**UL FRI Data Science - Introduction to Data Science, 2021-2022**
**Final report**

# Product recommendation system

David Ocepek[1], Klementina Pirc[2] and Matej Miočić[3]

**Abstract**

The goal of our project was to build a product recommendation system that would help the sales personnel at Siemens recommend more relevant products to their customers. We have evaluated 2 models (RankFM and DeepFM) that were trained on information about customer previous purchases and compared the results to the baseline model. We approached the evaluation from different standpoints and the trained model managed to outperform the baseline model in all of them. When evaluating the models based on hit-rate, the top hit@5 rate for RankFM was 0.68 compared to top hit@5 rate for baseline model which was 0.44. This means that the RankFM model correctly predicted at least 1 out of 5 predicted products for almost 25% more users than the baseline model.

**Keywords**
recommendation system, products, RankFM, DeepFM

[1] *do8572@student.uni-lj.si, 63160248*
[2] *kp3437@student.uni-lj.si, 63210492*
[3] *mm9520@student.uni-lj.si, 63180206*

## Introduction

Sales performance is one of the most important aspects of businesses therefore a lot of effort goes into its advancement. Suggesting new popular products or products adjusted to customers' needs contributes to mentioned goals and this is where recommendation systems are utilised. Our goal was to create a recommendation system for Siemens that provided us with information about their customer purchases. For the task we resorted to Factorisation Machines (RankFM) and Deep Factorisation Machines (DeepFM) that are specialised for recommendations and are therefore designed to learn from sparse data which is one of the characteristics of recommendation problems. Below we describe utilised methods and present our results.

## Data analysis

Our data set consists of 1249 consecutive days for which we have information about product sales for each customer. Given attributes are: *day, customer, prod, domestic, state, ind_code, ind_seg_code, new_orders*. *5HB8_4N50_1I6D* is an example of a product label, where *5HB8* is the family, *4N50* the product and *1I6D* product's variation. Same product and variation labels appear in different families, therefore we were attentive to also include family label when carrying out analysis and predictions. Data set contains 2762 customers, 126 families, 2013 products and 12304 variations. Additionally we have some information about each customer: their industry and industry segmentation code, whether they are a domestic customer (from Austria) or not and which state the customer is from. These features are uniquely determined by the customer id, something which is important as it enables us to impute said features in our negative samples. Starting date of the data set is assumed to be sometime in the late spring or early summer. We deducted this from Figure 1 which shows a clear seasonality in our data characterised by sharp periodic drops in sales.

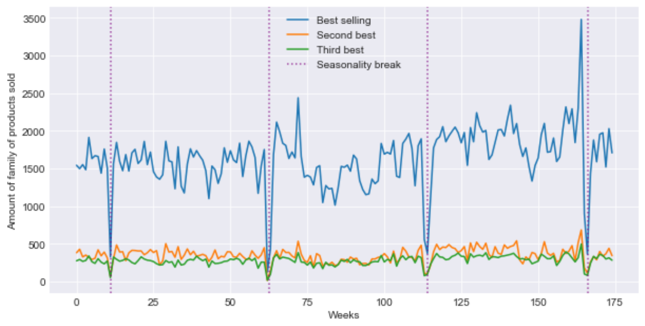

**Figure 1.** Amount of sales for **top 3 families of products** through time. Seasonality of our dataset can be seen in the form of periodic drops and lifts in sales. The figure also shows a clear dominance of the most popular family of products.

In Figure 1 we show a clear dominance in sales of the most popular family of products. We observe that the most frequently bought family of products has approximately three times as much sales as the second most frequently bought family of products, with second and third being similar, leading us to conclude that most families have expected sales similar to the bottom two families.

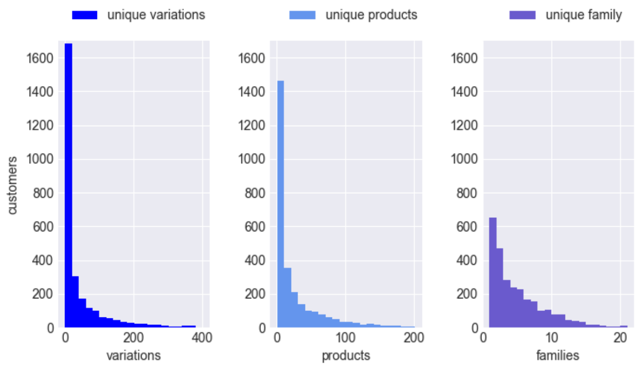

**Figure 2.** Distribution of unique families, products and variations per customer. Most customers only buy a few select product variations.

From Figure 2 we conclude that our prediction rate (even for a constant classifier) should be quite high due to the small number of purchases made by each customer as such most of our classification problem stems not from accurately assessing which items a customer will buy, but instead when a customer will buy a certain product.

## Methods

### Walk-forward cross-validation

We are working with temporal data and the model must not learn from the future data, therefore we decided to use walk-forward cross-validation (WF-CV), which is a technique that splits the data into folds and at the same time preservers the temporal ordering of the data. We decided to use 500 days as train data and 125 days as test data, this allows us to go through all the data in 6 folds (Figure 3).

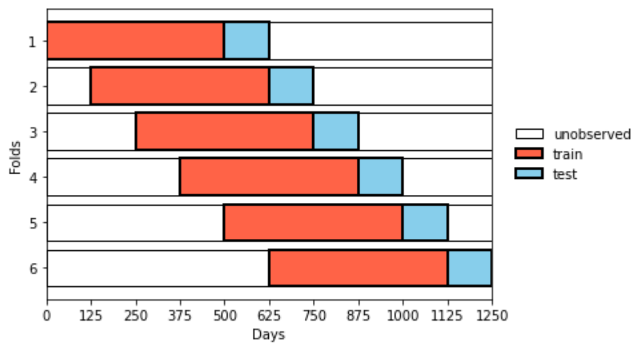

**Figure 3. Walk-forward cross-validation.** All folds have equal amounts of data, the train and the test set include 500 and 125 days respectively.

### Hyperparameter tunning

To determine the optimal values of hyperparameters for our models we applied nested cross validation, with grid search inside each fold. In each fold of nested CV we performed holdout estimation for each parameter in grid, choosing the parameters which minimised our mean squared error (MSE).

### Feature Embedding

Most recommender systems use some form of feature embedding to reduce high-dimensional customer-product feature spaces and extract valuable information from our data. This approach is used by FM, DeepFM or LSTM. All three models learn feature embeddings during training.

### Negative sampling

Examples of when customers didn't buy anything (negative samples) can be hard to acquire because it requires companies to also log customer's enthusiasm for a product which they did not buy. We could omit negative samples altogether, however it has been shown that most models perform better if they have access to data on customer behaviour not just when they make a purchase, but also when they decide not to purchase anything. We used a one-to-one negative sample ratio with uniform distribution across all customer-product pairs.

### The baseline model

The baseline model is important for understanding the true performance of our models. Baseline values for predictions are calculated by picking the most frequent family for each customer, based on their purchases in the train set, then predicting top $k$ variations or products from this family and finally checking if the customer bought any of them in the future (test set). Family prediction is done independent of customers by always using top $k$ families from the train set.

### Factorization Machines

Factorization Machines (FM) are a family of supervised learning algorithms that map arbitrary real-valued features into a low-dimensional latent factor space and can be applied naturally to a wide variety of prediction tasks including regression, classification, and ranking. FMs can estimate model parameters accurately under very sparse data and train with linear complexity, allowing them to scale to very large data sets — these characteristics make FM an ideal fit for real-world recommendation problems. Additionally we can set the complexity of FM, in practice the most commonly used are 1st and 2nd order FM which are also the two options that we evaluated in our work [1].

### RankFM

RankFM[1] is a cython implementation of the FM model adapted for collaborative filtering recommendation/ranking problems with implicit feedback user/item interaction data. For user implicit data we used user's state, where as for item implicit

[1] https://github.com/etlundquist/rankfm

data we used from which family each product is. However we are not able to utilise explicit data such as the exact day and the amount of items purchased.

### DeepFM

Deep Factorisation Machines [2] (DeepFM) are an improvement over FM, combining FM with deep neural networks (DNN), which enables the model not to only predict sparse features but also model higher degree non-linear interactions which FM struggle with as they can usually model only 1st and 2nd order polynomial feature interaction. Another benefit of using DeepFM is the huge expressiveness offered by neural networks. In our paper we used a vanilla deep neural network.

DeepFM and its components (mainly 1st and 2nd order FM) were implemented in Keras. Our personal implementations of these models enabled us to compare our model performances with the performance of popular library models.

### Hit rate

For estimating the performance of our models we used the hit-rate measure. If a customer buys at least one of the $k$ recommended products in the test set it counts as a hit. Finally, we calculated the portion of customers for which we got a hit. For recommending 5 products per user we denote hit rate as top hit@5 rate.

### F1-score

Instead of predicting a fixed number of products for each customer, another approach is to focus only on products whose probability is higher than a given threshold. This functionality was specifically requested by Siemens so the sales person can set a certain threshold and recommend only products that a customer will be very likely be interested in. In this case we estimated the performance of our models with the F1-score as it represents the harmonic mean between precision and recall. To calculate predictions we took 10 recommended products and filtered out items below the threshold, then calculated true positives (**TP**) - correctly recommended items, false positives (**FP**) - incorrectly recommended items and false negatives (**FN**) - items users bought but were not recommended. With obtained results we calculated precision (**Pr**), recall (**Re**), combining them into the **F1-score** (1), as follows

$$F1 = \frac{2 \cdot precision \cdot recall}{precision + recall} = \frac{TP}{TP + \frac{1}{2}(FP + FN)}. \quad (1)$$

To further illustrate why we think F1-score is a better metric for this approach we created a simple example in Figure 4. In this example a user has bought 3 items (A,B,C) in the test set (shown on the left). We recommend 5 items (X,B,C,D,E) and slowly increase the threshold, such that 1 less item is recommended in each row. If the filtered out recommendation was incorrect, precision and with that F1-score increases, meanwhile hit-rate stays the same (first 3 rows). When we reach a threshold, where we filter out correct recommendations, precision and with that F1-score decreases (4th row). Only when we filter out all correct recommendations the F1-score and hit-rate evaluate to 0 (bottom row).

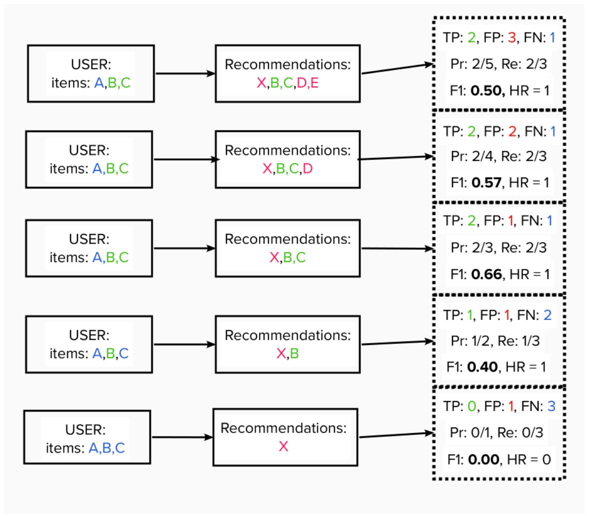

**Figure 4. A visualisation of evaluation metrics** illustrating why we think F1-score is a better metric than hit-rate when recommending based on probabilities, where TPs are labeled in green, FPs in red and FNs in blue. The left column shows the items that each user bought, the middle one recommendations and the right one shows calculated F1-scores and hit rates for different recommendations.

## Results

### Hyperparameter search

A subset of the most important tunned hyperparameter is shown in Table 1. The subset includes the dimension of our embeddings (k), model learning rates (lr), L1, L2 and dropout regularization (dr). All models were tunned using a Grid of parameter and if the optimal parameters were on the border of our search space, we increased its size guaranteeing that we found a minimum.

| Parameter | k | lr | dr | L1 | L2 |
|---|---|---|---|---|---|
| RankFM | 40 | 0.3 | - | - | 0.01 |
| Order 2 FM | 50 | 1e-5 | - | 0.0 | 1e-4 |
| DeepFM | 50 | 1e-3 | 0.3 | 0.0 | 0.0 |

**Table 1.** Optimal parameters for validation.

In Table 1 we can see that embedding dimension (k) for all model is approx. 50 for all models, this indicates that there is a large amount of redundancy between features such as customer id and product id, both of which have over 1000 distinct values. The low embedding dimension also leads us to assume that clusters should be present and clustering should be possible, since embedding dimension of above 100 are common for many machine learning task such as NLP. Additionally, it appears that all 3 models require a fair bit of regularization. This can be achieved by using a large L2 regularization constant

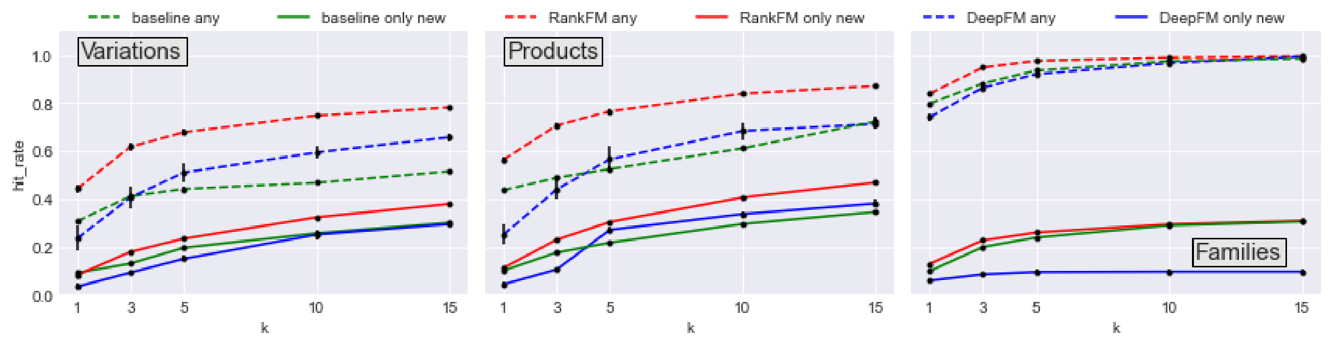

**Figure 5. Hit-rate and standard error** for baseline, RankFM and DeepFM for different number (k) of **any** (dashed line) and **only new** (full line) recommended **variations, products and families** separately.

(RankFM), a large dropout rate (DeepFM) or a small learning rate (2nd order FM and DeepFM). Finally, L1 regularization term was zero for both 2nd order FM and DeepFM is zero. Considering L1 regularization performs feature selection, this would indicate that our dataset has no redundant features.

## Hit-rate

In Figure 5 we show how our models perform compared to the baseline for different number of recommended items. We show hit-rates for both predicting any items and new, not yet purchased items by the individual customer. We split the plots into predicting **variations**, **products** and **families**. The more items we recommend the better the hit-rate. Since there are more different variations and products than families and customers are more likely to buy from the same family, hit-rate is the highest when predicting families and the lowest when predicting new families. We observe that RankFM outperforms the baseline and DeepFM in all possible combinations. DeepFM outperforms baseline only when recommending more than 5 any items and struggles to perform when recommending only new items.

## F1-score

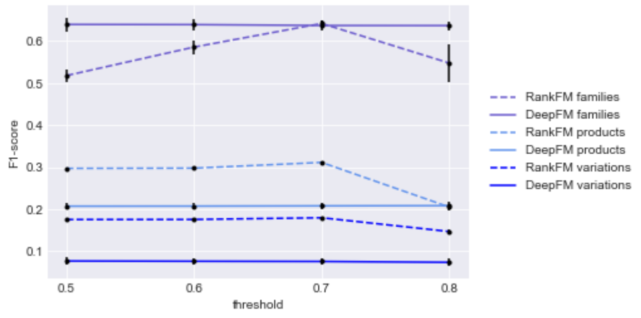

**Figure 6. F1-score and standard error** for **RankFM** (dashed lines) and **DeepFM** (full lines) for recommending variations, products and families for different thresholds. Maximum number of recommended items was capped at 10.

In Figure 6 we show how F1-score changes based on chosen threshold for RankFM and DeepFM models. The F1-score improves only for RankFM family, while appearing constant or decreasing for all other models. More in-dept analysis showed F1-score was slightly increasing elsewhere as well, but is not noticeable on the plot. After we reach a threshold where we filter out correctly predicted items we observe a decrease in F1-score.

## Conclusion

Our results have shown that due to customer tendencies to buy the same products, all three models predicting families give good results including the baseline model. Nevertheless, all algorithms improve upon the baseline, with RankFM achieving best performance regardless of number of recommended items. We observe a hit rate increase with number of predicted items for all models. The most significant improvement of the baseline model was accomplished by RankFM for variation and product prediction with hit@5 and hit@10.

Thresholding was analysed using F1-score to take into account both precision and recall. Thresholding improved F1-score only for RankFM family while increases were not substantial for other models. This means that thresholding cannot be used to improve F1-score for product and variation.

Hyperparameter search was shown to improve model performance by a couple of percent for all models, indicating that GridSearch or some other more sophisticated method such as Bayesian Optimisation should be used since even a small increase in hit rate can result in larger amounts of sold products and therefore higher revenue over time.

We conclude that RankFM is the best performing model, while both RankFM and DeepFM perform better than the baseline indicating that the difference in performance is truly due to some optimisations in the RankFM library, rather than errors in our code.

## References

[1] Steffen Rendle. Factorization machines. In *2010 IEEE International Conference on Data Mining*, pages 995–1000, 2010.

[2] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. Deepfm: A factorization-machine based neural network for ctr prediction, 2017.