Semantic Equivalence detection

DD2417 Mini-Project Report

Isac Lorentz
ilorentz@kth.se

Jiayi Guo
jiayig@kth.se

May 2022

1 Introduction

This project report is about our mini-project in semantic equivalence detection. Given the Quora Question pairs dataset, our task was to train and evaluate classifiers to determine whether pairs of Quora questions are the same or not.Question board platforms such as Quora, StackOverflow etc. enables users to post and answer questions on the platform. Naturally, popular topics gets asked about by several persons, thus leading to the possibility of duplicate questions. This can be problematic for the users since they need to browse several question threads on the same topic and also for the website owners that need to host duplicate information. By utilizing high-performing semantic equivalence classifiers, duplicate questions on such platforms could be merged or redirected, leading possibly to a better user experience and less duplicate content. We study this NLP task on the sentence, word, and character level.

2 Background

2.1 Dataset

The dataset features 404 000 pairs of questions from the Quora platform that are labeled 0/1 for semantic equivalence. The only other features of the data are IDs. 63.07% of question pairs belong to the 0-class (not semantically equivalent), thus a good performing classifier should have higher accuracy than this.

2.2 Bag of words Word embeddings

Bag of words is a simple model used in NLP where each sentence is represented as a bag of frequency count for each word in the context. In this project, the model assigns each word in a pair of questions an index.

We used the 300d Google News word2vec embeddings and the Wikipedia Glove embeddings for glove embeddings [1], [2].

2.3 Transformer Bert

Transformer solves the problem of parallelization and long-range dependencies in RNN based models. The transformer network initially appeared in the paper "*Attention is all you need*" [3]. The network consists of N stacked encoders and the same

number of decoders, both can be a RNN-based network. The encoder takes care of the input and the decoder outputs predictions. The input to the encoder unit is the word embedding and can be passed in paralel. The key mechanism of transformer is the so-called attention and specifically self-attention. The mechanism modifies the word embedding so that the values are more representative of the words they represent in the context of the sentence. Bidirectional encoder representations or BERT by Google AI, uses bidirectional transformer encoders that looks at a sequence from both directions, in contrast to a vanilla transformer.

2.4 Siamese Bert

The siamese Bert used in this project was proposed in the paper "*Sentence Embeddings using Siamese BERT-Networks*" by Reimers and Gurevych [4]. For classification the architecture consists of a pre-trained Bert layer with a mean-pooling layer on top, which means that the sentence embedding is the mean of the word embedding of the BERT layer.

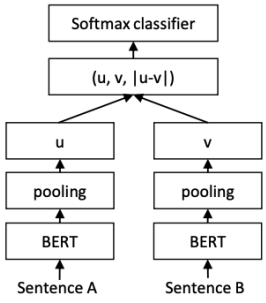

Figure 1: Siamese BERT-network

The sentence embeddings for u and v are then concatenated with the $|u-v|$ and multplied with the weights which outputs o to the softmax classifier. The modified Siamese Bert-network have proven to be more time-efficient and accurate than single BERT.

2.5 XGBoost

XGBoost is a library for efficient, scalable gradient boosted decision trees [5], [6], that perform well on many different types of classification and regression tasks. This is an implementation of the gradient boosting algorithm.

3 Description of implementation

3.1 Bag of words Word2vec (baseline)

In both Bag of words and Word2vec we are measuring accuracy by computing the cosine similarity between the vector representation of the sentences. The library used to implement both models and calculate the similarity was Gensim.

While the bag of words did not require any training since it is only a vector of frequencies of each word in both questions, for the Word2vec we used the pre-trained word embeddings.

3.2 LSTM

The deep learning network has a siamese architecture. First, we feed the sentences (not the strings but the word to indices mapping) into the embedding layer which is shared between both inputs. The embedding layer then retrieves the word embeddings weights from pre-trained Word2vec matrix. The output of the layer are sent to the share LSTM layer. In the end, the results are concatenated and fed to a sigmoid classifier. Regarding feature engineering, we tried to remove stopwords and stemming but none of the techniques did improve the performance significantly.

3.3 BERT with cross-encoder library

We also wanted to test the state-of-the-art network for NLP problems, which is a type of transformer called Bert. Out of interest, we wanted to see how pre-trained Bert models performed, therefore, we imported a pre-trained bert model called "distilroberta-base" and used the Cross-encoder library of the Sentence-Transformer framework. For each pair of questions, the cross-encoder will output the probability of them being duplicates. The classifier is either sigmoid or softmax depending on the number of labels.

To tune the pre-trained Bert model with our data, you have to train it with the dataset. Luckily, for a cross-encoder model we only need to call model.fit on the training data.

3.4 Siamese BERT-network

The Siamese BERT-Network described in the background section was implemented using the python framework Sentence-Transformers. We followed the provided example and used BERT as the base model with the mean pooling layer on top of the model. The vector representation of the sentences was created according to the formula and at the end, we applied the softmax classifier. When predicting the test data we retrieved embeddings for 40 000 questions (more than that was not possible in Google Colab because it would cause a memory error) and then measured the cosine similarity between the questions.

3.5 XGBoost model

We used the XGBClassifier using the default parameters for the Python library (see appendix A).

3.5.1 Feature engineering

From the words and characters in the questions, basic features were extracted such as the count of words and characters per question, difference in number of characters if the first and last words are the same and the number and percentage of words that are same between the two questions. Using the fuzzywuzzy library, the levenstein distance similarity ratio, substring match ratio, tokenized levenshtein distance similarity ratio and tokenized set similarity ratio were calculated. Using the NLTK library, the jaccard distance and edit distance between the two questions were calculated.

3.5.2 Word embeddings

Glove and W2V embeddings were loaded using the gensim library. Using gensim and the word embeddings, the Word Mover's Distance (WMD) [7] was calculated. WMD represents dissimilarity between the two questions as the minimum distance 'that the embedded words of one documents need to "travel" to reach the embedded words of another document'. It is inspired by the Earth mover's distance transportation problem and perform well on real world classification tasks in comparison to state of the art at the time of publication [7]. WMD distances were calculated using w2v and glove embeddings for normalized and non-normalized word embeddings. Using the w2v and glove embeddings, we created vector embeddings of the whole questions. This was done by tokenizing the words in the questions, remove stopwords and tokens not only containing letters, and then sum each vector embedding

for each word in the embedding and then normalize the result. Frow these vectors, we calculated distances between the vector embeddings of the questions: cosine, cityblock, euclidian and minkowski.

4 Results

Since this is a labeled dataset, evaluation is quite simple. Stratified K-fold was used to test the XGBoost classifier due to the unbalanced nature of the dataset.For the rest of the models we only trained them once with 70% of the dataset because it required a lot of time. The XGBoost model was trained and evaluated using the full dataset, for the other models we used a subset of the data. The feature importance graphs are made using the built-in feature importance plot in the xgboost library. It ranks the importance of the different features are plotted based on the F-score (how many times the variable is split on).

	predicted 0	**predicted 1**
Real 0	64046	13104
Real 1	17834	26321

Table 1: LSTM (Trained with more data by switching the order of q2 and q1), **Accuracy: 0.7450**

	predicted 0	**predicted 1**
Real 0	45748	30850
Real 1	11103	33604

Table 2: Bag of Words, Cosine,**Accuracy: 0.6541**

	predicted 0	**predicted 1**
Real 0	46950	29648
Real 1	12179	32528

Table 3: Word2vec, **Accuracy: 0.6551**

	predicted 0	**predicted 1**
Real 0	19951	5154
Real 1	1023	13872

Table 4: Siamese Bert, **Accuracy: 0.8456**

	predicted 0	**predicted 1**
Real 0	77150	0
Real 1	44155	0

Table 5: BERT base model, **Accuracy: 0.6360**

	predicted 0	**predicted 1**
Real 0	73055	4095
Real 1	33312	10843

Table 6: Trained sentence transformer cross-encoder, **Accuracy: 0.6916**

Word embedding	**Accuracy (STD)**	**Precision (STD)**	**Recall (STD)**	**F1 (STD)**
W2V	0.7639 (0.0016)	0.6584 (0.0022)	0.7492 (0.0021)	0.7009 (0.0019)
Glove	0.7684 (0.0016)	0.6617 (0.0025)	0.7628 (0.0030)	0.7086 (0.0018)
W2V + Glove	0.7726 (0.0013)	0.6677 (0.0021)	0.7647 (0.0029)	0.7129 (0.0015)

Table 7: XGBoost, 10-fold stratified cross validation results

Word Embedding	**class**	**precision**	**recall**	**f1-score**
W2V	0	0.84	0.77	0.80
W2V	1	0.66	0.75	0.70
Glove	0	0.85	0.77	0.81
Glove	1	0.66	0.76	0.70
W2V + Glove	0	0.85	0.78	0.81
W2V + Glove	1	0.67	0.76	0.71

Table 8: XGBoost, 30/70 test train split, we can see that the precision is significantly higher for the 0-class (not semantically equivalent)

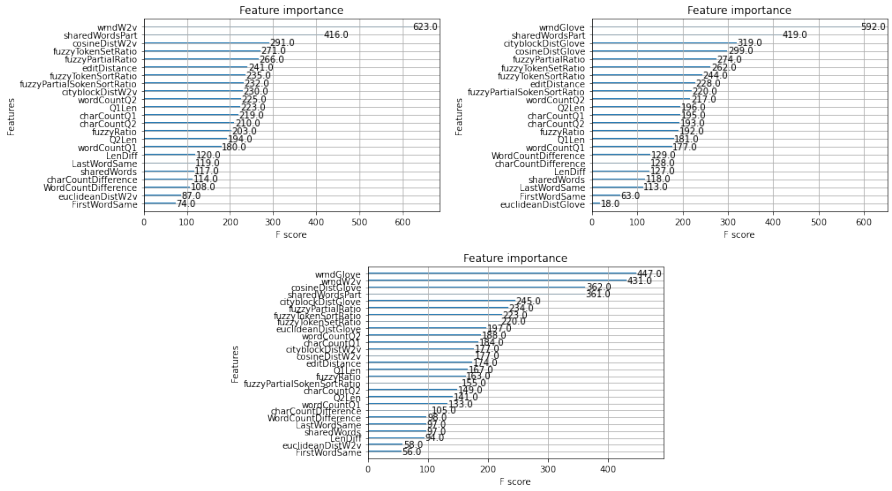

Figure 2: XGBoost Feature importance, W2V (Left), Glove (Right), both (bottom)

5 Discussion & Conclusions

According to the result, the model with the highest accuracy was the Siamese BERT-network and XGBoost was the second best (Tables 4,7,8). However, the of the Siamese BERT-network was measured with only 40 000 pairs of questions, additionally we could not perform a cross validation due to the fact that it took extremely long time to train the network.

The pretrained BERT model, that had not seen the dataset, only predicted 0 (Table 5). The model was outperformed by our baseline models, e.g. Word2vec and Bag of words (Tables 2,3). We were surprised by the result of Bag of Words as it based on a very simple idea and did not require any training at all.

Another factor that we discovered during the project was that the threshold of cosine similarity. Initially we used 0.5 as the threshold but as we tried to play around with the value, we found out that it could improve the accuracy significantly. For instance, with threshold as 0.5 we only got 55% accuracy on the siamese BERT-network. When that is change to 0.8, the accuracy increased to approximately 85%.

For the XGBoost model, Glove performed slightly better than W2V and W2V + Glove slightly better than only Glove. This can possibly be due to that the combination covers a larger set of words. All XGBoost modles had significantly higher precision for the 0 class (tables 7-8). When using both Glove and W2V embeddings, the Glove-based distance metrics are consistantly ranked as more important features than the w2v-based ones. The word movers' distance was the most split upon feature for

both W2V and Glove. Cosine distance and the ratios obtained using the fuzzywuzzy library were ranked in the top for all configurations (Figure 2).

We discovered that engineering some of the complex features for a dataset this large took a lot of time. What we could have done differenly was to first do a proper exploratory analysis of the dataset and maybe do some kind of upsamling. Another thing is that we did not define any baseline model at the beginning, we got distracted by trying new exciting models that we did not have much indepth knowledge of, i.e transformer. But testing a smaller number of models, we could have tested more configurations and more embeddings and engineered features per model. We also realized the importance of GPU in deep learning as we had to train the model on either Google Colab or Kaggle with GPU accelerator to finish the training in time. Overall we learned a lot and this problem had more depth than it seemed at the beginning.

References

[1] *GloVe: Global Vectors for Word Representation*. `https://nlp.stanford.edu/projects/glove/`. Accessed: 2022-05-18.

[2] Jeffrey Pennington, Richard Socher, and Christopher D Manning. "Glove: Global vectors for word representation". In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, pp. 1532–1543.

[3] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[4] Nils Reimers and Iryna Gurevych. "Sentence-bert: Sentence embeddings using siamese bert-networks". In: *arXiv preprint arXiv:1908.10084* (2019).

[5] Tianqi Chen and Carlos Guestrin. "Xgboost: A scalable tree boosting system". In: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 2016, pp. 785–794.

[6] *XGBoost Documentation*. `https://xgboost.readthedocs.io/en/stable/`. Accessed: 2022-05-18.

[7] Matt Kusner et al. "From word embeddings to document distances". In: *International conference on machine learning*. PMLR. 2015, pp. 957–966.

Appendix A: XGBoost Parameters

```
{'objective': 'binary:logistic', 'base_score': 0.5, 'booster': 'gbtree',
'colsample_bylevel': 1, 'colsample_bynode': 1, 'colsample_bytree': 1,
'gamma': 0, 'gpu_id': -1, 'interaction_constraints': '',
```

'learning_rate': 0.300000012, 'max_delta_step': 0, 'max_depth': 6,
'min_child_weight': 1, 'monotone_constraints': '()', 'n_jobs': 8,
'num_parallel_tree': 1, 'predictor': 'auto', 'random_state': 0, 'reg_alpha': 0,
'reg_lambda': 1, 'scale_pos_weight': 1, 'subsample': 1, 'tree_method': 'exact',
'validate_parameters': 1, 'verbosity': None}