taken from mostly the paper by Sharma et al.[6]:

- **Random Forest Classifier**: *Max depth = None, min samples per leaf = 5, num estimators = 50*
- **XGBoost Classifier**: *Max depth = 4, n estimators = 500*

These models can not learn from the questions as is, therefore feature engineering is needed where with better features we can attain higher accuracies. The following kinds were extracted and compared to not only ensure the right features were selected in training the models but also to look if there was a benefit in combining them in terms of both Accuracy and F1 scores:

- **Basic features** - Basic calculations of the features based on the length and number of occurrences of words and characters in and between the questions.
- **Fuzzy features** - The *RapidFuzz*[2] library, alternative cython implementation of the *FuzzyWuzzy*[4] library, uses Levenshtein Distance to give a similarity score between 0-100 for two question sentences.
- **Sparse features** - Calculating Unigram features (*CountVectorizer*) and TF-IDF features (*TfidfVectorizer*) gives sparse matrices representing token counts of the words in a question. These were excluded in the combination process as multiple attempts led to the program crashing caused by out-of-memory errors.
- **Distance features** - The python framework *SentenceTransformers*[5], based on the neural network *Sentence-BERT*, was used to derive semantically meaningful sentence embeddings from the questions. On these sentence embeddings, various distance metrics (e.g. Cosine, Minkowski, Braycurtis etc.) were calculated.

Following the results (see Appendix), we see a combination of features with the XGBoost classifier leads to the highest attained accuracy and F1 scores. As to increase the performance of the model a bit further, Hyperparameter tuning was initially done through HalvingGridSearchCV. While faster than traditional GridSearchCV, it still took a long time to compute. Instead, to improve this process, the python library *FLAML*[7] was used for searching more optimally through a larger range of parameter values within a smaller timeframe and with better results. From this we got the following optimal parameters: *n_estimators=13500, max_leaves=60, min_child_weight=0.008, learning_rate=0.007, subsample=0.597, colsample_bylevel=0.591, colsample_bytree=1, reg_alpha=0.252, reg_lambda=5.352*. These parameter values managed to bump the accuracy score from 0.83557 to **0.83958** and the F1-score from 0.78535 to **0.79051**.

3 CONCLUSION

Initial evaluations conducted using the Quora Dataset reveal that simple string matching with data preprocessing results in moderate accuracy due to the dataset being imbalanced (favouring classifying questions as non-duplicates) where a lot of false negatives are made. Several enhancements have been made where Feature engineering played a big role in improving the results. The different features standalone perform better than the initial solution. The sparse features made the lowest amount of false positive errors but still more false negative errors compared to the rest. Overall the distance features managed to score the best due to the usage of Sentence-BERT to calculate features that are more semantically representative of the questions. While no conclusions can be made on combining Sparse features with the rest due to memory issues encountered, for other features we see the results increase slightly more. This indicates that the combination and use of more features are meaningful and help the classifier model to learn semantics even better to predict whether questions are duplicates or not.

REFERENCES

[1] Navedanjum Ansari and Rajesh Sharma. 2020. Identifying Semantically Duplicate Questions Using Data Science Approach: A Quora Case Study. *CoRR* abs/2004.11694 (2020). arXiv:2004.11694 https://arxiv.org/abs/2004.11694

[2] Max Bachmann, layday, Jeppe Fihl-Pearson, Moshe Sherman, Delfini, Dan Hess, Henry Schreiner, and Hugo Le Moine. 2023. *maxbachmann/RapidFuzz: Release 2.0.8.*

[3] Andreas Chandra and Ruben Stefanus. 2020. Experiments on Paraphrase Identification Using Quora Question Pairs Dataset. *CoRR* abs/2006.02648 (2020). arXiv:2006.02648 https://arxiv.org/abs/2006.02648

[4] SeatGeek Inc. 2023. *fuzzywuzzy: Fuzzy String Matching in Python.* https://github.com/seatgeek/fuzzywuzzy

[5] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics. https://arxiv.org/abs/1908.10084

[6] Lakshay Sharma, Laura Graesser, Nikita Nangia, and Utku Evci. 2019. Natural Language Understanding with the Quora Question Pairs Dataset. *CoRR* abs/1907.01041 (2019). arXiv:1907.01041 http://arxiv.org/abs/1907.01041

[7] Chi Wang and Qingyun Wu. 2019. FLO: Fast and Lightweight Hyperparameter Optimization for AutoML. *CoRR* abs/1911.04706 (2019). arXiv:1911.04706 http://arxiv.org/abs/1911.04706

APPENDIX

	RandomForest Classifier		XGBoost Classifier	
Features	Accuracy	F1	Accuracy	F1
Basic	0.71876	0.60862	0.72151	0.60255
Fuzzy	0.72191	0.62927	0.72612	0.64087
Sparse (Unigram)	0.75267	0.56235	0.76930	0.63737
Sparse (TFIDF)	0.65918	0.15139	0.75336	0.64256
Distance	0.82141	0.76681	0.82614	0.77316
Basic + Fuzzy	0.74997	0.66338	0.75226	0.66699
Basic + Distance	0.82964	0.77760	0.83479	0.78430
Fuzzy + Distance	0.82566	0.77216	0.83031	0.77904
All Combined	0.83054	0.77858	**0.83557**	**0.78535**

Table 1. Accuracy and F1 scores on Development set for RandomForest and XGBoost Classifiers per different extracted features

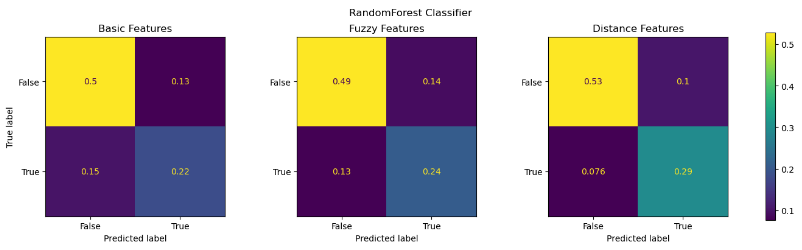

Fig. 1. Confusion matrices for extracted features on the RandomForest Classifier predictions

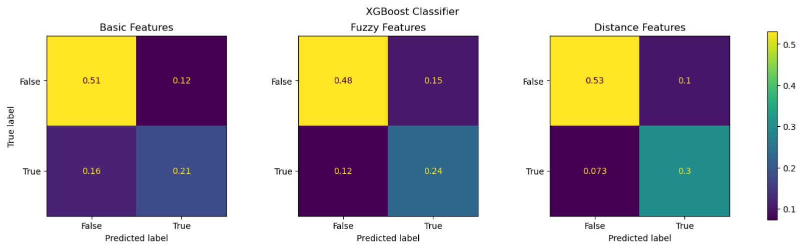

Fig. 2. Confusion matrices for extracted features on the XGBoost Classifier predictions

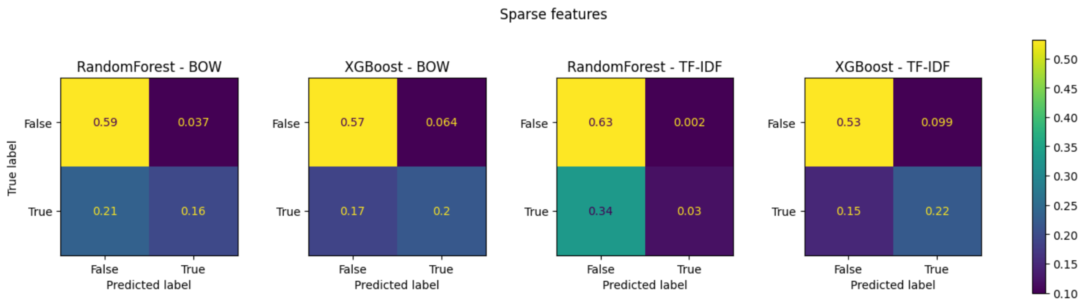

Fig. 3. Confusion matrices for extracted sparse features for both RandomForest and XGBoost Classifier predictions

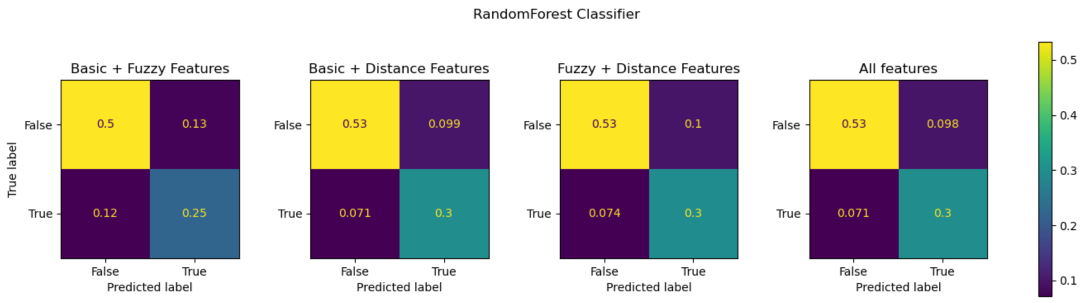

Fig. 4. Confusion matrices for combinations of extracted features for the RandomForest Classifier predictions

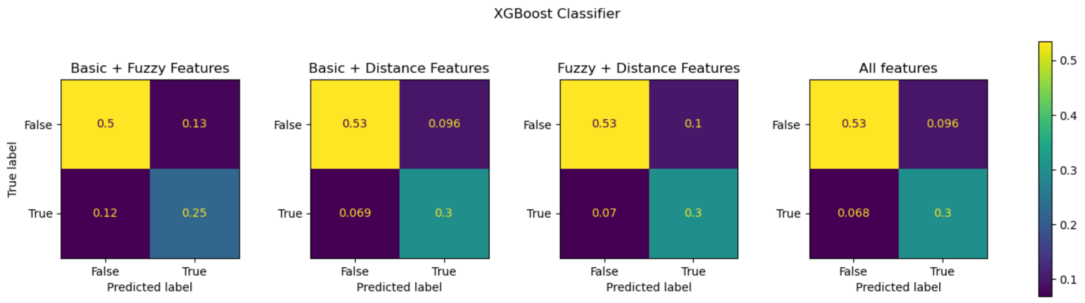

Fig. 5. Confusion matrices for combinations of extracted features for the XGBoost Classifier predictions