Robustness of Deep Entity Matching systems

Tianyu Wang (1006757948)

Abstract

In this project, I conduct experiments on a current state of art method, Ditto, for Entity Matching task, to study its robustness to noise. I develop a program to simulate Typographical error, add different levels of noise to the dataset, and experiment Ditto method using noisy datasets.The result shows the Ditto method is sensitive to noise, and its performance of the model trained on the dirty dataset is more robust than the model trained on the clean dataset.

1. Introduction

Entity matching is a task of determining whether two data entries refer to the same real-world entities, current state of art methods using pre-trained language models such as Bert (Devlin et al., 2019) based on transformers(Vaswani et al., 2017) and improve performance of Entity matching task by a large interval, but work (Di Cicco et al., 2019) have shown that current SOTA entity matching systems, making use of deep language models are prone to adversarial attacks changing their outputs, and may "rely on untrustworthy attributes". In the case of deep entity matching, a typical problem is a noise in the form of mistakes in the input data.

Ditto (Li et al., 2020) is one of the most recent SOTA entity matching systems, it is base on pre-trained language models, and introducing Domain Knowledge injection, TF-IDF-based summarization, Data augmentation to further improve the performance of Ditto.

In this study, I investigate the robustness of Ditto by comparing the performance of Ditto trained on clean and dirty datasets.

In section 2, I give a brief background on Ditto and introduce the method of simulating Typographical error.
In section 3, I describe the experiment set up and the result of the experiment, for each dataset, I train two version of the model, one trained on the clean dataset, and one trained on the dirty dataset with random typographical error, and I tested their performance on tested with different level of Typographical error. As shown by the result, the model trained on dirty dataset is much more robust.

2. Background and methodology

2.1. Ditto

Ditto applies pre-train language models to fine-tune the models for EM tasks. They added a fully connected layer and softmax layer for binary classification after the final layer of the LM, this is the model architecture they used in Ditto and this addition of the final layer can work on a different variant of the pre-trained language model. Ditto experiment is carried out on datasets form ER Benchmark datasets (Köpcke et al., 2010), Magellan data repository (Das et al.) and WDC product data corpus (Primpeli et al., 2019), datasets contain pairs of data entries, each pair is labeled as matching of non-matching, to be meaningfully ingested by the model they serialized each paired entry, indicating the start of attributes, values in data entries, and separation of two data entries as a pair.

In the basic version of Ditto, they fine tuned the basic model on the serialized dataset, and outperform previous SOTA methods, overall achieves significantly higher F1 scores. Besides the basic version of Ditto, they also implement versions of ditto with data argumentation and domain knowledge injections, which achieves similar performance as the basic version of Ditto.

2.2. Simulating typing error

To test method robustness, I implemented a program, TypingMesser to add Damerau-type errors (Damerau, 1964) to word tokens, following the idea of SpellMess (Agarwal et al., 2007). The program can introduce 5 types of error, insertion, deletion, substitution, transposition and duplication. It also requires a Kbmatrix which encodes the keyboard layout same as SpellMess.

On the keyboard, for each alphabetical letters L, the alphabetical letters on the left or the right side of L are called highly related to L, at most two letters are highly related to L, the other alphabetical letters surrounding L are called lowly related to L, at most 6 letters are lowly related to L. For example on British and American keyboards, ‘a’ and ‘d’ is highly related to ‘s’, ‘q’, ‘w’, ‘e’, ‘z’, ‘x’, ‘c’ are lowly related to ‘s’

At most one type of error can be introduced for a word token, and only a word token with length greater than 3 can have an error, each type of error is equally likely. Below is the effect of each type of error:

Insertion: add a letter to the word token at a random position except for the head of the word token, the letter added depends on the previous letter in this word token. If the previous letter is an alphabetic letter, then the letter added will be highly related to the previous letter for 70% of the time, or lowly related to the previous letter for 30% of the time. If the previous letter is a numeric digit, the letter added will be a random numeric digit. If the previous letter is not a numeric digit nor an alphabetic letter, no letter will be added.

Example: brown -> broiwn , KL2145 -> KL21545

Deletion: remove a random letter from the word token, except for the head of the letter.

Example: jump -> jmp

Substitution: select a random letter from the word token except for the head of the word, substitute it with another letter. If the selected letter is an alphabetical letter, it will be substituted by a highly related letter for 70% of the time or by a lowly related letter otherwise. If the selected letter is a numeric digit it will be substituted by a random numeric digit.

Example: fox -> fod, 100453 -> 105453

Transposition: Select a random letter from the word token except for the head and the tail of the word, transpose it with the letter that comes after it.

Example: quick -> qucik

Duplication: select a random letter from the word token except for the head of the word, insert the same letter after it.

Example: 10.34 -> 10..34

Same as SpellMess, We need set the overall error probability, for example, if we set the overall error probability to be 0.2, then given a text file 20% of the word tokens will have a typing error, and the 5 types of typing error are equally likely.

3. Experiment

3.1. Preprocess dataset

The dataset I use are 8 datasets from ER-Magellan datasets used in Ditto, other datasets are not experiment due to resource limit. I use TypingMesser to add spelling error to test set and validation set and create versions of the test set and validation set with overall error probability rate 0.0, 0.1, 0.2, 0.4, 0.6 for each dataset, for each overall error probability rate, 5 versions of the different test set and validation set is generated with 5 different random seed.

3.2. Experiment setup

The experiments are carried on UoT CsLab slurm cluster, compare to the setup of the experiment in Ditto, each model is trained with half of the batch size due to the memory of GPU, which means the batch size is 16 if MixDA is used and 32 otherwise. The number of epochs is also halved to

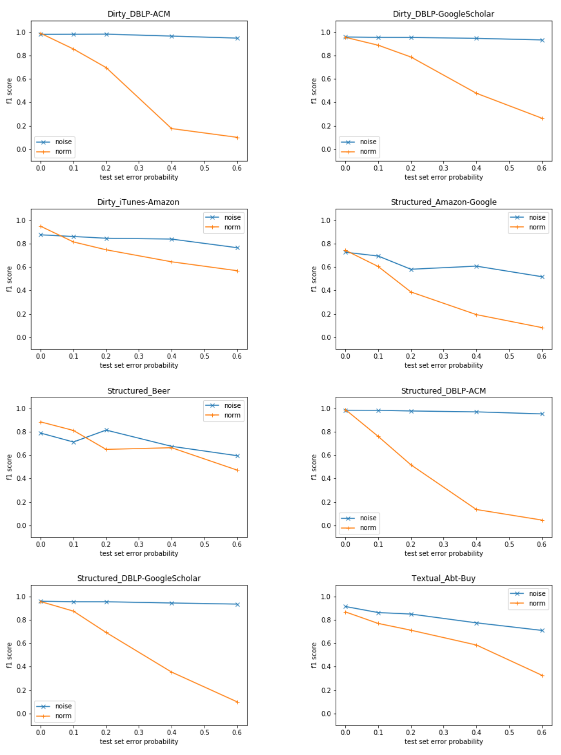

Figure 1: average f1 score of Clean Ditto (Orange) compare with Dirty Ditto (Blue)

make the number of iteration the same and reduce training time. In all the experiments, the max sequence length to be 256 and the learning rate to be 3e-5 with a linearly decreasing learning rate schedule, and the half-precision floating-point (fp16) optimization is also applied, these settings are the same as settings in Ditto.

3.3. Compared Methods

Two methods are compared, the performance of each method is test

Clean Ditto: the baseline version of Ditto. Compare to the complete Ditto method, it doesn't use data argumentation or domain knowledge injection but use TF-IDF summarization.

Dirty Ditto: Compare to Clean Ditto, typing error is injected into the training set and the overall error probability is 20%.

3.4. Results and analysis

Fig 1 shows the average f1 score of Clean Ditto compare with Dirty Ditto on a different dataset. We can see the differences in f1 score for the two methods is small when the test set is clean, . With the increase of noise level in the test set, the f1 score of Clean Ditto drop drastically, while the f1 score of Noisy Ditto is only slightly affected by the error level.

As the performance of Dirty Ditto is only slightly affected, this suggests that even the noise level is 60%, enough discriminating features are retained which enable the correct classification of pairs of data entries. (Agarwal et al., 2007) also observed a similar phenomenon, where the performance of a classifier trained on the dirty dataset is only slightly dropped even the noise level is 100%. The author suggests the classifier might learn the random pattern in the noise for classification, which may be the same reason for the robust performance of Dirty Ditto.

As the noise level of the training set in Dirty Ditto is only 20%, this suggests that a low level of noise would enable the model to classify data entries with a higher level of noise without a large drop in performance. This may also mean the model learned the pattern of noise when the noise level is only 20%.

(Agarwal et al., 2007) observed that 66.67% of word error rate doesn't affect the performance of traditional machine learning classifiers trained on clean dataset and test on a dirty dataset for a general document classification task. But in our case, 60% of word error rate cause the performance of Clean ditto drop drastically, there might be two reasons: The first reason is the entity matching task as a special case of the document classification task is sensitive to noise; The second reason is the use of the model, Ditto use Deep Neural Network and deep model is easily overfitting low-level features compare to the traditional machine learning model. The first reason is likely, as data entries have more concentrated information compare to general documents if important attributes are corrupted, paired data entries might be impossible to classify with human-level performance. The second reason is a known problem, a complex model is easier to overfit, which certainly is an important reason for the drop of the performance.

4. Conclusion and further work

This project have showed that ditto method trained on clean dataset is not robust to noisy data, for the performance of entity matching systems using deep model, it is important to make sure the training set and testing set are in the same domain, or at least add random noise training set to make the model robust. For further work, we can test using other more general data argumentation techniques to see if they help to improve performance, while keeping the training set and testing set in different domain, for example dropout data augmentation that used broadly in computer vision or the attribute-level data augmentation designed in Ditto, the techniques generally applicable to different types of data value unlike the method designed in this project that only words on alphabetic and numeric values and only augment the data in character level. We can also analysis the differences between Clean Ditto and Dirty Ditto using LIME method (Di Cicco et al., 2019), and compare the dependence of attributes in different classifiers.

References

Agarwal, Sumeet, Godbole, Shantanu, Punjani, Diwakar, and Roy, Shourya. How much noise is too much: A study in automatic text classification. In *Seventh IEEE International Conference on Data Mining (ICDM 2007)*, pp. 3–12. IEEE, 2007.

Damerau, Fred J. A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 7(3):171–176, 1964.

Das, Sanjib, Doan, AnHai, G. C., Paul Suganthan, Gokhale, Chaitanya, Konda, Pradap, Govind, Yash, and Paulsen, Derek. The magellan data repository. https://sites.google.com/site/anhaidgroup/projects/data.

Devlin, Jacob, Chang, Ming-Wei, Lee, Kenton, and Toutanova, Kristina. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

Di Cicco, Vincenzo, Firmani, Donatella, Koudas, Nick, Merialdo, Paolo, and Srivastava, Divesh. Interpreting deep learning models for entity resolution: an experience report using lime. In *Proceedings of the Second International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*, pp. 1–4, 2019.

Köpcke, Hanna, Thor, Andreas, and Rahm, Erhard. Evaluation of entity resolution approaches on real-world match problems. *Proceedings of the VLDB Endowment*, 3(1-2): 484–493, 2010.

Li, Yuliang, Li, Jinfeng, Suhara, Yoshihiko, Doan, AnHai, and Tan, Wang-Chiew. Deep entity matching with pre-trained language models. *arXiv preprint arXiv:2004.00584*, 2020.

Primpeli, Anna, Peeters, Ralph, and Bizer, Christian. The wdc training dataset and gold standard for large-scale product matching. In *Companion Proceedings of The 2019 World Wide Web Conference*, pp. 381–386, 2019.

Vaswani, Ashish, Shazeer, Noam, Parmar, Niki, Uszkoreit, Jakob, Jones, Llion, Gomez, Aidan N, Kaiser, Łukasz, and Polosukhin, Illia. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.