Project Report for WDPS 2022 Practical Assignment - Group 8

Yudong Fan 2662762
y.fan@student.vu.nl
Faculty of Science, Vrije Universiteit Amsterdam

Yihong Xi 2769073
y.xi2@student.vu.nl
Faculty of Science, Vrije Universiteit Amsterdam

Yue Zhang 2772421
y.zhang8@student.vu.nl
Faculty of Science, Vrije Universiteit Amsterdam

Zhuofan Mei 2757904
z.mei@student.vu.nl
Faculty of Science, Vrije Universiteit Amsterdam

I. Overview

This report is about the practical assignment of the Course Web Data Processing Systems (WDPS) held by prof. Jacopo Urbani at Vrije Universiteit Amsterdam. The goal of the assignment is to implement a program which is capable of recognizing and linking entities in web pages and extract relations between them. Throughout the report, the design of the program will be discussed step by step. Moreover, important techniques used and the decision making process including the major trade-offs will be elaborated.

II. Design of the pipeline

The general structure of the pipeline is shown in figure 1. The input data is initially stored in the format of WARC files. After processing and extracting text from a certain HTML page, the entity recognition section will select potential mentions (entities) inside the text. Then, the entity linking section generates candidates from Wikipedia for each mention and ranks the candidates based on some similarity measurements (details in later sections). The candidate which scored at the first place will be selected as the entity linking for the given mention. Finally, the relation extraction section takes all the linkable entities as input and extracts potential relations between them. The final result is printed out and stored.

A. Text Extraction

Initially, we used beautiful soup 4[1] to extract all texts inside HTML pages. However, the result was not satisfactory because we did not have a good way to determine what HTML tags contain useful information for our task and to what extent or depth we should split different contexts within the same page. After doing some research, we realize that this task is not trivial. Since the result of text extraction affect the final result of the pipeline significantly, we decide to utilize the Trafilatura [1] package, which is an accurate and efficient text scraping tool, to extract texts from HTML pages. Nevertheless, in order to achieve a high recall, naive scraping using BS4 is also kept as a backup approach in case Trafilatura does not work with certain pages.

[1]https://beautiful-soup-4.readthedocs.io/en/latest/

B. Named Entity Recognition

For named entity recognition (NER), we use the spaCy natural language processing framework [2]. More specifically, we use the pretrained transformer-based pipeline *en_core_web_trf* which is currently considered the state-of-the-art in terms of NER [2]. It is worth mentioning that the pipeline is built on top of RoBERTa [3], which is an enhanced variant of BERT. In addition, *en_core_web_md* which is a lighter and CPU-optimized NER pipeline is kept as a backup means for NER in case the running environment does not support the transformer-based pipeline.

C. Entity Linking

Particularly, we implement a 'Wikipedia-based method' to do entity linking. Notably, only context-independent features are considered and used in this implementation due to the concern of program complexity and efficiency.

1) Candidate Generation: In order to generate candidates for a given mention, we use Wikipedia python library[2] for easy access of the Wikipedia data. By requesting Wikipedia using the name of a certain mention, we get the corresponding Wikipedia entity page for it. If the entity page does not exist, we label the mention as unlinkable (a certain degree of typo handling and auto-suggestion is enforced due to the redirect mechanism of Wikipedia). In principle, the direct Wikipedia entity and all the entities listed in the disambiguation page of that entity page are considered to be the candidates for the corresponding mention.

2) Candidate Ranking: The candidates are ranked based on a weighted score which considers two metrics, namely the similarity between the mention and the candidate and the popularity of the candidate. Each metric weights 50% towards the final score. On the one hand, the similarity is calculated using a string similarity measurement inspired by Guo et al. [4]. Essentially, we compute the highest similarity score ranging from 0 to 1 between all possible names for a certain mention and all possible names for a certain candidate. Specifically, the possible names for the mention and the

[2]https://github.com/goldsmith/Wikipedia

candidate can be fetched from their Wikipedia redirect pages. Although this measurement is still on the surface, it goes one step further as it generates a more comprehensive similarity score between two bags of names instead of two names. Moreover, acronyms also get more chances to be recognized and analyzed correctly. Notably, a threshold value of 0.3 is set to eliminate unpaired candidates for better efficiency. On the other hand, we made an assumption that pages with more views are more likely to be the right entity links. Therefore, the page popularity is pegged to the page view, and the page views are normalized into a popularity score between 0 and 1, with 0 representing the most unpopular and 1 representing the most popular. After gathering the similarity score and the popularity score, the weighted score for a candidate is calculated following $WeightedScore = 0.5 * SimilarityScore + 0.5 * PopularityScore$. Finally, the surviving candidates are ranked based on their weighted score.

3) Linking Prediction: Given the fact that this implementation only considers context-independent features, and the score for ranking are computed during run-time, the prediction can be made right after the candidate ranking. Namely, the candidate with the highest weighted score will be selected as the entity link to the corresponding mention. If no candidate survives after pruning based on the similarity threshold, the corresponding mention will be labelled as unlinkable.

D. Relation Extraction

1) Dataset Selection: In the task of relation extraction, the selection of the dataset is crucial. After considering factors such as data size, quality, and distribution, we chose the wiki20m dataset[5] for training and development. The wiki20m dataset addresses the noisy labelling issues in the Wiki20 datasets and has a large, manually annotated test set. Furthermore, the dataset has been refined through entity linking.

2) Data Processing: In the preprocessing of the dataset, we first split the dataset into training, development, and test sets in an 8:1:1 ratio (in json format). For the content of the dataset, we used NLTK's WordPunctTokenizer to tokenize the text in each sentence. The tokenized sentences were then concatenated to form the single context for each input file. We also extracted the position of each entity in the context and stored it in the corresponding field. The label of each relation triple was also extracted and mapped to the corresponding relation name using the "rel2id.json" file. The preprocessed files have the same format as the input files, with the added *position* field and the mapped *relation ID* in the labels field.

3) Model Construction: Inspired by the work of Lin et al. [6], we construct the relation extraction model. For further clarification, the objective of our model is to take a set of sentences $\{x1, x2, \ldots, xn\}$ and two corresponding entities, and then employ a Convolutional Neural Network (CNN) to measure the probability of the relation between each entity in this set of sentences. First, the words in the sentences are transformed into a dense vector of real-valued features. We also refer to a dictionary[3] that maps tokens to ids developed by the Stanford Natural Language Processing team to facilitate this computation process. This process is depicted in Figure 2. Specifically, before inputting x to the CNN, we first convert the words into low-dimensional vectors. Here, each input word is converted into a vector by the Word Embedding matrix. In addition, Position Embedding is used for all words in the sentence to specify the position of each entity pair. Subsequently, the convolution, max-pooling, and non-linear transformation layers are used to construct a distributed representation x of the sentence. Furthermore, to solve the optimization problem, Stochastic Gradient Descent (SGD) was chosen to minimize the objective function by randomly selecting a small batch from the training set for iteration until convergence. Also, dropout [7] is employed in the output layer to prevent overfitting problem.

4) Model Deployment: Following previous work, we use validation on the training set to tune the model. For training, we followed the settings in the experiment of Zeng et al [8]. , setting the number of iterations to 25 for all training data. The parameters used in the model training process are shown in Table I. The eventually trained model will be applied to the extraction of relation in our pipeline.

III. Efforts Toward a better Scalability

Two techniques are implemented for a better scalability of the pipeline. That is, holding some offline dictionaries to cache the requested Wikipedia data for repetitive mentions to use later such that the total amount of requests sent to the Wikipedia is reduced, and utilizing python concurrent.futures library[4] to enforce parallel processing for the entity linking section. After evaluation, the efficiency of the pipeline increases significantly (depends on the max worker assigned to the thread pool, but generally 5-10 times faster) after implementing the techniques above.

IV. Evaluation of the result

For entity linking, gold annotations are generated using Falcon 2.0 [9]. In short, the results for Falcon 2.0 entity linking reports F-scores between 0.53 - 0.84 on different benchmark datasets. The evaluation for entity linking based on Falcon 2.0 is shown in Table II.

As for relation extraction evaluation, the gold annotation file was generated using OpenNRE's[10] toolkit. We pre-trained it with the wiki20m dataset and CNN which are the same as our model in the generation, and the final AUC was at 0.66. Our F1 value for the relationship extraction evaluation was 0.226, and the other parameters are shown in Table III.

References

[1] A. Barbaresi, "Trafilatura: A Web Scraping Library and Command-Line Tool for Text Discovery and Extraction," in *Proceedings of the Joint Conference of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: System Demonstrations.* Association

[3]https://nlp.stanford.edu/projects/glove/

[4]https://docs.python.org/3/library/concurrent.futures.html

for Computational Linguistics, 2021, pp. 122–131. [Online]. Available: https://aclanthology.org/2021.acl-demo.15
[2] M. Honnibal, I. Montani, S. Van Landeghem, and A. Boyd, "spaCy: Industrial-strength Natural Language Processing in Python," 2020.
[3] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.
[4] Y. Guo, B. Qin, Y. Li, T. Liu, and S. Li, "Improving candidate generation for entity linking," in *International Conference on Application of Natural Language to Information Systems*. Springer, 2013, pp. 225–236.
[5] H. Xiao, "Wiki20m. figshare. dataset." [Online]. Available: https://doi.org/10.6084/m9.figshare.19333988.v1
[6] Y. Lin, S. Shen, Z. Liu, H. Luan, and M. Sun, "Neural relation extraction with selective attention over instances," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2016, pp. 2124–2133.
[7] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.
[8] D. Zeng, K. Liu, S. Lai, G. Zhou, and J. Zhao, "Relation classification via convolutional deep neural network," in *Proceedings of COLING 2014, the 25th international conference on computational linguistics: technical papers*, 2014, pp. 2335–2344.
[9] A. Sakor, K. Singh, A. Patel, and M.-E. Vidal, "Falcon 2.0: An entity and relation linking tool over wikidata," in *Proceedings of the 29th ACM International Conference on Information amp; Knowledge Management*, ser. CIKM '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 3141–3148. [Online]. Available: https://doi.org/10.1145/3340531.3412777
[10] X. Han, T. Gao, Y. Yao, D. Ye, Z. Liu, and M. Sun, "OpenNRE: An open and extensible toolkit for neural relation extraction," in *Proceedings of EMNLP-IJCNLP: System Demonstrations*, 2019, pp. 169–174. [Online]. Available: https://www.aclweb.org/anthology/D19-3029

APPENDIX

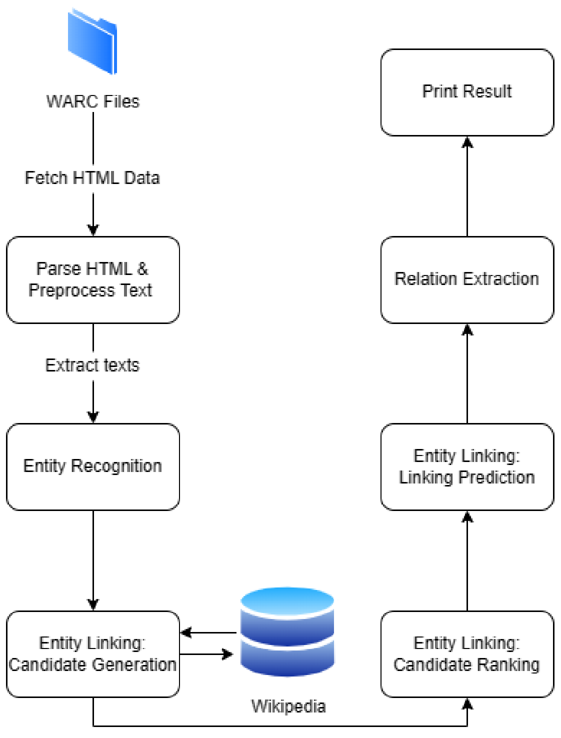
WARC Files
Fetch HTML Data
Parse HTML & Preprocess Text
Extract texts
Entity Recognition
Entity Linking: Candidate Generation
Wikipedia
Entity Linking: Candidate Ranking
Entity Linking: Linking Prediction
Relation Extraction
Print Result

Fig. 1. Overall Structure

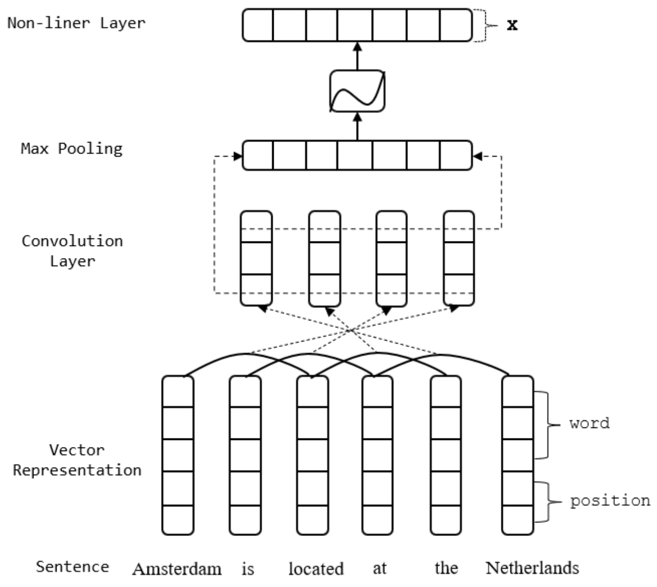

Fig. 2. Model structure

TABLE I
RELATION EXTRACTION MODEL TRAINING PARAMETER

Parameter	Value
Window size	3
Sentence embedding size	230
Word dimension	50
Position dimension	5
Batch size	16
Learning rate	0.01
Dropout probability	0.5

TABLE II
ENTITY LINKING EVALUATION

Parameter	Value
Gold	7347
Predicted	5577
Correct	1151
Precision	0.206
Recall	0.157
F1	0.178

TABLE III
ENTITY RELATION EXTRACTION EVALUATION

Parameter	Value
Gold	2855
Predicted	2858
Correct	646
Precision	0.2660
Recall	0.2662
F1	0.2661