RoBuEM: Robust Embedding for Deep Entity Matcher against Input Noise

For course CSC 2508 Advanced Data Systems, University of Toronto

Investigator: Miao Xi

Introduction

Entity Matching (EM) identifies and connects different data instances that represent the same real-world entity across different data sources. Its importance has been growing since it provides us the ability to integrate data fragmented across systems, websites and platforms and create a more holistic view of the real-world entity of interest. It has remained challenging because of poor data quality, large number of possible matches and dependency on external human knowledge and interaction (Barlaug & Gulla, 2021). Deep Learning (DL), with its high popularity and edge in performance, has been introduced to EM. Its most prominent advantage over earlier approaches is its ability to learn features instead of relying on carefully handcrafted rules (LeCun et al., 2015). There are much information about a real-world entity that are recorded as text. One major implementation of DL in EM is in the form of word embedding. Because of its power of converting text data into vectors that can be further processed with numeric operations. Word embedding is widely adopted in the state-of-the-art (SOTA) research in EM (Mudgal et al., 2018). It is proven to be successful in largely reduce human involvement in the end-to-end EM pipeline and increase accuracy. Yet, there still exists issues that word embedding has yet to solve for the field.

One of the major issues is the presence of misspellings in records caused by human errors. Although the study of word embedding has been popular in recent years, many SOTA word embeddings cannot work with noisy input such as typos. Because they are trained on large, well-constructed corpus such as entire Wikipedia dump or Google News. The rare occurrence of typos in these high-quality documents causes their absence from the training data. These SOTA word embeddings are not designed with working on typos in mind. They are more focused on understanding and representing the semantics of words. Typos are treated as out-of-vocabulary (OOV) tokens. And in many cases, for examples, GloVe (Pennington et al., 2014) and Word2Vec (Mikolov, 2013) are 2 most popular word embeddings, all OOV tokens are represented with one single token. This prevents deriving any meaning from the typos and inevitably leads to the loss of information. This would be potentially problematic in the field of EM, where many times typos made in the messy process of real-world data generation is prevalent. Without the ability to effectively deal with such issue, the potential of the implementation of word embedding and DL could not be fully realized.

In this paper, two different types of solution would be discussed. One is to implement a typo corrector module before the records are fed into the EM pipeline. The other would be creating a solution to generate robust word embeddings that could work with typos directly within the current EM pipeline. The main contribution of this paper is to introduce RoBuEM, a solution that generates robust word embedding using FastText (Bojanowski et al., 2017) from scratch based on the input datasets. Hence, it is customizable on the given EM task.

Related work

When dealing with non-standard text such as typos, there are two major approaches. One is to apply some preprocessing step that remove noise, such as spell checking or text normalization (Doval et al., 2020). Another one is to construct a character-level embedding that can capture the relationship of subwords (letter patterns within a word).

SymSpell. This is a fast single-word spelling correction algorithm with very high performance (more than 1000 times faster than BK-Tree when searching) (Garbe., 2012). It differs from traditional spelling correction because it only considers deletion operation when generating edit candidates and dictionary look up. While the standard approach requires considering four different operations (deletes, transposes, replaces and inserts).

In solving noisy input in EM, SymSpell could be leveraged as the main component of the typo corrector module to provide fast spelling correction before input file starts in the EM pipeline.

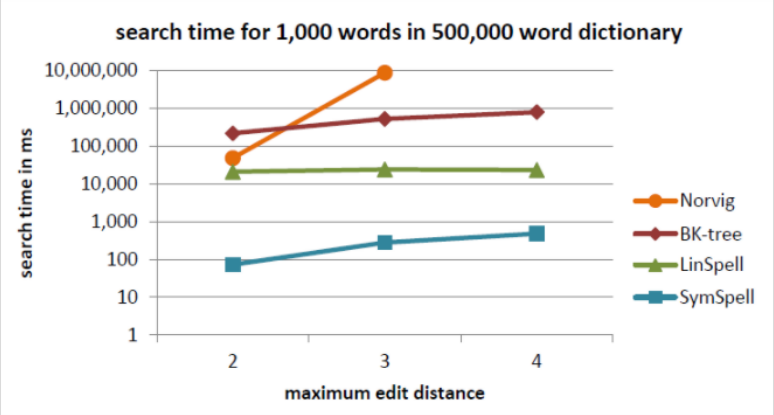

Figure 1.

FastText. This is a popular, recent proposal in the area of word embedding. It leverages the morphological information of words (how a word is formed by different subwords) and uses these features on the Word2Vec framework which enables building embeddings for OOV words (Piktus et al.,

2019). Based on FastText, there are several new proposals to build robust word embeddings. One is Misspelling Oblivious Embedding (MOE). It achieves this by modifying loss function and training on both correct corpus and corpus with artificially injected misspellings. The feature of deliberately training word embedding with misspellings is borrowed to this paper to create RoBuME.

DeepMatcher. This is a new development in the field of EM, which focuses on perform EM on structured datasets. It provides categorization of DL implementations in various matching tasks and developed four DL solutions (SIF, RNN, Attention and Hybrid) to tackle those tasks. It provides an open-sourced Python package (https://github.com/anhaidgroup/deepmatcher). In this paper DeepMatcher would be used as the end-to-end EM solution once the input noise reduction/typo correction is achieved.

RoBuEM architecture

There are two possible types of solution to reduce the impact of typos on the EM result. One is to deploy a Typo Corrector module. In terms of architecture, the Typo Corrector would be placed before the start of EM pipeline to pre-process the input data and make corrections. Then the processed text data where typos are converted, are then feed into the EM pipeline.

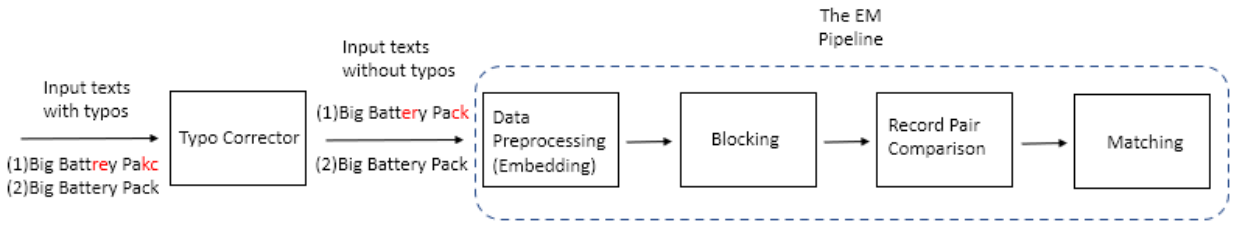

Fig 2. The reference architecture of the Typo Corrector in relation to generalized EM pipeline.

As illustrated in Fig 2., since the Typo Corrector does not participate in the EM pipeline, it can be seen as transparent to the EM pipeline. This particular architecture is not further pursued because of one major drawback that negatively impact its implementation is EM. It is constraint by dictionary. Automatic Spelling Correction (ASC) systems usually select correction candidates from a dictionary of correct words after detection of a spelling error (Hládek et al., 2020). It cannot process words that are outside of the correct dictionary. This is especially problematic because EM would be applied to fields such as e-commerce, manufacturing, where large number of terminologies/jargons that are specific only to one organization or even one department. And there would also be instances such as serial numbers, product IDs that would not be considered as words to be built into a dictionary. Yet, these information are curtail for the EM task. Hence, this architecture cannot be used as a general solution for noisy input with typos.

The other solution architecture proposed (shown in Fig 3.) could better overcome this drawback. RoBuEM module is divided into three sections.

- Input Duplication. It first combines the two datasets to be worked with A and B together as a single text file, and then duplicate the file by a number of iteration *i* and obtain the enlarged file *F*.
- Additional Typo Injection. At this point, it is unclear whether the input file *F* contains typos or not, and if it does, how much? The task of RoBuEM is not to identify and correct any typo inside *F*, but to learn, represent each token with a robust embedding that assign vectors to tokens that are likely to be similar both syntactically and semantically. To achieve this, the duplication and typo injection sections work together to create a training corpus with diverse subwords representing the same word in the same scenario. The result of duplication and typo injection is shown in Fig 4.
- FastText Embedding Training. *You shall know a word by the company it keeps* (Firth, J. R. 1957:11). With the typos surrounded by tokens that usually surround the correct words, the training of the FastText model (skip-gram) learned to blur the boundaries between them. And the subword morphological features could be borrowed from one typo to another to create understanding for those typos not being present in the training dataset.

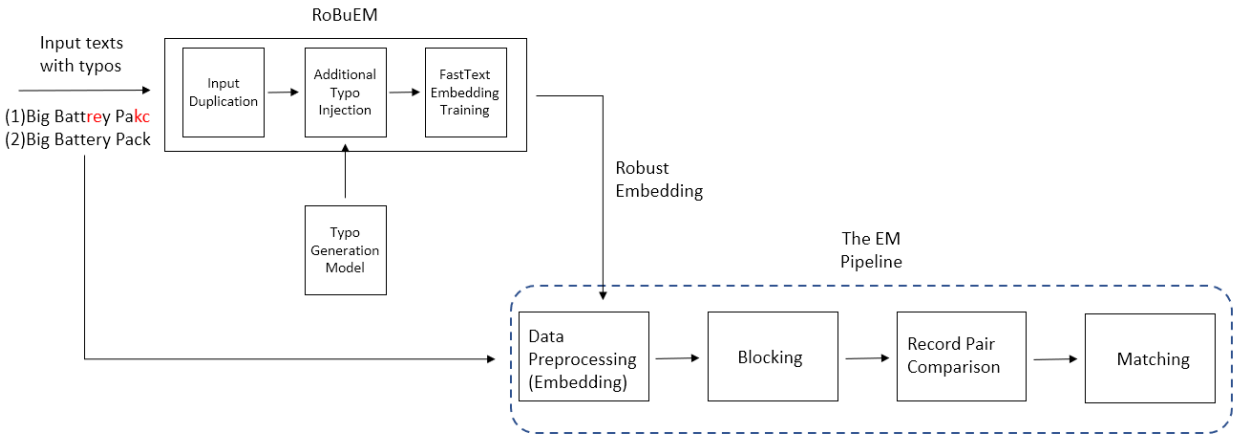

Fig 3. The reference architecture of the RoBuEM in relation to generalized EM pipeline.

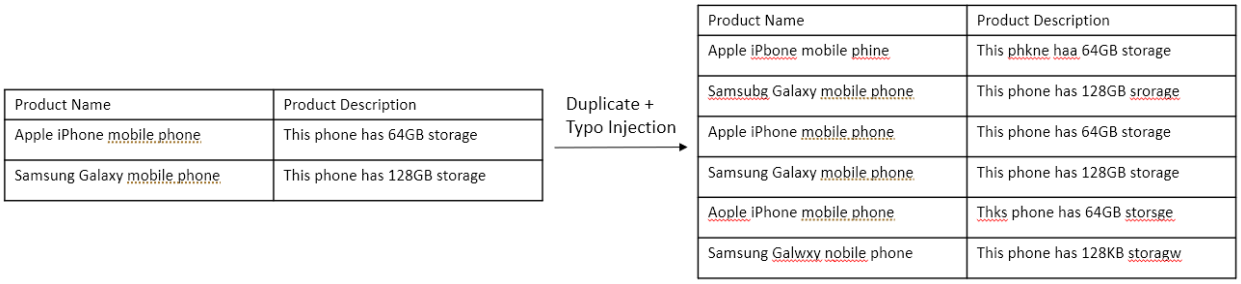

Fig 4. The illustration of the result obtained from Input Duplication and Additional Typo Injection

The Typo Generation Model (TGM) is independent from RoBuEM, but at the same time crucial to its performance. It is independent because it could be replaceable by different models with the same purpose. Yet, it needs to clearly mirror the process, through which the typos are generated, so that the embedding trained by FastText could reflect the actual misspellings within the input data. In this paper, the TGM used is an open-source, naïve rule-based implementation (https://github.com/alexyorke/butter-fingers) that randomly assigns characters based on keyboard difference provided certain probability (noise level).

After the word embedding is trained, it is integrated into the DeepMatcher package to conduct tasks such as attribute summarization, comparison, and classification.

Empirical Evaluation

Goals and Takeaways: The first goal is to determine whether RoBuEM could improve the performance of DeepMatcher with noisy input. The second goal is to determine if RoBuEM has any advantages over the current implementation of DeepMatcher.

The main takeaways are as follows. (1) On the dataset investigated, when injected 0.05 noise, RoBuEM achieves better validation F1 score compared to two proposed methods in DeepMatcher (using pre-trained FastText Eglish wiki embedding and training embedding from scratch on clean data). (2) When noise level increases, RoBuEM constantly outperformed DeepMatcher original methods. (3) RoBuEM trained embedding takes significantly smaller RAM than using pre-trained embedding, faster to train, and at the same time offer similar if not better F1 score. (4) RoBuEM performed significantly better on some structured data than on other structured data and text data with the same noise level.

Datasets: The datasets used in this paper are a subset of those used in the DeepMatcher paper. While the Walmart-Amazon dataset is more intensely focused.

Table 1. Datasets used in this paper

Type	Dataset	Domain	Size	# Pos.	# Attr.

Structured	Walmart-Amazon	electronic	10242	962	5
	iTunes-Amazon	music	539	132	8
Textual	Abt-Buy	electronic	9575	1028	3

Methods: All RoBuEM embedding is trained on datasets with 0.05 injected noise level while performing EM with DeepMatcher on data with varying noise levels. Following the original DeepMatcher paper, a data split of 3:1:1 between train, validation and test is applied. The validation F1 score is used to select the best model and represent the model performance. All DeepMatcher models are trained with 10 epochs and repeated 5 times to reduce the chance of outlier. Mean, max and min are obtained from the 5 repetitions to be used as measurement. To speed-up training time, most analysis on most datasets use a 500-row sample with negative to positive ration at 4:1. All models are trained with Attention attribute summarization.

Obtain the best learned embedding

As mentioned in previous section, the number of iteration *i* that dictates the fold of duplication when creating the training corpus is an important factor to impact the result of DeepMatcher using RoBuEM's learned embedding. Here five embeddings are trained with *i* euqals from 0, 5, 10, 15, 20. When *i*=0, there is no duplication and typo injection, the original input datasets are directly used to train an embedding from scratch. When *i*=20, the input datasets are duplicated 20 times with typo injection to all and then added with the original input sets. Below is the result on Walmart-Amazon data to determine *i* that offers highest validation F1. From Fig 5., *i*=15 is the optimal number that provided results constantly beat using pre-trained FastText word embedding on Wikipedia.

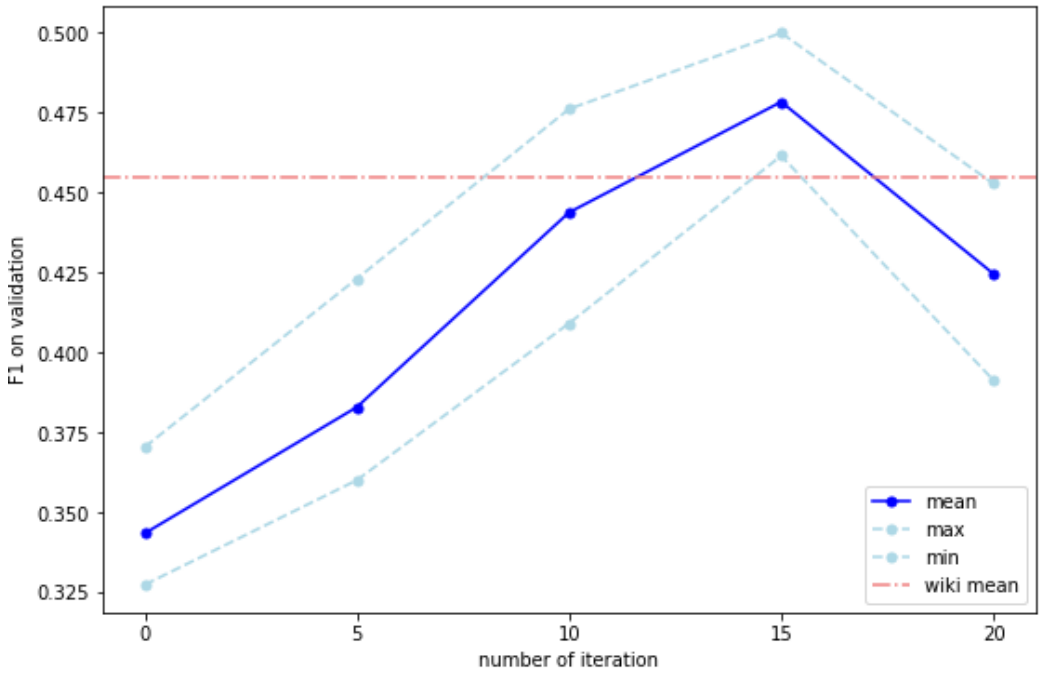

Fig 5. Comparison between word embeddings trained on different *i* and pre-trained embedding

Tabe 2. Comparison between different *i* and its embedding size (stored as .zip)

Pre-trained Wiki	*i=0*	*i=5*	*i=10*	*i=15*	*i=20*
8089 MB	221 MB	292.5 Mb	314.2 MB	326.4 MB	335.6 MB

From table 2. It could be observed that when *i=15*, RoBuEM trained embedding not only exceeds in F1 than pre-trained one but also takes less than 1/24 of space.

Comparing different noise level

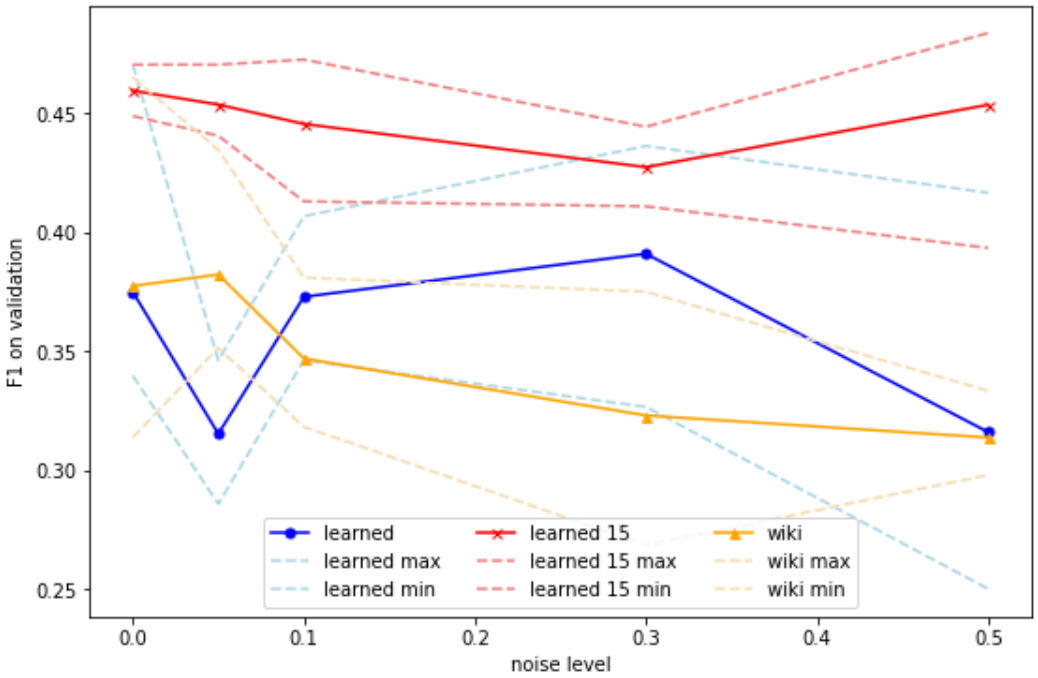

Fig. 6 Comparison between three studied embeddings in response to increasing noise level

Fig 6. Shows that the best RoBuEM embedding (learned 15) consistently outperforms on data with different noise level compared to learned from scratch and pre-trained embedding, Also, it is the least impacted among these three embeddings by the increased noise within the data. This shows that RoBuEM trained embedding is more robust to typos.

Comparing large and small training sets

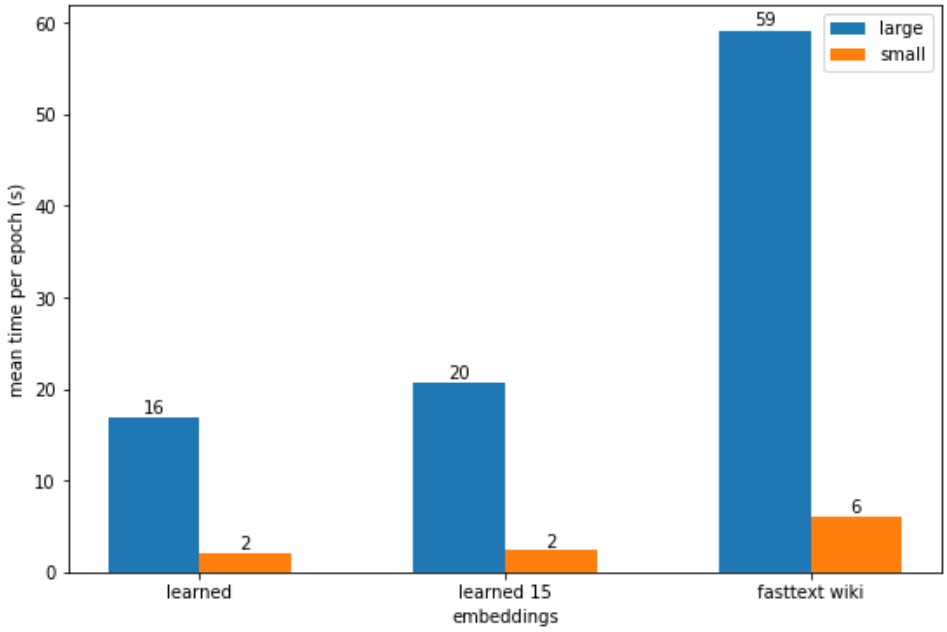

Fig 7. Comparison of training speed on both large and small datasets across three studied embeddings

Previous analyses are done on small samples of data (with 500 rows total between train, validation, and test). Here we compare the impact on large dataset (4810 rows) on the performance of the three embeddings. In terms of time, average epoch time increased linearly for all three embeddings. We could see the pre-trained embedding takes 3x the training time of the best RoBuEM embedding (learned 15).

Table 3. Comparison of mean validation F1 between large and small datasets for three embeddings

	learned	learned 15	pre-trained wiki
Large	0.4666	0.5449	0.5707
Small	0.4677	0.4655	0.4807

From Table 3., it could be observed that by increasing dataset by 10 times, there are significant improvement in F1 for RoBuEM and pre-trained embeddings. While in this test, pre-trained embedding provides slightly higher F1 score of 1.5-2.5%, considering the fact it would take 3x the time to train, RoBuEM is still a very good choice.

Comparing different datasets

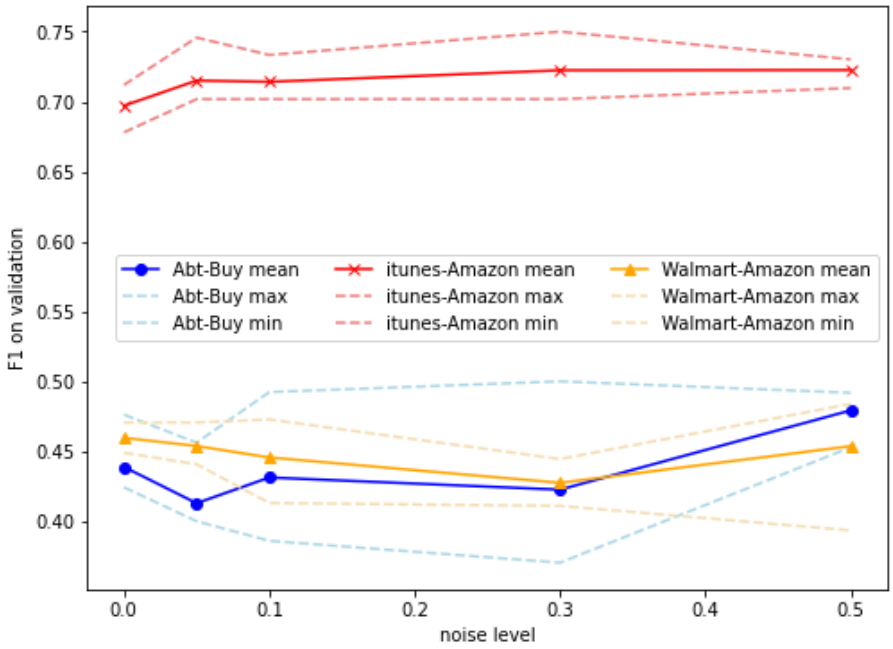

Fig 8. Comparison between the performance of RoBuEM learned 15 embeddings across three datasets and increasing noise levels

In this test, a different learned 15 embedding is trained with 0.05 noise level based on different datasets and evaluated across increasing noise levels with small samples (500 rows). It could be observed that iTunes-Amazon dataset yields the highest performance and consistency. This could be due to the relative larger number of attributes in this dataset. Big variation of performance is observed for the Abt-Buy dataset, which is the only textual data. It may indicate that the learning from scratch even with duplication and injection in RoBuEM may be difficult to work with this type of data.

Conclusion

In this paper, an architecture for reducing the impact of typos on EM is proposed. The approach centers around using duplication and manual injection of typos into the input dataset to create a learned-from-scratch, robust word embedding. Because of the simplicity of this approach, it is proven on the studied datasets that it could achieve similar, if not higher F1 score working on noisy data compared to the far larger pre-trained embedding currently implemented in DeepMatcher.

Future Work

One aspect of future activities could be expanding the analysis to more datasets. Also, ss mentioned in previous sections, even though the Typo Generation Model is not part of the RoBuEM architecture, to construct a truly robust embedding is as important. The current implementation is a naïve rule-based model. The next step could be to construct a more sophisticated, Machine Learning based model that could more closely reflect the typos created in the real world.

Reference

Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2017). Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Linguistics*, *5*, 135–146. https://doi.org/10.1162/tacl_a_00051

Doval, Y., Vilares, J., & Gómez-Rodríguez, C. (2020). Towards Robust Word Embeddings for Noisy Texts. *Applied Sciences*, *10*(19), 6893. https://doi.org/10.3390/app10196893

Hládek, D., Staš, J., & Pleva, M. (2020). Survey of Automatic Spelling Correction. *Electronics*, *9*(10), 1670. https://doi.org/10.3390/electronics9101670

Mikolov, T. (2013, January 16). *Efficient Estimation of Word Representations in Vector Space*. ArXiv.Org. https://arxiv.org/abs/1301.3781v3

Mudgal, S., Li, H., Rekatsinas, T., Doan, A., Park, Y., Krishnan, G., Deep, R., Arcaute, E., & Raghavendra, V. (2018). Deep Learning for Entity Matching. *Proceedings of the 2018 International Conference on Management of Data*. https://doi.org/10.1145/3183713.3196926

Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global Vectors for Word Representation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. https://doi.org/10.3115/v1/d14-1162

Piktus, A., Edizel, N. B., Bojanowski, P., Grave, E., Ferreira, R., & Silvestri, F. (2019). Misspelling Oblivious Word Embeddings. *Proceedings of the 2019 Conference of the North*. https://doi.org/10.18653/v1/n19-1326

Garbe, W. (2012, June 7). *1000x Faster Spelling Correction algorithm*. SeekStorm. https://seekstorm.com/blog/1000x-spelling-correction/