# Exploring the applicability of a data-driven approach on recommending in-store replenishments for fashion retailers

Master's Thesis

**Sander Meinderts**
s.meinderts@alumnus.utwente.nl

Supervisors

| | |
|---|---|
| dr. M. Poel | University of Twente |
| dr. A.B.J.M. Wijnhoven | University of Twente |
| D.W. Muller, MSc | Nedap |
| R. Weikamp, MSc | Nedap |

January 21, 2021

## Abstract

Having the right articles on the sales floor is essential for maximizing the profit of a fashion retailer. Identifying size gaps on the sales floor using RFID counts leads to recommendations on what articles should be refilled from the in-store stockroom to the sales floor. Although effective, not all size gaps should be resolved whereas some articles should be refilled even when they are still present on the sales floor. This research investigates the applicability of a data-driven approach on predicting what articles should be refilled. In order to determine this applicability, a dataset was formed from scratch for a single fashion retailer, the features and labels of which were based on historic data. Different machine learning models were tested on this dataset, of which a mixture of a Wide and Deep classification model and a residual regression network turned out to be the best performing, achieving 0.48 and 0.44 on the test set for the F1 score and Matthews Correlation Coefficient (MCC) respectively. This was an improvement of the current refill implementation, which achieved 0.30 and 0.24. The feature attributions of the model and the differences between outputs for specific categories and stores imply that the model is able to learn how to differentiate between these different categories or stores. Measuring the performance in actual stores turned out to not deliver statistical significant result. The evaluation of the model in actual stores could have been more exhaustive, were it not that difficulties were introduced by the global pandemic. The performance of applying the trained model on the data of a different retailer was considerably worse, resulting in an F1 score and MCC of only 0.24 and 0.26 respectively. In hindsight the chosen retailer used for training the model turned out to be unfortunate given the noise introduced by their frequency of counting. All in all the data-driven approach showed promising results and is definitely worth pursuing once data can be collected with less noise and online testing can be performed on a larger scale.

# Contents

# 1 Introduction

Product availability is of vital importance to fashion retailers. Customers are not likely to ask store employees whether a product or size is still available, meaning that an incomplete sales floor directly influences the amount of sales[1]. By extension, missing products on the sales floor decreases the amount of profit a store makes [7]. Stores are thus aiming to provide a sales floor with a wide variety of products for which at least one item of the popular sizes is physically present. However, given the limited space available on the sales floor, decisions need to be made regarding what items should be present on the sales floor. There are two stages that contribute to the presence of items, namely deciding on what should be sent from the distribution center to the in-store stockroom and what should be moved from this in-store stockroom to the sales floor. For this thesis the focus will be on the movements within the store itself, i.e. the movements from the in-store stockroom to the sales floor.

Accurate stock information is of vital importance when deciding upon what items should be moved from the in-store stockroom to the sales floor. Manually counting an entire store is tedious, so retailers have started adopting radio frequency identification (RFID) [50]. An RFID reader sends radio waves to tags in its vicinity, not necessarily line-of-sight, which answer with radio waves containing data used to identify that certain item. RFID can be used to perform quick scans on both the sales floor and the stockroom in order to get a better insight in what item is where, as long as the items have been tagged. Since counting an entire store using RFID is a matter of minutes, counts can be performed more frequently, leading to more accurate stock information[2].

When looking at retail, multiple studies have been performed discussing the possibilities of RFID [17, 39, 43]. Several companies saw a potential market and started offering RFID technology to retailers, including but not limited to Detego[3], Nedap[4] and RIOT[5]. The focus of solutions provided by these companies is mostly on smart inventory management, i.e. to provide a more detailed and accurate representation of what articles are where and in what quantities. This accurate data can be used in various ways, like computing how many items disappeared and determining what articles should be sent from the distribution center to the in-store stockroom.

Aside from store replenishment and anomaly detection there is a second area of interest, namely the replenishment of the sales floor. Even though a store may have all the required articles in stock and perfect insight in where an article is, an article is still more profitable when it is in view of the customer. The reason being that not all customers will ask a store employee whether his or her size is still available, but instead move to different products or even move out of the store. Some retailers refer to articles which have the potential of being on the sales floor while not being on the sales floor as Not On Shelf But On Stock (NOSBOS). Several possible states of an article have been sketched in figure 1, of which 1b and 1c concern NOSBOS articles.

[1]https://nedap-retail.com/resources/whitepapers/merchandise-refill-retail-stores
[2]https://gs1uk.org/rfidinretailing
[3]https://detego.com/detego-suite-en/for-the-store
[4]https://nedap-retail.com/solutions/id-cloud
[5]https://riotinsight.com/solutions

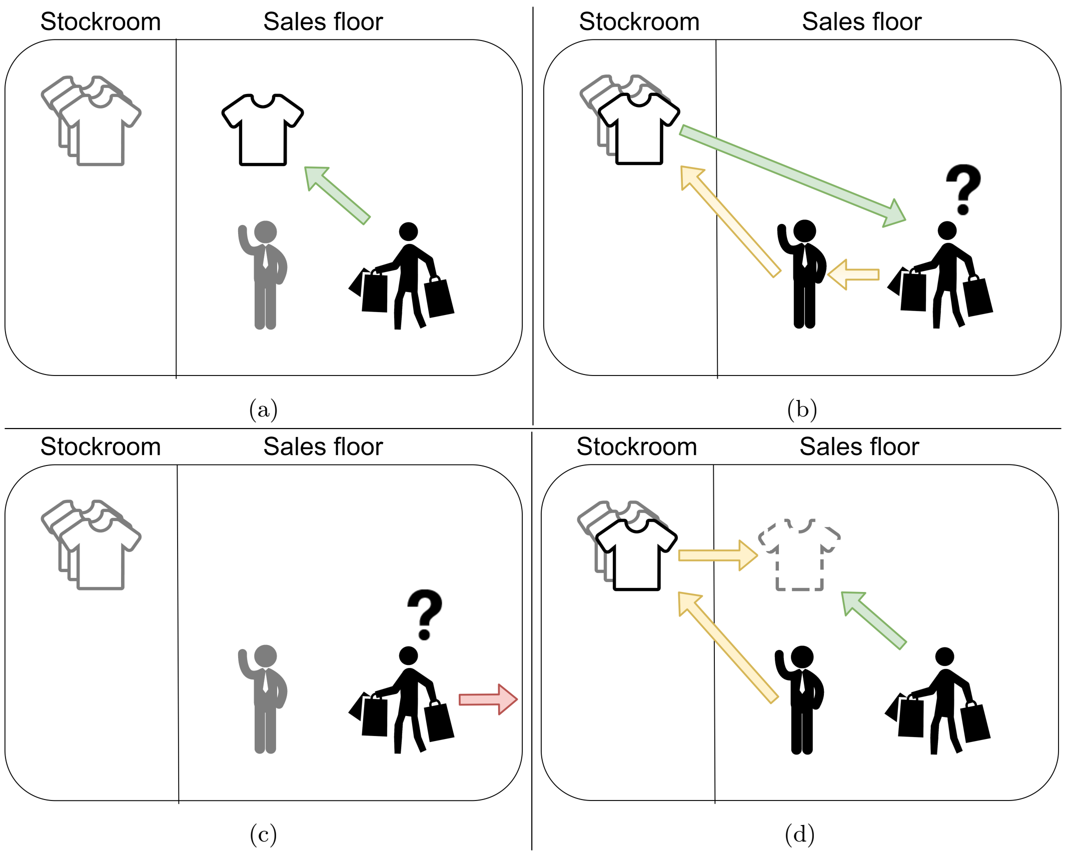


Figure 1: Different situations that might occur in the store of a fashion retailer. The objects are colored gray when they are not interacted with in that scene. In all the situations the customer is looking for a specific article to purchase.

(a) The article desired by the customer is on the sales floor, leading to a purchase.

(b) The article is not on the sales floor, so the customer asks an employee of the store about the article. The article is in the stockroom and can be retrieved by the employee, leading to a purchase.

(c) The desired article is not on the sales floor, so the customer leaves the store without making the purchase or asking the employee about the article.

(d) The store employee has been notified one way or another that the article stack should be refilled and has refilled the previously depleted article stack. Just like in situation (a), the customer observes the article on the sales floor, leading to a purchase.

When using a smart inventory management system, the accurate information of item quantities can be used to determine what articles should be replenished and what articles do not need special attention. The suggestions indicating which articles should be replenished from the stockroom to the sales floor will from now on be referred to as *refill recommendations*. Using refill recommendations, NOSBOS events can be prevented, as shown in figure 1d.

Refill recommendations have already been included in several of the solutions mentioned earlier. The way these refill recommendations are generated is mainly focused around resolving NOSBOS articles. It is not always clear to a store employee that an article is NOSBOS, e.g. it is not evident from a stack of ten t-shirts that one size is missing. Therefore, these refill recommendations are already appreciated, leading to an approximate increase in sales of about 0.5 to 1.5 percent. However, not all NOSBOS articles are proper refill recommendations since stores can also intentionally not place that article on the sales floor. A common example is shoes, of which stores rarely places every possible size on the sales floor, and winter coats, which take up too much space to be available in every size.

Aside from recommending unwanted NOSBOS articles, another unsolved issue is preventing certain articles from disappearing of the sales floor. Preventing NOSBOS articles is significantly more complicated than reacting to NOSBOS articles, since it involves predicting whether an article will sell-out between the store counts. Predicting the quantity to refill per article can, if sufficiently accurate, aid the stores in having a sales floor where unintended NOSBOS articles stop occurring.

This thesis will focus on improving said refill recommendations. In the remainder of this section the terminology used will be discussed, after which the potential improvements will be highlighted. The section will be concluded with the research goal and questions as well as an overview of the remainder of this report.

## 1.1 Terminology

Fashion retailers use different layers to group their products. The topmost layer is dividing the products across different *categories*. Examples of categories are t-shirts, shorts, socks and shoes. The products in a category are grouped per *option*, specifying different styles of the products within this category. Options within the category shorts could be green shorts, yellow shorts or camo shorts. The next level consists of *articles*, which divide an option into different sizes. Articles are groups such as yellow shorts in size L, green sweaters in size S or red shoes in size 43. Lastly, the articles are split in *items*, which indicate a specific instance of an article. A store can have several yellow t-shirts of size L on the sales floor, each and every one of which is a unique item. The layers are visualized in figure 2.

When scanning the tags of items using an RFID reader, the system can differentiate between different items using the item's *Electronic Product Code* (EPC). Every item has a unique EPC which can be used to track the item along the supply chain. Articles themselves are also identifiable using a unique *Global Trade Item Number* (GTIN). The GTIN can be used to scan a product on checkout, where it is included in the barcode, or to group the individual items.

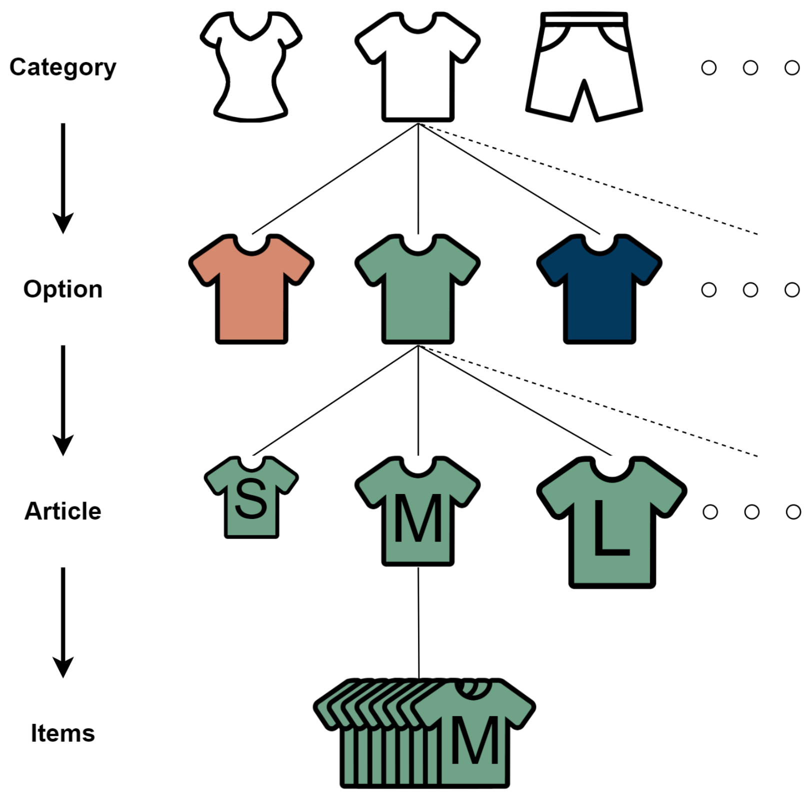


Figure 2: Visual breakdown of the terminology used by fashion retailers.

There are four locations an item can be, namely the *source*, the *distribution center*, in one of the in-store *stockrooms* and on one of the *sales floors.* All replenishments, or refills, discussed in the remainder of this thesis will target movements from the in-store stockrooms to the sales floor, unless stated otherwise.

As mentioned before, the term *NOSBOS* refers to articles that have no items on the sales floor whereas there are still items available in the stockroom.

Lastly the term *refill recommendation.* Specifically, using the terminology previously defined, a refill recommendation concerns the quantity of items that should be moved from the in-store stockroom to the sales floor for a specific article. An example of a refill recommendation would thus be: 'move 2 green t-shirts size M from the stockroom to the sales floor'. In practice a recommendation will not be as verbose but will use symbols to deliver the same message. In theory, refill recommendations can also be used to move items from the sales floor to the in-store stockroom, but since this might lead to missed sales opportunities these will be disregarded for this research. It is worth noting that when sharing refill recommendations with a retailer, they are grouped per option and as such the term refill recommendation will sometimes be used to indicate the group of individual refill recommendations. The accompanying text will make clear when this alternative definition is used.

The relevant terms have been summarized in table 1.

Table 1: Descriptions of the relevant terms used in fashion retail.

| **Term** | **Description** |
|---|---|
| Category | A subdivision of the set of products |
| Option | A subset of products within a category |
| Article | A specific size of an option |
| Item | A unique instance of an article |
| GTIN | Identifier of an article |
| EPC | Identifier of an item |
| Stockroom | The in-store area used for the storage of items |
| Sales floor | The in-store publicly visible area where items are displayed for sale |
| NOSBOS | Articles that are not available on the sales floor but are refillable from the stockroom |
| Refill recommendation | A quantity of items that should be moved from the stockroom to the sales floor for a specific article |

### 1.1.1 Data-driven approaches

Making predictions about what items should be available on the sales floor is a process largely influenced by the data available. The data gathered for this research consists solely of RFID counts, which result in detailed information regarding the item quantities for both the stockroom and the sales floor. Logically, this data becomes a less accurate representation of reality as more time passes, being updated only when a new count is performed. Other forms of valuable data include sales data and indications what item movements have occurred within the store. However, retailers are hesitant to supply their sales data to third parties. Data concerning in-store movements is usually not readily available since it is rarely part of the workflow. Furthermore, items are also frequently refilled during the day when an employee notices that the article quantity on the sales floor is off. Given that the count data is available for all retailers using RFID for inventory management, this data will be the focus of this thesis.

A direct effect of these limitations is that refill recommendations based solely on store counts are only relevant as long as the stock information is still relevant. Articles that go NOSBOS in between counts will not be considered a refill recommendation until the next count. A solution would be to count more frequently and hence have more up-to-date data, but this is often not desired given the labor-intensive nature of performing counts.

Even though the data available is far from perfect, it is still possible to predict refill recommendations. Simple quantity predictions can be made by looking at features like the average quantity of an article the past couple of weeks or the highest quantity that article has ever registered during a count on the sales floor. However, developing such an algorithm by hand is time-consuming and requires a substantial amount of tweaking to find the right balance between features. Furthermore, changes over time are difficult to reflect in a handcrafted algorithm without a complete overhaul of the algorithm itself. An alternative to manually developing such an algorithm is using machine learning to train a model to predict the quantities. A properly trained model would learn the balance between the features itself, whereas periodically retraining the model is able to reflect changes over time.

### 1.1.2 Machine learning

When applying supervised machine learning, one of the requirements is a labeled set of data. Creating such a set of labeled data is not as trivial as it might sound. Simply asking a retailer what they want to refill or what their target sales floor should be is not viable, since these targets differ widely depending on who you're asking. The headquarters of a retailer usually have a target in mind, but this target might not be achievable in every store due to spatial limitations. Even if these targets are achievable, the store managers themselves will likely have an opinion of their own based on their personal experience. A different approach would be to derive these target stocks from the data itself, which is not as trivial as it may sound. Given the absence of data indicating when certain items were moved from the stockroom to the sales floor, the only indication of such movements are found in the count data, which becomes less accurate when counts occur less frequently. Even though deriving the labels from the data will never be a completely accurate representation of what movements actually took place within a store, it does provide a better take on what a store actually wants when compared to what a store thinks it wants.

Assuming the target labels can be derived, the next issues arise from the question what machine learning method is most suitable for the problem at hand. There are three possible ways to look at the problem. The prediction task can be seen as classifying the quantity to refill or using regression to predict the quantity to refill. The third option is a combination of both, where first classification occurs to decide whether or not to refill and then regression is used to determine the quantity to refill. The question then becomes what method is best capable of dealing with the sparsity and noise of the dataset.

Considering the machine learning methods mentioned above it is not uncommon to consider either classification or regression. The usage of both approaches at once requires more explanation. The main idea is derived from recommender systems, which exploit similarities between different users and items to recommend items to users. Recommender systems are known for having to deal with a sparse dataset as well as continuously changing data. Examples of recommender systems used on a large-scale include YouTube and Netflix [12, 23], where the number of videos actually clicked on by a user is a small fraction of the total number of videos. Similarly, the number of target labels indicating that a refill should occur will likely be only a small portion of the data since most articles do not need to be refilled. Furthermore, new articles and changes of season require the machine learning model to continuously adjust its predictions accordingly, matching the goal of recommender systems.

## 1.2 Research goal

Given that the dataset is suffering from a lot of noise, originating from the uncertainty of the labels, the goal of this research is not to deliver a machine learning model capable of predicting refill recommendations on a level that is suitable for use in practice. Instead, the goal of this research is to determine whether machine learning, or a data-driven approach in general, is even applicable to predicting refill recommendations.

## 1.3 Research questions

The research goal led to the following main research question to be answered:

> *How applicable is a data-driven approach for generating refill recommendations for real-world fashion stores?*

Applicability is quite a broad term, therefore the details of the relevant aspects have been worked out in research sub-questions.

1. *What machine learning method performs best, judged by the highest average of the F1 score and Matthews Correlation Coefficient?*
2. *How do the best performing model's recommendations compare to the current refill implementation?*
3. *To what extent is the model able to differentiate between different stores and product categories?*
4. *What performance decay, if any, occurs when applying the model on a different retailer?*

5. *What performance decay, if any, occurs when applying the model on counts performed after the latest count that the model was trained on?*
6. *What are the requirements to fulfill in order to create a production-ready data-driven approach for generating refill recommendations?*

The first sub-question concerns the definition of the machine learning method to be used for the remainder of the research. The next sub-question compares the recommendations made by the model with the original refill application, creating a first impression of the potential of the machine learning model. The third sub-question is used to indicate whether the model is able to base its results on the product category or the store it belongs to. Such behavior differentiates a machine learning model from writing an exhaustive algorithm yourself. The next two sub-questions concern the consequences of using a machine learning model in practice, mainly focusing on the re-usability of the model and how often the model is deprecated. Closely related, the last sub-question is used to indicate what steps should be taken in order to actually deploy such a model for use in real-life.

## 1.4 Outline

The remainder of this thesis starts with an introduction to some basic concepts in section 2 and an outline of related work in section 3. The methodology used for this thesis will be detailed in section 4 and the results of this study are shown in section 5. A critical review of the research will be provided in section 6 after which the research is concluded in section 7. Lastly, the recommendations concerning what steps should be taken next for Nedap as well as the recommendations for future work are given in section 8.

# 2 Background

## 2.1 Neural networks

A neural network in its simplest form is a feed-forward network of perceptrons. A perceptron is a computational unit that takes an input, performs the computation and returns an output. A neural network can consist of multiple such perceptrons per layer and can contain any number of layers additional to the input and output layer, which are referred to as hidden layers. When one or more hidden layers are present the network is called a Multilayer Perceptron (MLP). The perceptrons are connected to the input, output or one another by edges with a certain weight. Each node computes the sum of the incoming nodes multiplied with their weights and outputs the result, usually after applying an activation function. The use of the activation function is what allows the neural network to find non-linear separation boundaries. When using $z_j^{(k)}$ as the output of node $j$ in layer $k$, $w_{ji}^{(k)}$ as the weight connecting node $i$ from layer $k-1$ to node $j$ of layer $k$ and $b^{(k)}$ as the bias of layer $k$, the computation of $z_j^{(k)}$ using activation function $f$ can be written either as a sum or as a matrix multiplication. In order to simplify the matrix multiplication, the bias term was incorporated in vector $\mathbf{z}$.

$$\begin{aligned} z_j^{(k)} &= f\left(\sum_{i=1}^{n}(w_{ji}^{(k)} z_i^{(k-1)}) + w_{j0}^{(k)} b^{(k-1)}\right) \\ &= f(\mathbf{w}_j^{(k)} \mathbf{z}^{(k-1)}) \end{aligned}$$

The output of the entire network is computed in a similar fashion. In order to train the network, the output is compared to the expected output, or ground-truth, using a loss function. The gradients of the loss $L$ with respect to the weights are back-propagated through the network in order to update the weights using gradient descent:

$$w_{ji} \to w_{ji} - \eta \frac{\partial L}{\partial w_{ji}}.$$

A schematic overview of a simple MLP is shown in figure 3.

### 2.1.1 Residual Networks

A residual network is a special kind of neural network which consists of residual blocks. Within deep learning, gradients have a tendency to become smaller when adding more layers, essentially making it impossible to properly train a network of many layers. This is known as the vanishing gradient problem. To combat this, residual networks use skip connections that add the output of a previous layer to a new layer, amplifying the gradient and thus easing learning [25]. The layout of a residual block is shown in figure 4.

## 2.2 Activation functions

The most common activation functions used in neural networks are visualized below. Figures 5a, 5b and 5c show the Linear, ReLU and Sigmoid activation functions respectively. It is difficult to plot the softmax activation function since it is based on multiple variables. As such only the function notation will be provided to calculate the Softmax for $x_i$, which is one element of the vector $\mathbf{x}$ of length $n$. This function is shown in equation 1.

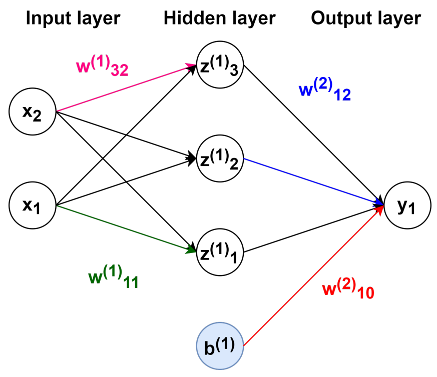


Figure 3: Simple Multilayer Perceptron. Some weights have been colored to indicate which weight is which.

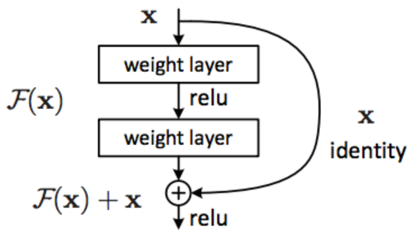


Figure 4: Residual block as shown in [25].

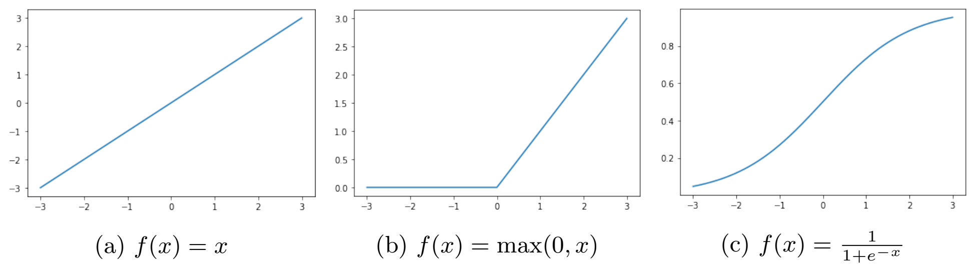


Figure 5: The Linear, ReLU [35] and Sigmoid activation functions.

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=0}^{n} e^{x_j}} \tag{1}$$

## 2.3 Metrics

The metrics discussed in this section will be used to evaluate the performance of the different models. Since this study deals with both classification and regression, both classification and regression metrics will be discussed. A more detailed overview of which metric will be used when can be found in section 4.

### 2.3.1 Classification metrics

The classification metrics will be used to facilitate a comparison between the different models on their performance when predicting whether an article should be refilled or not. Essentially this is binary classification, hence the following metrics can be used for evaluation.

#### Accuracy, precision, recall and the F1 score

Precision and recall are one of the most widely used accuracy metrics ever since their proposition in 1968 [16]. Precision measures the fraction of selected positive items out of the total amount of selected items, whereas recall measures the fraction of selected positive items out of the total amount of positive items. The accuracy of a classifier is simply the fraction of correctly predicted samples out of the total number of samples.

$$\begin{aligned} TP &= \text{True Positives} \\ FP &= \text{False Positives} \\ TN &= \text{True Negatives} \\ FN &= \text{False Negatives} \end{aligned} \qquad \begin{aligned} \mathit{Accuracy} &= \frac{TP+TN}{TP+FP+TN+FN} \\ \mathit{Precision} &= \frac{TP}{TP+FP} \\ \mathit{Recall} &= \frac{TP}{TP+FN} \end{aligned}$$

Closely related is the F1 score, which is the harmonic mean of the precision and the recall. The standard definition of the F1 score is as follows:

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}.$$

All the aforementioned metrics can have values between 0 and 1, with a value closer to 1 meaning a better performing classifier.

#### Receiver operating characteristic curve

A receiver operating characteristic (ROC) curve is a figure showing the trade-off between the true positive rate (TPR) and the false positive rate (FPR), which are calculated by dividing the true positives by the total positives and false positives by the total amount of negatives respectively. The area under the curve (AUC) can be measured to determine how well a classifier distinguishes between two classes. The ROC curve is shown in figure 6.

$$TPR = \frac{TP}{TP + FP}$$
$$FPR = \frac{FP}{TP + FN}$$

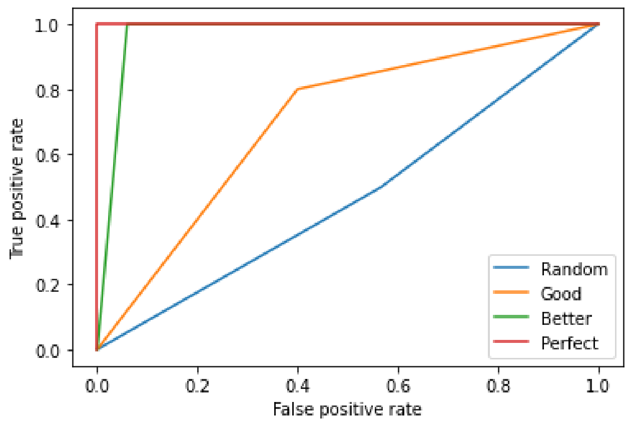


Figure 6: ROC curve

### Matthews correlation coefficient

A relatively unknown classification metric is the Matthews Correlation Coefficient (MCC). It has existed since 1975 and has been widely used in the field of bioinformatics [5, 32]. Recently, MCC has been proposed as a proper alternative to the F1 score and accuracy within the field of machine learning due to it being unaffected by class imbalance [8, 13]. The MCC can be calculated using the following equation:

$$MCC = \frac{(TP \cdot TN) - (FP \cdot FN)}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}.$$

Aside from being impervious to class imbalance, the MCC is also symmetric, meaning it does not matter which class is picked as the “positive" class. The MCC values range from -1 to 1, where 1 is a perfect classifier and -1 represents a classifier which predicts everything wrong. The value 0 is assigned to a random classifier.

### Cross-Entropy loss

The Cross-Entropy loss (CE), or log loss, calculates the errors between the output of a classification model and the expected output when the expected values are between 0 and 1. The definition uses $c$ for a specific class, $y_c$ for the expected output for class $c$ and $p_c$ for the predicted output of class $c$:

$$CE = -\sum_{c=1}^{C} y_c \log(p_c).$$

### 2.3.2 Regression metrics

The regression metrics will be used to facilitate a comparison between the different models on their performance when predicting the exact quantity of items to refill for a specific article.

#### Mean absolute error

The mean absolute error (MAE) calculates all errors between the actual quantities $q_i$ and the predicted quantities $\hat{q}_i$, then sums their absolute values and divides it by the total amount of samples.

$$MAE = \frac{\sum_{i=0}^{N} |\hat{q}_i - q_i|}{N}$$

#### (Root) mean squared error

The root mean squared error (RMSE) calculates the sum of the squared errors between the actual quantities $q_i$ and the predicted quantities $\hat{q}_i$, then divides that by the total amount of samples and takes the square root. The RMSE is similar to MAE, but the quadratic portion in the sum makes RMSE exaggerate the big differences. Hence it has been noted that MAE is usually preferable to RMSE [51].

$$RMSE = \sqrt{\frac{\sum_{i=0}^{N} (\hat{q}_i - q_i)^2}{N}}$$

The mean squared error (MSE) uses exactly the same formula only without the square root.

## 2.4 Recommender systems

In this section a short introduction will be given regarding recommender systems. For an exhaustive study on recommender systems consider reading the book *"Recommender Systems"* by Aggarwal [3]. This book is the main source of this section and the notation used throughout the book will be used for this section as well.

Recommender systems are used to predict a rating, or value, a specific user gives to a certain item [41]. Based on these ratings a subset of the available items can be recommended to that user. Recommender systems use (a combination of) the following three approaches:

1. Collaborative filtering,
2. Content-based filtering,
3. Knowledge-based filtering.

#### Collaborative filtering

Collaborative filtering exploits ratings given by other users based on their similarity to the target user. These similarities are calculated by comparing the ratings given by the target user and some other user. Aside from user similarities, recommender systems are also known to exploit similarities between the items to be ranked. Item similarities are computed by looking at the ratings given to the items by different users. Which of the two similarities is more applicable depends on the ratio between users and items, when there are more users than items the item-item similarities are preferable whereas user-user similarities are preferable when there are more items than users.

The similarities are usually calculated using either the Pearson correlation coefficient or the cosine similarity. Users $u$ and $v$ have a set of rated items $I_u$ and $I_v$ respectively. The rating of an item $k$ that user $u$ has given to it is denoted by $r_{uk}$, while $I_u \cap I_v$ denotes the set of items rated by both $u$ and $v$. Furthermore, the average rating $\mu_u$ given by user $u$ can be calculated as:

$$\mu_u = \frac{\sum_{k \in I_u} r_{uk}}{|I_u|}.$$

When defining $s_{uk} = r_{uk} - \mu_u$ as the normalized rating, the Pearson correlation coefficient for users $u$ and $v$ and the cosine similarity are defined as follows:

$$Pearson(u, v) = \frac{\sum_{k \in I_u \cap I_v} s_{uk} \cdot s_{vk}}{\sqrt{\sum_{k \in I_u \cap I_v} s_{uk}^2} \cdot \sqrt{\sum_{k \in I_u \cap I_v} s_{vk}^2}},$$

$$Cosine(u, v) = \frac{\sum_{k \in I_u \cap I_v} s_{uk} \cdot s_{vk}}{\sqrt{\sum_{k \in I_u \cap I_v} s_{uk}^2} \cdot \sqrt{\sum_{k \in I_u \cap I_v} s_{vk}^2}}.$$

In order to predict the rating of user $u$ for an unrated item $k$, the following prediction function can be used, where the ˆ is used to mark an estimate:

$$\hat{r}_{uk} = \mu_u + \frac{\sum_{v \in P_u(k)} Sim(u, v) \cdot s_{vk}}{\sum_{v \in P_u(k)} |Sim(u, v)|}.$$

$Sim(u, v)$ is either of the aforementioned similarity functions and $P_u(k)$ is the set of closest neighbors of user $u$ who have specified a rating for item $k$.

It is easily seen that in order to generate a list of recommendations using such a similarity-based method one would have to compare each pair of items, which is inefficient and expensive when the amount of items is large. In order to tackle this problem a new way of collaborative filtering was introduced in 1998 in which a predefined model is used to directly determine the recommendations [10]. There is a wide variety of models that fit this description, like decision trees and regular classification or regression models, however, most uses of models for collaborative filtering focus on Bayesian inference, latent factor methods and black box models.

### Content-based filtering

Content-based filtering bases the recommendations on the items previously rated by the same user. Therefore, newly added items have the same chance of being recommended as frequently rated items, alleviating the cold start problem for new items.

Content based filtering is performed in three stages. Firstly, the features of the items need to be determined and the items themselves need to be preprocessed to match the required features. Features can be a sentiment analysis of an item description, the price, the year of release and so forth. As is usual when applying machine learning, this selection of features is important and can make or break the resulting system, consequently careful consideration should be put into this feature selection. After the features have been selected a model needs to be trained that maps users to the item features. Examples of such models can be any regular classification or regression model as well as the nearest neighbor or Bayesian classifier.

### Knowledge-based filtering

Contrary to both collaborative and content-based filtering, knowledge-based filtering does not suffer from any cold start problem, since it does not depend on historical data. Knowledge-based filtering uses requirements specified by the user to recommend the list. An example would be the website of a car seller, who provides several filters with which the user can specify the price, the amount of seats and so on. The recommendation problem becomes fairly easy as the items can be matched against the criteria provided by the user.

The fact that knowledge-based filters do not take historical data or similarities into account is also the main disadvantage, since the quality of the recommendations is directly influenced by the user's ability to specify what he or she wants. Furthermore, it is up to the store to make the features of the items intuitive to the user, since the user is in direct contact with said features. Therefore it is unlikely that all possible features of an item can be used.

# 3 Related work

## 3.1 Retail

Little research has been performed regarding refill quantities within the store itself. Brahmadeep and Thomassey proposed a two-stage forecasting model for replenishing store quantities from a warehouse for fast fashion [9]. Since fast fashion focuses on a fast changing stock of articles, the replenishments from warehouses need to be based on little historical data. A sales forecast is made based on a long-term forecasting model, which uses historical article data as well as the attributes of the new article in order to generate a long-term forecast. This forecast is used as input to a short-term forecasting model, which uses the sales data of the past two weeks to readjust the forecast. The two-stage model structure is shown in figure 7.

Using a discrete simulation, the authors developed a replenishment model which simulated the replenishment of stores based on a 12-week season using data from a French retailer consisting of 482 products. The combined forecasting model was able to reduce the residual inventory of the simulated store from 5451 items to 1862 items and increase the amount of sales from 31.048 to 34.637. However, these results were simulated without taking into consideration the different sizes or the, sometimes valuable, input from the store managers. Especially the missing input from store manager might be vitally important, since the simulation performs all actions recommended, even if they might be considered too time-consuming in practice.

Park and Nam proposed to use a collaborative filtering based recommender system to recommend quantities to refill from the warehouse to the physical stores in order to optimize the store inventory [37]. Using stores as users and articles as items, the authors propose a collaborative filtering formula with which the quantities can be determined. A similarity matrix is proposed between stores based on the cosine similarity of product sales quantities. The product recommendations are computed by taking the normalized sum of the similarities between the target store and the other stores. The new recommendations are compared against a baseline using precision, recall and a newly defined novelty metric:

$$Novelty = \frac{\textit{Items recommended that have never been sold before}}{\textit{All recommended items}}.$$

Although the study showed promising results, increasing the F1 score from 0.06 to 0.12 and a novelty mean of 0.05, the data used was from a relatively short period of time without taking seasonal differences or trends into account. Other caveats of the data are that it includes sales data, which naturally makes it easier to predict the demand of individual products, and that the data is small-scale.

More research has been performed when looking at e-commerce, specifically product recommendation. Prévost et al. proposed to use wide and deep networks in combination with time translation and next-nearest neighbors in order to overcome the problem faced in retail that there is a constantly renewing inventory [38]. Time translation concerns the mapping of out of stock products, e.g. products from a previous collection, to products that are currently in stock. Specifically, the image features of the different products were determined using an ensemble of convolutional neural networks (CNN). Combined with other product

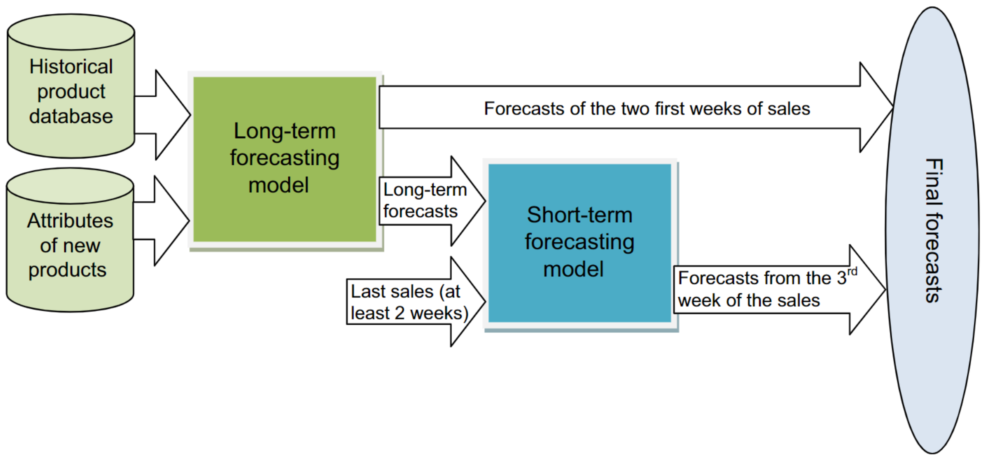

Historical product database
Attributes of new products
Long-term forecasting model
Forecasts of the two first weeks of sales
Long-term forecasts
Short-term forecasting model
Last sales (at least 2 weeks)
Forecasts from the 3rd week of the sales
Final forecasts

Figure 7: The two-stage forecasting model from [9].

features, the products that are currently in store are mapped to older products in a next-nearest neighbor approach. Using these similarities, the interaction matrix between users and items can be augmented, leading to a decrease in the sparsity of the data of the training set. Consequently, this led to increased performance when comparing a wide and deep network baseline with a wide and deep model including both time translation and next-nearest neighbors.

When it comes to considering in-store replenishments, Hübner et al. have proposed a model for a grocery retailer which combines regular shelf-space optimizations with the cost of replenishing and stocking items [27]. Whereas regular models only look at the optimal shelf-space allocation, the authors argue that this is closely related to the in-store replenishment process since items more frequently present on the sales floor are less likely to be refilled and thus restocked. The resulting model balances the costs of stocking items and replenishing items with the additional profit of showing the right items on the sales floor. The small scale experiment on canned food for a German retailer showed a staggering 29% increase in profits. While grocery retail may be a different field than fashion regarding the turnover of a store, it is an example of research targeting the in-store replenishments.

## 3.2 Recommender systems

Recommender systems in general are an essential part of big tech companies such as Amazon [6], Facebook [36], Google [12, 18] and Netflix [23]. Furthermore, several big companies in the world of fashion e-commerce have also adopted recommender systems, for instance Myntra [2], Taobao [49] and Zalando [26]. Especially YouTube is known for the progress they have made in the recent years regarding their video recommendation system, publishing three separate papers highlighting the improvements they made and detailing the changes.

The first paper from 2010 details the challenges that recommending YouTube videos poses, mentioning the noisy user interaction, sparse amount of metadata and short life cycle of videos [19]. The solution presented consists of both content-based and collaborative filtering. Firstly, a set of related videos is created based on the user's previously watched videos, where the relatedness is determined using the co-visitation count, i.e. how often videos $v_i$ and $v_j$ are watched in the same session. The set of related videos is extended with videos on distance $n$ away from the videos in the watched videos set in order to add more novelty and serendipity to the set. The resulting set of videos is ranked according to the quality of the video, how closely it matches the user's preferences and how much diversity the video adds to the set. In the end the recommendation engine improved former methods including 'Most Viewed' and 'Top Rated' with a staggering 200% in click through rate (CTR), where CTR is measured as the amount of users clicking on the recommended video over the total amount of users that see that video link.

The second paper was published in 2016 and provides a high level overview of the renewed recommendation system [18]. Similar to the previous approach, the system is split in two parts, candidate generation and ranking. The candidate generation utilizes a deep neural network with an embedding for the user's watch and search history combined with geographical and user information as input. The output of this network is a video vector containing the preferences of the user and is compared to video vectors in a nearest neighbor approach to generate a top-$n$ list of videos. These videos are then ranked using another deep neural network, which incorporates more features of the video in the embedding to rank the list. This contribution showed the potential of using deep neural networks for recommendations, while the split in two stages allows for using an ensemble of methods for candidate generation.

The third paper extends upon the second paper as they use an ensemble of candidate generation methods including their own from the second paper [54]. The main contribution of this paper was the extension of the ranking system, which is not as relevant for this research. However, this paper does highlight the strength of using the separate stages for recommendations since both parts can be extended or improved on their own.

### 3.2.1 Wide and Deep networks

As mentioned in the previous section, YouTube has shifted from traditional methods towards Deep Learning. However, one of the problems when applying deep neural networks in recommendation is the lack of relevant suggestions when the network is generalizing too much. This generalization can occur when the rating matrix is sparse and thus all new items will be given a low predicted value, making the differences between truly valuable content and rubbish small.

An important contribution addressing this issue was proposed in [12] in the form of a Wide and Deep network. The authors suggest that deep neural networks, i.e. neural networks consisting of multiple hidden layers, perform great when generalizing recommendations. However, deep neural networks can overfit on the sparse data leading to less relevant recommendations. On the other hand, linear models with a wide array of features are capable of making relevant recommendations but are less likely to provide diversity. Linear models require more exhaustive feature engineering using cross-products of features, e.g. a feature that is set to 1 only if two other features are also 1 or a feature that is 1 only when either of two other features are 1.

In order to overcome the problems of the individual approaches, the authors propose to combine a linear model and a deep neural network in one model. The wide linear part, consisting of a single layer, and the deep part of the model are jointly trained on the input data. The final label predicted by the wide and deep model is a linear combination of the output from the wide and the deep part.

The output of the wide part of the model is a bias term $b$ summed with the result of the weight vector $\mathbf{w}_{wide}$ applied on the raw features $\mathbf{x}$ and the cross-product feature transformations of those features. A single cross-product feature transformation, $\phi_k$, can be defined as the product of relevant feature values. Let $c_{ki}$ be a binary indicator of whether feature $x_i$ is relevant to cross-product transformation $\phi_k$, then $\phi_k$ can be computed as:

$$\phi_k(\mathbf{x}) = \prod_{i=1}^{d} x_i^{c_{ik}}.$$

When using binary features the transformation is 1 if and only if all relevant features are 1 as well. Using such cross-product transformations the output can be computed as:

$$y_{wide} = \mathbf{w}_{wide}^{\top}[\mathbf{x}, \phi(\mathbf{x})] + b.$$

The output of the deep part of the model has to be computed layer by layer. In order to compute the output of layer $l$ the weight vector $\mathbf{w}_{deep}^{(l-1)}$ needs to be applied on the output of the previous layer, $a^{(l-1)}$. The result is combined with the bias of layer $l-1$, $b^{(l-1)}$, and used as input of an activation function $f$:

$$a^{(l)} = f(\mathbf{w}_{deep}^{(l-1)\top} a^{(l-1)} + b^{(l-1)}).$$

Therefore, the output of the deep part of the model can be computed as follows, assuming that $l$ is the final layer of the network:

$$y_{deep} = f(\mathbf{w}_{deep}^{(l)\top} a^{(l)} + b^{(l)}).$$

The result of the wide and deep model can now be computed using a linear combination of both outputs and a bias term. When using an activation function $g$ the output becomes:

$$y = g(y_{wide} + y_{deep} + b).$$

This combination of a simple linear model and advanced deep model is capable of capturing both low- and high-order interactions. However, there is a disadvantage of this model when comparing it to regular deep models. Deep models are known for their lack of feature engineering, since the model performs the feature engineering itself when provided with just the raw features. The linear part of the wide and deep model needs to have predefined advanced features, making the inclusion of the cross-product transformation function $\phi$ necessary. Including the right transformations can be a hassle and as such increases the complexity of this model. A schematic example of a wide and deep architecture used for recommending apps in the Google Play Store can be seen in figure 8.

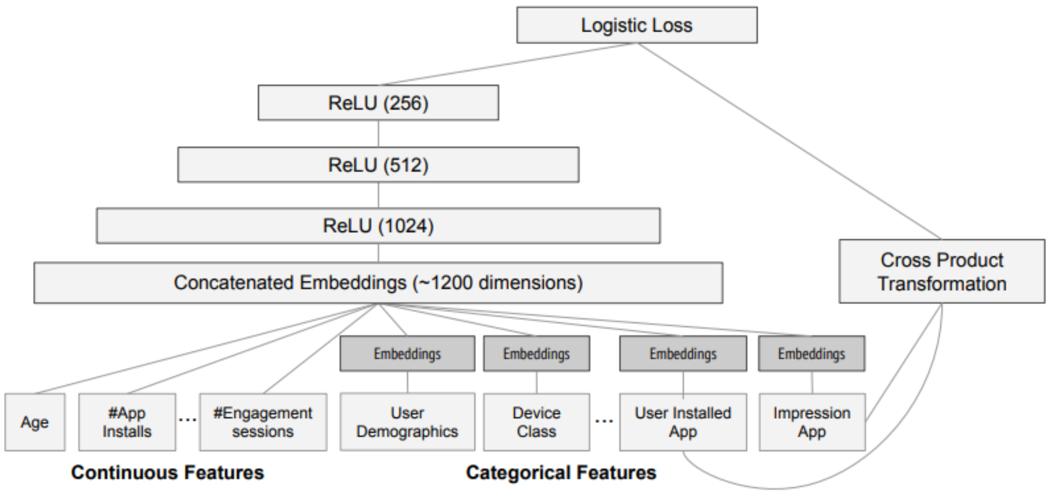


Figure 8: Wide and Deep network for app recommendation as featured in [12]. The deep part of the network consists of three fully connected layers with ReLU activation function. The input of the deep part consists of numerical features such as age and categorical features such as device class. The categorical features require embeddings in order to transform them from textual to numerical features. The wide part of the network creates cross-product transformations between the apps the user has installed and the apps the users are shown. The outputs of both parts are combined to compute one shared logistic loss.

## 3.3 Conclusion

The existing literature shows that plenty of studies have been performed regarding recommendation systems. Even though some studies have already attempted to apply such recommender systems to fashion retail, most of these are focused around e-commerce or the replenishments of the in-store stockrooms from the warehouses or distribution centers. As such this research is novel in the sense that it has not been researched before what recommendations can be made for in-store replenishments using solely the data of the store itself. Especially the limitations introduced by the missing sales data have not been studied before. The main lessons learned from the related work concern the use of wide and deep networks for recommendation, which have shown promising results in various areas including grocery retail. Furthermore, the mapping of stores as users and articles as items has proven to be viable, which will be taken into account for this study.

# 4 Methodology

## 4.1 Overview

The methodology can be roughly divided into three phases, namely the preparation of the data, the determination and training of the models and the evaluation of the trained model. A high-level overview of the process can be seen in figure 9. The details of the individual steps will be highlighted in the remainder of this section.

## 4.2 Data preparation

### 4.2.1 Choose retailer

For this research the data of a single retailer was used to train and compare the models with. Using a single retailer mimics the way a model would be initially used in practice, since this avoids any issues regarding retailers not willing to have their data used for the benefit of another retailer. The choice of retailer was made based on the accessibility of the stores and the nature of the data. Store accessibility was determined based on the connections Nedap has with the retailer and the presence of at least one store in the Netherlands that could be visited during the research. Assessing the nature of the data is obviously prone to subjectivity. Since this research can be considered exploratory, the decision was made to choose a retailer that uses refill on a regular basis with a widely varying set of articles that might appear in bigger quantities on the sales floor. It would be interesting to also look into retailers that have a limited amount of articles available or retailers that wish to only have one item available per article, but these can be considered edge cases and are therefore unsuitable for this exploratory study. Taking these measures into account a retailer was chosen, from now on referred to as retailer X.

A second retailer was chosen in order to test the model trained on retailer X. The test concerns measuring the performance of a model solely trained on one retailer when applying it on a second retailer. Since this second retailer is only used for verifying the final model it was not that important where the retailer was located and how many counts were performed, as long as there were enough counts to match the requirements of the feature vector. With these considerations a second retailer was chosen, from now on to be referred to as retailer Y.

There are several differences between retailer X and Y, the main one being their respective count frequency. Retailer Y counts more frequently than retailer X and thus has more up-to-date stock information. This will most likely influence the way they are currently using refill and in extension alter the nature of their data. Especially this difference in the nature of the data will make it interesting to see how well the model is able to adjust its predictions. The exact evaluation goals for retailer Y will be discussed in more detail in section 4.4.7.

### 4.2.2 Pre-process data

Generating refill recommendations requires count data, i.e. the results of a scan the retailer does to count the amount of items that are available in the stockroom and on the sales floor. A count entry is created for every count of either a stockroom or a sales floor and holds information regarding the GTINs and quantities observed during that count. Since retailers count the stockroom and sales floor simultaneously or closely after one another, the count

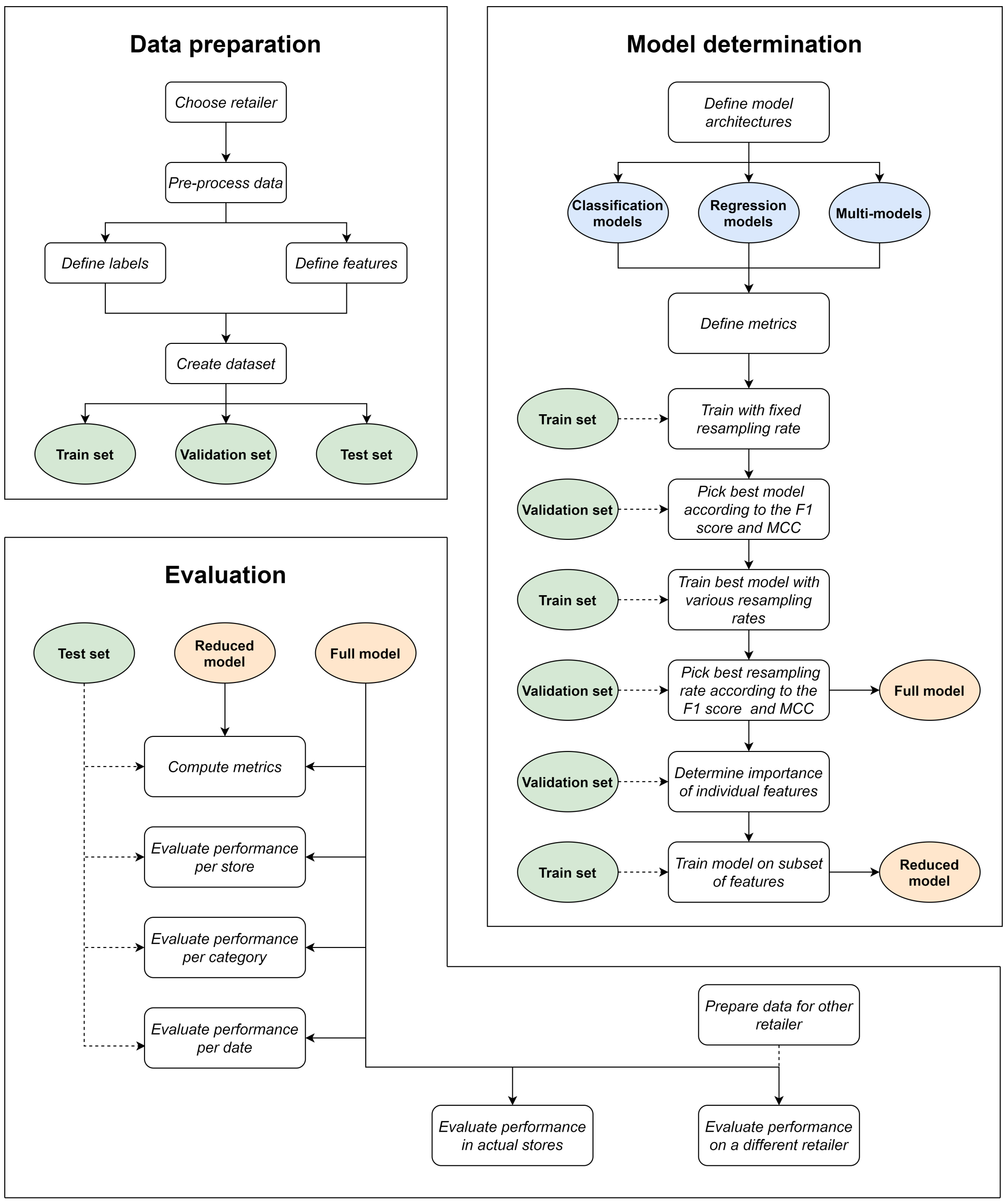

Data preparation
Choose retailer
Pre-process data
Define labels
Define features
Create dataset
Train set
Validation set
Test set
Model determination
Define model architectures
Classification models
Regression models
Multi-models
Define metrics
Train set
Train with fixed resampling rate
Validation set
Pick best model according to the F1 score and MCC
Train set
Train best model with various resampling rates
Validation set
Pick best resampling rate according to the F1 score and MCC
Full model
Validation set
Determine importance of individual features
Train set
Train model on subset of features
Reduced model
Evaluation
Test set
Reduced model
Full model
Compute metrics
Evaluate performance per store
Evaluate performance per category
Evaluate performance per date
Prepare data for other retailer
Evaluate performance in actual stores
Evaluate performance on a different retailer

Figure 9: A high-level overview of the methodology of this research. The large blocks indicate the different phases of the methodology, whereas the smaller blocks indicate actions taken. The ellipses indicate relevant outputs of actions used as input for future actions. Even though in theory every action results to an output of some kind, only the most important once are listed for brevity.

entries can be grouped per day in order to provide a representation of the stock inside a store. At this point it has become possible to determine for a store how many items of each article are present on the day of a count for both the stockroom and the sales floor.

From the list of articles the unique GTINs can be extracted, which can be used to retrieve additional article information. For every unique GTIN the gender, category, size, season, color and option of that article are retrieved. It does occur that a retailer assigns a new GTIN to an existing article, e.g. when a new batch arrives or the seasons change. Since it is essentially the same product, the data from these separate GTINs will be aggregated to prevent misleading data. If an article does not have any additional information, either intentional or due to this information being deleted by the retailer, empty strings will be used as a default.

The article and count information can be combined to create a table with detailed stock information. For each store the count data of the sales floor and the stockroom are combined per GTIN and date. If a certain GTIN does not appear in the stockroom whereas it does appear on the sales floor, or vice versa, the quantity for the area where it was not counted will be set to zero. Using this aggregation the count rows can be extended into a table where each store, date and GTIN combination has a separate row with the corresponding data. This unique combination will be used as an identifier for a single labeled data point.

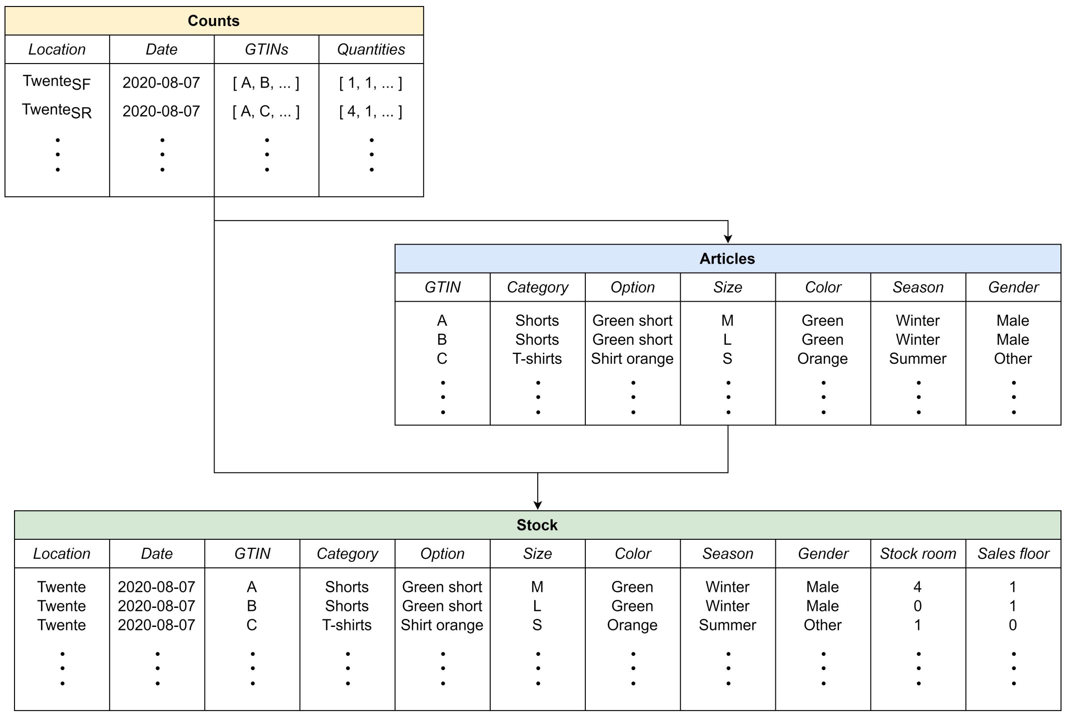
The data structure and the processing is visualized in figure 10. Relevant properties of retailer X have been summarized in table 2.

Table 2: Summary of the dataset used in the research. Some values have been given as ranges or approximations to prevent direct pointers to the retailer used.

| **Retailer X** | |
|---|---|
| Amount of stores | 50 - 100 |
| Amount of articles | 30.000 - 40.000 |
| Amount of product categories | 26 |
| Amount of counts | 5.000 - 7.500 |
| Date of first count | March 2019 |
| Average amount of items on the sales floor per article | 1.7 |
| Amount of stock entries | 8.000.000+ |
| Average time between counts | 9 days |
| Percentage of stores that count every week | 20% |
| Percentage of stores that count at least once every two weeks | 90% |
| Percentage of counts preceded by another count at most a week prior | 60% |

### 4.2.3 Define labels

There is no predefined set of labels to be predicted since no previous work has been done using this dataset. There are roughly two possibilities, namely predicting the exact quantity that should be on the sales floor for an article or predicting what quantity should be refilled



| Counts | | | |
|---|---|---|---|
| *Location* | *Date* | *GTINs* | *Quantities* |
| Twente$_{SF}$ | 2020-08-07 | [ A, B, ... ] | [ 1, 1, ... ] |
| Twente$_{SR}$ | 2020-08-07 | [ A, C, ... ] | [ 4, 1, ... ] |
| ⋮ | ⋮ | ⋮ | ⋮ |

| Articles | | | | | | |
|---|---|---|---|---|---|---|
| *GTIN* | *Category* | *Option* | *Size* | *Color* | *Season* | *Gender* |
| A | Shorts | Green short | M | Green | Winter | Male |
| B | Shorts | Green short | L | Green | Winter | Male |
| C | T-shirts | Shirt orange | S | Orange | Summer | Other |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |

| Stock | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| *Location* | *Date* | *GTIN* | *Category* | *Option* | *Size* | *Color* | *Season* | *Gender* | *Stock room* | *Sales floor* |
| Twente | 2020-08-07 | A | Shorts | Green short | M | Green | Winter | Male | 4 | 1 |
| Twente | 2020-08-07 | B | Shorts | Green short | L | Green | Winter | Male | 0 | 1 |
| Twente | 2020-08-07 | C | T-shirts | Shirt orange | S | Orange | Summer | Other | 1 | 0 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |

Figure 10: Simplified view of the data structures involved in preparing the data for feature generation. Columns that do not influence the refill recommendations, like identifiers, have been omitted. Some values have been shortened for readability.

to reach that target quantity. In the end both options lead to the same result but differ regarding the processing of the label to its final usage. When predicting the desired sales floor quantity, getting to a list of refill recommendations involves subtracting the desired quantity from the actual quantity. The advantage of this method is that the desired quantities can immediately be used for other use cases, like providing insights for store managers in how ‘optimal’ their store is. However, since the goal of this research is to generate a list of refill recommendations, the labels will concern the quantity that should be refilled as this can be used directly.

Determining the labels themselves is a challenge on its own since retailers differ greatly concerning what they want. Every store manager has an opinion, which frequently differs from the demands of the headquarters. Therefore, rather than doing field research and questioning employees about the desired refill quantities, the quantities will be derived from the data.

Three rules are defined which are used for generating labels. These rules are based on the assumption that the stockroom and sales floor counts of the subsequent count can be used to determine the label of the current count. For the rules the following notation will be used:

$$\begin{aligned} q_a^{(c)} &= \text{quantity of article } a \text{ on count } c \text{ on the sales floor,} \\ p_a^{(c)} &= \text{quantity of article } a \text{ on count } c \text{ in the stockroom,} \\ y_a^{(c)} &= \text{quantity of article } a \text{ to refill after count } c \text{ (i.e. the label).} \end{aligned}$$

A label distribution applying all three rules is shown below:

$$y_a^{(c)} = \begin{cases} \min(p_a^{(c)},\ q_a^{(c+1)} - q_a^{(c)}) & \text{if } q_a^{(c+1)} > q_a^{(c)} \\ \min(p_a^{(c)},\ 1) & \text{if } q_a^{(c+1)} = 0, \\ & \quad q_a^{(c)} > 0, \\ & \quad p_a^{(c+1)} - p_a^{(c)} \neq q_a^{(c)} \\ 0 & \text{otherwise.} \end{cases}$$

Before discussing the different cases it is important to note that these labels consider the refill quantity at count $c$. In order to generate the best labels, information from a future count will be used, count $c+1$, to essentially determine what in hindsight should have been the correct refill recommendation. With that in mind the rules will be discussed in more detail below.

The first rule is based on the assumption that if a larger amount of items is counted on the sales floor on count $c+1$ as compared to count $c$, a refill has taken place. The refill recommendation for that article after count $c$ should thus be to refill the difference between those two quantities, upper-bounded by the amount of items available in the stockroom. Intuitively this means that the information from future counts can be used to make assumptions about what has occurred after the current count. In this case that means that if the quantity of items has increased on the sales floor after the current count it has apparently been refilled and should thus be seen as a refill recommendation for the current count.

The second rule is focused on preventing NOSBOS articles from occurring. Currently, the refill algorithm is based on reacting to a NOSBOS article, i.e. refill when an article quantity reaches zero. Contrary to reacting to NOSBOS articles, this rule is used to target articles that had a non-zero quantity at the time of the first count, but have a zero quantity at the next count. The final condition for this rule is used to capture scenarios where a store has removed the article from the sales floor on purpose. To prevent recommending these articles, the condition checks whether the stockroom quantity has increased with the same amount as the quantity on the sales floor has decreased between counts $c+1$ and $c$. In case these conditions are met, the refill recommendation will be to refill one item. Naturally it is also possible to recommend more than just one item, but that could lead to problematic overstocking and more assumptions on the desired quantity. As such the decision was made to bound this rule at one.

If none of the aforementioned conditions are met, the quantity to refill should be zero, i.e. no action needs to be taken. It is highly unlikely that all edge cases will be captured using these rules, since the counts are only snapshots and all actions taken in between these snapshots are unknown. The rules try to approximate reality as much as possible, but every rule has several known deficits. The first rule will often be an underestimate of the actual quantity to be refilled, since the intermediate sales are not taken into account. The second rule suffers from overstocking since the NOSBOS can occur anywhere between the two counts whereas the refill recommendations will be given immediately after the first count. Even though the deficits are known, adjusting the label generation accordingly would introduce other deficits, essentially nullifying the use of the adjustments.

Applying these rules on the dataset yields labels with quantities ranging from 0 to 50. The label distribution across the different quantity values are plotted in figure 11. It can easily be seen that the label set is heavily skewed towards zero with over 90% of the labels being zero, which confirms the expected sparsity of the dataset. Quantities higher than 3 are extremely rare, accounting for only 0.17% of the labels. Considering that indicating whether an article should be refilled is more important than specifying the exact quantity, the labels are bounded on 3. The label set is still imbalanced, the effects of which will be discussed in more detail in section 4.3.3. Applying this upper-bound to the label generation leads to the final formula for computing labels:

$$y_a^{(c)} = \begin{cases} \min(p_a^{(c)},\ q_a^{(c+1)} - q_a^{(c)},\ 3) & \text{if } q_a^{(c+1)} > q_a^{(c)} \\ \min(p_a^{(c)},\ 1) & \text{if } q_a^{(c+1)} = 0, \\ & \quad q_a^{(c)} > 0, \\ & \quad p_a^{(c+1)} - p_a^{(c)} \neq q_a^{(c)} \\ 0 & \text{otherwise.} \end{cases}$$

### 4.2.4 Define features

Every unique combination of GTIN, date and location leads to a separate feature vector. Features originate from both article and store properties. In this section an overview will be given of what features were considered, a long-list of which can be found in appendix B.

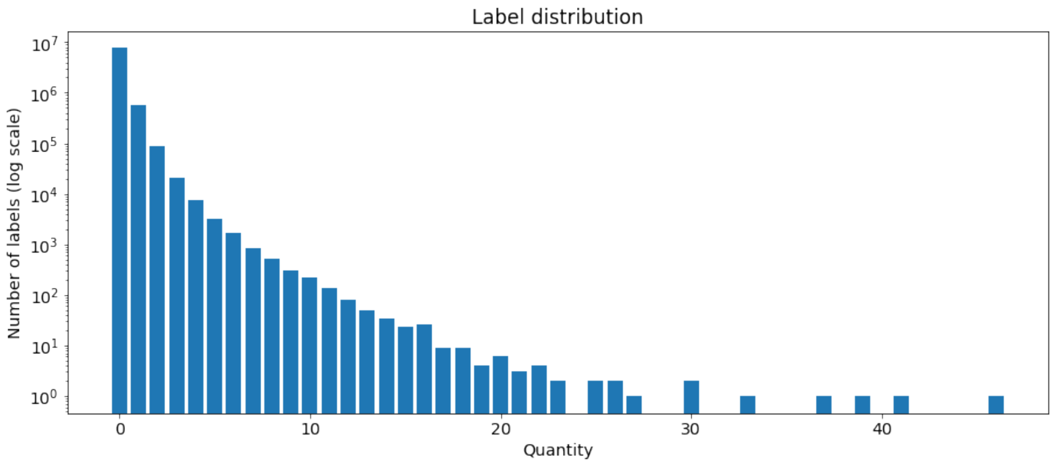


Figure 11: The label distribution of the dataset, clearly indicating the large difference in label counts. The y-axis is plot with log scale for readability.

### Numerical features

The first features to consider are article specific features per store. Originating from the count data, the sales floor quantity as well as the stockroom quantity of an article can be derived. When looking at the option that article belongs to, there are multiple numerical features that can be used. The minimum, mean and maximum sales floor quantity of an article within that option as well as the amount of articles belonging to that option are used as features. Lastly, the current stack size for the option, i.e. the total amount of items of that option on the sales floor, might be an important feature given that store appearance is important to the store manager and is therefore also included.

Looking at the store specifically leads to the features indicating the total amount of items on both the sales floor and in the stockroom as well as the ratio between these two quantities. Those features give a decent idea about the size of the store and what quantities are reasonable to refill. When taking this one step further, features can also be derived based on the count data of other stores. To get a good indication of what articles are currently trending, the count data of other stores within the past 7 days of the feature vector date will be taken into account. From that data a feature is added computing the mean sales floor quantity of the article across all stores. In an effort to also include seasonality directly in the feature set, the current month of the year is added as a feature.

Adding the month as a feature is not as simple as encoding the month as a number between 0 and 11. Such an approach would not capture the distance between December and January correctly. Instead the month value can be converted into a coordinate on a circle by taking the sine and cosine of the encoded month value. The circular properties of the sine and cosine ensure that January is actually close to December. Aside from just taking the sine and cosine of the month value, the values should be normalized to fit in the range $[0, 2\pi]$. Consequently the month value is spread across two feature values using equations 2 and 3.

$$month_{\cos} = \cos\left(\frac{2\pi \cdot month}{12}\right) \tag{2}$$

$$month_{\sin} = \sin\left(\frac{2\pi \cdot month}{12}\right) \tag{3}$$

To exploit the data available, the decision was made to include the aforementioned numerical features for a total of three counts, namely the current count and the two counts preceding that count. For the quantity means, which depend on other stores, the counts are grouped per three days to at one hand increase the amount of counts to average and on the other hand prevent a store that counted two weeks earlier from influencing the current result. By including more historical data, the expectation is that trends can be derived more easily, which should also help tackling the seasonality aspect of the problem.

All numerical features are scaled to the range [0, 1] to prevent features like the total number of items on the sales floor dominating the model training and preventing proper convergence. Furthermore, by including historical data of up to two counts ago, it is required for a store to have at least two counts to fully utilize the potential of the model. These values could be filled using baselines or averages to at least be able to generate refill recommendations, but this might lead to unexpected results. Therefore, only the samples with at least two previous counts will be taken into account.

### Binary features

The binary features mainly concern specific cases which are especially relevant for refill. These cases include a binary feature indicating whether the article is still available in the stockroom, a feature indicating whether the article is still available on the sales floor and whether the option itself is still available on the sales floor. The first two binary features are combined into an additional feature, which indicates whether an article is NOSBOS or not. In an effort to see how influential these binary features can be, a feature is added indicating whether the article is a shoe. The reason this is a separate feature is because shoes tend to show different behavior than articles from other categories, since there are a lot of sizes which are usually not put on the sales floor while they are available in the stockroom. The last binary feature indicates whether the article has more items on the sales floor than the other articles of that same option. This feature essentially indicates whether this article is the most popular size.

### Categorical features

The categorical features concern both article and store properties. When considering what aspects might contain useful information, an identifier for the store should definitely be included since it is to be expected that stores differ much from each other. Store-specific behavior could be achieved by training a model per store, but that would remove the advantage of exploiting store similarities, which could be used to for example identify popular articles. Adding store demographics as features is another possible solutions, but these are not trivial to identify. Instead, including an identifier for a store allows the models to learn similarities between stores based on the training data of both stores.

When looking at the articles themselves, the gender, size, option, category and age group of an article also have the potential to be relevant features since each of these properties tend to impact the way an article is usually displayed. Information from categories or options can be used to further deduce whether the article is relevant for the current season. Aside from providing useful information for the article itself, this information can also aid making sensible predictions about new articles with little historical data by exploiting the similarities between these articles. For each article the GTIN is also added as a feature, which in theory makes it possible for the model to learn which articles are similar when it comes to their respective refill recommendations.

These features cannot be added to the model as is, since they are not numerical and thus require an encoding. One possibility would be to one-hot encode the categorical values. One-hot encoding means creating a vector of all zeros with the length of the amount of categories and only setting the corresponding category to 1. Whereas this is a feasible solution for small categories, like gender, encoding stores or options like this will lead to large sparse feature vectors, which does not help when training a model. A second possibility would be to simply encode the categories in ascending order. While this does not increase the length of the feature vector, it does make it possible for the model to learn relationships between categories based on the number they have been assigned, while the order of these numbers has no meaning.

A third possibility, also the one used for this research, is to use embeddings to represent the categorical variables. Feature embeddings is a technique borrowed from natural language processing which attempts to find a compact representation of high-dimensional data [44]. An embedding is in essence a vector of a predefined size of floating point numbers. Each category receives its own embedding, i.e. its own vector of numbers. Initially, the numbers were generated using statistical techniques, but the numbers can also be learned using neural networks [24, 48, 52]. The network can learn the similarities between different categorical variables on its own during training, creating similar embedding vectors for similar categories. By including these mappings from category to embedding vector as a layer in the network, the embeddings can be jointly trained with the rest of the model, providing little overhead. Since the embedding vectors are of a fixed size, high-dimensional features such as the category of an article or the identifier of a store can be used as a feature. These learned similarities essentially represent the collaborative filtering part of recommender systems since the similarities between seemingly unrelated stores are derived from the distance of their embedding vectors. The size of the embedding vectors can be chosen freely and there is not a single correct answer for picking this size. For this research the following formula will be used, which was recommended in a blog post by Google[6]:

$$\text{number\_of\_dimensions} = \sqrt[4]{\text{number\_of\_categories}}.$$

One issue to overcome with any of the three approaches mentioned earlier is dealing with unseen values. These values did not occur in the training data so they have no representation yet. In order to overcome said issue, a special token will be added to the training set representing unseen values. Training this token without any adjustments will not lead to the desired effect since it has no meaning. Instead, during every epoch of training the token will be assigned to a random set of samples, replacing their original value. This

[6]`https://developers.googleblog.com/2017/11/introducing-tensorflow-feature-columns.html`

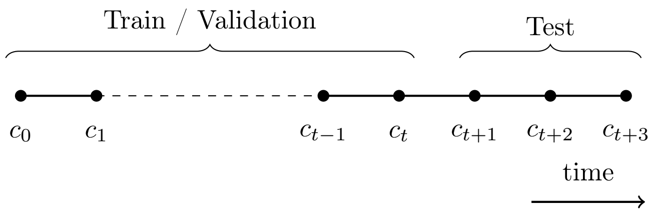


Figure 12: The division of the dataset per store. The test data consists of the last three counts per store, whereas the training and validation data is retrieved from the other counts. For this store the last count was performed at time $t + 3$, hence the test set for this store consists of counts $c_{t+1}$, $c_{t+2}$ and $c_{t+3}$. The remaining data is split in a 90:10 ratio for training and validating.

approach ensures that the samples in the training set are not continuously trained using the wrong embeddings while at the same time creating a sensible embedding for the token. Given the continuous stream of new data, it is to be expected that many unseen values will be introduced, especially during a collection change. Therefore, aside from learning a representation for the unseen values, the model itself should be retrained often enough to not get overwhelmed by new values. This will be discussed in more detail in section 8.

### 4.2.5 Create dataset

Using the pre-processed data and the definitions for the labels and the features, the dataset can be created. As is common, the dataset has to be split in a train, validation and test set. From the dataset the test set should be extracted first. Randomly sampling the dataset in order to create a test set is not suitable since information about these samples might still be included in the training dataset. As an example consider article $a$ at count $c_t$ for store $s$ to be included in the test set and that same article at count $c_{t+1}$ for store $s$ to be included in the training set. Since the label generation uses information from the next count, the expected label of the sample in the test set can be inferred from the training set.

Instead the test set consists of all samples in the last three counts of every store. Not only does this prevent information from the test set leaking into the training set, this also closely resembles what the models would encounter in practice, since it is unlikely that the model will be retrained after every count. After excluding the three counts per store for the test set, the remaining data will be split in a 90:10 ratio for training and validating. A graphical representation of this split is shown in figure 12.

## 4.3 Model determination

### 4.3.1 Define model architectures

There are multiple ways to deal with the prediction of recommendation quantities. For this research three different methods will be used to generate the recommendation quantities. The three methods to verify are classification, regression and a combination of both. The methods and their expected pros and cons will be discussed below.

### Classification

Using classification to predict the quantities boils down to estimating the following function, where $\mathbf{x}$ indicates the feature vector and $y$ marks the target class:

$$f(\mathbf{x}) = \arg\max_i P\left(y = i|\mathbf{x}\right), \quad i \in \{0, 1, 2, 3\}.$$

One of the advantages of classification is that it is rather straightforward and a widely studied area within machine learning. Furthermore, classification can return probabilities per class. These probabilities can be used to not only provide a quantity but also give a certain score or confidence to that prediction. Such scores can be used to rank predictions, by for example setting a certain threshold which needs to be passed before the recommendation is to be used.

A disadvantage of classification is that one loses the ordinality of the different classes, i.e. the notion that a quantity of 2 is closer to 1 than to 0. Another disadvantage originates from the different objectives. The first objective, deciding whether it should actually be a refill recommendation, is more important than predicting the exact quantity. This importance is not reflected in the classification target of regular classification. In order to reflect the differences of importance the loss function used for training will be modified. Using $\mathbf{y}$ as the target vector of zeros with only a one at the position of the target class $i$ and $\mathbf{p}$ as the vector of class probabilities returned by the model, the loss function is as follows:

$$\mathcal{L}(\mathbf{y}, \mathbf{p}) = \begin{cases} -\log(p_0) - \sum_{i=1}^{3} \log(1 - p_i) & \text{if } y_0 == 1 \\ \\ -\log(1 - p_0) - \sum_{i=1}^{3} y_i \log(p_i) & \text{if } y_0 == 0. \end{cases}$$

This loss function, from now on referred to as the modified cross-entropy loss, ensures that the probability the model assigns to instances of classes 1, 2 and 3 being class 0 is punished. Furthermore, the probability the model assigns to samples of class 0 being other classes will also be punished.

For classification two models were defined, one custom implementation and one scikit-learn model. The scikit-learn model will be used as a baseline to test how well the custom implementation fares against an out-of-the-box model. The scikit-learn model chosen is the default MLPClassifier[1] since it outperformed other scikit-learn models in small-scale experiments on a random subset of the data. The custom model is a Wide and Deep network with a similar layout to the one used in the original paper [12]. A summary of the architectures are shown in table 3 and a visual representation of the custom model is shown in figure 13. The training parameters will be discussed in section 4.3.3.

### Regression

A different approach would be to predict real values using regression. The function to be approximated will then have the following form, using $\mathbf{x}$ for the feature vector and $y$ for the target quantity:

$$f(\mathbf{x}) = y, \qquad y \in [0, \infty).$$

[1]`http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier`

Table 3: Classification model architectures. The custom approach is a Wide and Deep approach and therefore consists of two values for both the hidden layers and the activation. It is important to note that the activation functions concern the activation functions used for the hidden layers, for both models the final activation function is the softmax function. The hidden layer sizes of the custom approach were determined by performing several small-scale experiments on a random subset of the data, whereas the hidden layer size for the scikit-learn model was left at its default value. A visual representation of the custom model can be found in figure 13.

| **Model** | **Hidden layers** | **Activation** |
|---|---|---|
| Classification - scikit-learn | (100) | ReLU |
| Classification - custom | -, (128, 64, 32) | Linear, ReLU |

Even though quantities larger than 3 are non-existent in the label set, it is possible to recommend them using regression. However, a properly trained model should not do so even if it is possible. An advantage of using regression is the immediate conversion of the result to a quantity, which can be performed by rounding the result to the nearest integer. Furthermore, loss functions frequently used for regression, such as the Mean Average Error (MAE) and the Root Mean Squared Error (RMSE), also take the ordinality of the quantities into account.

A disadvantage of using regression is that there is no easy way to measure confidence or score a certain quantity. Surely one could conclude that a predicted quantity of 1.3 is more likely to actually be 1 than 1.1, but it is not such a useful indicator as the probabilities are for classification.

For regression two models were defined, one custom implementation and one scikit-learn model. The scikit-learn model is the default MLPRegressor[2] and can be viewed as the baseline for regression. The custom model is a residual network with shortcut connections in every layer. A summary of the architectures are shown in table 4, whereas the training parameters will be discussed in section 4.3.3. The layer sizes and amount of layers were chosen using the same method used for the classification network. A visual representation of the custom regression network is shown in figure 14.

Table 4: Regression model architectures. The custom approach is a residual network and contains skip connections within every layer. The architecture is shown in more detail in figure 14.

| **Model** | **Hidden layers** | **Activation** |
|---|---|---|
| Regression - scikit-learn | (100) | ReLU |
| Regression - custom | (128, 64, 32) | ReLU |

[2]`http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor`

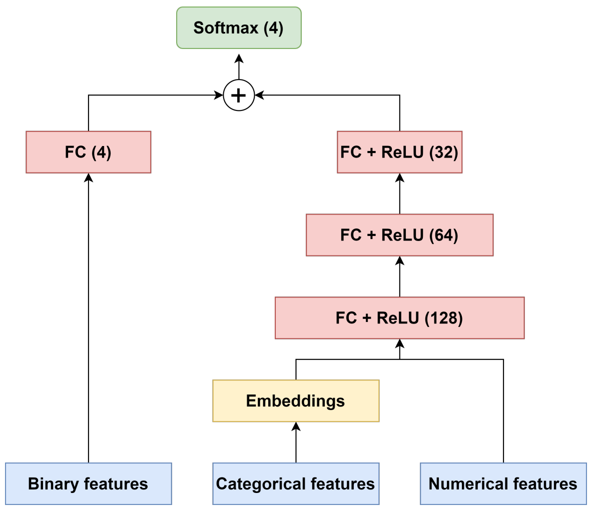


Figure 13: Network architecture of the custom classification model. The binary features form the wide part of the network, going straight to the output node via a Fully Connected (FC) layer without activation function. The categorical features are first translated into embeddings via an embedding layer and then used as input for the deep part of network together with the numerical features, passing 3 fully connected layers with ReLU activation. The output of the wide and the deep part are summed and used as input for the Softmax activation function.

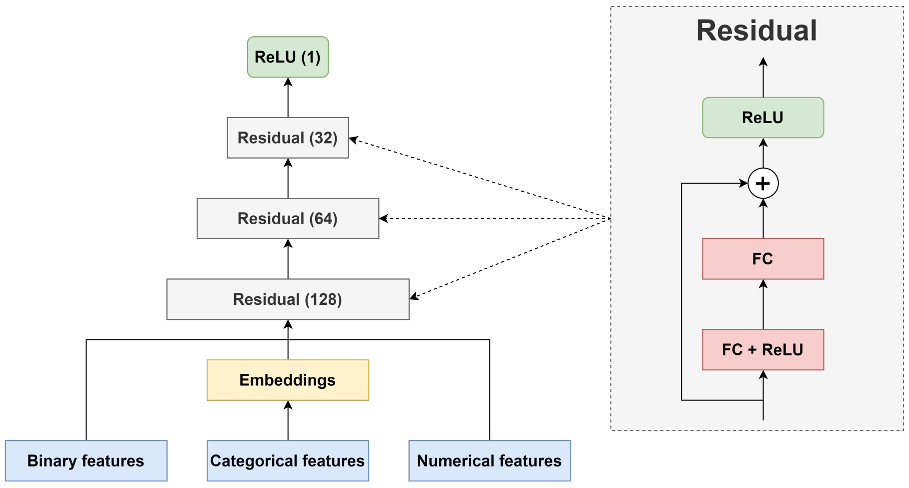


Figure 14: Network architecture of the custom regression model. The categorical features are first translated into embeddings via an embedding layer and then used as input together with the binary and numerical features for the residual network. Inside a residual block the input passes through one Fully Connected (FC) layer with ReLU activation and then one FC layer without activation. After the second FC layer the original input is added to this output and used as input for the final ReLU activation function.

**Multi-model**

The final option is to combine both classification and regression by training two separate models. The first model classifies an instance as 0 or 1, 0 being 'no-action required' and 1 being 'should-refill'. The results from the first model are used as filter to limit the samples used as input for the second model, which predicts the refill quantity in the range 1 - 3. Using $\mathbf{x}$ as the input vector, $t$ as a predefined threshold with which the model should accept a sample as a refill recommendation, $y_1$ as the binary indicator whether an article should be refilled and $y_2$ as the actual quantity to refill, the formulas for the two models are as follows:

$$f_1(\mathbf{x}) = \begin{cases} 1 & \text{if } P(y_1 = 1|\mathbf{x}) \geq t, \quad t \in [0, 1] \\ 0 & \text{otherwise} \end{cases}$$
$$f_2(\mathbf{x}_{f_1(\mathbf{x})==1}) = y_2, \qquad y_2 \in [1, \infty).$$

The first model produces the probability that the article should be refilled, based on feature vector $\mathbf{x}$. If that probability, $P(y_1 = 1|\mathbf{x})$, is larger than threshold $t$, function $f_1$ returns 1, otherwise the article should not be refilled and the function returns 0. Using the results from the first model, the second model determines the quantity to refill, $y_2$, for that article, i.e. $\mathbf{x}_{f_1(\mathbf{x})==1}$.

The biggest advantage of this setup is that the implementation directly reflects the split objectives. Since the second model is only used for predicting quantities that should actually be refills, the second model is not as heavily affected by the skewed label distribution as the first model. The first model can be implemented in the way recommender systems work, since these also deal with extreme sparsity and predicting whether to recommend or not. Furthermore, similar to the approach the authors proposed in the third paper discussing the recommender system developed for YouTube [54], multiple models can be used to provide recommendations to the second model, which could eventually lead to different specialized models.

Aside from the useful multi-model setup, this method is also capable of changing the amount of refill recommendations predicted without retraining the model. Since the first model is capable of producing a probability, the threshold value with which that article is actually passed on to the second model can be altered. If it turns out the model recommends too many or too few articles, the threshold can be increased or decreased accordingly. Naturally this does alter the nature of the samples the second model receives, so it should be noted that unexpected behavior could technically occur when drastically changing the threshold.

The disadvantage of using two models is the fact that two models need to be trained and saved. Training one proper machine learning model is resource-intensive, requiring approximately a day to achieve convergence on the relatively small dataset of retailer X, let alone training two. Considering that seasonality plays a big role for fashion retailers and that large amounts of data are added every week, it is likely that the models need to be retrained at a regular interval. It should be noted that the training time mentioned was observed while using a high end laptop, so improved training times are to be expected when training on a cluster.

For the multi-model approach the previously defined models for regression and classification will be combined and slightly altered. As such the baseline consists of the scikit-learn MLP-Classifier and MLPRegressor, whereas the custom approach consists of a Wide and Deep classification network and a residual regression network. Contrary to the original classification models the final activation function is a sigmoid activation indicating whether the article should be refilled. A visual overview of the entire custom model is shown in figure 15, whereas a textual representation of the architecture can be found in appendix A. The hyperparameters will be discussed in section 4.3.3.

An overview of all the different models used can be found in table 5.

Table 5: Overview of the models used. The loss functions are (Modified) Cross-Entropy ((M)CE), and Mean Squared Error (MSE).

| **Model id** | **Model type** | **Model 1** | **Model 2** | **Loss 1** | **Loss 2** |
|---|---|---|---|---|---|
| class-custom | Classification | WideDeep | - | MCE | - |
| class-scikit | Classification | MLPClassifier[1] | - | MCE | - |
| reg-custom | Regression | ResNet | - | MSE | - |
| reg-scikit | Regression | MLPRegressor[2] | - | MSE | - |
| mm-custom | Multi-model | WideDeep | ResNet | CE | MSE |
| mm-scikit | Multi-model | MLPClassifier[1] | MLPRegressor[2] | CE | MSE |

### 4.3.2 Define metrics

In order to reflect the split objectives, i.e. indicating whether an article should be refilled and predicting the quantity to refill, metrics are defined for both binary classification as well as regression. The metrics used are shown in the list below:

- Accuracy
- Precision
- Recall
- F1 score
- Matthews Correlation Coefficient (MCC)
- Cross-Entropy loss (CE)
- Mean Absolute Error (MAE)
- Root Mean Squared Error (RMSE)
- Fraction of NOSBOS articles recommended.

The accuracy can indicate how well the model is trained and whether it is able to cope with the skewed set of labels. The precision, recall, F1 score and MCC are used to determine how well the models perform on identifying whether an article should be refilled. The CE, MAE and RMSE are used to indicate how well the models predict the actual quantity to be refilled. The fraction of NOSBOS articles recommended is used to quickly compare the models to the current refill.

[1] `http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier`
[2] `http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor`

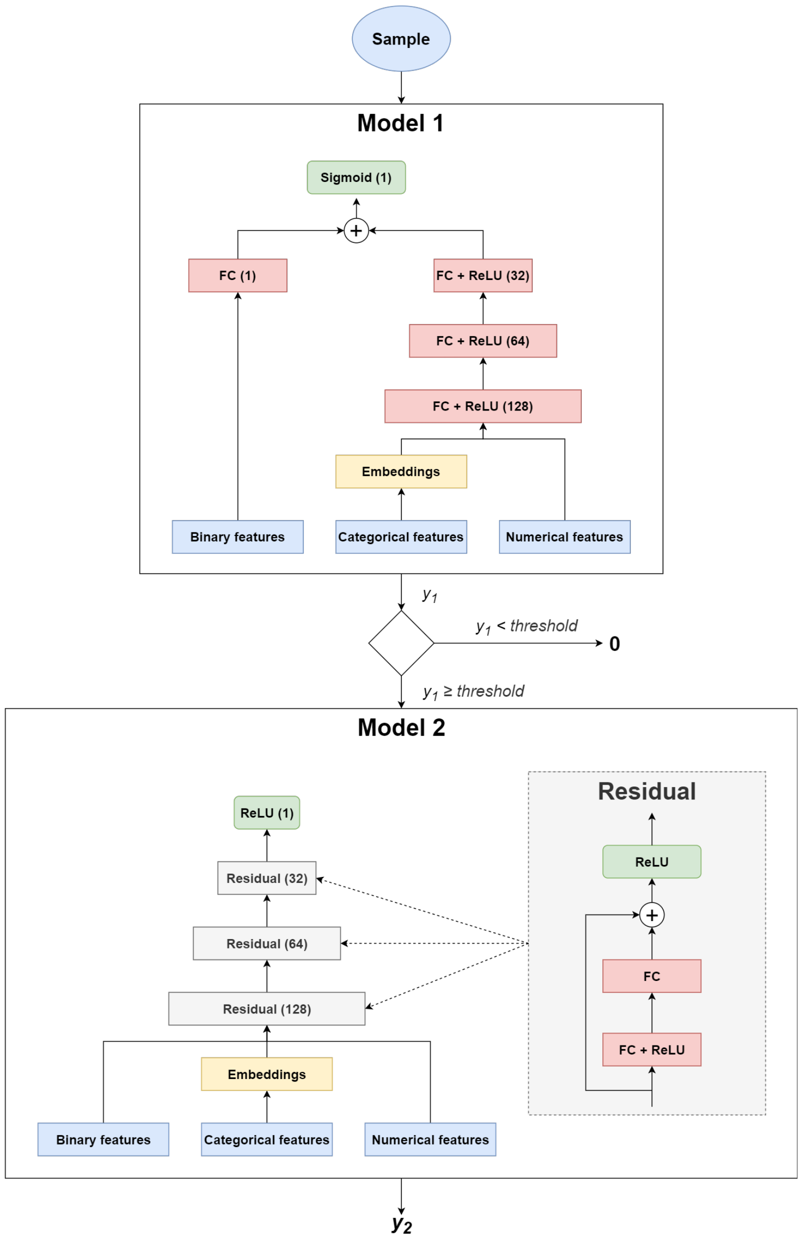


Figure 15: Network architecture of the custom multi model setup. The features of the sample are used as input for the first model. Similarly to the classification model, the features are split in binary, categorical and numerical features and passed through the first model. The result is a value $y_1$ which ranges between 0 and 1. If $y_1$ is less than a predefined threshold $t$ the quantity to refill, i.e. the result of the model, is 0. Otherwise the features of the sample will be used as input for the second model, which produces a quantity to refill, $y_2$, using a similar network as the one used for the custom regression approach.

### 4.3.3 Train with fixed resampling rate

In order to properly compare the model implementations, they were all trained using the same dataset. The validation data was computed using the same seed and consists of 10% of the dataset, where the test data has already been removed. As mentioned before, the training data is largely skewed, with over 90% of the samples being articles that should not be refilled, i.e. samples with label 0. There are a variety of ways to combat this imbalance, the most common ones being downsampling the majority class and upsampling, or resampling, the minority class. There is no clear answer regarding which of the two is better, but for this research the decision was made to resample the minority class to prevent the risk of losing valuable information. Even though there are ways to prevent such loss of valuable information when downsampling, resampling provided a safer alternative. Determining the optimal resampling rate is of big influence to the training of the model and should be optimized for the final model. However, since the effect of choosing a bad resampling rate will roughly be equal for all model implementations, the first training will be done using a fixed resampling rate of 0.25, i.e. for every four zero samples one non-zero sample will be included. This value was chosen after performing small-scale experiments with the scikit-learn classification model on various resampling rates and judging the model convergence.

All the models were trained using Adam [28] as optimizer for a total of 50 epochs. In order to determine the learning rate, batch size and L2 regularizer a grid search was performed. For the learning rate the values 0.01, 0.001 and 0.0001 were used, for batch size 32 and 128 and for the L2 regularizer the values 0.001, 0.0001 and 0.00001 were used. At the end of each epoch the validation loss was computed and training was stopped if the validation loss had not decreased for 5 epochs in a row. The parameters that led to the best results during training are listed in table 6.

Table 6: Hyperparameter setup for training the different models.

| **Parameter** | **Value** |
|---|---|
| Learning rate | 0.001 |
| Batch size | 128 |
| L2 regularizer | 0.001 |
| $\beta_1$ | 0.9 |
| $\beta_2$ | 0.999 |
| $\epsilon$ | 1e-8 |

### 4.3.4 Pick best model

The trained models can all be used on the validation set. The best performing model will be decided based on the average of the MCC and F1 score, since the F1 score is a good indicator of the performance of the models on the positive class whereas the MCC also takes the unbalance of the dataset into account. Taking the average of the two leads to making a decision based on both the performance specifically for the positive class as well as the balanced performance. The other metrics listed in section 4.3.2 will also be included to provide a better insight in what model implementation excels at which part of the task.

### 4.3.5 Train best model with various resampling rates

Continuing with the best performing model implementation, new models are trained using a variety of resampling rates. Using the hyperparameters listed in table 6, models are trained using resampling rates of 0, 0.2, 0.25, 0.3, 0.5, 0.75 and 1.0 on the training set.

### 4.3.6 Pick best resampling rate

Similar to picking the best model implementation, the different models trained using different resampling rates can be compared by looking at the average of the MCC and the F1 score. These are computed by running the different models on the validation set. The model corresponding to the resampling rate that achieves the highest average of the MCC and the F1 score will be used for the remainder of this research.

### 4.3.7 Determine importance of individual features

Aside from looking at the performance of the model using all of its features, the importance of the individual features can also be computed. Using these feature importances a new model can be trained on a subset of the features to determine what performance decay, if any, occurs when using less features. Calculating the attributions of the individual features can be performed using Shapley Value Sampling [11, 45, 46]. The Shapley Value originates from cooperative game theory as the unique solution satisfying three axioms of fairness for payoffs. The details can be found in the aforementioned papers, but short descriptions of the axioms are as follows:

| | |
|---|---|
| *Symmetry* | If players $i$ and $j$ are interchangeable, their payoffs should be as well |
| *Null player* | If a player does not contribute to the game, the payoff should be 0 |
| *Additive* | The payoff for playing two independent games should equal the payoff of playing both games together. |

This approach can be linked to feature importance by considering the change in the output produced by the model with or without a certain feature. If for some sample a feature is set to a baseline value, usually zero, and the output of the model does not change, that feature was apparently not of importance for that sample. However, this is a naive approximation of the attribution of that feature for the sample, since its attribution might also depend on the other feature values. In order to properly compute the attribution for a given sample, all the feature values should be set to some baseline. From this baseline the individual features of the sample will be added in a random order, or permutation, one-by-one. The difference in output from before and after the addition of each feature value will be the attribution of that feature for this permutation. The overall attribution of that feature for the given sample can then be determined by considering all possible permutations, i.e. all possible orders in which the features can be added, and averaging all the individual attributions belonging to the permutations. These attributions can then be used to explain the behavior of the model for that given sample.

Instead of only computing these attributions for a single sample, they can also be computed for the entire set of samples. For a given feature the attributions can be calculated per sample using the previously described method. Taking the average of the absolute values of these attributions gives a value indicating the average influence of that feature on the output of the model. Computing these values for all features allows for a direct comparison between the added values of the different features, which can in turn be used to select the top-$n$ most important features.

Ideally, one would compute the feature attribution for all possible permutations for all samples. However, since the amount of permutations per sample scales exponentially with the amount of features, this takes an infeasible amount of time. Instead, only 100 permutations will be computed per sample. The entire procedure is written as pseudocode in algorithm 1.

There are several advantages of using Shapley Values instead of more traditional approaches such as information gain. Firstly, the Shapley Values can be computed using a fully trained model and provide insight in the contributions of the features. Other feature selection methods are either based on the dataset itself or performed during training by iteratively dropping or adding features. The contributions of the features can not only be used to determine a subset of features on which another model can be trained, but also for explaining why the model makes certain decisions [31]. Since the Shapley Values are computed based on permutations of the input features and the change in output, these can be computed regardless of the machine learning approach used [42].

The second, more important, advantage of Shapley Values is that they are computed using permutations of the input features. This means that the contribution of a feature also take into consideration the added value of that feature in combination with other features [1]. Such an approach allows feature contributions to be recognized that would have been neglected if only their individual contribution would be considered [47].

For this research the Shapley Values will be computed using the best performing model and resampling rate on the samples in the validation set. The resulting list of feature contributions are split per target class, which provides insight in what features are important for individual targets, e.g. what features are important for samples with target label 0 and what features are important for samples with target label 1.

### 4.3.8 Train model on subset of features

Using the previously determined feature contributions, a feature selection can be made by selecting the top $n$ most important features. For this research a new model is trained using the fifteen most important features. The model will be trained using the same hyperparameters as seen in table 6 and using the same model implementation and resampling rate as the model using the complete set of features. The resulting model will be used alongside the model trained on the complete set of features for the initial evaluation on the test set.

**Algorithm 1** Compute feature attribution of features $f_0, f_1, \dots, f_{n-1}$ for model $m$ based on samples $s_0, s_1, \dots, s_{k-1} \in S$

```
 1: $b \leftarrow [0, 0, \dots, 0]$                          ▷ set baseline feature vector to all zeros
 2: $b_{out} \leftarrow m(b)$                    ▷ compute model output for the baseline feature vector
 3: $res \leftarrow [0, 0, \dots, 0]$                  ▷ initialize total attributions to all zeros
 4: for sample $s \in S$ do                                              ▷ loop over all samples
 5:     $P \leftarrow [\pi_0(s), \dots, \pi_{99}(s)]$            ▷ create 100 random permutations of features
 6:     $attr \leftarrow [0, 0, \dots, 0]$          ▷ set feature attributions to zero for this sample
 7:     for permutation $p \in P$ do
 8:         $x \leftarrow b$                                        ▷ set feature vector to baseline
 9:         $out_{old} \leftarrow b_{out}$                              ▷ set output to baseline output
10:         for feature value $v_i \in p$ do
11:             $x[i] \leftarrow v_i$                                    ▷ set feature $i$ to value $v_i$
12:             $out_{new} \leftarrow m(x)$                                    ▷ compute new output
13:             $attr[i] \mathrel{+}= out_{new} - out_{old}$                     ▷ add feature attribution
14:             $out_{old} \leftarrow out_{new}$                                     ▷ set new output
15:         end for
16:     end for
17:     for $a_i \in attr$ do
18:         $res[i] \mathrel{+}= |\frac{a_i}{n}|$     ▷ add the absolute average feature attributions to the total
19:     end for
20: end for
21: for $i \leftarrow 0$ to $n - 1$ do
22:     $res[i] \mathrel{/}= k$                    ▷ average attribution over the amount of samples
23: end for
```

## 4.4 Evaluation

### 4.4.1 Compute metrics

The first evaluation step concerns running both the model trained on the full set of features as well as the model trained on the reduced set of features on the test set. Furthermore, the current refill implementation will also be executed on the test set in order to facilitate a proper comparison with the current situation. The metrics computed in this step are listed in section 4.3.2. Aside from the list of metrics, visualizations will be made using ROC curves and confusion matrices to provide more insight in what errors the models make.

The metrics generated in this way present a good way of comparing the individual models, but the representation of reality can be improved. Currently, the test set still includes samples that would be automatically disregarded when used in a real-world scenario. To investigate whether these filtered samples are of influence to the performance of the model on the test set, the test set itself will be filtered using the same criteria. The samples that will be filtered concern articles that are not available on the stockroom, which can thus not be refilled, and options which are entirely not available on the sales floor.

Unless there is practically no difference between the performance of the entire test set and the filtered test set the remainder of the experiments will be performed using the filtered test set.

### 4.4.2 Evaluate performance per store

In order to determine whether the model's performance deviates between stores, the samples in the test set will be grouped per store. This enables the computation of metrics per store as well as other indicators such as the fraction of options recommended. Combined with store demographics this can be used to analyze what kind of stores are more difficult for the model to handle and what stores are easier.

### 4.4.3 Evaluate performance per category

Since there are big differences between individual categories, e.g. shoes and raincoats, one might expect big differences in model behavior when dealing with different categories. Especially considering the way current refill disregards differences between categories when predicting the quantities, it is interesting to see whether the newly trained model does differentiate between the categories. Performance per category as well as the fraction of options recommended per category and the errors made per category should give a good indication of how well the model deals with categorical differences.

### 4.4.4 Evaluate performance per date

Given that the test set contains the last three counts per store, the performance of the model on the test set can be split across the count number of the store. Depending on how frequently a store counts, the individual counts may not be that far away from each other, but it still remains interesting to investigate whether the performance at count $c_{t+1}$ differs greatly from the performance at count $c_{t+3}$. This can be done by taking a closer look at the list of metrics, the details of which are discussed in section 4.3.2.

### 4.4.5 Evaluate performance in actual stores

The post-processed options to refill can be used in order to get an indication of how well they are received by the stores themselves. Getting enough results for making statistical significant claims is not possible using the current setup, which is fine considering that the goal of the evaluation is getting an idea of whether the new recommendations are promising. Essentially this can be seen as a validation step of the label generation. The online evaluation consists of two phases, one concerns providing a couple of options to refill remotely via e-mail whereas the other concerns actually visiting a store.

For the e-mail results the goal is to make it as easy as possible for a store manager to provide feedback. Naturally one cannot expect the store manager to respond when sending a list with all options that were recommended by the model each and every count. Therefore, in order to get a good overview of the potential of the new implementation, the decision was made to include only six options per e-mail. The six recommendations are divided in three groups:

2 options unique to the new method

2 options unique to the current method

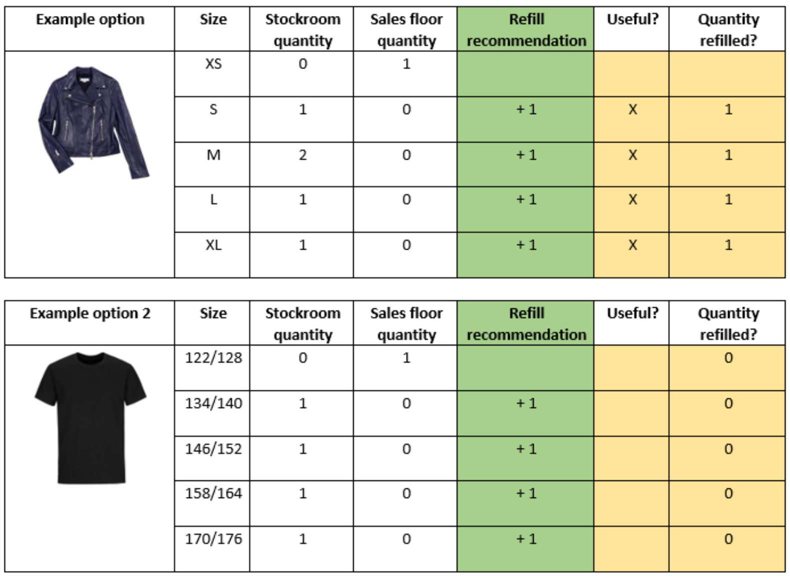
2 options shared by both methods.

An example entry of an option within the e-mail can be seen in figure 16. There is only one limitation on the options send using these e-mails, namely shoes. Shoes is a category which is almost guaranteed to never make a proper refill recommendation while containing lots of NOSBOS articles. The current refill method will recommend shoes as an option to refill, but it will not lead to interesting results. Therefore, shoes are excluded from the suggestions send in the e-mail.

Aside from the e-mails a visit to an actual store will also be executed. The setup will be to generate the refill recommendations as soon as possible after the store has performed a count. With the generated list of recommended options in hand, the list will be assessed with the store manager or employee while executing the options to refill from that list. Since this can be seen as a replacement for their regular refill schedule, more time can be invested in working through this list. Therefore, the setup is slightly different from the one used in the e-mail, with a long-list of all options to refill grouped by category visualized on a simple web page. The layout of said web page is similar to that of the e-mail, albeit being significantly easier to create since this can be fully automated. The layout is visualized in figure 17.

### 4.4.6 Prepare data for other retailer

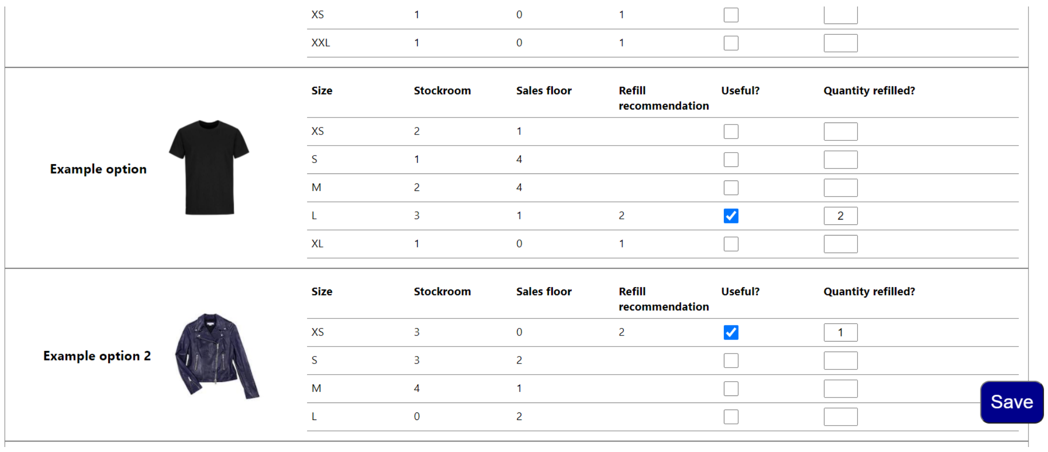
Essentially this phase requires the same steps to be performed as the data preparation of the original retailer, discussed in section 4.2. Instead of retrieving all the data for retailer Y, only the last three counts per store were retrieved. This ensures that the data originates from approximately the same time frame as the test data from retailer X. The same filters will be applied as were applied to the test set of retailer X to ensure its resemblance to real-life refill.



| Example option | Size | Stockroom quantity | Sales floor quantity | Refill recommendation | Useful? | Quantity refilled? |
|---|---|---|---|---|---|---|
| | XS | 0 | 1 | | | |
| | S | 1 | 0 | + 1 | X | 1 |
| | M | 2 | 0 | + 1 | X | 1 |
| | L | 1 | 0 | + 1 | X | 1 |
| | XL | 1 | 0 | + 1 | X | 1 |

| Example option 2 | Size | Stockroom quantity | Sales floor quantity | Refill recommendation | Useful? | Quantity refilled? |
|---|---|---|---|---|---|---|
| | 122/128 | 0 | 1 | | | 0 |
| | 134/140 | 1 | 0 | + 1 | | 0 |
| | 146/152 | 1 | 0 | + 1 | | 0 |
| | 158/164 | 1 | 0 | + 1 | | 0 |
| | 170/176 | 1 | 0 | + 1 | | 0 |

Figure 16: Example entries of e-mail recommendations to stores.



| | | | | | |
|---|---|---|---|---|---|
| XS | 1 | 0 | 1 | ☐ | |
| XXL | 1 | 0 | 1 | ☐ | |

Example option

| Size | Stockroom | Sales floor | Refill recommendation | Useful? | Quantity refilled? |
|---|---|---|---|---|---|
| XS | 2 | 1 | | ☐ | |
| S | 1 | 4 | | ☐ | |
| M | 2 | 4 | | ☐ | |
| L | 3 | 1 | 2 | ✓ | 2 |
| XL | 1 | 0 | 1 | ☐ | |

Example option 2

| Size | Stockroom | Sales floor | Refill recommendation | Useful? | Quantity refilled? |
|---|---|---|---|---|---|
| XS | 3 | 0 | 2 | ✓ | 1 |
| S | 3 | 2 | | ☐ | |
| M | 4 | 1 | | ☐ | |
| L | 0 | 2 | | ☐ | |

Figure 17: Example entries of recommendation at store visit.

### 4.4.7 Evaluate performance on a different retailer

Similarly to the evaluation of the model on the test set of retailer X, the performance of the model on retailer Y will be focused around the metrics of section 4.3.2. The evaluation will be performed on the entire dataset, since no training has been performed on this data. Furthermore, this will be extended by including confusion matrices and ROC curves to create a more detailed view of the strengths and weaknesses of the model. The results of this evaluation are used to draw conclusions about the usability of a trained model on a different retailer, which might prove valuable for performing quick tests or making retailers aware of the possibilities of this approach.

# 5 Results

## 5.1 Model implementation

The results of the offline evaluation of all the different model implementations when run on the validation set are shown in table 7.

Table 7: Results of the evaluation per model implementation when run on the validation set. The details of the model implementations can be found in section 4 and table 5.

| | **class-custom** | **class-scikit** | **reg-custom** | **reg-scikit** | **mm-custom** | **mm-scikit** |
|---|---|---|---|---|---|---|
| **Accuracy** | 0.90 | **0.91** | 0.87 | 0.85 | 0.90 | 0.90 |
| **Precision** | 0.43 | **0.47** | 0.38 | 0.34 | 0.45 | 0.44 |
| **Recall** | 0.47 | 0.31 | 0.61 | **0.64** | 0.54 | 0.31 |
| **F1 score** | 0.45 | 0.37 | 0.47 | 0.45 | **0.49** | 0.37 |
| **MCC** | 0.40 | 0.34 | 0.42 | 0.40 | 0.44 | **0.45** |
| **CE Loss** | 1.34 | **1.22** | 1.62 | 1.85 | 1.31 | 1.28 |
| **MAE** | 0.34 | 0.31 | 0.17 | 0.21 | 0.18 | **0.11** |
| **RMSE** | 0.60 | 0.59 | **0.34** | 0.36 | 0.40 | 0.37 |
| **% NOSBOS** | 45.4 | 30.8 | 53.3 | 58.0 | 47.0 | 26.7 |
| $\mathbf{\frac{F1+MCC}{2}}$ | 0.43 | 0.36 | 0.45 | 0.43 | **0.47** | 0.41 |

Across the entire line the custom models outperform the out-of-the-box scikit-learn models when looking at the average of the F1 score and MCC, even though the scikit-learn classification model does achieve the best performance on accuracy and precision. The caveat with the scikit-learn classification model is that especially the recall takes a big hit, being much lower than that of the custom model. When comparing the different approaches, the custom classification approach is performing worse than the custom regression approach. Interestingly, the custom regression approach even outperforms the custom classification approach when it comes to the F1 score and MCC, only being beaten by the classification approach for the Cross-Entropy loss.

Comparing the regression approach to the combined approach shows the strength of combining classification and regression. The metrics show that the combined model achieves better results on the binary classification metrics than the regression approach, while also achieving similar results on the regression metrics. Especially the MCC, which is not influenced by the unbalanced dataset, is superior for both the scikit-learn and the custom multi-model approaches when compared to the classification and regression approaches.

One could argue that the differences between the approaches might be considered too small. However, it should be noted that the values were obtained using a validation set of approximately 700.000 samples randomly sampled from the original dataset. Given such a large set of samples, even tiny differences in performance can be viewed as significant from a practical point of view, since it requires a large amount of samples to increase, or decrease, said performance. When looking at the results of the two best performing models on the average of the F1 score and the MCC, the *reg-custom* and *mm-custom* models, there are

Table 8: Contingency table of the *reg-custom* and *mm-custom* models showing the results of both models on the validation set.

| | *reg-custom* correct | *reg-custom* incorrect |
|---|---|---|
| *mm-custom* correct | 650814 | 40253 |
| *mm-custom* incorrect | 19105 | 57363 |

several major differences between the other metrics. Even though the F1 score only differs 0.02 between the two approaches, the origin of this difference can be found in the larger differences between the precision and recall, which are 0.07 and -0.07 respectively.

The different behavior of the two models can also be shown using a statistical test. In this case the McNemar test can be used, as recommended by Dietterich for use on binary classifiers trained on a single validation set [20, 33]. The McNemar test compares the predictive accuracy of two models for paired nominal data by looking at the disagreements between the models. As such the test does not show whether one model performs better than the other, but instead is used to show that the models show significantly different behavior. The hypotheses can thus be formulated as follows:

$$H_0 : \text{there is no significant difference between the models,}$$
$$H_1 : \text{there is a significant difference between the models.}$$

The test statistic can be calculated using a 2x2 contingency table as seen in table 8. Since all values from the contingency table are at least 25, the test statistic has a Chi-squared distribution with 1 degree of freedom and can be calculated as follows:

$$\chi^2 = \frac{(40253 - 19105)^2}{40253 + 19105} \approx 7535.$$

The resulting p value is $< 0.01$, so the result is significant for $\alpha = 0.01$ and the null hypothesis $H_0$ can be rejected, meaning that there is a significant difference between the behavior of the *reg-custom* and *mm-custom* model.

All in all this leads to the conclusion that for the remainder of this research the custom multi-model combination of a classification and regression model will be used for answering the remaining research questions, i.e. the *mm-custom* model.

## 5.2 Resampling rate

Continuing with the model *mm-custom*, i.e. a combination of a Wide and Deep model and a residual model, the next parameter to tweak is the resampling rate. As mentioned in section 4.3.3, the models were trained using the hyperparameter specifications of table 6 using resampling rates 0, 0.2, 0.25, 0.3, 0.5, 0.75 and 1.0. Given that the model implementation is a multi-model, the threshold with which to accept a sample as a refill recommendation should also be tweaked. Since increasing the amount of non-zero samples in the training data will lead to a larger amount of samples being accepted as refill recommendations, the optimal threshold which needs to be passed in order to accept such a sample will differ per resampling rate. To determine the best threshold, all the models were evaluated using thresholds in the range [0, 1] to find what threshold led to the maximum average of the

MCC and F1 score when applied on the validation set. Logically, decreasing the threshold leads to a higher recall and a lower precision. Therefore, the F1 score and MCC are a proper balance to find a good threshold. The results of resampling rates as computed on the validation set are shown in table 9.

Table 9: Results of the offline evaluation for the WideDeep-ResNet model combinations, *mm-custom*, using different resampling rates and acceptance thresholds when run on the validation set.

| | **0.0** | **0.20** | **0.25** | **0.30** | **0.50** | **0.75** | **1.0** |
|---|---|---|---|---|---|---|---|
| **Threshold** | 0.20 | 0.40 | 0.45 | 0.50 | 0.60 | 0.70 | 0.75 |
| **Accuracy** | 0.83 | 0.86 | **0.88** | 0.86 | 0.84 | 0.86 | 0.86 |
| **Precision** | 0.32 | 0.36 | **0.41** | 0.35 | 0.33 | 0.34 | 0.34 |
| **Recall** | 0.65 | 0.62 | 0.62 | 0.59 | **0.68** | 0.60 | 0.60 |
| **F1 score** | 0.42 | 0.46 | **0.50** | 0.44 | 0.44 | 0.44 | 0.44 |
| **MCC** | 0.37 | 0.42 | **0.45** | 0.39 | 0.40 | 0.39 | 0.38 |
| **CE Loss** | 2.14 | 1.72 | **1.49** | 1.76 | 2.0 | 1.83 | 1.80 |
| **MAE** | 0.25 | 0.21 | **0.19** | 0.23 | 0.27 | 0.27 | 0.28 |
| **RMSE** | 0.52 | 0.46 | **0.42** | 0.47 | 0.51 | 0.50 | 0.51 |
| **% NOSBOS** | 72.4 | 63.0 | 54.0 | 60.2 | 67.3 | 60.0 | 59.6 |
| $\frac{\mathbf{F1+MCC}}{\mathbf{2}}$ | 0.40 | 0.44 | **0.48** | 0.42 | 0.42 | 0.42 | 0.41 |

The results confirm the suspicion that an increased resampling rate also demands an increased threshold to optimize the performance. The resampling rate of 0.25 performs best for all metrics except recall, where it performs third best, and the percentage of NOSBOS articles recommended. The percentage of NOSBOS is a proper indicator measuring how close the new method is to the old approach, but it is no deal-breaker when comparing the individual new models. When looking at the significance of the differences between the models the same arguments can be made as the ones from the previous section, this time with even larger differences than the ones observed in table 7. Given these considerations, the results calculated in the remainder of this research are based on the *mm-custom* model with resampling rate 0.25 and threshold 0.45.

To get an idea of how influential the threshold value actually is, the accuracy, precision, recall, F1 score and MCC have been plotted for the model with resampling rate 0.25 across different thresholds in figure 18. The figure reaffirms the choice of the threshold value 0.45.

## 5.3 Feature importance

The Wide and Deep model has two different parts with two different targets, while the residual network consists of one part with three different targets. In the end this means a total of seven feature importance bar charts can be computed. An example of such a bar chart is shown in figure 19, the others have been included in appendix B for brevity. The bar charts use abbreviations for the features in order to improve readability. The abbreviations of the individual features have been described in appendix B. The top 5 most important features for the wide portions of the model and the top 10 most important features for the deep part are listed in tables 10 and 11 respectively.

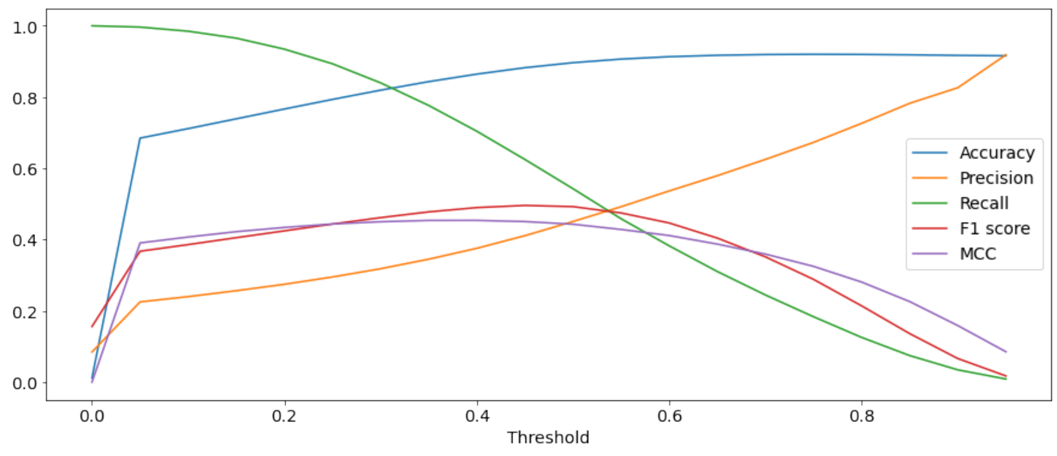


Figure 18: Accuracy, precision, recall, F1 score and the MCC for different thresholds using model *mm-custom* with a resampling rate of 0.25. Looking at the F1 score and the MCC, one can see the maximum of the average occurs at a threshold of approximately 0.45.

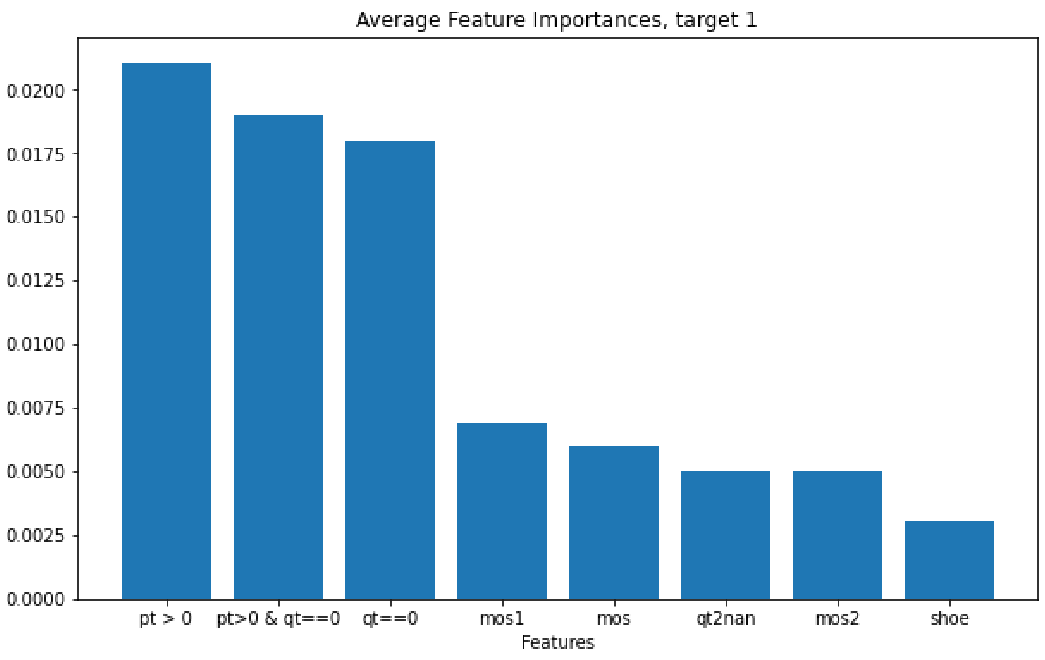


Figure 19: Feature attributions calculated using Shapley Value Sampling from the wide part of the model targeting samples with label 1. *pt* and *qt* refer to the stockroom and sales floor quantity at the target count, whereas *mos* stands for a binary indicator whether this article is the most available out of all articles within its option. *mos1* and *mos2* are the same as mos but then for earlier counts. Lastly, the *shoe* feature indicates whether the article belongs to the category shoes and *qt2nan* is an indication of whether enough historic counts were available to properly compute the other features.

Table 10: Top 5 most important features of the wide part of the model per target class. *pt* and *qt* refer to the stockroom and sales floor quantity at the target count, whereas *mos* stands for a binary indicator whether this article is the most available out of all articles within its option. *mos1* is the same as *mos* except that it is computed for the count before the target count.

| | **Wide target 0** | **Wide target 1** |
|---|---:|---:|
| **pt > 0** | **0.007** | **0.021** |
| **pt > 0 & qt == 0** | 0.006 | 0.019 |
| **qt == 0** | 0.006 | 0.018 |
| **mos1** | 0.006 | 0.007 |
| **mos** | 0.005 | 0.006 |

Table 11: Top 10 most important features of the deep part of the models per target class. *cat*, *age* and *loc* are embeddings for the category, age group and store identifier respectively. *pt* and *qt* refer to the stockroom and sales floor quantity at the target count. *qtmin* and *qtmax* refer to the minimum and maximum sales floor quantity of the article of the last three counts. *opta* and *optavg* refer to the availability and the average article quantity of the option belonging to the article. Lastly, *dm* refers to the average sales floor quantity of the article at the given date across all stores.

| | **Deep (0)** | **Deep (1)** | **ResNet (1)** | **ResNet (2)** | **ResNet (3)** |
|---|---:|---:|---:|---:|---:|
| **pt** | **0.205** | 0.021 | **0.138** | **0.483** | **0.833** |
| **qt** | 0.104 | **0.250** | 0.012 | 0.066 | 0.095 |
| **loc** | 0.068 | 0.104 | 0.015 | 0.056 | 0.105 |
| **opta** | 0.043 | 0.069 | 0.007 | 0.017 | 0.030 |
| **cat** | 0.027 | 0.035 | 0.011 | 0.034 | 0.046 |
| **age** | 0.017 | 0.031 | 0.006 | 0.019 | 0.030 |
| **dm** | 0.028 | 0.041 | 0.004 | 0.015 | 0.036 |
| **optavg** | 0.028 | 0.038 | 0.002 | 0.010 | 0.019 |
| **qtmin** | 0.035 | 0.057 | 0.003 | 0.012 | 0.016 |
| **qtmax** | 0.040 | 0.048 | 0.003 | 0.011 | 0.018 |

Both for the wide and the deep part of the model the features dealing with stockroom quantities are the dominating features. For the wide part the binary feature indicating whether the article is still available on the stockroom, $pt > 0$, is the feature attributing most to the output for both the samples with target label 0 and 1. However, for samples with target label 0 the differences are negligible. When classifying samples with target class 1, the wide part is more heavily influenced by the features indicating NOSBOS, paying little attention to the features indicating whether the article is the most available article on the sales floor for that option. Interestingly, the specific feature targeting shoes is not that important for both targets.

The deep part of the model also mainly focuses on the stockroom quantity when classifying samples of class 0. However, when dealing with samples of class 1 the deep part is largely influenced by the sales floor quantity with the second and third place going to the store embedding and the availability of the option. Interestingly, both the store embedding and the category embedding are of a large influence to the model, meaning that it does extract valuable information from these properties. Comparing these feature attributions to the ones from the wide part of the model also suggest that the decisions of the classification model are largely based on the deep part of the model. Even the feature contributing the most from the wide part of the network is only contributing as much as the 10th most important feature of the deep part.

When looking at the regression part of the model, the stockroom quantity is the most dominating. This is to be expected since the stockroom quantity provides an upper limit on the quantity that can be refilled. Once again the embedded features are also relatively important, especially the store embedding is the second, third and second most important feature for targets 1, 2 and 3 respectively. Even though the most importance stems from the stockroom quantity, the fact that the store embedding is an important feature is promising given that one would expect differences in refill quantities based on what store is targeted. Furthermore, the current sales floor quantity and category embedding are also important, which is to be expected when refilling an article and the categorical differences. All in all this is a promising result since the features that one might expect to be important are also of importance.

Even though some feature attributions are fairly small, especially for the wide part of the classification network, the attributions are based on a large number of samples. Hence these averages present a clear view of the dominating features of the model as well as what features are not at all influential. Using these averages the features from tables 10 and 11 were selected for the training of the model with the reduced set of features.

## 5.4 Test set

### 5.4.1 Overall performance

In this section the results of the model on the test set will be examined in further detail in order to identify potential improvements or common mistakes. The model with the complete feature set, the model with the reduced feature set and the current refill algorithm are all executed on the test set. The resulting metrics, as defined in section 4.3.2, are shown in table 12.

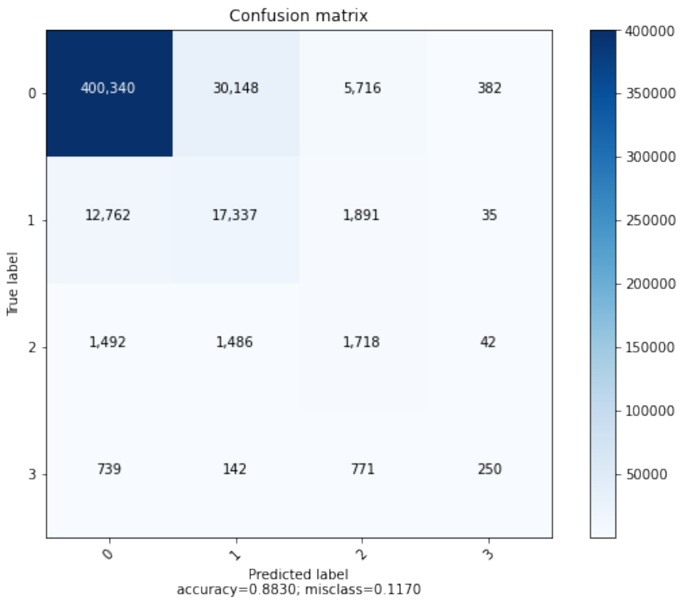
Table 12: Comparison of the current refill algorithm, the model trained on the complete feature set and the model trained on the reduced feature set when executed on the test set.

| | **Model (full features)** | **Model (reduced features)** | **Current refill** |
|---|---|---|---|
| **Accuracy** | **0.88** | 0.87 | **0.88** |
| **Precision** | **0.40** | 0.35 | 0.29 |
| **Recall** | **0.61** | 0.58 | 0.32 |
| **F1 score** | **0.48** | 0.44 | 0.30 |
| **MCC** | **0.44** | 0.39 | 0.24 |
| **CE Loss** | **1.49** | 1.67 | 3.26 |
| **MAE** | **0.18** | 0.19 | 0.25 |
| **RMSE** | **0.42** | 0.43 | 0.52 |
| **% NOSBOS** | 56.0 | 56.0 | 100 |

The current refill implementation bases its recommendations on NOSBOS articles. The only condition that has to be met is that at least one item within the entire option is available on the sales floor. Not surprisingly, solely recommending NOSBOS articles does not lead to good performance on the other metrics. This makes sense since the label definition targets more cases than just NOSBOS. Essentially the current refill metrics provide a baseline with which the other models can be compared. From the current refill metrics it can be derived that of the positive samples in the test set approximately 32% are NOSBOS, whereas the precision shows that approximately 29% of the NOSBOS samples is actually a sample that should be refilled.

Comparing the models trained on either the complete or reduced set of features to the current refill immediately shows improvements for all evaluation criteria. The differences in the regression metrics can be ignored since the current refill does not do quantity predictions, i.e. if it should be refilled it always assigns a quantity of 1 to the refill action. Both model implementations predict approximately 56% of the NOSBOS samples as positive samples. These 56% accounted for 82% and 78% of the total amount of positive NOSBOS samples for the model trained on the complete and reduced set of features respectively. This implies that both models are able to capture most positive NOSBOS samples, but are still including many negative NOSBOS samples as well.

Looking at the two models, the performance does decrease when using less features. This was to be expected since with less features there is less information available. Especially the classification part definitely shows worse results, the regression part of predicting the exact quantities does not suffer that much, making it arguably worthwhile to use the reduced feature set for the regression part of the model. It is not that surprising that the performance of the regression part is roughly equal, given that the stockroom quantity features were considerably more important than any of the other features. The question whether the difference in performance is small enough to consider using a simpler model, i.e. a model using fewer features, will be discussed in more detail in section 8. For the remainder of this section the results will be computed using the model with the complete feature set.



|  | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 400,340 | 30,148 | 5,716 | 382 |
| 1 | 12,762 | 17,337 | 1,891 | 35 |
| 2 | 1,492 | 1,486 | 1,718 | 42 |
| 3 | 739 | 142 | 771 | 250 |

Figure 20: Confusion matrix of the model results when applied on the entire test set.

A different view of the performance of the model can be observed in the form of a confusion matrix, as shown in figure 20. An interesting result is that a higher target label also leads to a higher predicted label, albeit still not always being entirely correct. A possible explanation for the drop in performance between target label 2 and 3 is that the target label 3 also includes all samples with quantities higher than 3, which alters the nature of that set of samples.

The ROC curve of the classification part of the model is shown in figure 21. The marker indicates the value in the ROC curve belonging to the chosen threshold 0.45. Even though it might not seem reasonable to pick this threshold when looking at the ROC curve, one should keep in mind that the imbalance of the dataset cannot be derived from such a curve. What can be concluded from this curve is that a large section of samples which should be labeled with 0 can be correctly classified with relative ease. This can be derived from the flattened curve at approximately a false positive rate of 0.3.

To verify whether the performance is influenced by the inclusion of samples that are not even available for refill, the test set will be filtered to exclude these samples. As mentioned in section 4.4.1, the samples that will be filtered concern articles that are not available on the stockroom, which can thus not be refilled, and options which are entirely not available on the sales floor. With these alterations a new confusion matrix and ROC curve can be computed, which are shown in figures 22 and 23 respectively. The test metrics are summarized in table 13 alongside the results of the model on the complete test set.

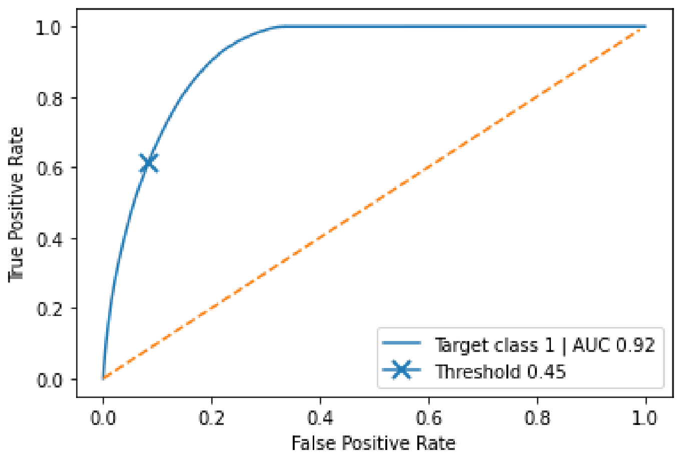
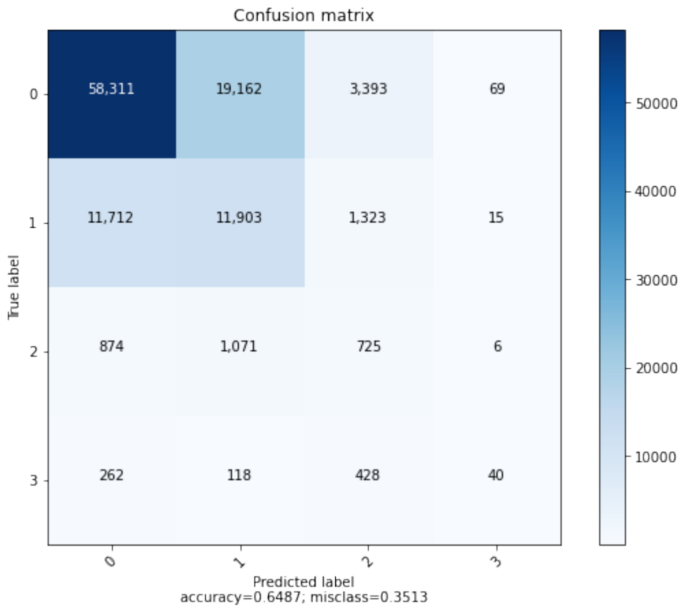


Figure 21: ROC curve of the classifier part of the model when applied on the entire test set.



| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 58,311 | 19,162 | 3,393 | 69 |
| 1 | 11,712 | 11,903 | 1,323 | 15 |
| 2 | 874 | 1,071 | 725 | 6 |
| 3 | 262 | 118 | 428 | 40 |

Figure 22: Confusion matrix of the model results when applied on the filtered test set.

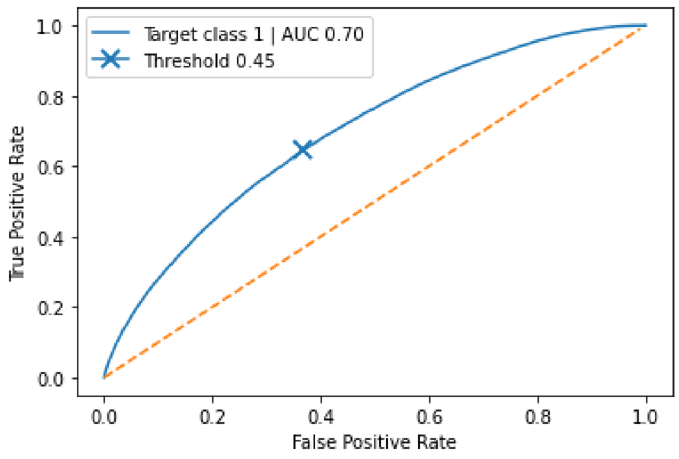


Figure 23: ROC curve of the classifier part of the model when applied on the filtered test set.

Table 13: The test results for both the full test set and the filtered test set.

| | **Full test set** | **Filtered test set** |
|---|---|---|
| **Accuracy** | 0.88 | 0.65 |
| **Precision** | 0.40 | 0.41 |
| **Recall** | 0.61 | 0.55 |
| **F1 score** | 0.48 | 0.47 |
| **MCC** | 0.44 | 0.25 |
| **CE Loss** | 1.49 | 4.48 |
| **MAE** | 0.18 | 0.45 |
| **RMSE** | 0.42 | 0.54 |
| **% NOSBOS** | 56.0 | 49.0 |

The figures and table clearly paint a different picture. Removing the options that would also be filtered when using the model in practice from the test set leads to similar performance when looking at target labels 1, 2 and 3. However, the performance of target label 0 has plummeted, leading to more than 20% of the target label 0 samples to be classified as 1. Even though relatively more zero samples are accepted as refill recommendations, the ratio between zero and non-zero samples that are accepted remains roughly the same. The recall decreased just slightly whereas the precision even increased a tiny bit. Both make sense since one would expect the model to not recommend impossible refill recommendations in the first place, even though they could be labeled as such. Since the MCC also takes true negatives into account, the MCC, together with the accuracy, is the clearest indicator that the performance has degraded.

For the next sections, which dive into details such as performance per category, the filtered test set will be used since it is a better representation of what the model would actually present to the user.

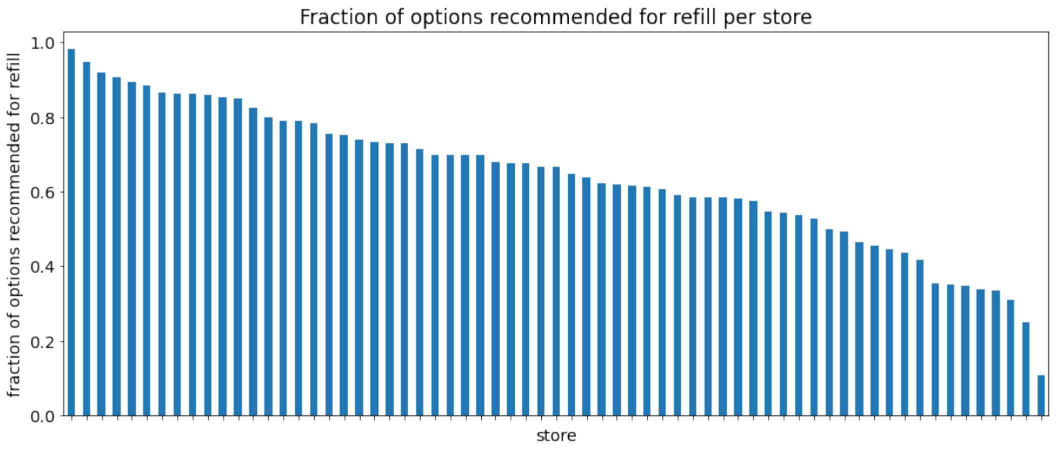


Figure 24: Fraction of options recommended per store.

### 5.4.2 Performance per store

Using the filtered test set, the fraction of options recommended per store can be computed and are shown in figure 24. There is a significant difference between the store with the highest fraction of recommended options and the store with the lowest fraction of recommended options. Naturally, lots of factors can be of influence regarding these differences, but looking at aspects such as the size of the sales floor or stockroom, the ratio between these two sizes, the amount of NOSBOS options or the category distributions of the options did not lead to big enough differences to explain the gap. Even though the aforementioned factors might be of influence, the biggest difference between the left-most and right-most stores in the figure are the frequency with which they count. For the first five stores the average amount of days between the three counts are 7, 4, 6, 18 and 9 whereas for the last five stores the average amount of days are 2, 2, 6, 2 and 1. On second thought it is not that surprising that this interval is of importance to the amount of refill recommendations, since more frequent counting leads to more accurate stock levels and hence less refills necessary to compensate. Aside from less refills being necessary the model's training data for these stores is also more accurate than for stores that count less frequently, since less events introducing noise in the labels can occur.

### 5.4.3 Performance per category

In addition to the filters already in place, the samples for which no article details were provided are also removed from the test set, since these could be part of any of the other categories and are thus inconclusive when it comes to deriving patterns. For the category comparison the refill recommendations are grouped per option, since the number of articles per option is irrelevant when using the model in practice. Furthermore, categories with a large number of articles per option, such as shoes, could create misleading results if the performance would be measured per article.

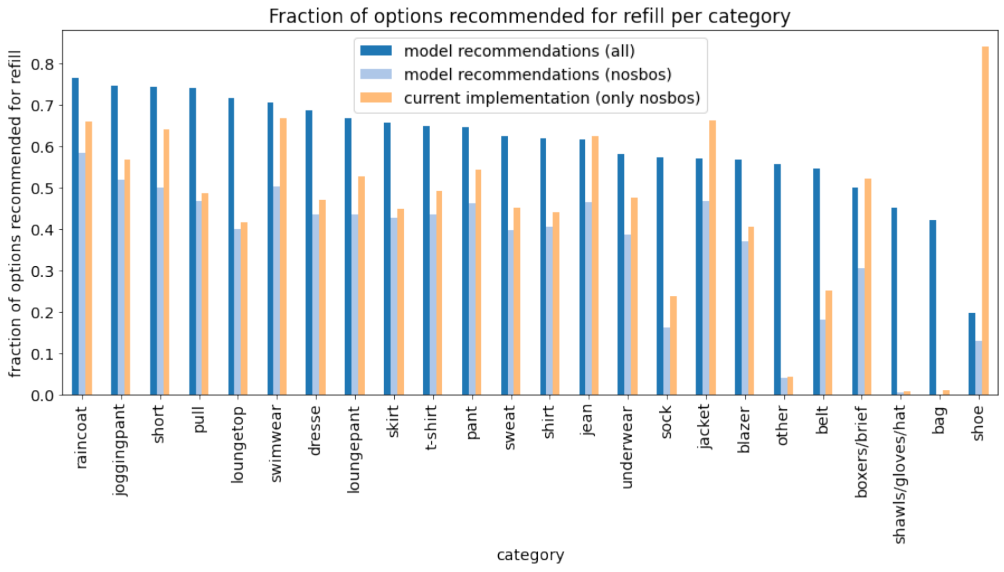


Figure 25: Fraction of options recommended out of the total amount of options which can potentially be refilled per category. The bars concern the model suggestions, the model suggestions which are also NOSBOS and the current refill suggestions.

The options were grouped per store and count. For each group the maximum model prediction of the individual articles belonging to that option was computed. If that prediction is bigger than the threshold with which the model accepts a refill recommendation, which in this case is 0.45, that option is counted as a refill recommendation. The remaining options were grouped per category. Using the total amount of options per category, the fraction of options that were recommended per category was computed. In order to compare the new model with the current implementation, the fraction of model recommendations which are also NOSBOS articles were computed alongside the entire fraction of NOSBOS articles in the filtered test set. The entire fraction of NOSBOS articles represent the current implementation's recommendations. These fractions are shown per category in figure 25.

For the categorical comparison it is noteworthy that the test set consists of counts performed in September and October by stores in Western Europe. This time period can be considered a tough test set since stores will probably be in the middle of or finishing the exchange of the summer collection with the autumn collection. When looking at the results with that consideration in mind, it is favorable that categories like raincoats, jogging-pants and pulls end up on the higher end of the spectrum, with over 70% of the available options being recommended to refill. Considering the climate in Western Europe, these categories seem reasonable to refill frequently. On the other hand, the categories shorts and swimwear are definitely questionable when it comes to their results. They are not necessarily wrong, however, since stores often want to get rid of their stock from the previous season.

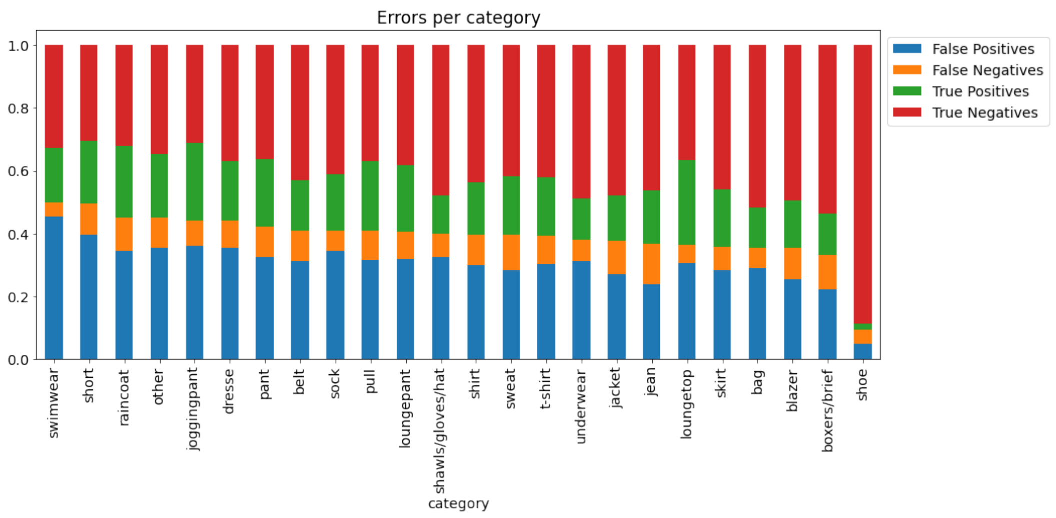


Figure 26: The true positives, true negatives, false positives and false negatives for the filtered test set split by category.

Comparing the model's recommendations with the ones from the current implementation does show the possibilities of applying a data-driven approach. The decrease in shoes recommended is huge and the fact that one-size options such as bags and shawls are even recommended at all is promising. The model overall does seem to largely agree with the current implementation, as indicated by the fraction of model recommendations which are also NOSBOS. It is thus not surprising that categories with fewer sizes, like bags and shawls, end up being less frequently recommended. Another observation is that for most categories the model is recommending more options than the current implementation. This means that even though the model does not recommend all NOSBOS articles, it compensates these articles with other articles to refill.

Using the same filtered test dataset, the mistakes can be computed per category. The true positives, true negatives, false positives and false negatives per category are shown in figure 26. These values illustrate the expected errors for both swimwear and shorts, but also indicate that a significant fraction of these articles should actually have been recommended. Naturally this might be the desired behavior, but it does reinforce the importance of the label generation, which might be erroneous in these cases. Another conclusion that can be drawn from this figure is the ease with which shoes are classified correctly, which is partly due to the large amount of negative samples within that category since for each category the fraction of true negatives outperforms the false negatives by a big margin.

One aspect that is not illustrated in the previous figures is the absolute amount of options recommended. For example, the first figure, figure 25, indicates that approximately 70% of the swimwear options are recommended, but this does not give insight in what the effect of these options are. Therefore, it is interesting to look at the category frequency within two different subsets, one being the filtered test set and the other one being the set of recommended options. These frequencies are shown in figures 27 and 28 respectively.

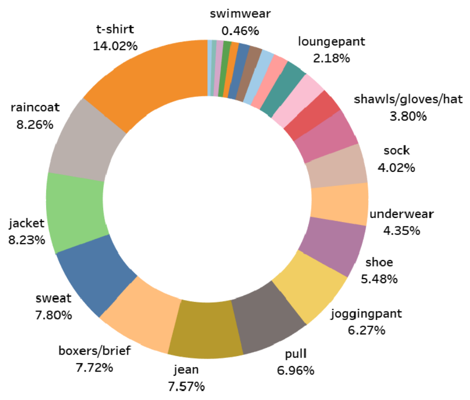


Figure 27: Category frequencies within the filtered test set.

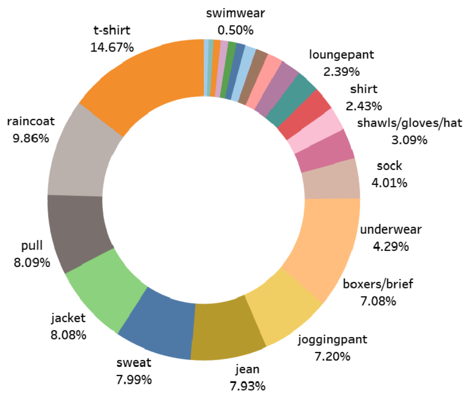


Figure 28: Category frequencies within the set of recommended options.

When looking at these distributions it becomes clear that both shorts and swimwear are not occurring frequently, occupying less than 2% of the total amount of options. Comparing the other categories in these figures shows that the categories raincoats and pulls are relatively more frequent in the recommended set than in the complete set, whereas shoes are less frequent. These numbers put the number of errors as shown in figure 26 into perspective by indicating that even though the errors made for shorts and swimwear are quite large, the effect on the total list of options is small.

In the end it seems likely that, when looking at these figures and the feature contributions, the model is capable of adjusting its decisions based on the category of the article. Especially considering the shoe category, for which the binary feature was not deemed important, the results are large enough to conclude that the categorical embedding is of importance. However, the model does appear to have trouble dealing with seasonality, even though the errors made while recommending swimwear and shorts are not conclusive evidence. More counts should be included over a longer time period to make proper claims about seasonality. This will be discussed in more detail in sections 6 and 8.

### 5.4.4 Performance per date

When applying the model on the test set, a division can be made considering how far away in the future each count is, as shown in figure 12. Table 14 shows the computed metrics per count in the test set, grouped by how far away it was from the last count of the store. Even though there are some indications of the performance decaying over time, e.g. when looking at the regression based metrics, the differences are not as big as one might expect. Interestingly, the classification performance is best for the samples gathered from the second and third counts after the last count per location. When looking at some other statistics about the data this might be related to the distribution of zero and non-zero samples in the test data set per count. The means of the labels are 0.098, 0.089 and 0.11 respectively, indicating that aside from the moment the samples are gathered there are also other factors influencing the performance. All in all its hard to determine whether the performance actually degrades, especially considering that only three counts per store were used.

Table 14: Performance of the model per count in the test set.

| | **Test set** | **Count $c_{t+1}$** | **Count $c_{t+2}$** | **Count $c_{t+3}$** |
|---|---|---|---|---|
| **Accuracy** | 0.88 | 0.88 | **0.89** | 0.87 |
| **Precision** | 0.40 | 0.39 | 0.37 | **0.42** |
| **Recall** | 0.61 | 0.56 | **0.65** | 0.63 |
| **F1 score** | 0.48 | 0.46 | 0.47 | **0.51** |
| **MCC** | 0.44 | 0.41 | 0.44 | **0.46** |
| **CE Loss** | 1.49 | 1.49 | **1.46** | 1.52 |
| **MAE** | **0.18** | **0.18** | **0.18** | 0.19 |
| **RMSE** | **0.42** | **0.42** | **0.42** | 0.43 |
| **% NOSBOS** | 56.0 | 57.0 | 58.0 | 54.0 |

## 5.5 Online evaluation

### 5.5.1 Store visit

The store visit did not necessarily produce results related to the relevance of the model recommendations. Even though the store visit was carefully planned beforehand with the store manager, the store employee present who was executing the count was never responsible for using the refill section of the Nedap app. Therefore, the employee did not have any prior expectations as to what kind of recommendations could show up and how this differs from the current implementation. With that in mind it is difficult to draw conclusions regarding the effectiveness of the model as compared to the current implementation. For this specific store visit it did become clear that the store employee kind enough to provide feedback had a vision of having a broad selection of sizes available on the sales floor while not adding a size when it was already present. According to the store employee this was not in line with what the store manager wanted, since he favored refilling as many articles as possible, regardless of whether that size was already present on the sales floor. This not only reinforces the statement that there is no single solution at hand satisfying all parties, but also provides some context for the results gathered while performing refill at this store visit.

Working through the entire list of refill recommendations generated was an impossible task given the time required to do so. To get the most out of the time available, several recommendations were handpicked per category and shown to the store employee. In line with the aforementioned policy of having as many sizes available as possible, many refill recommendations that were suggesting to add another item to an existing article stack were not relevant. On the other hand, the most relevant recommendations were executed by adding an item of a missing size while simultaneously removing an item from another size. When taking a closer look at the list, it became clear that higher quantities were recommended for this store than in general for other stores, which makes sense considering that the store manager is refilling additional sizes the day before counting. This leads to larger stacks during the count itself, which in turn triggers the model to recommend larger quantities, which in this specific case is the exact opposite of what the store employee performing refill wanted.

Given these considerations the most important conclusion that can be drawn from this store visit is that generating refill recommendations is a complicated task which requires a proper feedback system to be set up before reasonable conclusions can be drawn from online testing.

### 5.5.2 Store e-mails

The store e-mails present two sides of roughly the same story. For starters, the suggestions included that were only recommended by the current implementation were found useful and executed for approximately half of the cases provided. The ones executed were mostly performed partially, i.e. not all recommended size refills were actually found useful.

The same could be said for the shared recommendations. Almost 80% of the refill recommendations were performed, albeit several of them only partially. Whereas it should be said that the current experiment was small-scale and might thus not be conclusive or representative for the entire retailer, it does hint towards the notion that the samples for which the model agrees with the current implementation are promising. Furthermore, the model does also recommend to refill items for other sizes within the option which are not NOSBOS. Those are found useful for roughly half of the cases.

Some interesting recommendations were provided by the model which the current implementation would never do, like recommending shawls or face masks. At the time of writing face masks are of vital importance given that they are mandatory to wear in public areas due to COVID-19. So even though the option has only been added recently, it has no negative influence on the model's predictions. Other popular recommendations include underwear and t-shirts, typical categories for which it is not uncommon that all sizes are available in larger quantities on the sales floor.

## 5.6 Other retailer

Contrary to the data used for retailer X, the dataset of retailer Y only contains the last three counts per store. Therefore, the entire dataset will be used for generating the metrics, since no train or validation split is necessary. The metrics were computed on both the entire dataset as well as on the filtered dataset, using the same filters as the ones described in section 5.4.1. The test metrics are shown in table 15. The resulting ROC curve and confusion matrix for the complete dataset can be seen in figures 29, 30, whereas the ROC curve and confusion matrix for the filtered dataset can be seen in figures 31 and 32.

Given the performance of the model on the complete and filtered test set of retailer X, it was to be expected that the performance on the filtered test set is worse than on the complete test set. Interestingly, even though the embeddings for this retailer are all replaced with the token indicating that the embedding has not yet been seen and that no prior training has been performed on this retailer, the performance is not that bad. The metrics that stand out the most are the precision, recall and the percentage of possible NOSBOS articles recommended. The higher recall and lower precision as compared to retailer X indicates that the model is overly optimistic when predicting for retailer Y.

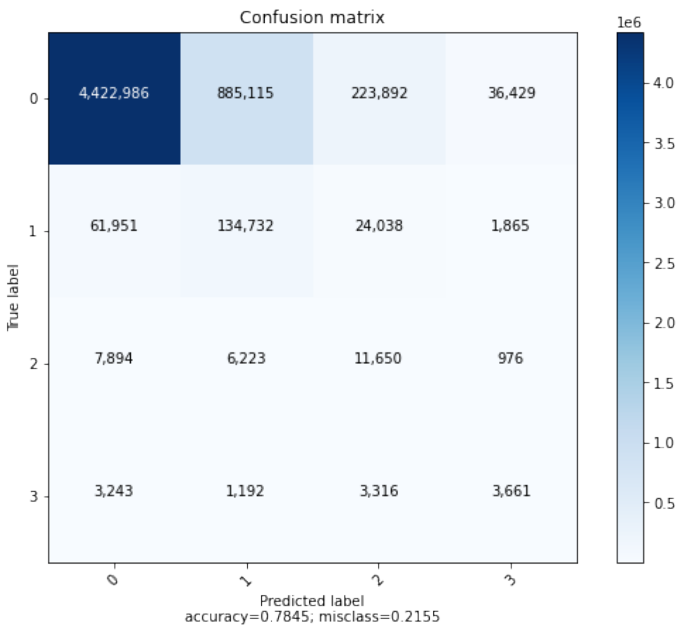


Figure 29: Confusion matrix of the model results when applied on the complete dataset of retailer Y.

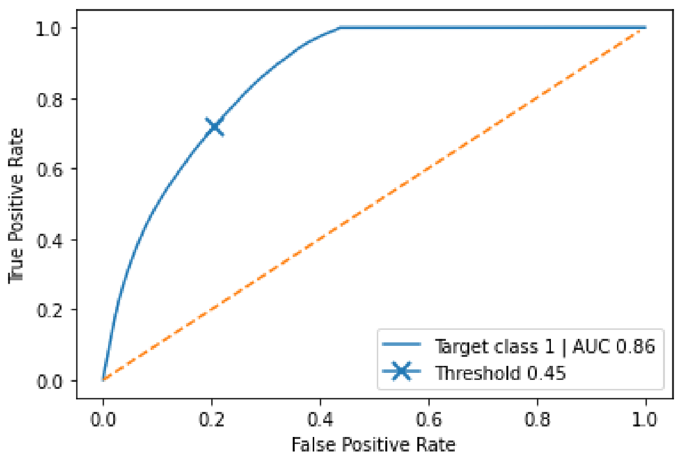


Figure 30: ROC curve of the classifier part of the model applied on the complete dataset of retailer Y.

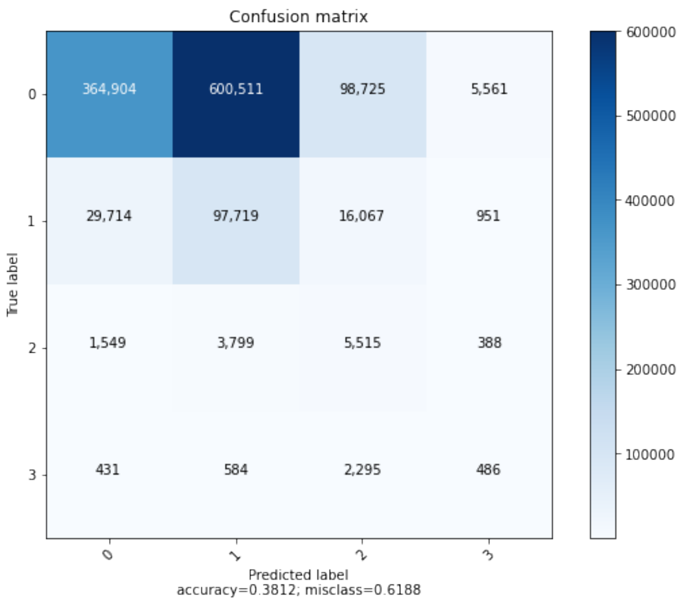


Figure 31: Confusion matrix of the model results when applied on the filtered data of retailer Y.

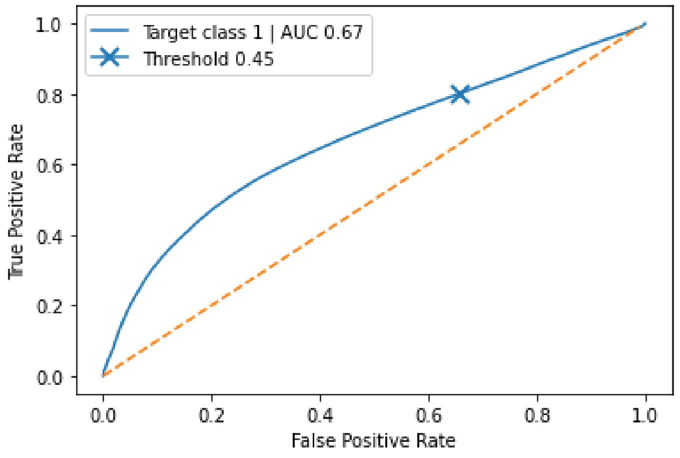


Figure 32: ROC curve of the classifier part of the model applied on the filtered data of retailer Y.

Table 15: The test results of the model when applied on retailer Y for both the full dataset and the filtered dataset.

| | **Retailer Y - full dataset** | **Retailer Y - filtered dataset** |
|---|---|---|
| **Accuracy** | 0.78 | 0.38 |
| **Precision** | 0.14 | 0.15 |
| **Recall** | 0.72 | 0.80 |
| **F1 score** | 0.24 | 0.26 |
| **MCC** | 0.25 | 0.10 |
| **CE Loss** | 2.89 | 8.28 |
| **MAE** | 0.33 | 0.51 |
| **RMSE** | 0.63 | 0.56 |
| **% NOSBOS** | 90.0 | 90.0 |

Looking a little closer at the features of the samples in the filtered dataset, the largest difference between retailers X and Y occurs at the sales floor quantities and the amount of NOSBOS articles available. Retailer X has on average 0.93 items on the sales floor per article, whereas retailer Y has 1.53 items on the sales floor per article. This metric immediately affects the fraction of articles which are NOSBOS. For retailer X, 38% of the articles that can be recommended are NOSBOS, while for retailer Y only 14% of the articles that can be recommended are NOSBOS. These feature differences are a possible explanation for why the model recommends more articles, since it takes historical sales floor quantities into account which are on average higher for retailer Y than for retailer X.

The feature differences themselves can largely be explained by the fact that retailer Y is using RFID on a more frequent basis, counting practically every day as well as using the current refill algorithm after almost every count. This ensures that especially the NOSBOS articles are quickly responded to if they occur, explaining why the fraction of NOSBOS articles is so low. As such it is not all that surprising that the model is recommending too many articles considering that it was trained on a retailer which counted less frequently and features smaller sales floors.

All in all this experiment highlights the differences between retailers and the need of applying retailer-specific algorithms. Whether or not this should be done using machine learning will be discussed in section 8.

# 6 Discussion

## 6.1 Methodology

### 6.1.1 Data

The first decision influencing the entire research was the choice of retailer. Even though it was not necessarily a bad decision to take a retailer that uses refill in an ordinary way, picking a retailer who counted more frequently would remove noise from the dataset. Especially the labels will be more accurate since less time passes between counts, which would make the labels derived from the data a better representation of what actually happened. The noisy dataset led to several rules for labels to be discarded. An example of such a rule dealt with changes in stockroom quantities. If the stockroom quantity reduced between counts, one could argue that this is due to a refill. However, if the time between counts is sufficiently large, other factors such as shipments between stores or the distribution center get a bigger influence on the stockroom quantity. The amount of noise introduced by choosing this retailer thus made it harder to draw valuable conclusions from the data.

Apart from the noise introduced in the dataset, another consideration is the way features were engineered. Even though all the readily available information in the dataset was used, combinations of features were not used as much. More effort could have been put in deciding upon which combination of features would make sense to use as a separate feature. An example of such a feature combination might be a binary feature indicating that the article is a sock and that the quantity on the sales floor is lower than 3.

Concerning the dataset split, the way the test set is generated does present the most accurate representation of reality. However, big changes can occur when taking the data of a month later, since changes in season can have a major influence on the model performance. Since data from numerous counts was available, it would in hindsight have probably been better to include more than just the last 3 counts in the test set.

Lastly, as was shown in section 5.4.1, filtering the test set has a large influence on the overall performance. Since these samples will be filtered out eventually, these samples might as well be removed from the dataset all together. This would most likely also increase the usefulness of the feature importances since the domination of the stockroom quantity is likely to decrease. The decrease is mainly due to the nature of the samples that would be removed, a large part of which consists of samples which have stockroom quantity 0 and are therefore inherently not able to be refilled.

### 6.1.2 Model determination

Naturally it is impossible to test every different kind of model with a variety of layers. However, given that only a small subset of models were trained, there are numerous other ways to implement and train models. The resulting best model is thus not necessarily be the best model possible. Furthermore, contrary to taking scikit-learn neural networks as baseline, it would also have been valuable to check the performance of several other scikit-learn default models, such as a decision tree or a support vector machine. Especially decision trees would have been a valuable addition since they can easily be inspected by humans to get insights in how the model makes its decisions.

Other than that, the baselines were barely tuned, which probably had influence on the metrics they produced. Even though it might not be worthwhile to spend a lot of time on improving the baseline, the scikit-learn models were used in a direct comparison to the custom approaches, on which more time was spent.

## 6.2 Results

### 6.2.1 Feature importance

As mentioned before, the results for the feature importances are disproportionately in favor of the stockroom indicators since a large portion of the samples can be correctly classified based solely on the stockroom quantity. Even though big differences can be seen between other features, the attributions for the stockroom quantities in the regression network are so skewed that the other features contributions are almost negligible.

### 6.2.2 Test set performance

When looking at the performance per store there are several caveats. The conclusion that there are significant differences between the stores can easily be reached, but finding the reason why is more difficult given the amount of factors that are of influence. Similarly, the conclusions drawn for the categories should also be placed in perspective. Obviously categories like raincoats and jogging-pants being on the top end of the spectrum is desirable, but whether swimwear and shorts are actually bad recommendations is tougher to judge. Even though the season is shifting, a retailer might decide to place all available swimwear on the sales floor in order to clean the stockroom. This once again shows that choosing a retailer that counts more frequently would have been the better choice, since these conclusions could have been drawn more easily.

Lastly, the performance per date metrics would have been more valuable if a larger amount of time would have passed between the counts. If the test set consisted of more counts, the interval between counts could have been increased and would be a better indicator of whether the performance actually decays.

### 6.2.3 Online evaluation

The online evaluation could definitely have been more exhaustive. The e-mails send to the stores provided enough information to get a good first indication of the potential of the proposed method. Especially the fact that multiple e-mails were sent to multiple stores over the weeks led to a diverse set of results. The store visit, however, did not lead to any satisfactory results. Since the intention was clear from the start, there was no reason to suspect that the store employees present had never used refill. It would technically have been possible to retry at another store at another time, but based on the prior experience it was likely that the results would be largely skewed by the preferences of the employee. Furthermore, given the current pandemic, it was deemed unwise to travel across the country to perform another test at a more suitable store.

### 6.2.4 Other retailer

In hindsight it would have been preferable to use retailer Y as the retailer for the larger part of this research and to use retailer X for verification. This would mean that the verification would be done on the noisier data, based on which conclusions could be drawn regarding the effect of counting less frequently.

# 7 Conclusion

To come to a conclusion about the main research question of this thesis the research sub-questions will first be discussed.

The first research sub-question concerned what machine learning method was the best performing. The offline evaluation showed that the model combining a Wide and Deep classification model with a residual regression model performed best when considering the average of the F1 score and the Matthews Correlation Coefficient. It turned out that using an acceptance threshold of 0.45 for the classification model combined with a resampling rate of 0.25 for the training data led to the best results.

The second research sub-question deals with the comparison of the newly defined model and the current refill application. Using an exhaustive list of 9 metrics it was shown that the model outperformed the current refill application on all of the metrics considered.

The third sub-question covers the way in which the model is able to differentiate between different stores and categories. The fraction of options recommended and the amount of errors made showed that there are big differences between individual stores and categories. Naturally this is partly due to other factors, but the feature contributions confirm that both the category and store embedding are deemed important for the decision process, reaffirming the differentiating potential of the model.

The fourth and fifth sub-question concern the applicability of the model on a different retailer and the frequency with which the model should be retrained. The performance on retailer Y was significantly worse than the performance on retailer X. Even though this might partly be due to the completely different nature of the other retailer, it is unlikely that using the model without (partial) retraining on another retailer works well. On the other hand, the performance of the model barely decayed over time. Consequently, it can be concluded that for this dataset the model does not necessarily need to be retrained after each and every count.

The last research sub-question is about the requirements that should be met in order to create a production-ready data-driven model. The details concerning this sub-question will be discussed in next section, but essentially it boils down to creating pipelines for large scale online testing as well as registering article movements in-store. With these two measures, models can be trained on proper data and easily be tested to verify whether they are satisfactory.

In conclusion, a data-driven approach is definitely applicable for generating refill recommendations for real-world fashion stores. Even though the current setup is far from perfect, mainly due to the noisy dataset and the missing ability to perform large scale verification, the results were promising and far exceeded the performance of the current refill.

# 8 Recommendations and future work

With the results gathered an analysis can be made regarding what should definitely happen in order for a data-driven approach to properly work for generating refill recommendations. Aside from these adjustments this section will also make several recommendations concerning what methods to use and what considerations can be made regarding aspects such as features and generalizability. These recommendations will include both recommendations for Nedap specifically as well as recommendations for future research.

## 8.1 Labeled data

The most decisive factor is the collection of a labeled dataset without the extreme uncertainty that the dataset used for this research has. Such a dataset is impossible to gather using the current setup, at least for most retailers, since the movements between the stockroom and the sales floor are not measured consistently. There are a variety of ways to improve upon this situation. In short these options are as follows:

1. Recount the stockroom after refill
2. Mark articles as refilled
3. Register moved articles.

The first option involves doing a new count of the stockroom after refill has been performed. Using the newly counted quantities, the difference per article for the stockroom can be computed. A positive difference means the quantity in the stockroom has increased, so items from that article have been removed from the sales floor. On the other hand, a negative difference indicates that the article has been refilled.

Whereas this is an easy approach to get an accurate picture of what was refilled, the main drawback is that performing yet another count after refill is time consuming and might feel redundant for the store employees. Furthermore, the process involves counting the entire stockroom, which is quite inefficient given that nothing has changed for the articles that were not refilled.

The second option would be to enable retailers to indicate what they have refilled while working through the list of refill recommendations. Contrary to performing another count, this method offers a way to improve the dataset while being less intrusive for the store employees performing the refill. The challenge in implementing this method lies in making it intuitive and efficient. Given that refill is performed while holding a handheld device, it is important to make it as easy as possible for the store employee to provide feedback about the refill recommendation. Especially indicating what articles were refilled and in what quantities can be quite a challenge to implement in such a way that the store employee can easily provide the correct information without being slowed down too much, since the refill recommendations shown are grouped per option. If the employee does not feel like filling in the required information, or forgets about it, this will introduce more noise in the dataset, essentially creating the same issue as before.

Another downside of gathering feedback from the employee during refill is that the only refills for which data is gathered are the ones that are recommended. If one takes into account that a store employee will most likely not work through the entire list of refill recommendations, the model used will suffer from confirmation bias. Confirmation bias also occurs using the other solutions since employees are less likely to refill articles not shown in the list of refill recommendations. However, for this method the problem is amplified since the other methods do measure refill recommendations not in the list, whereas this approach provides no way to register them.

The last possibility is to recount the articles that were refilled. When performing refill, the store employee can gather all the articles which should be refilled. Instead of immediately placing them on the correct spot on the sales floor, the stack of articles to refill can be counted using an RFID scanner. This process takes significantly less time than scanning the entire stockroom while having the advantage of registering the exact quantity of items refilled per article. Furthermore, this method can also be used at other occasions where items are moved from the stockroom to the sales floor, or vice versa, in order to get an improved stock representation of the store.

While the visual aspects of this method can be implemented quite easily, the main problem lies in identifying the items that should be refilled from the new RFID scan. Given that it is likely that also other items will be scanned, these need to be filtered out in order for this method to correctly register which items have been refilled. However, this should not be too big of a problem and can definitely be dealt with accordingly, for example by cross-referencing the newly refilled articles with previous count data.

In the end none of the methods discussed are perfect. It is safe to say that performing an additional count of the stockroom is the worst of the three options given the inefficient nature of that method. The main downside of the second option is that it still does not provide an accurate view of refill and is limited as to its applicability outside of refill. Therefore, the third option is preferable for gathering an accurate labeled dataset.

## 8.2 Testing

Closely tied to the gathering of a labeled dataset is the performing of extensive testing. Generating proper test results not only involves receiving accurate data registering what movements have occurred during refill, but also creating a system in which different methods can be tested against each other in a large-scale A/B test. This becomes more relevant when a machine learning model is actually deployed, since then it is an essential part of verifying whether a newly trained machine learning model is actually better performing than the previous one.

Concretely, this would mean implementing a machine learning pipeline where it becomes possible to deploy new models to a small number of stores per retailer in order to verify whether it actually performs better.

## 8.3 Machine learning method

Considering the performance of the different machine learning methods tested, the logical choice would be to pursue the combination of classification and regression. As mentioned in the results section, specifically table 7, a combination of models not only performs the best on the offline metrics but also enables users to set the acceptance threshold based on how many refill recommendations they want.

Splitting the model for two different objectives also allows for additional classification models to be used. These models could then independently classify whether an article should be refilled. The results of these classifications can be combined and used as input for the secondary model, which predicts the exact quantity. Such a setup can be used to create specific classification models for subsets of articles, for example based on categories or seasonality, while leaving the secondary model untouched. Combining this with a suitable machine learning pipeline could lead to a dynamic model where separate parts can be exchanged and models can be customized per retailer, or even per store, based on what combination of models works best. For future work it would be interesting to verify whether defining several specialized classification models actually leads to an increase in performance.

When considering the regression part of the model it might also be interesting to remove the samples from the training set with a stockroom quantity of just one item. This would probably lead to the stockroom quantity feature to be a little less important, giving room for other features to contribute more.

Aside from continuing with the current approach, this problem can also be seen from a reinforcement learning point of view [29, 53, 55]. The biggest issue with reinforcement learning is that it requires either an offline simulator or a large amount of sequential events with some kind of reward system. Given the uncertainty of the labeled dataset it was deemed unwise to enter more noise into the system by designing such a simulator or experimenting with the reward function. This is definitely one of the routes that can be pursued in a later stage and might be an interesting consideration for future research. One of the reward functions to specifically look at would be considering the total distance from the current sales floor to the optimal sales floor. Every refill would then contribute to reaching that optimal sales floor, which would immediately provide a way to rank the refill recommendations.

A different approach would be to apply unsupervised learning. Instead of directly predicting the quantities to refill, unsupervised learning could be used to identify clusters in the data by minimizing the distance between several objects [4]. Whereas research has been performed applying unsupervised topological learning to recommender systems [15, 21, 22, 34], these papers focus on identifying groups of similar users in order to estimate what rating an item should get based on ratings of similar users. This is certainly an interesting way of dealing with recommendation since it closely resembles collaborative filtering, but it only deals with a small fraction of the information available. Hence it was not the preferred methodology for this research. However, for future research it might be interesting to use unsupervised learning to cluster stores and articles. Features of the relevant stores and articles can then be used as input to another network for the final prediction.

## 8.4 Model retraining

The results observed in this research regarding the performance of the model on future counts do not directly imply that the model needs to be retrained after every new count. Still it remains difficult to say whether this can actually be used to directly disregard the decaying performance, since for some stores the future counts were only a few days away from the last count trained upon. A better guideline indicating how often the model needs to be retrained is the embeddings it learns. Since the embeddings can only be learned for values observed in the training set, this set should be frequently updated to prevent too many new items being assigned the special token used for unseen values.

Given the counting frequency, a possible starting scheme would be to completely retrain a model once a month. The counts that occur within that month can then be used to update the current model with, ensuring that trends and sales are properly dealt with. Alternatively, one might consider training a model per season. The only caveat of such an approach being that changes in season occur over the span of several weeks, making it difficult to pinpoint when the model should be altered. For both approaches it becomes easier to measure the decay in performance once the testing is properly implemented, based on which this training scheme can be adjusted.

When looking at different retailers it has become clear that the differences between retailers are large enough to deploy different models. It thus seems viable to train a different model for each retailer. Not only does this possibly lead to improved performance, it also prevents any problems that might arise from training a model on data from multiple retailers. It is questionable whether a retailer allows for another retailer to profit from a model partially trained on their data.

Alternatively, one generic model could be trained, which can be used as a basis for applying transfer learning. It is an interesting topic for further research, where problems need to be discussed such as the way the embeddings are trained and with what data the generic model should be trained.

## 8.5 Feature set

Regarding features for the wide part of the model, the binary features related to NOSBOS, sales floor and stockroom quantities should definitely stay since they contributed the most to the model outputs, as was seen in section 5.3. The other features were not as important and could probably be ignored.

Looking at the features of the deep part of the model, the embeddings are definitely important enough to keep. The only embedding that can be excluded is the embedding for GTINs. This embedding is not only the largest but also has barely any effect on the outcome. Furthermore, this embedding is likely to change quite frequently since every new article introduced requires this embedding to be altered. Other features that can be excluded concern data points from previous counts. The only features worth keeping that concern earlier counts are the stockroom and sales floor quantities.

The recommended features are listed in table 16, the meaning of the abbreviations can be found in appendix B. It should be noted that these recommendations are based on the features used in this research. There are numerous other features to consider which might also be of interest, e.g. the count frequency or the amount of days since the previous count, so this list is by no means final.

Table 16: Recommended features

| **Wide features** | **Deep features** | |
|---|---|---|
| pt > 0 | cat | qtmax |
| qt == 0 | opt | opta |
| pt > 0 & qt == 0 | size | optmin |
| | gender | optavg |
| | age | optmax |
| | loc | optsum |
| | qt | optlen |
| | qt1 | sd |
| | pt | nsr |
| | pt1 | nsf |
| | dm | month |
| | qtmin | sf/sr |
| | qtavg | |

Aside from these features, more research should definitely be done on what combinations of features lead to satisfactory results. The question remains whether these features can be defined in a general settings or if these features need to be redefined per retailer to match the specific requirements of that retailer.

## 8.6 Sales data

Including sales data as a feature creates more insight in what movements occur within a store and can thus be used to make a more direct connection to demand forecasting. This is not necessarily a requirement if movements from the stockroom to the sales floor are registered since that already gives an indication regarding the demand of an article. However, the sales data is more direct and capable of showing popular articles or sales events. It remains to be seen how effective including sales data actually is, but in conjunction with proper testing this can easily be verified.

## 8.7 Demand forecasting

If factors like sales data come into play, it will also be interesting to include the expected rate of sale of an article as a feature. Not only might this be of influence to the quantity that should be refilled, it can also aid in determining the order of the list of refill recommendations. How this rate of sale or expected rate of sale is to be computed depends on the situation and the type of sales data the retailer provides, but plenty of research has been done within this domain [14, 30, 40]. Ultimately, it might be possible to let the demand forecasting model and the refill model work hand-in-hand, allowing the models to improve each other. However, given the current situation and the novelty of using recommender systems for physical fashion stores, this is not worth pursuing for the foreseeable future.

## 8.8 Ranking recommendations

As shown in this research, there is a big difference between offline and online performance, especially when taking into account the different points of view employees within a store tend to have. An improved dataset already improves the offline evaluation, but online evaluation requires some more tuning. The problem with solely looking at whether a refill recommendation was executed is that inherently this suffers from position bias. It would be incorrect to assume that every store works through the entire list of refill recommendations every time. As such the order in which the refill recommendations appear in the list directly influences the rate at which these recommendations are actually executed. It would be interesting to take a look at ranking algorithms for refill recommendations and whether aspects common for ranking models such as novelty and diversity are important for this specific use case.

## 8.9 List recommendation

A different approach which might be interesting to pursue would be to take the other refill recommendations into account when predicting whether an article should be refilled. This would mean that aside from the article specific features, more features should be added that indicate what other articles are possible to refill. It might be difficult to implement such a system during the prediction phase of the model, but optimizing the list of refill recommendations afterwards should be possible. Such an optimization could look at the total amount of items that the model recommends, comparing this amount to quantities such as the amount of items on the sales floor or the average amount of items refilled after other counts. This optimization closely resembles a ranking model and could thus be implemented alongside a ranking model.

## References

[1] Fatemeh Afghah, Abolfazl Razi, S. M. Reza Soroushmehr, Somayeh Molaei, Hamid Ghanbari, and Kayvan Najarian. A game theoretic predictive modeling approach to reduction of false alarm. In Xiaolong Zheng, Daniel Dajun Zeng, Hsinchun Chen, and Scott J. Leischow, editors, *Smart Health*, pages 118–130, Cham, 2016. Springer International Publishing.

[2] Pankaj Agarwal, Sreekanth Vempati, and Sumit Borar. Personalizing similar product recommendations in fashion e-commerce. *CoRR*, abs/1806.11371, 2018.

[3] Charu C. Aggarwal. *Recommender Systems*. Springer International Publishing, 2016.

[4] Xavier Amatriain, Alejandro Jaimes*, Nuria Oliver, and Josep M. Pujol. *Data Mining Methods for Recommender Systems*, pages 39–71. Springer US, Boston, MA, 2011.

[5] Pierre Baldi, Søren Brunak, Yves Chauvin, Claus AF Andersen, and Henrik Nielsen. Assessing the accuracy of prediction algorithms for classification: an overview. *Bioinformatics*, 16(5):412–424, 2000.

[6] M. Brian Blake. Two decades of recommender systems at amazon. 2017.

[7] Pol Boada-Collado and Victor Martínez-de Albéniz. Estimating and optimizing the impact of inventory on consumer choices in a fashion retail setting. *Manufacturing & Service Operations Management*, 22(3):582–597, 2020.

[8] Sabri Boughorbel, Fethi Jarray, and Mohammed El-Anbari. Optimal classifier for imbalanced data using matthews correlation coefficient metric. *PLOS ONE*, 12(6):1–17, 06 2017.

[9] Brahmadeep and S. Thomassey. 8 - intelligent demand forecasting systems for fast fashion. In Tsan-Ming Choi, editor, *Information Systems for the Fashion and Apparel Industry*, Woodhead Publishing Series in Textiles, pages 145 – 161. Woodhead Publishing, 2016.

[10] John S. Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, UAI'98, page 43–52, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[11] Javier Castro, Daniel Gómez, and Juan Tejada. Polynomial calculation of the shapley value based on sampling. *Computers  Operations Research*, 36(5):1726 – 1730, 2009. Selected papers presented at the Tenth International Symposium on Locational Decisions (ISOLDE X).

[12] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. Wide & deep learning for recommender systems. *CoRR*, abs/1606.07792, 2016.

[13] Davide Chicco and Giuseppe Jurman. The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation. *BMC Genomics*, 21, 12 2020.

[14] T.-M. Choi. Launching the right new product among multiple product candidates in fashion: Optimal choice and coordination with risk consideration. *International Journal of Production Economics*, 202:162–171, 2018. cited By 10.

[15] Debby Cintia Ganesha Putri, Jenq-Shiou Leu, and Pavel Seda. Design of an unsupervised machine learning-based movie recommender system. *Symmetry*, 12(2):185, Jan 2020.

[16] C. Cleverdon and M. Kean. Factors determining the performance of indexing systems. Aslib Cranfield Research Project, Cranfield, England, 1968.

[17] Cosmin Condea, Frédéric Thiesse, and Elgar Fleisch. Rfid-enabled shelf replenishment with backroom monitoring in retail stores. *Decision Support Systems*, 52(4):839 – 849, 2012. 1)Decision Support Systems for Logistics and Supply Chain Management 2)Business Intelligence and the Web.

[18] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16, page 191–198, New York, NY, USA, 2016. Association for Computing Machinery.

[19] James Davidson, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet, Ullas Gargi, Sujoy Gupta, Yu He, Mike Lambert, Blake Livingston, and Dasarathi Sampath. The youtube video recommendation system. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys '10, page 293–296, New York, NY, USA, 2010. Association for Computing Machinery.

[20] Thomas G. Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10:1895–1923, 1998.

[21] Issam Falih, Nistor Grozavu, Rushed Kanawati, and Younès Bennani. A recommendation system based on unsupervised topological learning. In Sabri Arik, Tingwen Huang, Weng Kin Lai, and Qingshan Liu, editors, *Neural Information Processing*, pages 224–232, Cham, 2015. Springer International Publishing.

[22] Issam Falih, Nistor Grozavu, Rushed Kanawati, and Younès Bennani. Topological multi-view clustering for collaborative filtering. *Procedia Computer Science*, 144:306 – 312, 2018. INNS Conference on Big Data and Deep Learning.

[23] Carlos A. Gomez-Uribe and Neil Hunt. The netflix recommender system: Algorithms, business value, and innovation. *ACM Trans. Manage. Inf. Syst.*, 6(4), December 2016.

[24] Cheng Guo and Felix Berkhahn. Entity embeddings of categorical variables, 2016.

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[26] Sebastian Heinz, Christian Bracher, and Roland Vollgraf. An lstm-based dynamic customer model for fashion recommendation. *CoRR*, abs/1708.07347, 2017.

[27] A. Hübner and Kai Schaal. Effect of replenishment and backroom on retail shelf-space planning. *Business Research*, 10:123–156, 2017.

[28] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[29] Feng Liu, Ruiming Tang, Xutao Li, Weinan Zhang, Yunming Ye, Haokun Chen, Huifeng Guo, and Yuzhou Zhang. Deep reinforcement learning based recommendation with explicit user-item interactions modeling, 2019.

[30] A.L.D. Loureiro, V.L. Miguéis, and L.F.M. da Silva. Exploring the use of deep neural networks for sales forecasting in fashion retail. *Decision Support Systems*, 114:81–93, 2018. cited By 26.

[31] Scott Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *CoRR*, abs/1705.07874, 2017.

[32] Brian W Matthews. Comparison of the predicted and observed secondary structure of t4 phage lysozyme. *Biochimica et Biophysica Acta (BBA)-Protein Structure*, 405(2):442–451, 1975.

[33] Quinn McNemar. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157, jun 1947.

[34] Erwan Le Merrer and Gilles Trédan. The topological face of recommendation: models and application to bias detection. *CoRR*, abs/1704.08991, 2017.

[35] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 807–814, Madison, WI, USA, 2010. Omnipress.

[36] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G. Azzolini, Dmytro Dzhulgakov, Andrey Mallevich, Ilia Cherniavskii, Yinghai Lu, Raghuraman Krishnamoorthi, Ansha Yu, Volodymyr Kondratenko, Stephanie Pereira, Xianjie Chen, Wenlin Chen, Vijay Rao, Bill Jia, Liang Xiong, and Misha Smelyanskiy. Deep learning recommendation model for personalization and recommendation systems. *CoRR*, abs/1906.00091, 2019.

[37] Jihoi Park and Kihwan Nam. Group recommender system for store product placement. *Data Mining and Knowledge Discovery*, 33, 11 2018.

[38] Bobby Prévost, Jonathan Laflamme Janssen, Jaime R. Camacaro, and Carolina Bessega. Deep inventory time translation to improve recommendations for real-world retail. In *Proceedings of the 12th ACM Conference on Recommender Systems*, RecSys '18, page 195–199, New York, NY, USA, 2018. Association for Computing Machinery.

[39] Yacine Rekik, Evren Sahin, and Yves Dallery. Analysis of the impact of the rfid technology on reducing product misplacement errors at retail stores. *International Journal of Production Economics*, 112(1):264 – 278, 2008. Special Section on Recent Developments in the Design, Control, Planning and Scheduling of Productive Systems.

[40] S. Ren, H.-L. Chan, and P. Ram. A comparative study on fashion demand forecasting models with multiple sources of uncertainty. *Annals of Operations Research*, 257(1-2):335–355, 2017. cited By 21.

[41] Paul Resnick and Hal R. Varian. Recommender systems. *Commun. ACM*, 40(3):56–58, March 1997.

[42] Raquel Rodríguez-Pérez and Jürgen Bajorath. Interpretation of machine learning models using shapley values: application to compound potency and multi-target activity predictions. *Journal of Computer-Aided Molecular Design*, 34(10):1013–1026, may 2020.

[43] G. Roussos. Enabling rfid in retail. *Computer*, 39(3):25–30, 2006.

[44] Sam T. Roweis and Lawrence K. Saul. Nonlinear Dimensionality Reduction by Locally Linear Embedding. *Science*, 290(5500):2323–2326, 2000.

[45] Lloyd S. Shapley. *A Value for n-Person Games.* RAND Corporation, Santa Monica, CA, 1952.

[46] Erik Strumbelj and Igor Kononenko. An efficient explanation of individual classifications using game theory. *J. Mach. Learn. Res.*, 11:1–18, March 2010.

[47] Erik Štrumbelj and Igor Kononenko. Explaining prediction models and individual predictions with feature contributions. *Knowledge and Information Systems*, 41(3):647–665, aug 2013.

[48] Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 1225–1234, New York, NY, USA, 2016. Association for Computing Machinery.

[49] Jizhe Wang, Pipei Huang, Huan Zhao, Zhibo Zhang, Binqiang Zhao, and Dik Lun Lee. Billion-scale commodity embedding for e-commerce recommendation in alibaba. *CoRR*, abs/1803.02349, 2018.

[50] R. Want. An introduction to rfid technology. *IEEE Pervasive Computing*, 5(1):25–33, 2006.

[51] C. Willmott and K Matsuura. Advantages of the mean absolute error (mae) over the root mean square error (rmse) in assessing average model performance. *Climate Research*, 30:79, 12 2005.

[52] Wei Xu and Alexander Rudnicky. Can artificial neural networks learn language models? pages 202–205, 01 2000.

[53] Xiangyu Zhao, Liang Zhang, Zhuoye Ding, Dawei Yin, Yihong Zhao, and Jiliang Tang. Deep reinforcement learning for list-wise recommendations. *CoRR*, abs/1801.00209, 2018.

[54] Zhe Zhao, Lichan Hong, Li Wei, Jilin Chen, Aniruddh Nath, Shawn Andrews, Aditee Kumthekar, Maheswaran Sathiamoorthy, Xinyang Yi, and Ed Chi. Recommending what video to watch next: A multitask ranking system. In *Proceedings of the 13th*

*ACM Conference on Recommender Systems*, RecSys '19, page 43–51, New York, NY, USA, 2019. Association for Computing Machinery.

[55] Guanjie Zheng, Fuzheng Zhang, Zihan Zheng, Yang Xiang, Nicholas Jing Yuan, Xing Xie, and Zhenhui Li. Drn: A deep reinforcement learning framework for news recommendation. In *Proceedings of the 2018 World Wide Web Conference*, WWW '18, page 167–176, Republic and Canton of Geneva, CHE, 2018. International World Wide Web Conferences Steering Committee.

## A Model architecture

```
MultiModel(
  (model1): WideDeep(
    (wide): Linear(
      (main): Linear(in_features=8, out_features=1, bias=True)
    )
    (deep): DeepEmbedding(
      (embeddings): ModuleList(
        (0): Embedding(26, 3)
        (1): Embedding(8597, 10)
        (2): Embedding(134, 4)
        (3): Embedding(3, 2)
        (4): Embedding(6, 2)
        (5): Embedding(66, 3)
        (6): Embedding(36024, 14)
      )
      (main): Sequential(
        (0): Linear(in_features=79, out_features=128, bias=True)
        (1): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True,
              track_running_stats=True)
        (2): ReLU()
        (3): Linear(in_features=128, out_features=64, bias=True)
        (4): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True,
              track_running_stats=True)
        (5): ReLU()
        (6): Linear(in_features=64, out_features=32, bias=True)
        (7): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True,
              track_running_stats=True)
        (8): ReLU()
        (9): Linear(in_features=32, out_features=1, bias=True)
      )
    )
    (activation): Sigmoid()
  )
  (model2): ResNet(
    (embeddings): ModuleList(
      (0): Embedding(26, 3)
      (1): Embedding(8597, 10)
      (2): Embedding(134, 4)
      (3): Embedding(3, 2)
      (4): Embedding(6, 2)
      (5): Embedding(66, 3)
      (6): Embedding(36024, 14)
    )
    (activation): ReLU()
    (main): Sequential(
      (0): ResidualFCBlock(
        (blocks): Sequential(
          (0): Linear(in_features=79, out_features=128, bias=True)
          (1): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True,
                track_running_stats=True)
          (2): ReLU()
          (3): Linear(in_features=128, out_features=128, bias=True)
          (4): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True,
                track_running_stats=True)
        )
        (shortcut): Sequential(
          (0): Linear(in_features=79, out_features=128, bias=True)
          (1): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True,
                track_running_stats=True)
        )
        (activation): ReLU()
      )
      (1): ResidualFCBlock(
        (blocks): Sequential(
          (0): Linear(in_features=128, out_features=64, bias=True)
          (1): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True,
                track_running_stats=True)
          (2): ReLU()
          (3): Linear(in_features=64, out_features=64, bias=True)
          (4): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True,
                track_running_stats=True)
        )
        (shortcut): Sequential(
          (0): Linear(in_features=128, out_features=64, bias=True)
          (1): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True,
                track_running_stats=True)
        )
        (activation): ReLU()
      )
      (2): ResidualFCBlock(
        (blocks): Sequential(
          (0): Linear(in_features=64, out_features=32, bias=True)
          (1): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True,
                track_running_stats=True)
          (2): ReLU()
          (3): Linear(in_features=32, out_features=32, bias=True)
          (4): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True,
                track_running_stats=True)
        )
        (shortcut): Sequential(
          (0): Linear(in_features=64, out_features=32, bias=True)
          (1): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True,
                track_running_stats=True)
        )
```

```
        (activation): ReLU()
      )
      (3): Linear(in_features=32, out_features=1, bias=True)
    )
  )
)

----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                    [-1, 1]               9
            Linear-2                    [-1, 1]               0
         Embedding-3                    [-1, 3]              78
         Embedding-4                   [-1, 10]          85,970
         Embedding-5                    [-1, 4]             536
         Embedding-6                    [-1, 2]               6
         Embedding-7                    [-1, 2]              12
         Embedding-8                    [-1, 3]             198
         Embedding-9                   [-1, 14]         504,336
           Linear-10                  [-1, 128]          10,240
      BatchNorm1d-11                  [-1, 128]             256
             ReLU-12                  [-1, 128]               0
           Linear-13                   [-1, 64]           8,256
      BatchNorm1d-14                   [-1, 64]             128
             ReLU-15                   [-1, 64]               0
           Linear-16                   [-1, 32]           2,080
      BatchNorm1d-17                   [-1, 32]              64
             ReLU-18                   [-1, 32]               0
           Linear-19                    [-1, 1]              33
    DeepEmbedding-20                    [-1, 1]               0
          Sigmoid-21                    [-1, 1]               0
         WideDeep-22                    [-1, 1]               0
        Embedding-23                    [-1, 3]              78
        Embedding-24                   [-1, 10]          85,970
        Embedding-25                    [-1, 4]             536
        Embedding-26                    [-1, 2]               6
        Embedding-27                    [-1, 2]              12
        Embedding-28                    [-1, 3]             198
        Embedding-29                   [-1, 14]         504,336
           Linear-30                  [-1, 128]          10,240
      BatchNorm1d-31                  [-1, 128]             256
           Linear-32                  [-1, 128]          10,240
      BatchNorm1d-33                  [-1, 128]             256
             ReLU-34                  [-1, 128]               0
           Linear-35                  [-1, 128]          16,512
      BatchNorm1d-36                  [-1, 128]             256
             ReLU-37                  [-1, 128]               0
  ResidualFCBlock-38                  [-1, 128]               0
```

```
         Linear-39                 [-1, 64]           8,256
     BatchNorm1d-40                [-1, 64]             128
         Linear-41                 [-1, 64]           8,256
     BatchNorm1d-42                [-1, 64]             128
           ReLU-43                 [-1, 64]               0
         Linear-44                 [-1, 64]           4,160
     BatchNorm1d-45                [-1, 64]             128
           ReLU-46                 [-1, 64]               0
 ResidualFCBlock-47                [-1, 64]               0
         Linear-48                 [-1, 32]           2,080
     BatchNorm1d-49                [-1, 32]              64
         Linear-50                 [-1, 32]           2,080
     BatchNorm1d-51                [-1, 32]              64
           ReLU-52                 [-1, 32]               0
         Linear-53                 [-1, 32]           1,056
     BatchNorm1d-54                [-1, 32]              64
           ReLU-55                 [-1, 32]               0
 ResidualFCBlock-56                [-1, 32]               0
         Linear-57                  [-1, 1]              33
           ReLU-58                  [-1, 1]               0
         ResNet-59                  [-1, 1]               0
================================================================
Total params: 1,267,595
Trainable params: 1,267,595
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.02
Params size (MB): 4.84
Estimated Total Size (MB): 4.86
----------------------------------------------------------------
```

# B Features

Table 17: Descriptions of the feature abbreviations.

| Abbreviation | Description |
|---|---|
| pt > 0 | is article available in stockroom |
| qt == 0 | is article not available on sales floor |
| pt > 0 & qt == 0 | is article NOSBOS |
| qt2nan | is there data from three counts |
| mos | is the article the most present within its option |
| mos1 | is the article the most present within its option last count |
| mos2 | is the article the most present within its option two counts ago |
| shoe | is the article a shoe |

| **Abbreviation** | **Description** |
|---|---|
| cat | category embedding |
| opt | option embedding |
| size | size embedding |
| gender | gender embedding |
| age | age group embedding |
| loc | location embedding |
| gtin | gtin embedding |
| qt | sales floor quantity |
| qt1 | sales floor quantity last count |
| qt2 | sales floor quantity two counts ago |
| dm | sales floor quantity mean across stores |
| dm1 | sales floor quantity mean across stores last count |
| dm2 | sales floor quantity mean across stores two counts ago |
| pt | stockroom quantity |
| pt1 | stockroom quantity last count |
| pt2 | stockroom quantity two counts ago |
| qtmin | minimum sales floor quantity of last three counts |
| qtavg | average sales floor quantity of last three counts |
| qtmax | maximum sales floor quantity of last three counts |
| opta | is option available |
| optmin | minimum sales floor quantity of articles in option |
| optavg | average sales floor quantity of articles in option |
| optmax | maximum sales floor quantity of articles in option |
| optsum | sum of sales floor quantities for option |
| optlen | amount of articles within the option |
| opta1 | is option available last count |
| optmin1 | minimum sales floor quantity of articles in option last count |
| optavg1 | average sales floor quantity of articles in option last count |
| optmax1 | maximum sales floor quantity of articles in option last count |
| optsum1 | sum of sales floor quantities for option last count |
| optlen1 | amount of articles within the option last count |
| opta2 | is option available two counts ago |
| optmin2 | minimum sales floor quantity of articles in option two counts ago |
| optavg2 | average sales floor quantity of articles in option two counts ago |
| optmax2 | maximum sales floor quantity of articles in option two counts ago |
| optsum2 | sum of sales floor quantities for option two counts ago |
| optlen2 | amount of articles within the option two counts ago |

*Continued on next page*

| | |
|---|---|
| sd | fraction of article sales floor quantity within option |
| sd1 | fraction of article sales floor quantity within option last count |
| sd2 | fraction of article sales floor quantity within option two counts ago |
| nsr | total number of items on sales floor |
| nsr1 | total number of items on sales floor last count |
| nsr2 | total number of items on sales floor two counts ago |
| nsf | total number of items in stockroom |
| nsf1 | total number of items in stockroom last count |
| nsf2 | total number of items in stockroom two counts ago |
| month | month |
| sf/sr | stockroom sales floor ratio |

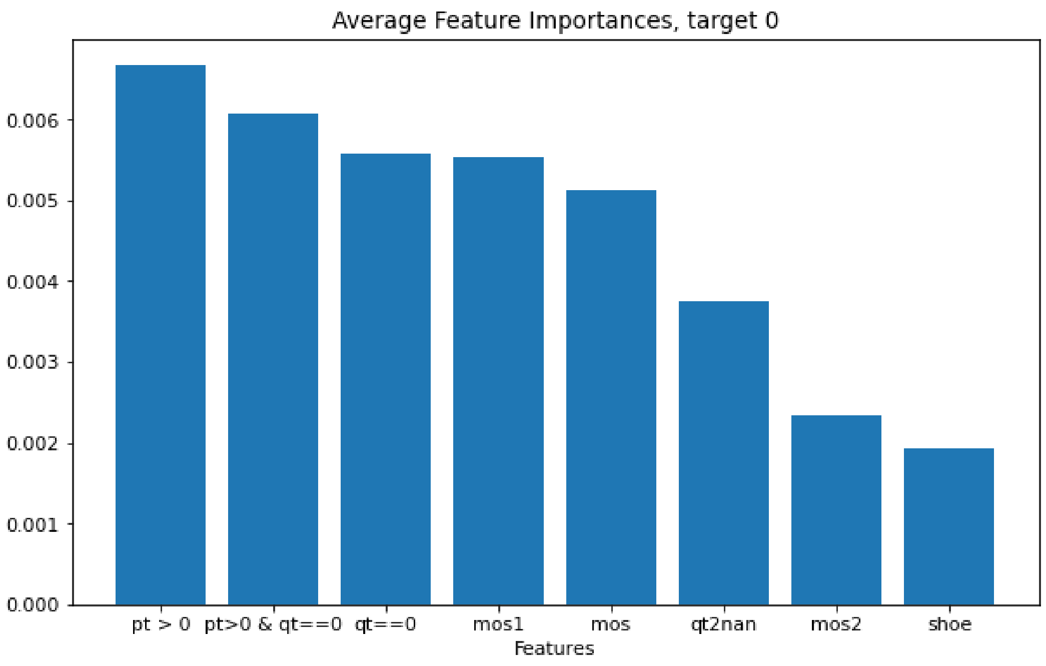


Figure 33: Feature attributions from the wide part of the model targeting samples with label 0. The meaning of the abbreviations can be found in table 17.

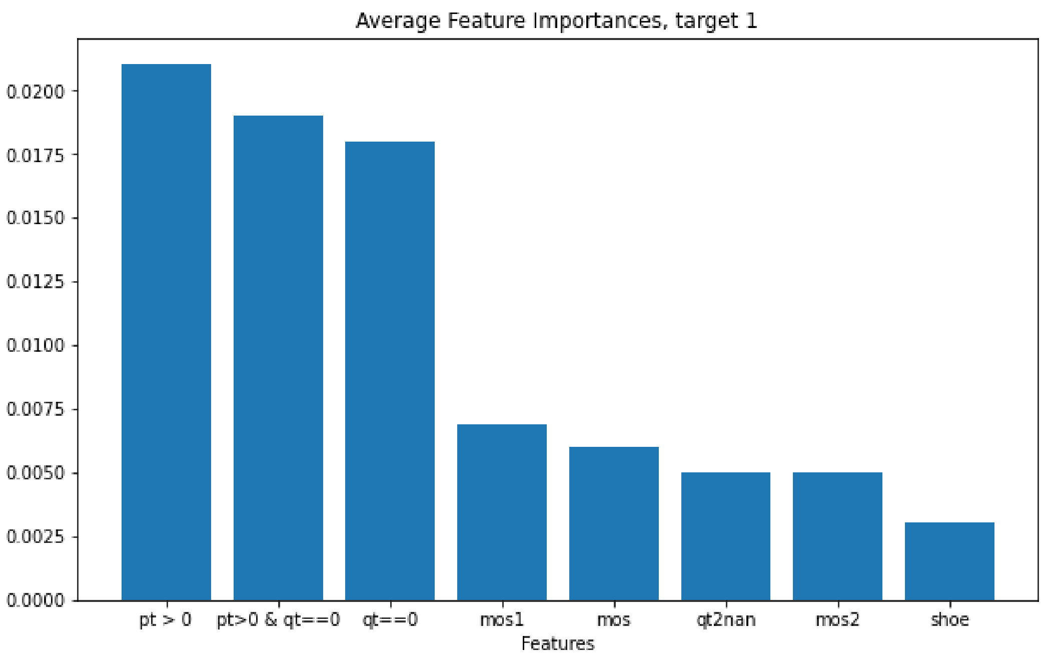


Figure 34: Feature attributions from the wide part of the model targeting samples with label 1. The meaning of the abbreviations can be found in table 17.

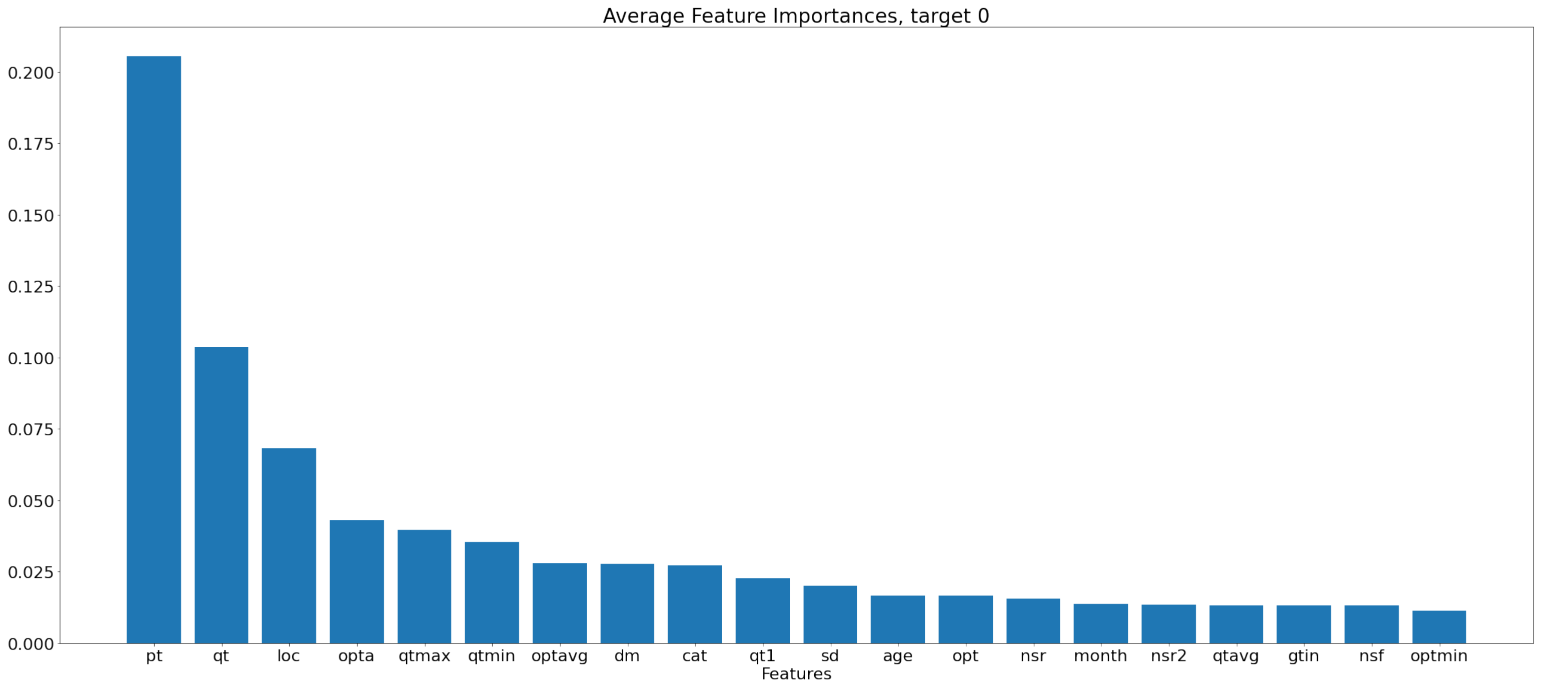


Figure 35: Feature attributions from the deep part of the model targeting samples with label 0. For readability only the top 20 most important features are listed. The meaning of the abbreviations can be found in table 17.

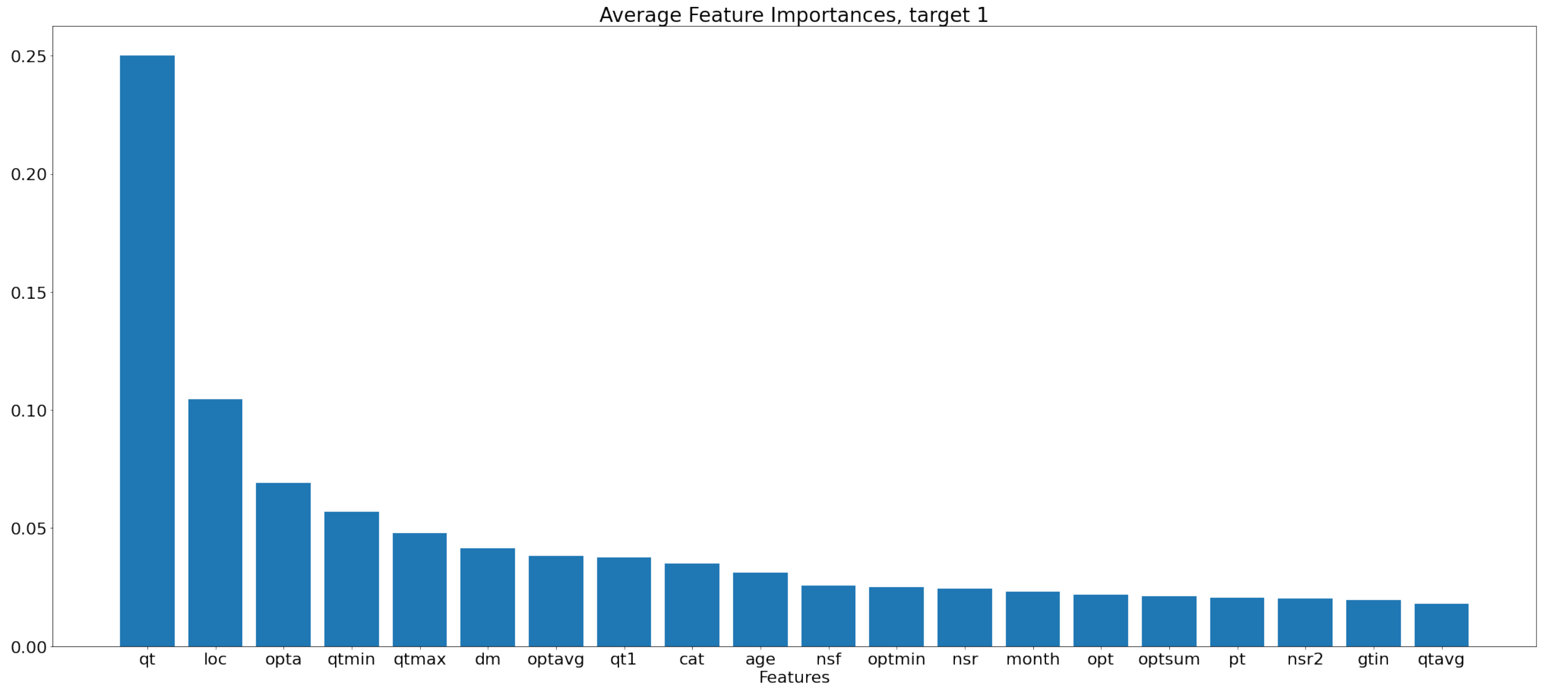


Figure 36: Feature attributions from the deep part of the model targeting samples with label 1. For readability only the top 20 most important features are listed. The meaning of the abbreviations can be found in table 17.

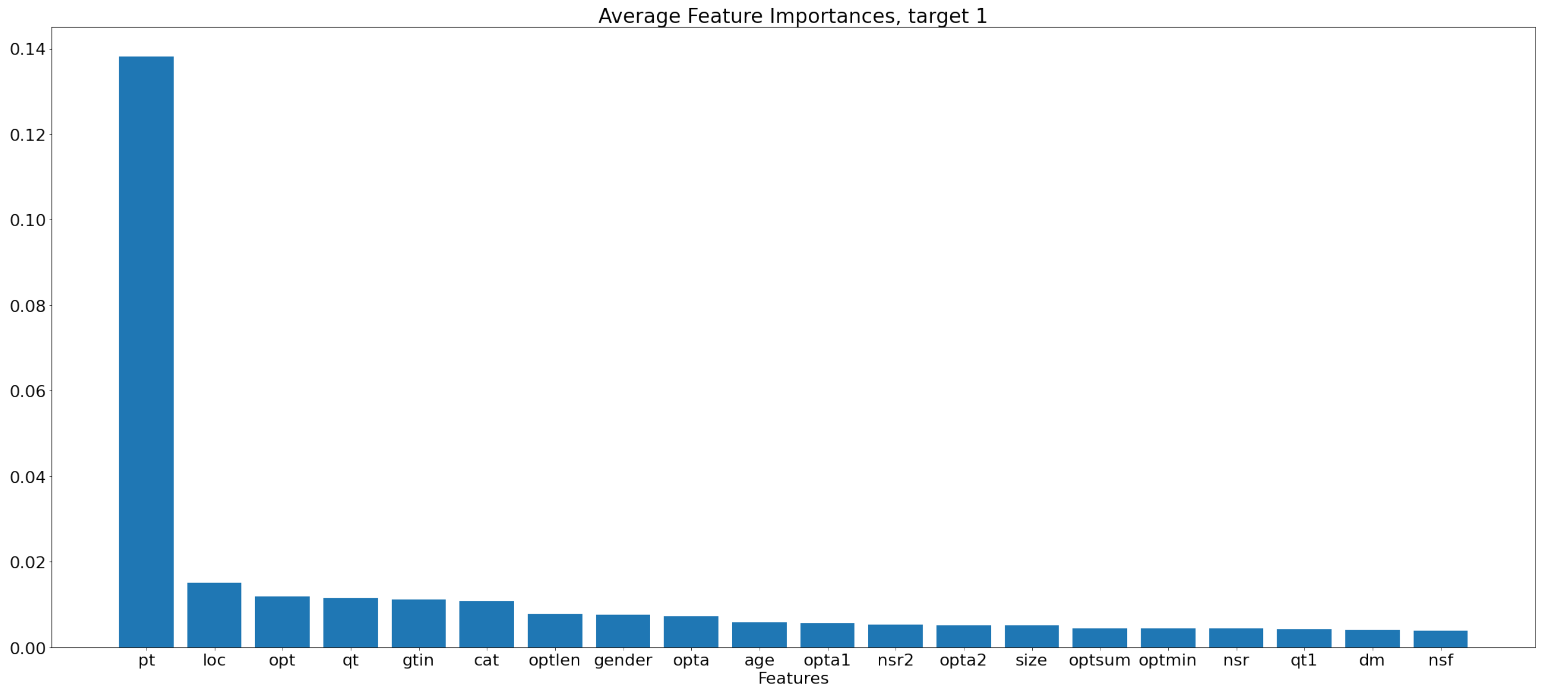


Figure 37: Feature attributions from the regression part of the model targeting samples with label 1. For readability only the top 20 most important features are listed. The meaning of the abbreviations can be found in table 17.

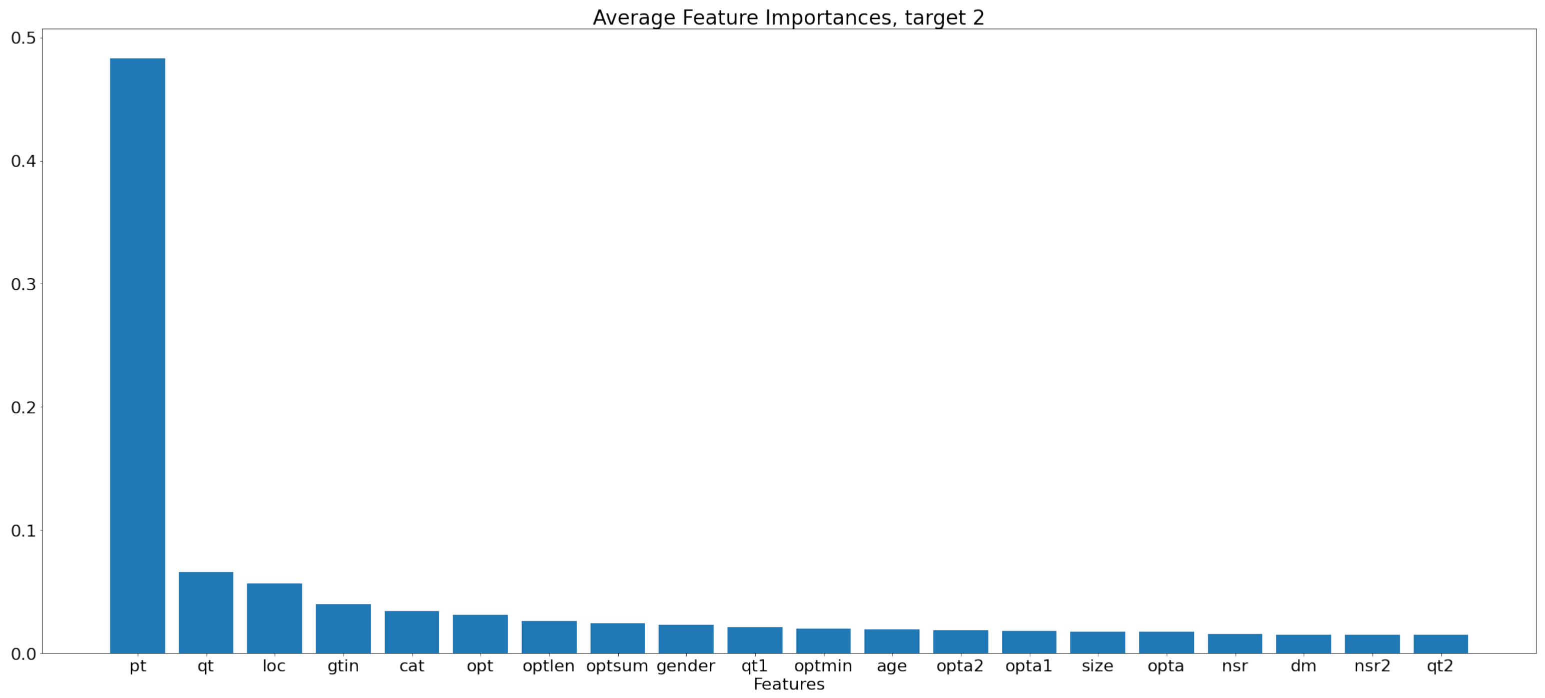


Figure 38: Feature attributions from the regression part of the model targeting samples with label 2. For readability only the top 20 most important features are listed. The meaning of the abbreviations can be found in table 17.

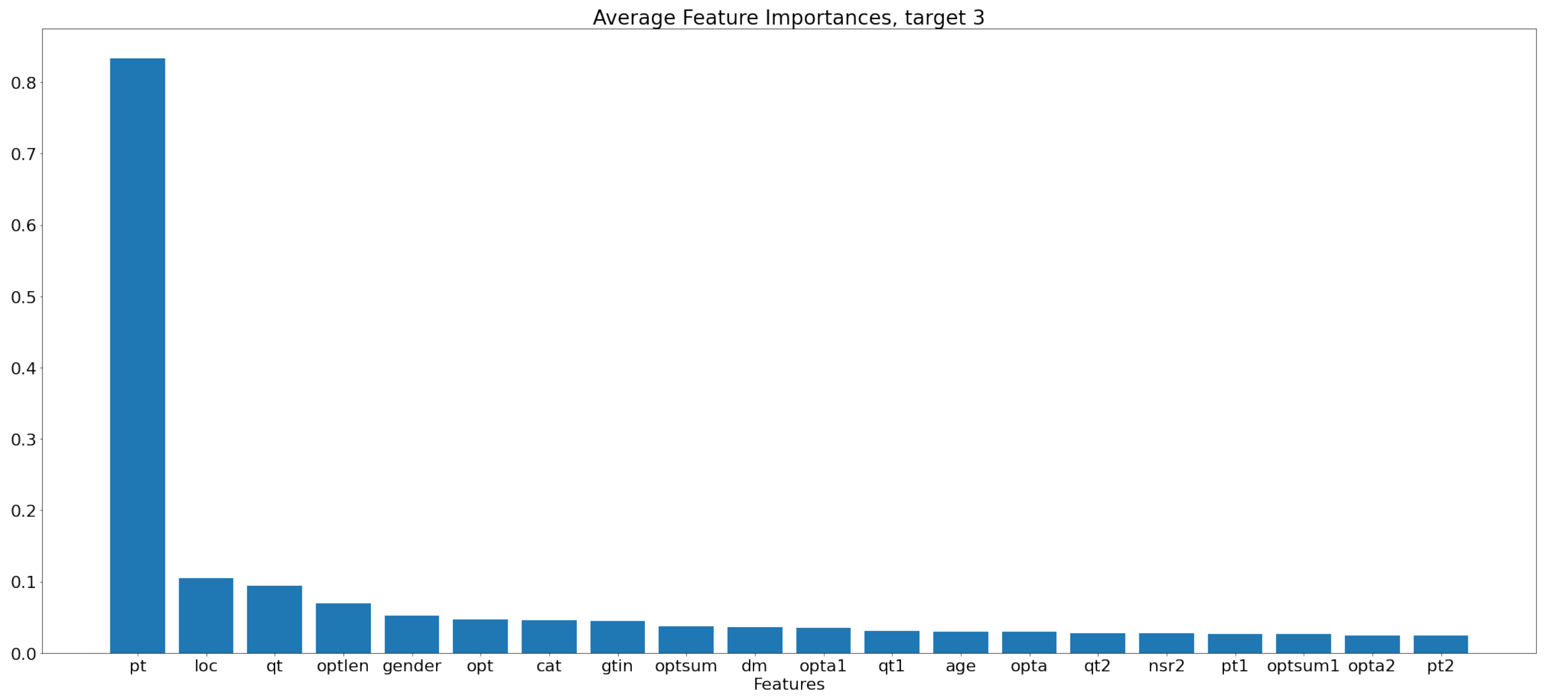


Figure 39: Feature attributions from the regression part of the model targeting samples with label 3. For readability only the top 20 most important features are listed. The meaning of the abbreviations can be found in table 17.