ERASMUS UNIVERSITY ROTTERDAM

Erasmus School of Economics

Master Thesis Econometrics and Management Science

Business-wise interests added on customer-based recommendations

Wietze Janssen (455018)

January 31, 2021

Supervisor: Bouman, P Second assessor:

Abstract

The increase in e-commerce has created an enormous amount of data. With these data, companies know more about their customers than ever before. So, companies can take their advantage due to all this information. In this paper, we are using the customer information to create a model that can provide recommendations of items. To create these recommendations we use a neural network with multiple layers. The most important layer for creating the recommendations is the Gated Recurrent Unit. In this layer, the model is able to recognize connections in the training data and apply this information to new data. After we created this model, we are including business-wise interests to the recommendations. This is done by a re-ranking algorithm with different kind of restrictions. The goal of this paper is to include business-wise interests into the customer-based recommendations and see what the effects of these additional restrictions are. Results show, that the neural network without additional interests of the company gives the best recommendations for the customers. However, adding certain restrictions to the re-ranking problem can provide very useful options for the company. These options can lead to a higher customer/retailer satisfaction and higher profits.

Contents

1 Introduction

In the last decade, e-commerce has evolved a lot. Increasingly more brands and companies have started to sell their items online and try to earn their spot in the e-commerce market. One of the problems, the companies in the e-commerce market are dealing with, is that they have an enormous amount of items they are offering. The customer visiting the website cannot go through all the items because this would simply take too much time. So, the goal for the companies is to first show the items that the customers are more likely to buy. This can be different for every single customer because no one has the same taste. If this is done correctly it increases customer satisfaction because customers can find the items they were looking for a lot faster. It is also positive for the company itself because the sales increases and the references of the company are more positive. Showing the items the customer is interested in can be called in different ways. For instance, "Items recommended for you", "Similar items" or "Others also viewed". These sections are called recommenders, the goal of these recommenders is to show items the customer is most likely to buy. Recommendation algorithms are best known to solve this problem, where the interest of customers is used as input and a list of recommendations is created as output. The interest of customers can be obtained by surveys and reviews, but also by click history on the site, items put in a wish list and previously made purchases. A lot of research has been done for the problem where only user interest is used as input. For this problem, recurrent neural networks and collaborative filtering are tools that provide very accurate recommendations. We will be doing literature research with a combination of testing and fine tuning, to find the best way to create a recommendation model for our data set. We are also looking to use more input to add on the recommendation model build based on the literature research. This input is from the companies preferences which items they want to show first. We are especially interested in the effects of including business-wise requirements into the recommendations. Are these changes smart adjustments to contribute to the goals and ambitions of the company? What are the effects of the requirements by itself? Are there any combined effects of the requirements? To know what the requirements of the company are we need to know what kind of company we are dealing with. With these requirements, the company might be able to personalize the recommendations in a way they think it improves customer satisfaction and results in increasing their market share. In this report we have seven chapters, in Chapter 2 we give an overview of already existing literature about our problem. Thereafter, in Chapter 3 we provide a clear overview of the problem we are dealing with. In Chapter 4 the methods we have used to find

our results can be found. In Chapter 5 we show the results of these methods. In Chapter 6 the conclusions of the method and results are stated. Lastly, in Chapter 7 the methods and results used are discussed.

2 Literature review

To solve the recommendation problem, one of the tools being used are neural networks. The general structure of a neural network consists of input nodes, output nodes and internal processing. The internal processing can be built by different kind of layers and activation functions. Some traditional applications of the neural network are classification, noise reduction and prediction (Masters, 1993). Especially Long Short-Term Memory (LSTM) networks have been shown to work particularly well for recommendation problems. These networks include additional gates that regulate when and how much to take the input into account and when to reset the hidden state. This helps with the vanishing gradient problem that often plagues the standard Recurrent Neural Network (RNN) models. A slightly simplified version of LSTM, that still maintains all their properties, are Gated Recurrent Units (GRUs) (Quadrana et al., 2017). Quadrana et al. (2017) use a session-based RNN. Their RNN is based on a single GRU layer that models the interactions of the user within a session. A session is stated here as a group of interactions that take place within a given time frame. The RNN takes the current item ID in the session as input and a score for each item representing the likelihood of being the next item in the session as output. Quadrana et al. (2017) created a personalized session-based Hierarchical Recurrent Neural Network (HRNN). They have built this HRNN from adding a GRU layer to the RNN they already had, and by using a so-called powerful user-parallel mini-batch mechanism for efficient training. They have tested the different neural networks on two real-world data sets, and the HRNN model significantly outperforms the other RNNs and other basic personalization strategies for a session-based recommendation. Donkers et al. (2017) also use a neural network. They created a Sequential User-based Recurrent Neural network where they use GRU layers as well. They compare a Linear User-based GRU, a Rectified Linear User-based GRU and an Attentional User-based GRU In the way they formulate their RNNs, they are able to model user behaviour and to capture dependencies between consumption events more adequately. Linden et al. (2003) use Item-to-item Collaborative Filtering to solve the problem we are facing using only user interest as input. The idea of this method is that they match items from the users' interest with similar items, then combine these similar items into a recommendation list. The similarity is measured by the cosine of the angle of two different items (vectors).

The papers discussed before only use user interest as input, while we are taking more business-related factors into account like profit and stock. However, Jambor and Wang (2010) use Collaborative Filtering for optimizing multiple objectives, where they focus on availability, profitability

and usefulness of an item. They start with a recommendation without the additional objectives using Collaborative Filtering, and after that, they are trying to re-rank the items keeping in mind for example availability. This is done by including the probability of an item being chosen by the customer. With this probability, they add a constraint and after solving the problem with this extra constraint the items are being re-ranked. Jannach and Adomavicius (2017) discuss multiple methods to solve the recommendation problem including profit as an extra factor. They used a matrix factorization algorithm to get an initial ranking of the items. Then they greedily re-rank the items for each user with a method provided by Adomavicius and Kwon (2011) according to the profitability of the items. Optimizing a problem with for each item stated the profitability, availability and other item-specific characteristics can be written in an integer linear programming problem. These kinds of problems can be transformed into network flow problems to increase optimization speed. In our case, we are facing a min-cost-max-flow network problem. Ahuja and Orlin (1995) came up with a method called capacity scaling which can solve the problems where the cost of flow on each arc is a linear or convex function of the amount of flow. Hagberg et al. (2008) also provides a method called capacity scaling to solve min-cost-max-flow network problems.

If we want to provide one of these algorithms with our data we need to solve an interesting problem as well. In our data set, we are dealing with around 50,000 different items that can be shown on the website. This is leading to a high dimension of our input data. We need to reduce the dimension of the data set to be able to use the data. This can be achieved in different ways. Zhou et al. (2018) are solving a recommendation problem with the use of a neural network with different kinds of layers. One of the layers they use is a recurrent layer where they have two different options: the GRU and the LSTM layer, which is similar to other papers mentioned before. To solve the high dimensional problem they use an embedding layer. These embedding layers are mainly used for Natural Language Processing. In these problems, they have a fixed vocabulary and every word is connected to an integer. This integer then is transformed into a vector of the required dimension length. However, Zhou et al. (2018) solve a problem, which is similar as our problem, where they are dealing with items and activities instead of words, so each item or activity gets its own integer value which in the end are transformed in a vector of the required dimension. Wang and Zhu (2017) look at the more familiar dimension reduction techniques where they look further into Principal Component Analysis (PCA) and Singular Value Decomposition (SVD). Both of these methods are based on eigenvalues and eigenvectors. These methods are performed before the neural network unlike the embedding layer described before, which is included in a neural network layer. We would

start with a high dimensional data going into the PCA or SVD algorithm, which transforms the data into input data for the neural network of the required dimension.

3 Problem Description

First, we are looking at the problem of recommending items based on user interest. We are recommending N different items, the goal is that these N items have the highest chance for a certain customer to be clicked on. For this problem, we need certain data that allows us to use customer interest in our algorithm. The customer data we are using comes from an online store that sells luxury fashion. We base user interest on the historic activity of the customers on the website. The user activity data, we need and is available for our research is the activity of clicking on items, putting items in a shopping basket, putting items on a wishlist and wanting a restock notification for items. All these activities are stored in set A. These activities are all linked to a user with a user ID, the time the activity took place and what item was involved by the action. We are dealing with around 50,000 different items stored in set I, and around 250,000 different customers stored in set U. To solve the problem where we are providing recommendations based on user interests we are using a neural network. For the neural network, we use an input vector of the activity data of a customer over all the different items. So, the input of the model is the data of user $u \in U$, where we look at the n latest activities before a certain output activity (the last click-on activity). We define the data as the sequence $S_u = [e_1, e_2, ..., e_n]$, where each $e_t = (p_l, a_k)$ is formed out of item $p_l \in I$ and activity $a_l \in A$. The output data we use to train the neural network with, is a vector with a value of zero or one for every possible item that can be clicked on. For every customer, we look at the last item that is clicked on. This item has a value of one in the vector all the other items have a value of zero. So, the goal of the algorithm, in our case the neural network, is to predict the next item to be clicked on. This means predicting the item that will have a value of one.

Because the length of the output vector is very large and the output vector has only one value being non-zero, we are facing two problems with our data set. First, we have a sparsity of the output vector very close to one, which can lead to long running times. And secondly, we have high-dimensional data set for our algorithm. This results in a large memory and long running time as well. For the sparsity problem, we can store this data in a so-called sparse-matrix. The sparse-matrix can be stored very efficiently and the neural network can use the sparse-matrix very efficiently as well. To solve the high-dimensional problem we started with different general dimension reduction techniques like Principal Component Analysis and Singular Value Decomposition for sparse and normal matrices. Those techniques did not give the required results for our data set and the algorithms were pretty time-consuming. Zhou et al. (2018) used a different kind of technique to

solve this problem, which is called an Embedding Layer. This is a certain layer in our neural network that transforms our input into a smaller dimension. How the method is adjusted in our case is explained further in the next section.

For the second problem, we are also taking business-wise requirements into account while providing recommendations for the customers. These requirements depend on the company, the e-commerce company of the data we are using sells items provided by different retailers. Multiple retailers may sell the same item. The retailer who offers the item for the lowest price is chosen. So, including requirements from the business point of view, we focus on items with more profit, items with a higher stock and items from certain retailers. We chose the first requirement to focus on items with a higher profit because if based on user interest it might be possible that only items with low profit are recommended and this can lead to less total profit in the end. Keeping the stock level of the items into account is important. We do not want to recommend items with low stock level and disappoint more customers if they want to buy an item, while it is out of stock. For the last requirement, we want to focus on balancing the number of shown items from all retailers instead of recommending all items of one retailer. In this way, retailers have a more equal chance of selling their items. For this problem, we have two sets I and J. I is the set of items and J is the set of retailers. Now we have the likeliness for each item $i \in I$ to be clicked on next l_i, obtained from the neural network. However, now we want to include the multiple aspects, profit per item $i \in I$ p_i, the total stock value per item $i \in I$ S_i and if item $i \in I$ is offered by company $j \in J$ (o_{ij}). The overall goal of this problem is again to maximize the chances of the N recommended items for a certain customer to be clicked on. But now these N items need to meet the additional requirements. If profit is included, the chances are combined with profit, we maximize the expected profit of the N items.

4 Methodology

4.1 Neural Network

For our recommendation algorithm, we are using a neural network with different layers. In the network we have to deal with several problems. First, for the input of our neural network, we use a sequence of the n latest activities for a customer $u \in U$. The input is very sparse and high-dimensional. Besides, the activities have different importances, so we need to make sure all the activities can add their impact to the model. To make sure we tackle all these problems, we created a network consisting five different layers: an input layer, an embedding layer for the sparse and high-dimensional input, a GRU layer, an attention layer to capture the different effects of the activities and an output layer. The architecture of our network is shown in Figure 4.1, here we can briefly see how the layers are used. In the following subsections, we describe each layer. The structure and layers we use are based on Zhou et al. (2018).

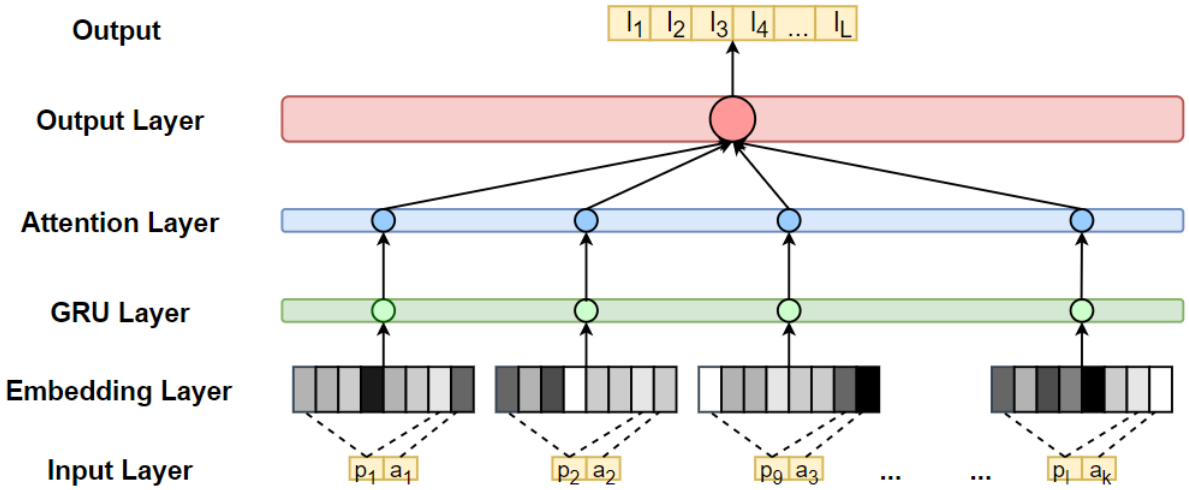


Figure 4.1: Architecture of the proposed neural network, with as output the chance for the items to be clicked on (l_i). This is firstly introduced at the re-ranking problem.

4.1.1 Input Layer and Embedding Layer

The input of the model we described in the data section: for a certain customer u we have a sequence $S_u = [e_1, e_2, ..., e_n]$, with $e_t = (p_l, a_k)$ is formed out of item $p_l \in I$ and activity $a_k \in A$. In set I we have L different items and in set A we have K different activities. So, we can make $L \cdot K$ different kinds of combinations. We have two different embedding layers. One layer is to reduce the dimension of the items and the other one is to reduce the dimension of the activities. We need to split the input into two parts for each e_t, to have input for both embedding layers. So, the input for every embedding layer is a unique integer value for an item between one and L, and for an activity between one and K. Those integer values will be transformed in a vector of the new dimension length with values between zero and one. At the end of the two embedding layers, we combine the output into

$$x_t = concatenate(W_p \cdot p_l, W_A \cdot a_k), \tag{1}$$

where $W_p \in \mathbb{R}^{d_p \cdot L}$ and $W_A \in \mathbb{R}^{d_A \cdot K}$, where $d_p << L$ and $d_A << K$ are the new dimensions for items and activities respectively. The weights W_p and W_A are trained by the network. The dimension of x_t is $d_p + d_A$ which is much smaller than $L \cdot K$.

4.1.2 GRU Layer

The goal of the current x_t we obtained from the embedding layers is to predict the next clicked item by valuing all the possible items. With this method we try to search for useful information from the input, so it is able to model the behaviour from the previous activities.

In order to do this we use a Gated Recurrent Unit (GRU). The GRU is activated by linear interpolation between the previous hidden state h_{t-1} and the candidate hidden state $\hat{h_t}$:

$$h_t = (1 - z_t)h_{t-1} + z_t\hat{h_t}, \tag{2}$$

where the update gate z_t is obtained as follows

$$z_t = \sigma(W_z x_t + U_z h_{t-1}) \tag{3}$$

and x_t is the item embedding x_t, and W_z and U_z are weight matrices. The candidate activation

function $\hat{h_t}$ is determined by:

$$\hat{h_t} = tanh(Wx_t + U(r_t \cdot h_{t-1})), \tag{4}$$

where the reset gate r_t is computed as

$$r_t = \sigma(W_r x_t + U_r h_{t-1}), \tag{5}$$

where W, U, W_r and U_r are weight matrices. In the end, the current hidden state h_t gives us a value for all the items. The general architecture of a GRU is shown in Figure 4.2. In the GRU we have a reset and an update gate. The reset gate is derived from the previous hidden state and the current input. Both of these get multiplied by a weight matrix before it goes into the σ function. The σ function is a nonlinear activation function, for example, ReLU, sigmoid or tanh which transforms the values to fall between zero and one to make sure we can filter the less important out of the more important information. The weight matrices get updated after the network is trained so the useful features are learned by the GRU. The update gate is computed the same way as the reset gate but the difference is the weight matrices. So, the final vectors from the two gates are different. The goal of the update gate is to determine how much information about the previous hidden state is used for the current hidden state. The way the GRU operates allows the model to retain long-term dependencies. A main problem RNN's are facing, but a GRU does not have to deal with, is the exploding/vanishing gradient problem. This problem happens during the training of the model when the error gradients are calculated, and during back-propagation, these gradients then undergo a lot of matrix multiplications and either vanish or explode exponentially for long sequences. The GRU solves this problem by keeping most of the existing hidden state and adding the new content on top of it, instead of replacing the entire content of the hidden state at every step. By this way, the error gradients can be back-propagated normally.

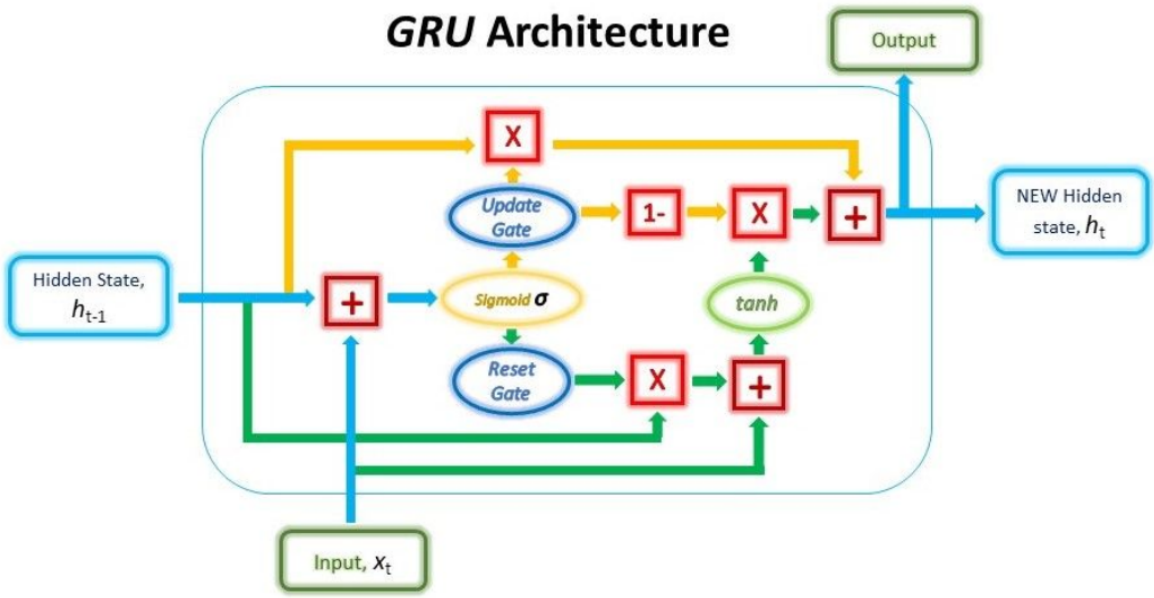


Figure 4.2: Inner workings of GRU cell. How to create the new hidden state h_t with the previous hidden state h_{t-1} and the current input x_t. With the clear split between the update gate and the reset gate. In this figure tanh is the activation function. Adapted source: Loye (2019)

During the training, the scores of h_t are compared to the next clicked item in the data set to compute the loss and update the weight matrices. We are estimating probabilities which means we are using a probabilistic loss function like cross-entropy, binary cross-entropy, categorical cross-entropy and Poisson. After trying these loss functions out on our data set we found that categorical cross-entropy gave the best results and fits our data the best. The categorical cross-entropy function looks as follows

$$H(p(x), q(x)) = -\sum_{x} p(x) \cdot \log(q(x)), \tag{6}$$

where $p(x)$ is a distribution of the real outcome where the clicked item has a value of one and $q(x)$ a distribution of the expected outcome from the data. This function can measure the dissimilarity between $p(x)$ and $q(x)$.

4.1.3 Attention and Output Layer

To capture the impacts of the varied effects of the different activities we use an attention layer that assigns proper weight on each hidden unit. It helps to get a more balanced output. We based our

attention layer on the attention layer of Zhou et al. (2018). With trying similar functions to find what fits best for our data we find the combination of the following functions for our attention layer

$$M_t = \tanh(W_m h_t + b_m), M_t \in R^{GRU} \tag{7}$$

$$c_u = \text{concatenate}(M_1, M_2, ..., M_n), c_u \in R^{GRU \cdot n} \tag{8}$$

$$\text{output} = \text{softmax}(W_a c_u + b_a). \tag{9}$$

First, we use the output of our GRU layer to transform it into a real-valued score with a tanh activation layer. Then we combine all the n activities into c_u with a concatenate merging layer. Lastly, we use a softmax activated dense layer to get an output for every possible item between zero and one. If the final value is closer to one the chance of being clicked on is higher. The softmax function for the example vector z with K components, re-scales each value of z between the interval (0,1) and all the values together add up to one. The new values of vector z after the softmax function look as follows:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \text{ for i = 1, ..., K} \tag{10}$$

4.2 Re-ranking problem

To start with the re-ranking algorithm, we use the output of the neural network. The output we are looking at is the chances for all the possible items to be clicked on (l_i). We sort these chances from highest to lowest and we use the n highest items as input for our re-ranking algorithm. In the re-ranking problem, we are taking three requirements into account: profit, stock-value and items offered per retailer. For the profit requirement, we focus on the profit per item. In our situation, this is simplified to the profit being equal to the price because of complex aspects being involved by calculating the profit per item. Also, the profit of the company we are looking into is mainly based on the price. If the company sells an item of a higher price the profit also is higher however, there are also some small other aspects which affect the profit that we ignore in our research. The stock-value requirement is met if the average stock-value over all the sizes of an item is equal or larger than 0.8. Lastly, when we look at the items offered per retailer we set a maximum on the number of items that can be linked to a single retailer. The performance measure we are looking at for this restriction is the total number of different retailers offering the N recommended items. If a single item can be offered by five different retailers, then we already have a value of five. The

next item can be offered by three new retailers and two retailers that also offered the first item, we have a total value of eight. In the re-ranking problem, only one retailer is linked to an item. So, if the maximum number of items a retailer can offer is two, it is possible that there is a retailer that can offer three items of the first N recommended items. Now, this retailer can only be linked to two items. However, if another retailer can offer the third item as well this item can still be offered. In this case, the value of the performance measure does not change by using this restriction. But, if the third item can only be offered by the retailer that already offered the two other items. A new item has to be chosen, which can change the value of the performance measure. We start off with writing the problem down in a integer linear programming problem. To solve the problem we transform it into a network-flow problem. Lastly, we show the method we used to solve the network-flow problem.

4.2.1 Integer linear programming problem

Here we have the two sets I and J. I is the set of items and J is the set of retailers. Now we have the likeliness for each item $i \in I$ to be clicked on next l_i. However, in this part we want to include multiple aspects like the profit per item $i \in I$ p_i, the total stock value per item $i \in I$ S_i and if item $i \in I$ is offered by company $j \in J$ (o_{ij}). The problem we are facing now is that we want to maximize the profit and keep the stock value of the items in mind and from which retailer the items come from. We can formulate this problem as a integer linear programming problem. First, we start with the expected value of the profit of item $i \in I$

$$E(p_i) = l_i p_i. \tag{11}$$

We have also introduced the parameter S_i, which is one if the stock value of item i is meeting the stock requirements and zero otherwise. We also use o_{ij}, which is one if item i is offered by retailer j and zero otherwise. Now we want to solve the following integer linear programming problem

$$\max \sum_{\forall i \in I} E(p_i) S_i x_i \tag{12}$$

$$\text{s.t.} \sum_{\forall i \in I} x_i \leq N \tag{13}$$

$$\sum_{\forall i \in I} o_{ij} x_i \leq M \qquad \forall j \in J \tag{14}$$

$$x_i \in \{0, 1\} \qquad \forall i \in I, \tag{15}$$

where the decision variable x_i is one if item i is chosen to be shown and zero otherwise. In equation (12) we have our objective function where we maximize the total expected profit, where an item only can be chosen if the stock value is high enough. Equation (13) makes sure there are only N items selected, and equation (14) allows a maximum of M items per retailer to be selected.

If we want to exclude the requirements from our re-ranking algorithm it is done in different ways. For the profit requirement we replace $E(p_i)$ with the likelihood of being clicked on l_i. Now we do not maximize the profit but the chance of the items being clicked on. If we want to exclude the stock requirement we set all values of S_i equal to one. In this way there is now separation based on stock value between different items. Lastly looking into the number of items offered per retailer requirement. We set M equal to N because now one retailer is able to offer all the recommended items. The retailers have no restrictions based on the number of items to offer. For the upcoming min-cost-max-flow problem we deal with exlcuding the requirements in the same way.

4.2.2 Min-cost-max-flow problem

The integer linear programming problem we are facing for the re-ranking problem can be written as a min-cost-max flow problem. The graph look as follows.

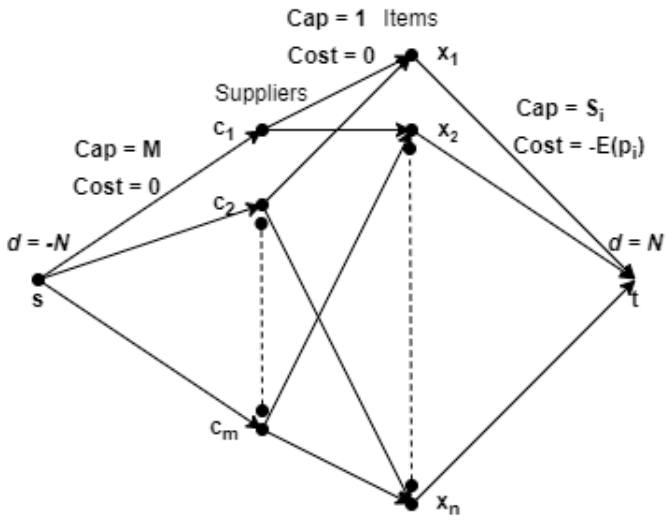


Figure 4.3: Example min-cost-max-flow network for all suppliers and items, with all the items and suppliers on the dots of the dotted lines

We call the graph we are using $G = (V, E)$, where V is the set of nodes and E is the set of edges. In V we have a starting node s and ending node t, with accordingly demands $-N$ and N which is the number of items we want to recommend. The other nodes are all the suppliers and the first n items we obtain from the recommendation algorithm, all these nodes have a demand of zero. In E we have an edge from the starting node to every supplier, with a capacity of M and cost of zero. We use a capacity of M to make sure that a supplier can only have a maximum of M items that are recommended. Then we have an edge from company j to item i if company j offers item i ($o_{ij} = 1$). These edges have a capacity of one and a cost of zero. The last edges are from each item to the ending node, these edges have a capacity of S_i, where S_i is one if the stock value of item i is meeting the requirements and zero otherwise, but the cost is equal to $-E(p_i)$. The costs are the negative of the expected profit, we are now minimizing the costs which later can be transformed into maximizing the expected profit. For all the edges lets call u the capacities and c the costs. So the flow of $f(v, w)$ units on edge (v, w) between node v and w has costs $c(v, w)$ and capacity $u(v, w)$. For every node we have a demand d, so the demand of node v is $d(v)$, which is negative if the node wants to send and positive if the node receives.

The solution of this problem is obtained by finding the routes that minimizes the costs and maximizes the flow of the graph in Figure 4.3. The mathematical notation of this problem is as

follows.

$$\min \sum_{(v,w)\in E} c(v,w)f(v,w) \tag{16}$$

$$\text{s.t.} f(v,w) \leq u(v,w) \qquad \forall (v,w) \in E \tag{17}$$

$$\sum_{w\in V} f(v,w) - \sum_{w\in V} f(w,v) = b(v) \qquad \forall v \in V \tag{18}$$

$$f(v,w) \geq 0 \qquad \forall (v,w) \in E \tag{19}$$

Equation (16) minimizes the costs of the network. Equation (17) makes sure the flow does not exceed the capacity of all arcs. Equation (18) is for every node being satisfied for the corresponding demand. Lastly, equation (19) let the flow be non-negative. When we have solved this min-cost-max-flow problem, the solution we obtained for $f(v,w)$ for the items to the ending node, is the same as x_i for the integer linear programming problem explained in the section above. We can show that these results are the same by first of all assume that $f(v,w)$ is an integer because all the demands and capacities are integer as well. By all the capacities, demands and costs we can also regulate the restrictions stated in the integer linear programming problem. Restriction (13) is retained by the demands of the starting and end nodes. Restriction (14) by the capacity of the edges from the starting node to the supplier nodes, and because there is only an edge between a supplier and an item if the supplier can offer the item. Lastly the binary restriction (15) is held by setting the capacity of the edges from the items to the ending node as S_i which either is one or zero. In the end, we need to be aware that we are minimizing in the min-cost-max-flow problem and maximizing in the integer linear programming problem. This means that the if we multiply the total minimum cost of the min-cost-max-flow problem with -1 we obtain the total maximum expected profit of the integer linear programming problem.

4.2.3 Capacity scaling

To solve the min-cost-max-flow problem we use an algorithm called capacity scaling provided by Hagberg et al. (2008). The algorithm we used is stated by NetworkX (2020). This is a website with the exact code of the capacity scaling algorithm, here you can find a very detailed version of the algorithm. To give an idea on how capacity scaling works we show a general form of capacity scaling for a constrained maximum flow problem provided by Ahuja and Orlin (1995). The idea of capacity scaling is in each step to find the path with the highest bottleneck which increases the

flow as much as possible. This is achieved by limiting our scope to a residual graph $G_f(\Delta)$ that only contains edges with a capacity that is at least Δ. The pseudocode of the algorithm for graph G with starting node s, ending node t and capacities c looks as follows.

Algorithm 1 Capacity scaling maxflow(G, s, t, c)

```
for e ∈ E do f_e = 0
Δ = 2^⌈log2(C)⌉
while Δ ≥ 1 do
    G_f(Δ) = Δ residual graph
    while there is augmenting path P in G_f(Δ) do
        f = augment(f, c, P)
        update(G_f(Δ))
    Δ = Δ/2
return f
```

To solve the algorithm we need to assume that all capacities are integers between one and C. The algorithm gives a maximum flow because by integrality we have $G_f(1) = G_f$. Therefore, there are no remaining augmenting paths after the last step. So, this means we have found a maximum flow.

4.3 Performance measures

To see what the effects of using a re-ranking algorithm are, we need to have performance measures to compare the different models. The performance measures are calculated as the average over the total customers used in the test data. The first performance measure is, the percentage of the times the target item is included in the top-N recommended items. So, if for five out of ten customers, the target item is included in the top-N recommended items, the value of the performance measure is 50%. This performance measure is relevant for every single situation because with this result we are able to see how well the method is predicting the next step. Next, we will look at the average chance of the top-N recommended items to be clicked on. This performance measure and the following performance measures are calculated by first taking the average of the top-N recommended items for a single customer, then taking the overall average of the total customers. This performance measure will most likely be in line with the previous performance measure. The average chance of the top-N recommended items is interesting to look at in combination with the following performance measure, the average expected profit of the top-N recommended items. Especially, when we use profit in our re-ranking algorithm. The goal is to improve the average expected profit, but what are the effects

on the average chance of the recommended items. When we are taking the stock restriction into account, we will be looking at the percentage of the top-N recommended items that are meeting this stock restriction. Using this stock restriction can result in some target items not satisfying the requirements. If this happens, it will automatically lead to a lower value for the percentage of the target item being included in the recommended items. So, it is interesting to also just look at the target items satisfying the requirements and see what the results are. Lastly, we have the performance measure, the number of total retailers offering the top-N recommended items. This performance measure is used to see what the effects of using the retailer restriction are.

4.4 Implementation

To be able to apply the methods we use different kinds of software and tools. The programming language we used is Python, to be precise we are using version 3.8. The data we use is stored in a Google data warehouse called, Google Cloud BigQuery. With the Google package in Python, we use a direct communicator from Python to the Google Cloud BigQuery. For this communicator, we use SQL to obtain and create the data files we need. These files are formed from different data sets stored in the Google data warehouse. Now we have the data we need ready. However, this cannot be directly used as input for our methods. We use the Python packages Numpy, Pandas and Scipy to transform the data in the correct format. Now we have the training input and output data for our neural network. We create the structure of our neural network by using the machine learning package Keras. Adding and connecting different kinds of layers. To be able to use the neural network, we run it in a Tensorflow environment. After training and using the neural network, we obtained the chances for all the items to be clicked on. This is part of the input data for our re-ranking algorithm. For our re-ranking algorithm, we are solving a network-flow problem. To create the network and find the optimal solution, we are using the Networkx package. This package allows us to create all the nodes and edges, with the corresponding costs, capacities and demands. It also has built-in methods to solve these problems quickly.

For this research, the use of hardware can lead to different performing results. Here we are looking at the running speed of the programs. The memory that can be used during the programs, which can lead to different sizes of training sets. We used a Laptop with an Intel Core i5 processor with four cores. The laptop has a DDR4 memory type with a maximum of internal memory of 8 GB. With this hardware, we are able to train the neural network for 200,000 customers in around eight hours. To obtain experimental results of our re-ranking algorithm for 25,000 customers, it takes

around three hours. To get a recommendation for a single customer of just the neural network, it is done under half a second. Using the re-ranking algorithm for making a recommendation for one customer, it takes around one second. With these running times, it is able to use the recommendations in a live environment. However, training the network takes some time and has to be done at strategic timestamps to have the most recent/accurate model, without having to train the model constantly.

5 Results

For the results, we are mainly focusing on the impact of our re-ranking method on top of the recommendations of our neural network. We will look at different variants of the re-ranking method where we include and exclude the profit per item, the total stock value per item and by which retailers the items are offered. By this way, we can see all the effects of the different and combined aspects of the re-ranking algorithm. The results we are focusing on in our comparison is the percentage of times that the real item that is clicked on (target item) is included in the top-N recommended items of our recommendation algorithms, the average expected profit of the top-N recommended items, the number of different retailers included in the top-N recommended items and the percentage of the top-N recommended items that satisfy the stock value requirements. In the evaluation of our results, we are looking at three different top-N values which are 5, 10 and 20. For the re-ranking algorithm, we use the first n items with the highest chances to be clicked on (l_i) from the output of the neural network. In Figure 5.1 we zoomed in on the top two hundred chances of 20,000 customers. We see that after the first fifty items there are not any significant items left for any customer. To make sure we can find feasible solutions for every customer we use the first 150 items as input for our re-ranking algorithm.

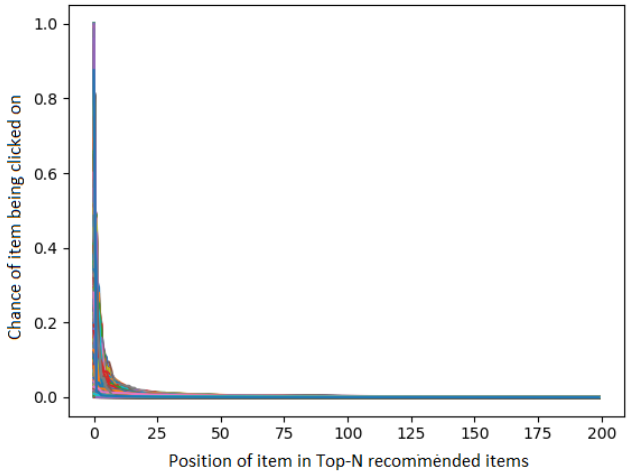


Figure 5.1: Output of the neural network for 20,000 customers. Zooming in on the top two hundred chances to be clicked on for every customer.

5.1 Parameter tuning

Before we were able to start obtaining results we needed to tune the hyperparameters of our neural network. The hyperparameters are the variables that determine the structure of the neural network and the variables which determine how the network is trained. For the structure of the neural network, we are looking into the number of nodes in different layers, activation functions and, in our case, the embedding dimension as well. The variables for training the network are the number of epochs, learning rate and the batch size.

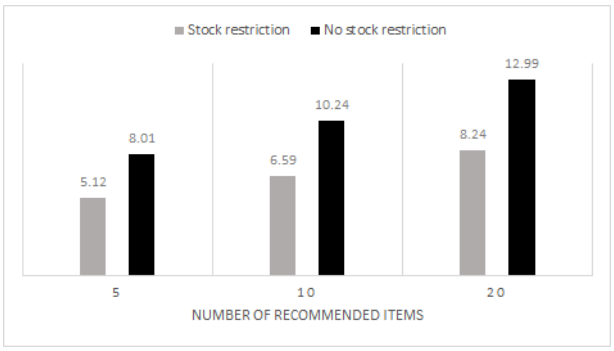
For the variables of the structure of our network, we used a relatively small version of our data. We used a smaller part of our data because the running time is less and we could test a larger amount of different values to obtain more results. Another reason was that with this amount of data we were already seeing the impact of different parameters, and how well the network was learning the data. For the training variables we started with a quarter of our whole data set. With this we tested the more extreme values to give a general idea in which direction we needed to go. Later on, we were increasing the amount of data we were using and making smaller changes in the parameter values to get as close as possible to our optimal values. We aimed to end with a data set close to our full data set to get the most representative effects. In such a way, we have found the following outcome for our hyperparameters shown in Table 5.1.

Table 5.1: Summary results parameter tuning

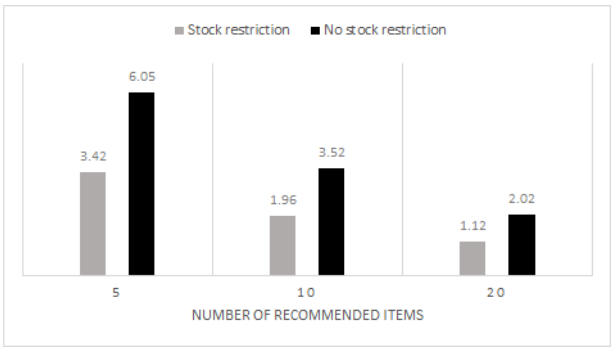
Item embedding	Activity embedding	Gru nodes	Epochs	Batch size	Learning rate
40	1	50	20	20	0.006

5.2 Stock value

To start with looking at the impact of including the stock value in our re-ranking algorithm, we compare the results of the performance measures of our neural network with the results of the performance measures of our re-ranking algorithm when we only use the stock value restriction. This stock value restriction can be anything you want using information about the stock value. In our case we used the restriction that the average stock value of all the sizes from a specific item is at least 0.8. We have chosen for this restriction because the customer would see the item on stock in their size most of the times if this restriction is met.



(a) Percentage, for the times the target item is included in the top-N recommended items.



(b) Average chance for the top-N recommended items to be clicked on, in percentage.

(c) Percentage of the top-N recommended items meeting the stock requirements.

Figure 5.2: Performance measures neural network compared to performance measures of the re-ranking algorithm only using the stock value restriction. Looking at different values of N.

In Figure 5.2a and 5.2b we see that the use of the stock restriction results into a circa 35% lower percentage where the target item is included in the top-N items and a circa 45% lower average chance of the top-N items being clicked on. In Figure 5.2c we see that the percentage of the top-N items meeting the stock requirements is twice as low if we do not use the stock requirements in our re-ranking algorithm compared to when we do use the requirements. So for the company, there is a trade-off to be made: do they want to show the customer items they want to see but might not be able to buy or are they showing items with a high chance being on stock but are less likely to be clicked on? A remarkable point in the test data we used to obtain our results, is that 41.5% of the target items that we tried to recommend did not meet the stock requirements. Thus, when we use the re-ranking algorithm where we include the stock requirements 41.5% of the times it was impossible to find the target item in our top-N recommended items. Because of this, we showed the percentage where the target item is included in the top-N items for the recommendations with the stock restriction in Table 5.2, but this time without these 41.5% of target items that were impossible

to recommend. In this table, we see that for each value of N the case with the stock restriction gives a circa 8% higher percentage than the situation without the stock restriction for all cases. So, you could say that the items that meet the stock requirements are "easier" to predict, but at least we can say the re-ranking method with stock requirements is not performing worse on cases they have an equal chance to succeed.

Table 5.2: Percentage of times the target item is included in the top-N recommended items. We compare the results of using and not using the stock restriction. For using the stock restriction results we excluded the impossible cases.

top-N	Stock restriction (%) (excl. impossible cases)	No stock restriction (%)
5	8.598	8.006
10	11.071	10.244
20	13.847	12.986

5.3 Retailers

When we include the number of retailers into our re-ranking algorithm, we try to give more retailers a chance to sell their items. Right now it might be the case that one retailer is linked to all the top-N recommended items. To diversify the number of different retailers included in the top-N items we set a maximum of items a retailer can be linked to. So, for example, if we want to recommend ten items we can set the maximum of items for a single retailer to four and in this way we have at least three retailers which are included to the top-N recommended items. For the different values of N we set the maximum of items for a retailer as lowest possible, while the re-ranking algorithm gives a feasible solution for every customer. We have set the values as lowest possible to see the biggest effects.

(a) Percentage, for the times the target item is included in the top-N recommended items.

(b) The number of total retailers offering the top-N recommended items.

Figure 5.3: Performance measures neural network compared to performance measures of the re-ranking algorithm only using the retailer restriction. Looking at different values of N

In Figure 5.3a there is no significant difference between the percentages of the target item being included in the top-N recommended items for the model with retailer restriction and the model without the retailer restriction. In Figure 5.3b we see that for each value of N the number of different retailers is higher if we include the retailer restriction, but the difference is less than 1% for every value of N. We see that the use of the retailer restriction does not lead to specific results. This can be explained by the fact that we used the total number of different retailers that can offer a certain item. In this way, if one of those retailers already has the maximum of items to recommend the other retailer can take this item. So, the total number of different retailers does not change and the percentage of the target item being in the top-N items does not change either, this is because the same item is chosen and the results stay the same. If we would have linked one item to strictly one retailer the difference could have been more significant. It also is possible that the initial recommendations from our neural network are already pretty well divided among the retailers, and the restriction of a maximum number of items per retailer is not necessary.

5.4 Profit

If we include profit in our re-ranking algorithm we link them to the costs in our network flow problem. In our situation, we link price directly to profit. So the higher the price the higher the profit. In other situations it also can be possible that different items have different type of profit margins, which can lead to different results. The costs in our network flow are not exactly the price, we first multiply the price with the chances we obtained from our neural network. This combination is the expected profit of the items.

(a) Percentage, for the times the target item is included in the top-N recommended items.

(b) Average expected profit in euros of the top-N recommended items.

(c) Average chance for the top-N recommended items to be clicked on, in percentage.

Figure 5.4: Performance measures neural network compared to performance measures of the re-ranking algorithm only using profit. Looking at different values of N.

In Figures 5.4a and 5.4c we see that the average chance of the top-N items being clicked on and the total percentage where the target item is included in the recommended items is significantly lower if we use profit in our algorithm. This is in line with the expectations, because if we assume that the output of the neural network (not using profit) is correct then the average chances can only go down and so the target percentage goes down as well. Now if we look at Figure 5.4b we see that the average expected profit of including profit in our re-ranking method results into a circa 10% increase. This means if we include profit to our re-ranking method we are recommending items with a significantly higher price than if we would not use profit, those percentages are shown in Table 5.3.

Table 5.3: The average price in euros of the top-N recommended items. For when we do and do not use profit in our re-ranking method. Also we show the percentage change between the two different situations.

top-N	Using profit	Not using profit	Percentage change
5	904.76	375.07	+141%
10	769.58	372.78	+106%
20	783.38	444.26	+76%

Another result we found while using profit in our re-ranking problem is that the percentage for meeting the stock requirements of items included in the top-N recommended is higher if we take profit into account. Those results are shown in Figure 5.5. This can be explained by, if we take profit into account the average price is higher and more items have a price of a thousand euro or higher. For this group are on average more items meeting the stock requirements as we see in Figure 5.6.

Figure 5.5: Percentage of the top-N recommended items meeting the stock requirements. For when we do and do not use the profit in our re-ranking method.

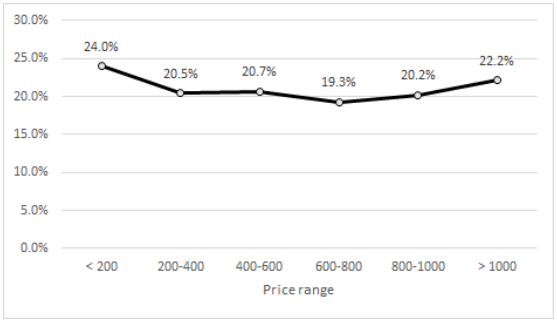
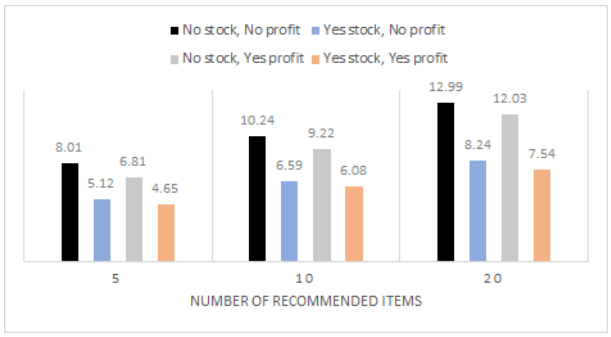


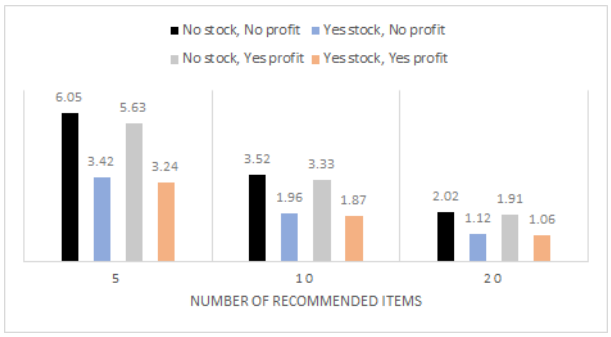
Figure 5.6: Percentage of items meeting stock requirements in different price ranges.

5.5 Combined effects

Because the results of the retailer restriction were not significant at all, we had a quick look at the combined effects including the retailer restriction. Here we also saw no significant effects, so we will only look further into the combined effects of using the profit and stock restriction.



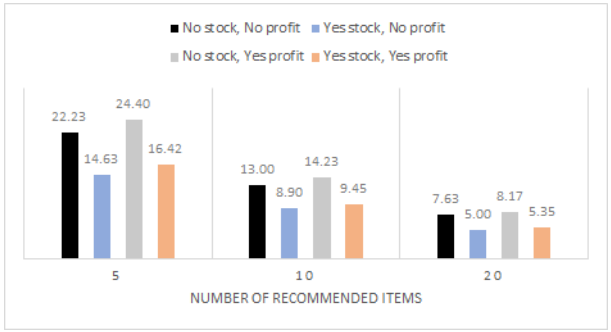
(a) Percentage, for the times the target item is included in the top-N recommended items.



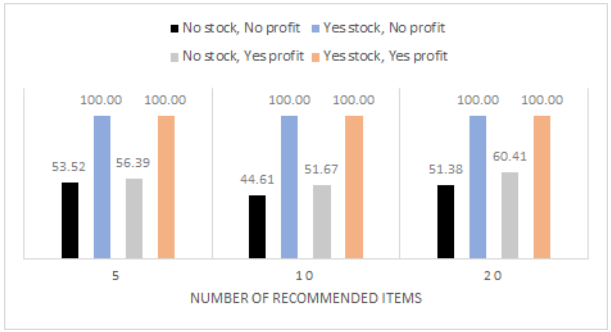
(b) Average chance for the top-N recommended items to be clicked on, in percentage.

Figure 5.7: Performance measures neural network compared to performance measures of three different cases of the re-ranking algorithm. Looking at different values of N.

In Figures 5.7a and 5.7b we compare the four different cases for the three N values. We see that the use of the stock restriction has the biggest effects, and we also notice that the use of both restrictions does not have an extra combination effect; just the two effects on top of each other.



(a) Average expected profit in euros of the top-N recommended items.



(b) Percentage of the top-N recommended items meeting the stock requirements.

Figure 5.8: Performance measures neural network compared to performance measures of three different cases of the re-ranking algorithm. Looking at different values of N.

For the business-wise results we have a look at Figures 5.8a and 5.8b. We can conclude the same for the effects of profit and stock restrictions as for the previous two Figures. The difference in this case is, that the use of these restrictions leads to better results in their corresponding performance measures.

So, to summarize these four Figures we could say that if we use the stock requirements the chances for the top-N recommended items are decreasing, which leads to a lower percentage of the target item being in the recommended items, a lower average chance and so a lower expected profit. On the other hand, all the items that are recommended, meet the stock requirements. For the use

of profit in our re-ranking algorithm, we also see that the chances for the top-N recommended items decrease, and so we also have a lower percentage of the target item being in the recommended items and a lower average chance. However, in this case, we are maximizing the profit which leads to higher prices for the N recommended items, and this shows a higher average expected profit and a higher percentage that is meeting the stock requirements. If we use both restrictions we have all of these effects combined.

6 Conclusion

The goal of this paper was to provide user-interest based recommendations and improve these recommendations with business-wise interests. We started by making a model only based on user interests. We wanted to use a neural network including a recurrent layer as LSTM or GRU. We have tested both of these layers. In our case, the GRU layer was not performing significantly worse and was performing significantly faster, so we decided to move on with the GRU layer. Because we were dealing with a relatively sparse and high-dimensional data set we quickly realised that we needed to make some smart changes to improve running time. We started with trying out general dimension reduction techniques like Principal Component Analysis and Singular Value Decomposition; those did not improve the running time we wanted. Then we looked into a model provided by Zhou et al. (2018), where they use an embedding layer to deal with a high-dimensional data set. We used these embedding layers to transform our data before going into the GRU layer. At this point, we needed to get a real-valued score out of the output of the GRU layer. For our data set, we used a tanh activation layer and to shape the output into a value between zero and one we used a softmax output layer. For the neural network, we needed to tune some parameters. We are taking a closer look at two, which are the new item embedding dimension and the new activity embedding dimension. Originally we have 42,681 different items and five different activities we take into account in our data set. After tuning these parameters we have come to an item embedding dimension of forty and an activity embedding dimension of one. So originally we had $42,681 \cdot 5 = 213,405$ different combinations. In the way, we use our embedding layers we now have a dense dimension of $40+1 = 41$ which is much lower and results in a faster training and performing network.

To include the business-wise interests we created an integer linear programming which can be transformed into a min-cost-max-flow network problem. This min-cost-max-flow problem can be efficiently solved by the use of the capacity scaling algorithm. For this re-ranking method, we use the output of the neural network as chances for all the items that it can be clicked on. We also include the stock level per item, the profit per item and the number of retailers involved per item. Because we used the negative of the expected profit we can look for the minimal cost, which is the maximal profit if we remove the minus sign. To see the results of our re-ranking problem we compared the three aspects separately with the initial results of the neural network and we combined these aspects to see what the effects are. If we look at only using the stock restriction we see that the average chance of the recommended items and the percentage of the times that the target item

is included in the recommended items is respectively around 45% and 35% lower. This can be partly explained by the fact that if we use the stock restriction 41.5% of the target items does not satisfy the stock requirements. If we look at the results excluding these "impossible" cases we see that the percentage of target items being included in the recommended items is around 7% higher. While using a restriction for the number of retailers being included with the recommended items we see no significant results and we do not take this aspect into account in our further results. If we use profit in our re-ranking algorithm we see that the average chance of the recommended items and the percentage of the times that the target item is included in the recommended items are both around 10% lower. Nevertheless, the expected profit is increased with circa 10%. This is due to the fact that the average price is almost doubled when using profit in our re-ranking method. If we use both the stock requirements and the profit in our re-ranking method, we do not see any combined increasing effects or decreasing effects. The effects of both restrictions are noticed, as we see for the model where we use both aspects; we have the lowest percentages for the average chance of the recommended items and the percentage of the times that the target item is included in the recommended items. The average profit for the combined model is performing better than when we only use stock but performs worse than the initial output of the neural network as well as when we only use profit.

For a company to decide which aspects to include in the re-ranking algorithm depends on what the intentions are with the recommender. If they want to have as many clicks as possible then they need to use the initial results of the neural network, but if they want to show only items meeting certain stock restrictions then they need to include those requirements to the re-ranking algorithm. Using those requirements have negative effects on the average chance of the recommended items and the percentage of the times that the target item is included in the recommended items. The re-ranking method which is taking profit into account has a higher chance of increasing the profit of the company. However, it leads to higher average prices of the items and the chances for the items of being clicked on are lower. So, the re-ranking algorithm does not give clear results which are always the best, but it gives the opportunity to the company to create recommendations they think fits best for the goals of their company.

7 Discussion and optional extensions

To improve the results we have so far we were looking at a few things. First of all, we could have spent more time finding the best parameter values possible. However, in our case we were time-restricted and at some point, we needed to be content with the outcome. Next, we can have a further look into trying different stock requirements because in our situation the stock requirement might have been too strict which resulted in too many items not satisfying these restrictions. This made it harder to compare the results in the cases we did not use the stock requirement. Lastly, the results of the re-ranking algorithm, where we used a retailer restriction, were not significantly different from the results where we did not use this restriction. We think this occurred because we linked all the retailers that could offer a certain item to that particular item, but in the end, only one retailer sells this item if the item is bought on the website. Because of this, the top-N recommended items were not changing as much as we thought upfront and the results stayed nearly the same. We do think that the results could become more different if we would link a single item to a single retailer. We also think that it can be useful for a company selling items through different retailers to diversify the retailers for the items of their recommender. This could give the retailers a more "fair" chance to sell their items. Also, this feature is something for the companies to decide whether they believe it is in line with their strategies and goals.

On a longer-term, we also wanted to include more restriction types next to profit, stock and the number of retailers. For example, including specific attributes of the current item that the customer is looking at; for instance, gender, price range, colour or a particular garment. We also assume it is interesting to look into the changes in the data set after the company starts using a recommender. Are we creating a more clear pattern in the data set? Will these lead to better neural network training and performance? Or will these lead to the neural network including aspects of the re-ranking algorithm? Are we seeing the popular items getting even more popular or are we giving all the items there chance to "shine"? These are questions that are interested to look into after introducing the recommender or the re-ranking algorithm. Lastly, in our data set, we are not experiencing any correlations between the different restrictions. However, it can be interesting for companies that have fewer retailers offering higher priced items than lower priced items. Or lower priced items having a lower stock value overall. To take a deeper look into those correlations and see what the effects are on the results of the different re-ranking algorithms.

Bibliography

G. Adomavicius and Y. Kwon. Improving aggregate recommendation diversity using ranking-based techniques. *IEEE Transactions on Knowledge and Data Engineering*, 24(5):896–911, 2011.

R. K. Ahuja and J. B. Orlin. A capacity scaling algorithm for the constrained maximum flow problem. *Networks*, 25(2):89–98, 1995.

T. Donkers, B. Loepp, and J. Ziegler. Sequential user-based recurrent neural network recommendations. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, pages 152–160, 2017.

A. A. Hagberg, D. A. Schult, and P. J. Swart. Exploring network structure, dynamics, and function using networkx. In G. Varoquaux, T. Vaught, and J. Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA, 2008.

T. Jambor and J. Wang. Optimizing multiple objectives in collaborative filtering. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 55–62, 2010.

D. Jannach and G. Adomavicius. Price and profit awareness in recommender systems. *arXiv preprint arXiv:1707.08029*, 2017.

G. Linden, B. Smith, and J. York. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing*, 7(1):76–80, 2003.

G. Loye. Gru architecture, 2019. URL `https://blog.floydhub.com/content/images/2019/07/image14.jpg`.

T. Masters. *Practical neural network recipes in C++*. Morgan Kaufmann, 1993.

NetworkX. Source code for networkx.algorithms.flow.capacityscaling, 2020.

M. Quadrana, A. Karatzoglou, B. Hidasi, and P. Cremonesi. Personalizing session-based recommendations with hierarchical recurrent neural networks. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, pages 130–137, 2017.

Y. Wang and L. Zhu. Research and implementation of svd in machine learning. In *2017 IEEE/ACIS 16th International Conference on Computer and Information Science (ICIS)*, pages 471–475. IEEE, 2017.

M. Zhou, Z. Ding, J. Tang, and D. Yin. Micro behaviors: A new perspective in e-commerce recommender systems. In *Proceedings of the eleventh ACM international conference on web search and data mining*, pages 727–735, 2018.