MATCHING IDENTICAL PRODUCTS

USING TEXTUAL DATA

By

Attils Szűts

Submitted to

Central European University

Department of Economics and Business

In partial fulfillment of the requirements for the degree of

MSc in Business Analytics

Public Project Summary

Budapest, Hungary

2021

Introduction

My client is a business facing (B2B) company providing IT services to their clients. One of their services is procurement analytics. This service stands for helping their clients analyze their spend on resources and services necessary for everyday business operations (such as office supplies, transport, etc.). My client receives purchase order data, transforms it through a series of text processing steps, and at the end of the pipeline they create a dashboard for the customer that they can analyze with the help of Business Intelligence officers.

My project is about developing a proof of concept for a new analytical use-case in this dashboard. I developed an item-to-item matching framework using only textual data (purchase order description). This enables my client to aggregate identical transactions and help their client to negotiate better terms for future procurement.

Methodology

Product matching is a growing field in data science and e-commerce where researchers want to identify similar and identical products using various data available (title, description, meta-data, text, etc.). It is essential for large web shops to be able to find duplicate products in their catalogs, and this becomes ever more difficult as the number of products grows exponentially. There has been research already conducted in this field to identify whether advanced machine learning algorithms can be used to find identical products. (Ristoski, Petrowski, Mika, & Paulheim, 2018) (Tracz, et al., 2020)

I have followed a similar strategy in my project as in previous research: using a two step approach by first calculating item similarity for each product pair and then clustering all the products into different categories. My client already had existing frameworks to calculate the similarity of textual data using classical metrics such as term frequency-inverse document frequency (tf-idf) and Levenshtein distance and I used this as a benchmark.

I wanted to introduce deep learning tools to see if I can improve the performance of finding identical products. Therefore, I developed two approaches side-by-side: a “classic” approach, using already implemented techniques to find item similarities and an “embedding” approach using Google’s Universal Sentence Encoder to encode cleaned purchase order text into 512 by 1 vectors. These vectors can then be directly compared to each other, on how similar they are which can be done by simply taking the Euclidean distance of any two vector.

After having both a classic and embedding metric on item similarity we could find identical products by plugging the pre-computed distances into a DBSCAN clustering algorithm. After running the clustering algorithm, we will have a predicted group for each transaction, and this will be the output of the program.

There is one more important step for the project, which is to optimize the parameters of the DBSCAN algorithm. The minimum number of points is a straightforward parameter, it must be one, since there can be unique elements which should not be grouped together with other items in the transactional data. The other parameter that needs to be optimized is the ε, epsilon (EPS) parameter which gives a distance around each core point within which every point will

be assigned to that group. To find this best ε we need two things: a loss function that determines a trade-off between finding identical and finding different products and we need a training sample to test our results.

Creating a training sample was not straightforward as it is the case in unsupervised learning tasks. After having item similarity metrics, we ran a DBSCAN on the inputs and with the predicted outputs we took a random sample to contain only around 3.000 transactions. We then exported the results, and without looking at the predicted groups, I manually clustered together similar products.

The final step was to iterate over all the values in the series: 0.02, 0.04, 0.06, … 0.98 and cluster the processed transaction similarity metrics for each ε above (for both classic and embedding). After the clustering, we took the transactions that were validated and calculated the recall for identical and different products. This was done by taking the validated transactions, pairing them up with all the rest of the validated transactions comparing their predicted group if they matched (m), and comparing the validated groups if they matched (mV). If m = mV then that means a correct classification, if m ≠ mV then that transaction was incorrectly clustered in that group. We could then calculate precision and recall for the validated transactions to see how well it could cluster identical and different products.

Conclusion

Our loss function was semi formalized, and it required for the best epsilon to maximize recall for identical products, while having at least 0.98 recall for different products. You can see in Figure 1 that for the classic approach this epsilon was 0.02 and achieved a 0.19 recall for identical products while the epsilon for embedding approach was 0.48 and achieved a 0.28 recall for identical products which is a 150% increase in performance.

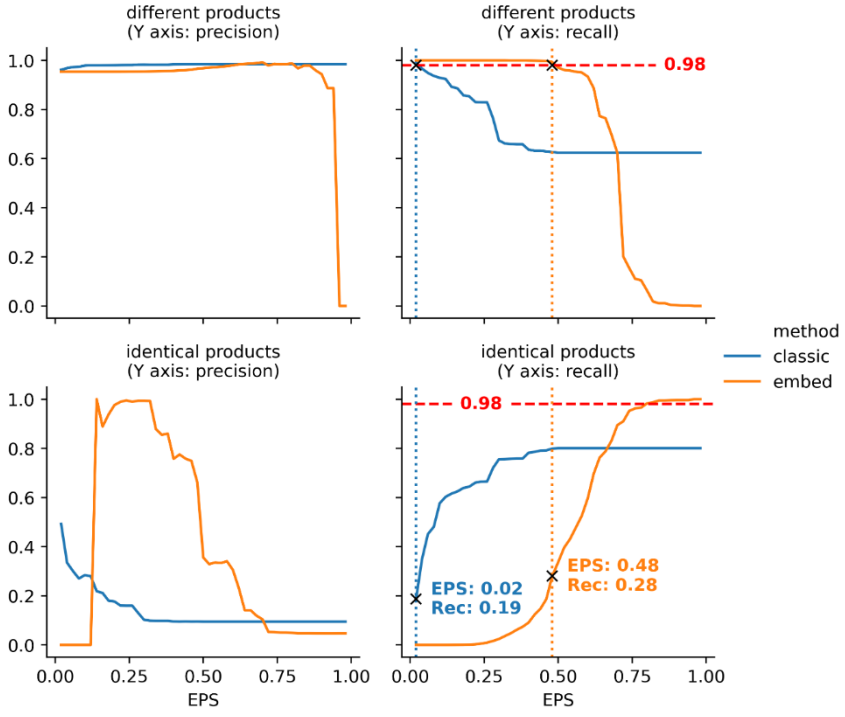

Figure 1 Precision and recall for different and identical products for classic and embedding methods

As this was a proof-of-concept project, this can be considered a success for two reasons. We managed to create a codebase for matching identical products that can be immediately put into production, and we have developed a framework for continuously testing and improving the solution.

My recommendations for future work include formalizing a more appropriate loss function as initial data shows that with a marginal decrease in different product recall, we can approximately double the recall of identical products. This suggests that by formalizing a more comprehensive loss function we can achieve better performance with little to no additional work required. The next idea for future work I have is to run the whole process using data from another client to see how the models perform (both embedding and classic for a benchmark) with different data. We expect to see similar metrics both in absolute and relative terms, which would corroborate the external validity of the framework. Finally, it would be worthwhile to test whether additional data cleaning and feature engineering could improve model performances. This is an important question because my client must decide how much work they invest in their clients and if models have to be tailored individually this might be enough overhead to make this approach hard to adapt to every client and might be considered as a premium product for higher paying or more loyal customers only.