Using a Skip-gram Architecture for Model Contextualization in CARS

Dimitris Poulopoulos and Athina Kalampogia
QIVOS, Athens, Greece

Keywords: Context-aware Recommender Systems, Proximity Marketing, Deep Learning.

Abstract: In this paper, we describe how a major retailer's recommender system contextualizes the information that is passed to it, to provide real-time in-store recommendations, at a high level. We specifically focus on the data pre-processing ideas that were necessary for the model to learn. The paper describes the ideas and reasoning behind crucial data transformations, and then illustrates a learning model inspired by the work done in Natural Language Processing.

1 INTRODUCTION

IKEA is the world's largest furniture retailer, with a presence in over 45 countries. Every day, thousands of customers roam the stores' exhibitions, considering new purchases for their homes or workplaces. In this work, we present how the recommender system that was developed for the Greek branch of the organization, contextualizes the information that receives as input, and helps the customers discover new, interesting products in real-time. For example, if a customer navigates through the living room section of the exhibition, the system should push recommendations for products that are in close proximity to the customer. Our approach should cover the following challenging requirements:

- **Real-time:** Recommendations should arrive in real-time, in less than 30 seconds, while the ideal goal is under 10 seconds. This requirement derives from the fact that customers that pass a specific location rarely return to pick up a late recommended product. Thus, the time requirement drives us to either scale up complex solutions or turn to more straightforward techniques.
- **Proximity:** IKEA exhibitions define a precise path that every customer follows. For example, the customer passes through the living room sector, then enters the bathroom region, and finally, ends up exploring kitchen products. Thus, customers should be able to quickly match what they already have in their basket with products from the same category.
- **Up-selling:** One of the requirements is to detect similar products to what a customer has already in the basket, and recommend those that, although a bit pricier, present an opportunity due to some in-store special offer, or a stock policy.

Common recommendation systems employ traditional matrix factorization techniques to offer personalized content to the users. In our work, we also consider the context in which a purchase was made, using dynamic and fully observable factors that deployed sensors in-store provide to the system. Our method is based on the work done in Natural Language Processing using deep learning, and it can be thought of both as *contextual pre-filtering* and *contextual modelling* (Adomavicius et al. 2011).

Moreover, in contrast to the much more researched matrix factorization methods [2, 3], especially those that feed on explicit datasets, little work has been done on recommender systems using deep learning. However, we seem to get into a paradigm shift. Neural networks are utilized to recommend news (Oh et al. 2014), citations (Vázquez-Barquero et al. 1992), and review ratings (Tang et al. 2015), while YouTube recently converted its method to follow the deep learning trend in the organization (Covington, Adams, and Sargin 2016). Moreover, the standard technique in recommender systems is collaborative filtering, and it has been recently developed as a deep neural network (Wang,

Poulopoulos, D. and Kalampogia, A.
Using a Skip-gram Architecture for Model Contextualization in CARS.
DOI: 10.5220/0008256304430446
In *Proceedings of the 8th International Conference on Data Science, Technology and Applications (DATA 2019)*, pages 443-446
ISBN: 978-989-758-377-3

Wang, and Yeung 2014) and autoencoders (Sedhain et al. 2015).

Our system is built on TensorFlow (Xu et al. 2016), an open sourced version of Google Brain (Dean and Corrado 2012). Our model learns approximately 3 million parameters and is trained over millions of customer transactions.

This work is concerned with the first part of IKEA's recommender, namely the one that captures the context and generates lists of candidate products. It is structured as follows: We present a brief, bird's eye view of the system in Section 2. Section 3 describes the reasoning and process behind data pre-processing. Finally, Section 4 illustrates the inspiration and architecture of the model.

2 SYSTEM OVERVIEW

Figure 1 depicts a high-level overview of the IKEA recommender system. It consists of two discrete components; the data pre-processing unit and the component responsible for producing recommendations. The customer shopping cart and location are also sketched, as two other sources of input.

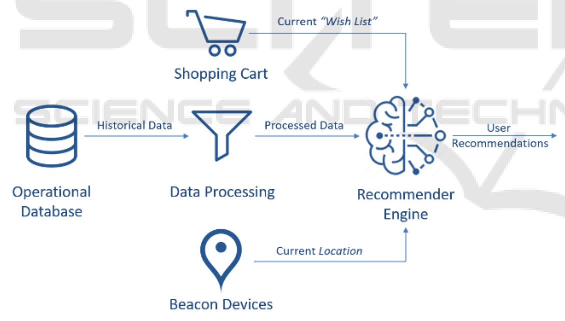

Figure 1: IKEA skip gram context-aware recommender conceptual diagram. Data are retrieved by the operational database and fed to the neural network through the data pre-processing unit. In real-time the customer's location and shopping cart are fed as extra input to the model.

Initially, we feed the customer's historical data, stored in the database, in the pre-processing unit. The unit must shape them in a form that the neural network accepts. Moreover, it logically transforms the data so the network can discover intricate patterns behind customers' purchases and behaviour (more on this in Section 3).

The training of the algorithm is done periodically, offline. In real-time, we feed the customer's location and shopping cart into the model. By the time a customer adds something to the shopping cart, or moves to a new in-store location, the system produces candidate items. The primary job of the algorithm, that is in the scope of this paper, is to create a list of items, that might be of interest to the user and are located nearby. During the next step a ranking algorithm sorts these candidate items to create personalized recommendations. This will be addressed in a following publication.

The candidate generation algorithm is inspired by the work done in language models (Collobert and Weston 2008; Mikolov et al. 2006, 2013),. Its job is to find the correlations between the items that customers choose together, as well as the reasoning behind these purchases (e.g., same color, brand, style, etc.). In Section 4, we present the model architecture and logic in detail. During training, we make use of offline metrics like precision, recall, and ranking loss, but the real value of a recommender is not only to predict the held out data in a test set, but also to discover new items that might be of interest to the customers, even if they were unaware of their existence. Thus, we can only draw a safe conclusion using specifically designed A/B tests in production.

3 DATA PRE-PROCESSING

To take advantage of the work done in language models, and especially the word2vec notion (Goldberg and Levy 2014), we need to transform the products in a way that they can be viewed as words in a document. This transformation would permit us to learn informative distributed vector representations, for each product, in a high dimensional embedding space. To achieve this, we need a data pre-processing component. We cannot convey this as feature engineering because the features that fed into the algorithm are still raw product IDs.

We view the customers' purchases on a specific visit, i.e., on a unique date, as a set of purchased products P, whose elements are taken from a set of items I, which encloses every product. If we assume a set C, that contains every product category, we can filter the set P into distinct product categories, thus creating subsets P_c, such as $P_c \subseteq P$, that consists of what each customer bought on a specific date, grouped by product category. We treat the resulting sequences of transactions, i.e., the elements of P_c, as *"purchase sentences"*, where each product $i \in I$ is a *"word"* composing the *"sentence"*. This way we can compose *"chapters"* for each category, that in the end concatenate into a *"book"* or *"document"* to train our model. Figure 2 depicts the process.

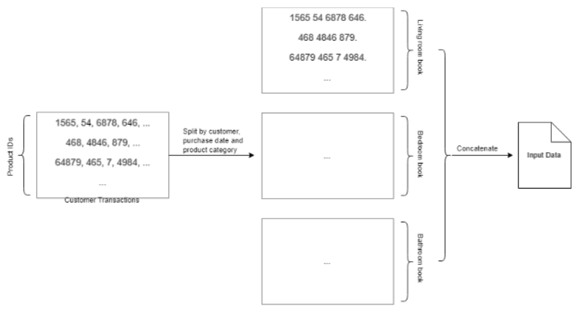

Figure 2: Data pre-processing procedure. Customers' transactions are split on purchase date and product category to form sequences of transactions within a specific product category. These sequences are viewed as "purchase sentences", with which we can create a so-called "purchase book".

For instance, consider a customer who enters the store and purchases products from three different product categories; kitchen accessories, bedroom furniture and children toys. We split the customer's transactions by product category and get three sequences of continuous events, one for each category. If we do this for every customer, on each date, we can create those *"purchase chapters"* we need for each category. Finally, we concatenate those chapters to build the resulting "book" or "document", that serves as the input to our model.

The intuition behind splitting on transaction date, as well as unique customers and product categories, is because the same user can enter the store and purchase a sink and its accessories from the kitchen department in one visit, and kitchen furniture in a second visit. Although the customer makes these transactions within the same product category, we treat them as different events, because furniture products can have different correlations than kitchen hydraulics.

4 MODEL OVERVIEW

Inspired by the work done in Natural Language Processing, and especially the ideas concentrating around the word2vec model architecture, we employ a similar network engineering to learn a distributed representation for each product, in a high dimensional embedding space. The idea is that given the "book" we generated in the data pre-processing phase, we exploit the capabilities of a skip-gram model, trying to predict the items that complete a "purchase sentence", given one item that exists in a random position inside this sequence.

Figure 3 depicts the model architecture. Given an item in position t_0, we try to predict its surrounding items. During this process, we learn a high dimensional vector representation for this item, in the Embedding layer of the network. At serving time, we throw away the prediction layer keeping the learned embeddings. The model generates the candidate items by taking the *top-n* closest items to the product that a customer added to the shopping cart.

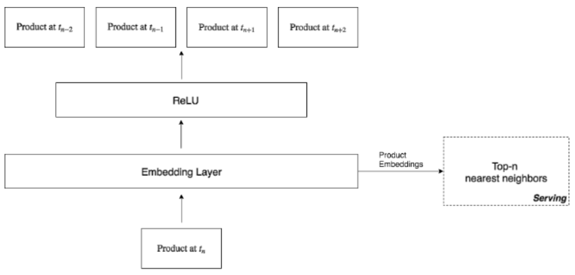

Figure 3: IKEA skip gram recommender model. Inspired by the work done in NLP, we try to predict the items that are in the same context as the item we feed as input to the neural network. At serving time, we throw away the prediction layer keeping the learned item embeddings. The candidate generation problem is then reduced to a simple k-nearest neighbours question.

While experimenting with different variants of the architecture, we found that adding more hidden layers assisted with integrating different item features into the model. Although it did not help much with the model accuracy by itself, it provided a way of passing more item features to the model, such as the item's age, brand, pricing level, etc., which helps to discover better candidates. Other aspects of data pre-processing in language models, such as the notion of stop words and sub-sampling, did not help us in this case, and they were skipped. Moreover, we did not use the idea of windows or padding to consider only a small, random number of surrounding items in a sentence, as we consider all items in the sentence of equal importance.

ACKNOWLEDGEMENTS

This work has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement number 732051, CloudDBAppliance project.

REFERENCES

Adomavicius, Gediminas, Bamshad Mobasher, Francesco Ricci, and Alex Tuzhilin. 2011. “Context-Aware Recommender Systems.” : 67–80.

Collobert, Ronan, and Jason Weston. 2008. “A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning.” http://wordnet.princeton.edu.

Covington, Paul, Jay Adams, and Emre Sargin. 2016. “Deep Neural Networks for YouTube Recommendations.” In *Proceedings of the 10th ACM Conference on Recommender Systems - RecSys ’16*,.

Dean, Jeffrey, and Greg S. Corrado. 2012. “Large Scale Distributed Deep Networks Jeffrey.” *Cambridge University Press*: 1–9.

Goldberg, Yoav, and Omer Levy. 2014. “Word2vec Explained: Deriving Mikolov et Al.’s Negative-Sampling Word-Embedding Method.” (2): 1–5. http://arxiv.org/abs/1402.3722.

Koren, Yehuda, Robert Bell, and Chris Volinsky. 2009. “Matrix Factorization Techniques for Recommender Systems.Pdf.” : 42–49.

Mikolov, Tomas, Kai Chen, Greg Corrado, and Jeffrey Dean. 2006. “Distributed Representations of Words and Phrases and Their Compositionality.” *CrossRef Listing of Deleted DOIs* 1: 1–9.

Mikolov, Tomas, Chen, Kai

Corrado, Greg Dean, Jeffrey, 2013. “Efficient Estimation of Word Representations in Vector Space.” : 1–12. http://arxiv.org/abs/1301.3781.

Oh, Kyo Joong, Won Jo Lee, Chae Gyun Lim, and Ho Jin Choi. 2014. “Personalized News Recommendation Using Classified Keywords to Capture User Preference.” *International Conference on Advanced Communication Technology, ICACT*: 1283–87.

Sedhain, Suvash, Aditya Krishna Menon, Scott Sanner, and Lexing Xie. 2015. “AutoRec: Autoencoders Meet Collaborative Filtering.” *Www-2015*: 111–12.

Su, Xiaoyuan, and Taghi M. Khoshgoftaar. 2009. “A Survey of Collaborative Filtering Techniques.” *Advances in Artificial Intelligence* 2009(Section 3): 1–19.

Tang, Duyu, Bing Qin, Ting Liu, and Yuekui Yang. 2015. “User Modeling with Neural Network for Review Rating Prediction.” *IJCAI International Joint Conference on Artificial Intelligence* 2015-January(Ijcai): 1340–46.

Vázquez-Barquero, J. L. et al. 1992. “A Neural Probabilistic Model for Context Based Citation Recommendation.” *Psychological Medicine* 22(2): 495–502.

Wang, Hao, Naiyan Wang, and Dit-Yan Yeung. 2014. “Collaborative Deep Learning for Recommender Systems.” http://arxiv.org/abs/1409.2944.

Xu, Tianyin et al. 2016. “TensorFlow: A System for Large-Scale Machine Learning.” *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI ’16)*: 619.