Ukrainian Catholic University

Bachelor Thesis

Warehouse Layout Optimization for Order Picking in E-commerce Industry

Author:
Maksym Kmet

Supervisor:
Oleh Lukianykhin

A thesis submitted in fulfillment of the requirements
for the degree of Bachelor of Science

in the

Department of Computer Sciences
Faculty of Applied Sciences

Lviv 2022

Declaration of Authorship

"It is a capital mistake to theorize before one has data."

Sherlock Holmes

UKRAINIAN CATHOLIC UNIVERSITY

Faculty of Applied Sciences

Bachelor of Science

Warehouse Layout Optimization for Order Picking in E-commerce Industry

by Maksym Kmet

Abstract

This project focuses on the order picking process of a fulfillment center supplying products of e-commerce websites. The fulfillment center employs a manual picker system where pickers move on foot, collecting products from a batch of orders that were requested. Efficiency of an order picking process is defined by the time the pickers spend on collecting orders. This time can be separated into 2 components: orders picking time and orders sorting time. The goal of this project is to reduce order picking time by solving a storage location assignment problem (SLAP). The project consists of two main parts: picking time prediction and picking time optimisation by solving SLAP. As a result a Random Forest model was trained to predict orders picking time based on storage location of products. This model was used to estimate walking time change for new assigned product storage locations (new warehouse layout). New product storage locations were generated using suggested heuristic based on SHAP values obtained from the trained Random Forest model. The new layout satisfies required constraints and can be implemented in the corresponding real world warehouse, while being 23% more efficient in terms of order collection time.

- Project repository link:
  https://github.com/MaxKmet/OrderPickingTimeOptimization

Acknowledgements

I am very grateful to Andriy Slyusar, the owner of a fulfillment center, who told me about opportunities for improvement in his job, so together we came up with an idea for my project. He also shared the data with me, using which I did all my experiments. I also want to say thank you to my supervisor Oleh Lukianykhin and Stepan Fedynyak, for mentorship and valuable comments that helped me structure my work and gave inspiration. Finally, I am grateful to Ukrainian Catholic University and the Faculty of Applied Sciences for providing students with opportunities and bringing us together to get knowledge, get inspired and make the world better.

Contents

List of Figures

List of Tables

List of Terms and Abbreviations

Warehouse layout	the way storage units (like shelves) and products are located in a warehouse
SKU	Stock Keeping Unit, a single item of inventory
OPT	Order Picking Time
SLAP	Storage Location Assignment Problem
MAE	Mean Absolute Error
SHAP	SHapley Additive exPlanations

Dedicated to my parents, who always support me and provided
with everything I ever needed to grow personally and professionally.

Chapter 1

Introduction

1.1 Problem Background

We are dealing with a fulfillment company, which handles product storage and delivery for multiple e-commerce websites. The processes inside different fulfillment centers may vary, but the principles remain similar. In this project we are optimizing processes of a particular fulfillment warehouse, located near Kyiv, whose owners provided us their data, so our solution is based on the assumptions and principles described by owners of this particular fulfillment company. This company owns several warehouses in close locations and would like to distribute workers' labor across them. One option to do so is by minimizing time required for collecting orders in a warehouse. This way the fulfillment company would be able to increase productivity of individual workers, hire less people (or expand their e-commerce clients base), resulting in increased revenue.

1.2 Problem Setup

Let's discuss the processes happening inside our warehouse. The orders are collected in bulk called "collection". Usually there are several orders in one collection, and every order may have several products. The collections are generated from orders that were requested by e-commerce customers. In most cases several collections are generated per day. The whole process of orders collection involves the following steps:

1. A number of orders that were requested during the day are waiting to be shipped.
2. A collection of 1-50 orders is generated.
3. A program generates a path to the cells where needed products for this collection are located. The locations are provided to the picker in alphabetical order, and the picker approaches the cells in that order. Usually a worker has to do the whole walk in store along all the shelves to collect the orders. But in the case of a low number of products some rows/cells might be skipped and these cases are not possible to track using our data. The products for a current collection are collected in a shopping cart all together. Picking time usually takes 30 mins-2hrs per collection. This picking time is our main point of interest and object of optimization.
4. After the products are collected, they are sorted into individual orders on a dedicated table. It takes 30 mins-1.5hrs.
5. The orders are then loaded to a truck and shipped via a delivery provider.

Product storage locations are grouped in rows. Every location (place) has a dedicated number or letter. Some places have several levels (on different heights) and/or sub-cells as you can see in the image. On one cell/level/sub-cell several different products may be located. Locations with names starting with Π are pallets and a location named X is a refrigerator.

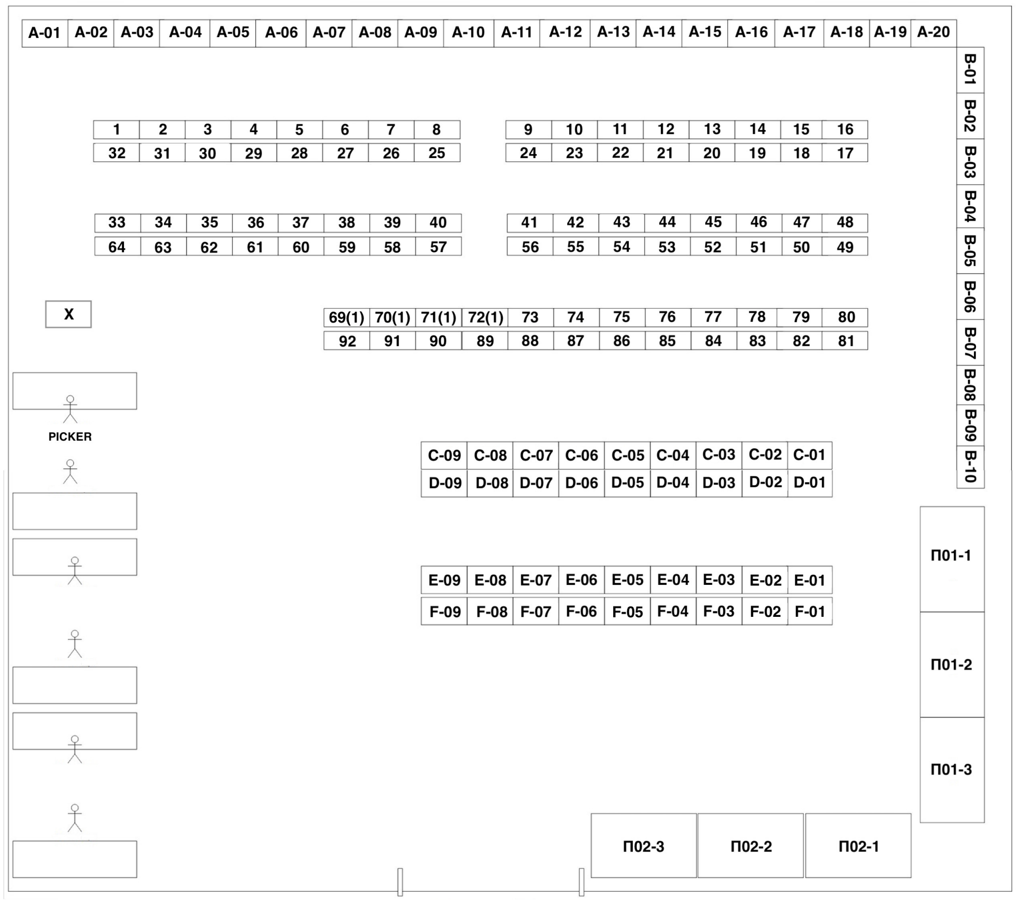
A-01 A-02 A-03 A-04 A-05 A-06 A-07 A-08 A-09 A-10 A-11 A-12 A-13 A-14 A-15 A-16 A-17 A-18 A-19 A-20
B-01 B-02 B-03 B-04 B-05 B-06 B-07 B-08 B-09 B-10
1 2 3 4 5 6 7 8
32 31 30 29 28 27 26 25
9 10 11 12 13 14 15 16
24 23 22 21 20 19 18 17
33 34 35 36 37 38 39 40
64 63 62 61 60 59 58 57
41 42 43 44 45 46 47 48
56 55 54 53 52 51 50 49
X
69(1) 70(1) 71(1) 72(1) 73 74 75 76 77 78 79 80
92 91 90 89 88 87 86 85 84 83 82 81
PICKER
C-09 C-08 C-07 C-06 C-05 C-04 C-03 C-02 C-01
D-09 D-08 D-07 D-06 D-05 D-04 D-03 D-02 D-01
E-09 E-08 E-07 E-06 E-05 E-04 E-03 E-02 E-01
F-09 F-08 F-07 F-06 F-05 F-04 F-03 F-02 F-01
П01-1
П01-2
П01-3
П02-3 П02-2 П02-1

Figure 1.1: warehouse map

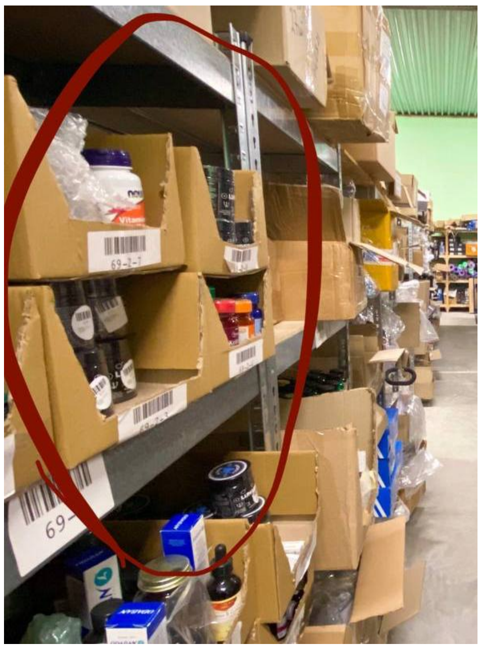

Figure 1.2: sub-cell photo

1.3 Project Goal

Time spent on collecting orders is defined by 2 components: orders picking time and orders sorting time. Considering the needs of the fulfillment company, the main objective of this project is to propose a new product storage assignment that will require less time to pick orders on average than the current one. In order to do this, the first step is to create a model that will estimate order picking time in existing setup (relying on features, applicable to make time estimations in a new setup). The second part of the project is suggesting a new warehouse layout (new product placement locations) that will have better average order picking time estimates according to our model than the original setup.

1.3.1 Restrictions and assumptions

After a discussion with a domain expert, who has been working in our warehouse, we came up with a set of limitations and suggestions that we should consider during our modeling:

1. Products stored in the warehouse are low volume, light weight and particular products generally don't cause delay in collection time (due to weight or else). But some products have to be stored only on pallets or only in a refrigerator, so accessing and locating these products requires special treatment. That is why during our storage locations assignment step we decided to leave the products stored on pallets and in the refrigerator in the same places as before.

2. Sometimes it happens that for 1-2 orders a new collection was generated and processed, though generally collections contain 20-30 orders. Such collections with low number of orders may require special treatment

3. Errors may occur during time logging (due to human factors or program bugs). So it is important to filter out collections with anomaly order collection times. Details of filtering will be discussed in further sections.

4. Every cell/ sub-cell has a particular number of unique products. We consider that every cell/ sub-cell can store at most that number of products that are placed there during the original setup.

5. There is no limits of products that can be stored in a pickers cart

6. Every unique product type is stored only in 1 place

7. Product required for picking is always available in the place it is stored

8. Order picking always starts and finishes from the place where pickers are shown in the warehouse map

9. Set of products available in the warehouse is constant

Chapter 2

Related Works and Background Information

2.1 SLAP problem

Warehouse optimization has been studied for more than 50 years and different approaches and case studies have been made. Most of them incorporate different heuristics for product storage location assignment (SLAP), which we are mostly interested in for our project. They include class-based clustering approaches [3] ,order quantity and product popularity criteria for determining the location of stock items within a distribution center [6], criterions based on complementarity, compatibility, popularity, and size of products [7], clustering based on demand dependencies between SKUs [2]. Most of these approaches are very case specific and researchers use assumed walking distance based measures instead of time (actual metric we want to optimize) to evaluate goodness of new storage layout. Still, some of these principles can be applied in our project.

2.2 Random Forest

Our first task in this project is to build a model that will be able to estimate order picking time regardless of the assigned storage locations. We will further use this model to estimate improvement provided by new storage location assignment. The features we are going to use include locations of products, number of different types of products in the same cell, number of cells on different level of height etc. These features have complex interrelations, so we need to use models that will be capable of learning them. That is why we suggest using Random Forest.

Random forest is a supervised learning algorithm which uses ensemble learning method for classification and regression. In particular, random forest uses assembling method called bagging, which involves training several independent predictors and combining their predictions by some averaging techniques in order to reduce variance. In random forest, an independent predictor is a decision tree. A decision tree is a tree where each node represents a feature, each branch represents a rule and each leaf represents an outcome(categorical or continuous value). This approach to modeling is close to human thinking and considered good for learning complex, non-linear, relations.[1]

2.3 SHAP values

SHAP (SHapley Additive exPlanations) is a game theoretic approach to explain the output of any machine learning model. As we have already mentioned, SHAP method attributes to each feature an importance value (named SHAP value) that represents the contribution of that feature to the final outcome of the model. [4] Suppose for example that we have a model $f(x)$ and that we want to explain the prediction of the model on a specific sample x^*. To do so, SHAP method decomposes the prediction into the sum of the contributions of all the features, namely $f(x^*) = \phi_0 + \sum_{j=1}^{M} \phi_j^*$ where ϕ_0 is the average model's prediction. To compute each contribution ϕ_j, SHAP method relies on a concept that comes from cooperative game theory, known as Shapley value, which is defined as the average marginal contribution of a feature value over all possible coalitions. Implementation of computing SHAP values for tree based models is available in a python package called shap.

Chapter 3

Data and Evaluation

3.1 Data description

The data provided consists of 3 files that have primary keys allowing us to merge them. The values ranges in the description are shown after filtering to be more informative. The filtering itself is described in the following section.

3.1.1 Orders in collections data

name	type	values
collect_id	Integer	2683 -17841
order_id	Integer	169994 -381779
products_quantity	Integer	1-424
collect_date	Timestamp	2020-12-12 14:04:00 - 2022-04-23 15:03:00
num_orders	Integer	1-103
created_at	Timestamp	2020-12-12 12:25:08 - 2022-04-23 14:58:26
collect_time	TimeDelta	0 days 00:00:30 - 0 days 05:39:31
approx_store_walk_end	Timestamp	2020-12-12 14:04:00 - 2022-04-23 15:03:00
approx_store_walk_time	TimeDelta	0 days 00:00:30 - 0 days 02:59:43

Table 3.1: Order in collections data values

name	description
collect_id	Unique identifier of a collection (primary key)
order_id	Unique identifier of an order. One collection may have several orders
products_quantity	total number of products in order (considering some products may appear several time in the order)
collect_date	Time when collection was finished, meaning it was picked from warehouse and sorted
num_orders	Number of orders in collection
created_at	Time when collection was generated. We consider this time to be time when order picking started
collect_time	collect_time - created_at
approx_store_walk_end	Time when the first order of a collection was sorted. We consider that this is approximately the time when order picking was finished.
approx_store_walk_time	approx_store_walk_end - created_at

Table 3.2: Order in collections data description

3.1.2 Products list data

name	type	values
product_id	Integer	616-45851
sku_code	Integer	181-17895
name	String	
place	String	

Table 3.3: Products list data values

name	description
product_id	Unique identifier of a product (primary key)
sku_code	Code identifying a product. Not used in other tables
name	Name of a product. May be in Ukrainian, English, number or empty
place	Storage location identifier in format <location_number> - <level> - <sub_level> or <location_letter>-<location_number>-<level> - <sub_level>. Level and sub_level are optional.

Table 3.4: Products list data description

3.1.3 Orders list data

name	type	values
order_id	Integer	169994 -381779
product_id	Integer	616-45851

Table 3.5: Orders list data values

name	description
order_id	Unique identifier of an order (primary key)
product_id	Unique identifier of a product. One order may have several products

Table 3.6: Orders list description

3.2 Data Pre-processing

3.2.1 Filtering

A domain expert, who has been working in the warehouse as an order picker for a while, provided us with limitations of time of order picking/sorting that we can consider as top and bottom limits. Collections that don't satisfy these limits were deleted because they are considered errors in time tracking. This is the list of collections that were deleted:

- collections with negative total collection time
- collections with 3+ hours of walk time
- collections with < 30 seconds of picking time
- collections with 3+ hours of sort time

3.2.2 Transformation

The following transformations are performed on the input data:

1. Transform time deltas from TimeDelta data type to seconds
2. Cubic root of target. Our target value (*approx_store_walk_time*) has a skewed distribution. The skewness (Fisher-Pearson coefficient of skewness) is 1.675, which is considered high. Some models, especially linear, may have difficulties fitting to skewed data. That is why we compare predicting raw seconds of target and predicting cubic root of target and transforming it back to seconds using power 3 function. As will be presented later, this approach gave us improvement in most cases compared to not applying cubic transform. Cubic transform of *approx_store_walk_time* has a distribution with skewness 0.38 (fairly symmetrical). Also after applying cubic root transformation, Pearson correlation between number of products in collection and target slightly raised from 0.59 to 0.65.

Further feature transformations, related to individual experiments, are described in a section dedicated that experiment.

3.2.3 Splitting data

Length of data (number of collections in the dataset) – 3097
Train and test set were randomly splitted in proportion 80/20.

During the experiments, additional splitting of the dataset was done to experiment how training different models for different splits of data (with respect to number of orders as suggested by domain expert) affects the quality of models performance. This process will be explained in more detail in following sections.

3.3 Evaluation

Now that we understand the project goals and the data, let us define the key metrics for measuring quality of order picking time prediction and changes in order picking time provided by SLAP solution.

MAE – mean absolute error. This metric was chosen because it is less influenced by outliers, which appear in the data due to human factor and other reasons, which are not detectable from our data (for example someone called worker while order picking and it caused a delay, or some product appeared missing, which caused the delay). Also this metric intuitively translates into real life values, in our case – number of seconds. The lower the value of this metric, the better. We also use it as a criterion in our Random Forest regressor.

$$\mathbf{MAE} = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n}$$

MAE = mean absolute error y_i = prediction x_i = true value n = total number of data points

Percentage of collections in validation set with MAE less than 5 minutes. After discussion with a domain expert about handling human factor which causes variance in results, we agreed to consider that collection times that are predicted with error less than 5 minutes are considered to be predicted perfectly. That is why this metric is also useful for monitoring.

Coefficient of determination $\mathbf{R^2}$ – is the proportion of the variance in the target variable that is described from the independent variables by a model. The desired value of the metric is 1, but in some cases of poor model fitting, computational definitions may yield negative values.

Average order picking time improvement - average difference between order picking time predicted in an original setup and order picking time predicted in new setup suggested by SLAP solution. This metric is used to measure changes in order picking time provided by a suggested SLAP solution.

Chapter 4

Predicting order picking time

In order to predict order picking time, we performed several experiments involving different approaches to data splitting, different models, target value transformation and ensembling of previous results. Below we list models, target transform, data splitting approaches and features which are later used in different combinations.

4.1 Models

1. Linear regression - actually linear regression assumptions (like homoscedasticity and normally distributed residuals) were not met, but this is a common baseline model to compare more complex models with.

2. Random forest – fine tuned using Grid Search with 5-fold cross-validation and tested parameters { 'n_estimators': $[50, 100, 200]$, 'min_samples_split': $[2, 10, 30, 0.02, 0.05, 0.1$, $0.2]$, "criterion": ["absolute_error"']}

4.2 Target transform

1. No target transformation

2. Cubic root of target. Motivation behind it described in Pre-processing section

4.3 Data splitting

1. No splitting

2. Separate modeling for collections with [1;2] orders (40% of all data) and [3; inf) orders (60% of the data). This idea is based on a suggestion number 2 from a domain expert described in Restrictions section.

4.4 Features

Every model was trained on a separate set of features out of the ones suggested below. We did not use multiple suggested features at once because they are highly correlated and might worsen the models performance. Still, in order to experiment with combining different features, we used an ensemble of models trained on different features. Here is a list of suggested extracted features:

feature_id	feature	explanation
1	Total number of products in collections	A baseline feature, because it is not optimizable. You can't change the number of products needed because it does not depend on storage layout.
2	Number of products required to take from locations	The feature is a vector of 156 entries, where each entry represents the number of products required to be taken from a particular location. Sub-cell and level on location is not encoded, because then we have too low (number of samples) / (number of features) proportion and models fail to fit, returning a negative R^2 score. This feature is the most valuable for optimization we want to make in part 2 of our project, because representation of a product location assignment is straightforward in this case.
3	Number of cells on different level as a feature	The feature is a vector of 3 elements because we have at most 3 levels of location. If a level is not mentioned, we consider the product to be placed on level 2.
4	Number of product to pick up from a particular line (location cluster) in warehouse	We identified 13 location clusters in the warehouse which are shown in a figure below. Every location cluster is painted on a map in a different color. Refrigerator was added to the cluster with pallets because it appeared a low number of times in the dataset, and we decided not to create a separate cluster for it to avoid creating a noisy feature.
5	Number of products to pick up from cells with many other unique products	In many cases different products are located in the same location (sub-cells and levels considered). In particular, here are some statistics. 574 locations have < 5 unique products located, 411 locations have [5;15] products, 129 - (15;30] products, 83 - (30; inf] The feature itself is a vector with 4 entries, each of which is the number of products from the current collection to be picked up from a location falling into one of 4 above mentioned categories.
6	Number of products to pick up from cells of a particular class	By class here we mean: 1) a location without subcell 2) a location with a subcell 3) a pallet 4) a refrigerator. The feature is a vector with 4 entries each of which is the number of products from the current collection to be picked up from a location falling into one of 4 above mentioned categories.

Table 4.1: Features

Below is a visualization of location clusters described in feature 4. Locations highlighted in different colors belong to different location clusters.

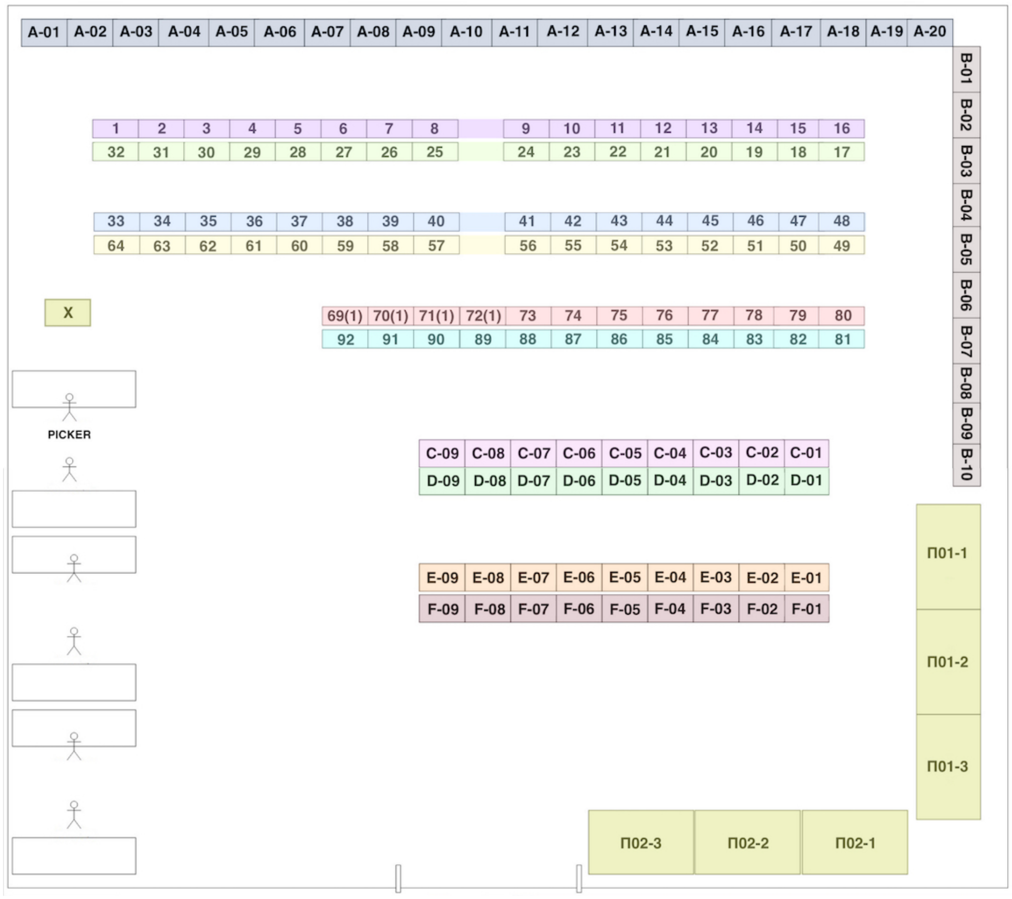
A-01 A-02 A-03 A-04 A-05 A-06 A-07 A-08 A-09 A-10 A-11 A-12 A-13 A-14 A-15 A-16 A-17 A-18 A-19 A-20
B-01 B-02 B-03 B-04 B-05 B-06 B-07 B-08 B-09 B-10
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17
33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
64 63 62 61 60 59 58 57 56 55 54 53 52 51 50 49
X
69(1) 70(1) 71(1) 72(1) 73 74 75 76 77 78 79 80
92 91 90 89 88 87 86 85 84 83 82 81
PICKER
C-09 C-08 C-07 C-06 C-05 C-04 C-03 C-02 C-01
D-09 D-08 D-07 D-06 D-05 D-04 D-03 D-02 D-01
E-09 E-08 E-07 E-06 E-05 E-04 E-03 E-02 E-01
F-09 F-08 F-07 F-06 F-05 F-04 F-03 F-02 F-01
П01-1
П01-2
П01-3
П02-3 П02-2 П02-1

Figure 4.1: location clusters

4.5 Ensembling

After separate models were trained and we made train/validation predictions using them, we used these predictions to experiment with combining outputs of models trained on different features to test if it gives better validation results. In such way we incorporate different features into one ensemble model. The approaches of ensembling that were tried:

Method	Explanation
Linear regression ensembling	Use outputs of models trained on different features as inputs and predict order picking time using them. The features are highly correlated and we may not need all of them so we did 2 things to avoid using unnecessary features and make the most of what we have: Use up to 4 features for 1 linear regression ensemble model Use L1 regularization, which allows penalizing unnecessary features.
Random forest ensembling	Use outputs of models trained on different features as inputs and predict order picking time using them. All features are used for training the model. Also GridSearch implementation with 5-fold cross-validation from python library sklearn is used to tune models parameters. The parameters tested are {n_estimators':[50, 100, 200], 'min_samples_split':[2, 10, 30, 0.02, 0.05, 0.1, 0.2], "max_features":[0.4]}

Table 4.2: Ensembling approaches

Different combinations of the above mentioned configurations were tried. In total we have 2 models, 2 target transforms, 2 data transformation approaches, 2 data splitting approaches (1 of them involves repeating all experiments twice, one for each subset of data), and 6 features. Also 2 sampling techniques were tried. So in total 2 * 2 * 2 * 3 * 6 + 2 = 146 experiments were performed.

4.6 Order picking time prediction results

Below are presented metrics for each of the models predicting order picking time. They are sorted by validation (test) MAE.

4.6.1 All data experiment results

feature_id	model_type	train_mae	val_mae	perc_lt _5mins
6	Rand forest cbrt y preprocessing	18.11	9.47	0.46
6	Rand forest no preprocessing	18.06	9.64	0.45
5	Rand forest cbrt y preprocessing	17.55	9.85	0.46
3	Rand forest cbrt y preprocessing	16.66	10.0	0.44
4	Rand forest cbrt y preprocessing	16.49	10.32	0.41
5	Rand forest no preprocessing	17.52	10.39	0.42
6	Lin reg cbrt y preprocessing	21.21	10.97	0.29
3	Linreg cbrt y preprocessing	20.78	10.97	0.3
3	Rand forest no preprocessing	15.91	11.01	0.37
2	Rand forest cbrt y processing	14.36	11.1	0.35
5	Lin reg cbrt y preprocessing	20.78	11.15	0.32
4	Lin reg cbrt y preprocessing	21.02	11.35	0.26
4	Rand forest no preprocessing	15.41	11.67	0.35
2	Rand forest no y processing	13.89	13.2	0.26
10	Lin reg ensembling features: (0, 1, 6)	15.49	14.2	0.21
2	Lin cbrt y processing	19.03	14.24	0.28
11	Rand forest ensembling	6.61	15.07	0.27
5	Lin reg no preprocessing	21.59	15.13	0.11
6	Lin reg no preprocessing	21.71	15.29	0.1
3	Linreg no preprocessing	21.3	15.4	0.1
4	Lin reg no preprocessing	21.41	15.45	0.11
2	Lin reg no y processing	19.65	16.57	0.09
1	Rand forest no processing	16.78	17.49	0.36
1	Rand forest cbrt y	17.1	17.51	0.36
1	Lin reg x,y cbrt	17.47	17.6	0.31
1	Lin reg no processing	19.56	19.86	0.13

Table 4.3: All data experiment results sorted by performance

The best validation MAE, achieved by a Random forest model, is 9,47 minutes. It corresponds to feature number 6 (number of products to pick up from cells of a particular class) with target transformation using cubic root. The best parameters found using GridSearch are { 'min_samples_split': 0.05, 'n_estimators': 50}. Corresponding train R^2 for this experiment is 0.43.

4.6.2 Data split by number of orders results

In the Data splitting section we described an approach to splitting data into 2 datasets and training models for them separately. The suggestion came from the domain expert, but it was confirmed by observations when plotting data. As you notice from plotting the number of products in order against the target, we have a concentration of observations with just a few products in collection and very different picking times. It makes us think that these collections with small amounts of orders may behave differently than in other range, and require special treatment. As a reminder: 40% of collections contain only 1-2 orders.

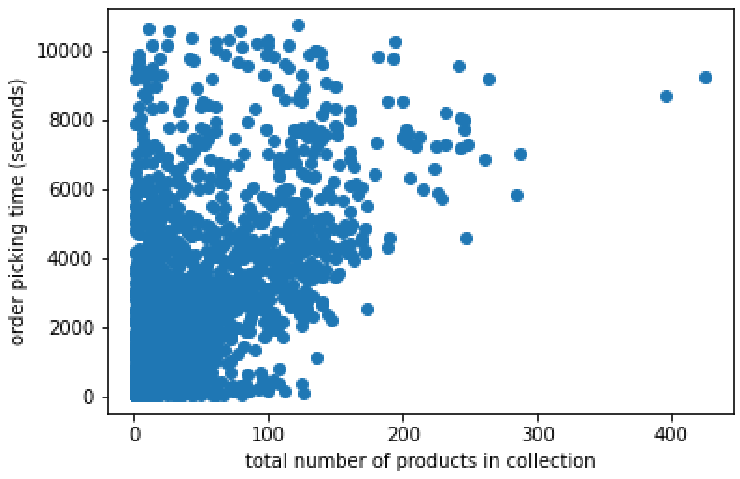

Figure 4.2: correlation plot

<= 2 orders data split results

feature_id	model_type	train_mae	val_mae	perc_lt_5mins
6	Rand forest cbrt y preprocessing	15.1	7.26	0.61
6	Rand forest no preprocessing	14.79	7.39	0.61
3	Rand forest cbrt y preprocessing	14.62	7.52	0.65
5	Rand forest cbrt y preprocessing	14.68	7.73	0.62
4	Rand forest cbrt y preprocessing	13.67	8.01	0.54
5	Rand forest no preprocessing	14.31	8.14	0.58
3	Rand forest no preprocessing	14.12	8.16	0.57
6	Lin reg cbrt y preprocessing	16.96	8.49	0.38
4	Rand forest no preprocessing	13.61	9.57	0.41
2	Rand forest cbrt y processing	12.34	9.92	0.43
4	Lin reg cbrt y preprocessing	16.23	10.71	0.39
2	Rand forest no y processing	12.3	12.5	0.33
10	Lin reg ensambling features: (10, 18, 19)	16.7	12.86	0.17
5	Lin reg cbrt y preprocessing	16.24	13.29	0.37
6	Lin reg no preprocessing	18.93	13.71	0.09
3	Linreg no preprocessing	18.76	14.09	0.09
5	Lin reg no preprocessing	18.66	14.18	0.07
4	Lin reg no preprocessing	18.63	14.29	0.07
11	Rand forest ensambling	6.67	15.48	0.34
1	Rand forest cbrt y	13.13	15.78	0.56
1	Rand forest no processing	12.7	15.89	0.54
1	Lin reg x,y cbrt	13.37	16.24	0.45
3	Linreg cbrt y preprocessing	16.52	17.07	0.39
1	Lin reg no processing	16.17	19.5	0.09
2	Lin reg no y processing	16.47	20.87	0.18
2	Lin cbrt y processing	14.13	27.16	0.33

Table 4.4: Experiment results for data with <= 2 orders

>= 3 orders data split results

feature_id	model_type	train_mae	val_mae	perc_lt_5mins
6	Rand forest cbrt y preprocessing	20.61	10.94	0.36
5	Rand forest cbrt y preprocessing	20.08	11.68	0.32
4	Rand forest cbrt y preprocessing	16.68	12.11	0.3
3	Rand forest cbrt y preprocessing	17.23	12.29	0.27
5	Lin reg cbrt y preprocessing	23.03	12.38	0.24
2	Rand forest cbrt y processing	15.21	12.57	0.27
6	Rand forest no preprocessing	18.82	12.58	0.28
3	Linreg cbrt y preprocessing	22.49	12.72	0.2
6	Lin reg cbrt y preprocessing	22.95	12.73	0.22
4	Lin reg cbrt y preprocessing	22.95	12.96	0.2
5	Rand forest no preprocessing	18.39	13.39	0.23
3	Rand forest no preprocessing	17.19	13.45	0.23
4	Rand forest no preprocessing	16.46	13.66	0.25
2	Rand forest no y processing	11.51	14.9	0.2
10	Lin reg ensambling features: (1, 2, 7)	10.99	15.09	0.25
2	Lin cbrt y processing	19.84	15.29	0.2
11	Rand forest ensambling	3.16	15.68	0.24
5	Lin reg no preprocessing	23.35	16.02	0.14
6	Lin reg no preprocessing	23.3	16.47	0.11
3	Linreg no preprocessing	22.63	16.76	0.09
4	Lin reg no preprocessing	22.7	16.87	0.11
2	Lin reg no y processing	20.17	19.26	0.09
1	Lin reg x,y cbrt	19.51	21.29	0.21
1	Rand forest cbrt y	19.4	21.67	0.22
1	Rand forest no processing	18.66	22.0	0.19
1	Lin reg no processing	21.04	22.02	0.15

Table 4.5: Experiment results for data with >= 3 orders

The performance of models on data split should be calculated together using waiting. Since the data was split in proportion 40/60, metric = metric_for_split1 * 0.4 + metric_for_split2 * 0.6. The best combination of models by this "weighted" validation MAE is **9.45** minutes. Corresponding train R^2 is 0.43. Weighted percentage of collections in validation set with MAE less than 5 minutes is 46%.

For split 1 (<= 2 orders): Random forest model, feature number 6 (number of products to pick up from cells of a particular class) with target transformation using cubic root. The best parameters found using GridSearch are { 'min_samples_split': 0.1, 'n_estimators': 100}.

For split 2 (>= 3 orders): Random forest model, feature number 6 (number of products to pick up from cells of a particular class) with target transformation using cubic root. The best parameters found using GridSearch are { 'min_samples_split': 0.2, 'n_estimators': 100}.

4.7 Conclusion for order picking time prediction

The best quality of order picking prediction was achieved by combining 1) data splitting into 2 categories (<=2 orders and >=3 order), 2) using the number of products to pick up from cells of a particular class as a feature 3) applying cubic root transform to target during training 4) and using Random forest models with appropriate reported parameters. This approach might be the best of explored for predicting picking time itself, but it is not very suitable for part 2 of our project - optimization by solving storage location assignment problem, because this feature considers only location classes, not the locations themselves.
So for part 2 of our project we will use the best approach for order picking time prediction using feature 2 - number of products required to take from locations, because it directly deals with locations of products, and therefore can be optimized using SLAP. Best MAE involving feature 2 is **11.4** mins. Corresponding percentage of collections in validation set with MAE less than 5 minutes is 0.34.
As for the reasons why location classes gave slightly better performance than locations themselves, we explain it by a relatively small size of the warehouse, which made spacial features slightly less important than information about location of products in the context of one place.

Chapter 5

Optimization of order picking time

In this part of the project we use the model trained to predict order picking time in the previous part, in order to estimate improvement achieved by solving the SLAP problem with different methods.

5.1 Methodology

We use the same validation set that was used for measuring validation metrics in the first part of the project but for a different purpose. First we save validation predictions from our model for original setup features (original product storage locations). Then we solve the SLAP problem using one of the suggested approaches (change product storage locations) and generate validation predictions for a new setup. After that we compare the two predictions using the average order picking time improvement metric explained in the Evaluation section.

5.2 Approaches

5.2.1 Alphabetical sorting

A domain expert explained, the program they use for generating collections returns locations where these products are located, sorted in alphabetical order. Then order picker approaches these locations in given order to pick up the needed products, Based on that, our first approach is the following:

1. Sort products by number of orders in which they appear in train set
2. Assign more frequently appearing products to locations starting from first products coming in alphabetical order

Note: We fill every location with at most the number of products that were located there in an original setup to address restriction 4 in the Restrictions section.

5.2.2 Rows locations clustering

Every row in the warehouse has a particular distance from the exit of the warehouse. Based on that knowledge (from a map visualization) we sorted rows in the warehouse based on that distance and intuition and started placing popular products starting from rows closer to exit, within the rows products are placed on locations in alphabetical order. In the picture below we marked rows of cells with order in which they start being filled (yellow to red). Indexes are also added next to each cluster in order which they start being filled, starting from 1. Palettes and the refrigerator are left blank on purpose because the locations of products placed there are not altered.

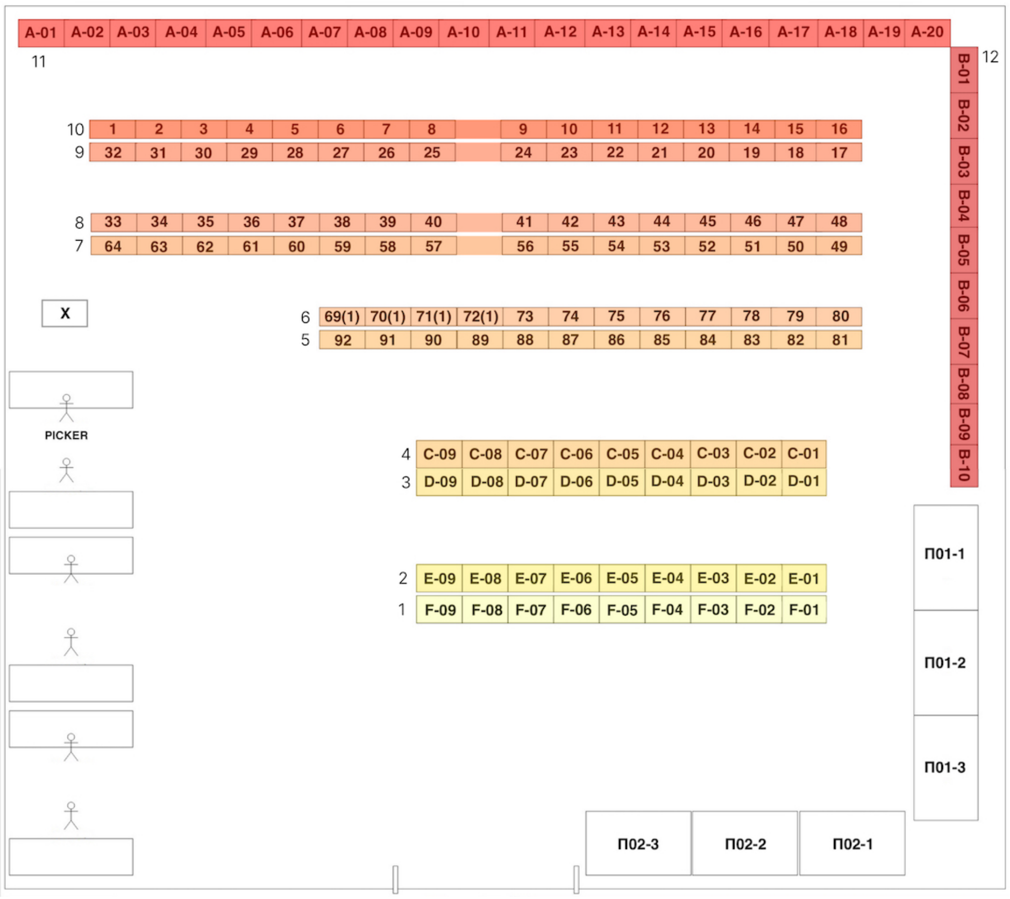
A-01 A-02 A-03 A-04 A-05 A-06 A-07 A-08 A-09 A-10 A-11 A-12 A-13 A-14 A-15 A-16 A-17 A-18 A-19 A-20
11
12
B-01 B-02 B-03 B-04 B-05 B-06 B-07 B-08 B-09 B-10
10 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
9 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17
8 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
7 64 63 62 61 60 59 58 57 56 55 54 53 52 51 50 49
X
6 69(1) 70(1) 71(1) 72(1) 73 74 75 76 77 78 79 80
5 92 91 90 89 88 87 86 85 84 83 82 81
PICKER
4 C-09 C-08 C-07 C-06 C-05 C-04 C-03 C-02 C-01
3 D-09 D-08 D-07 D-06 D-05 D-04 D-03 D-02 D-01
2 E-09 E-08 E-07 E-06 E-05 E-04 E-03 E-02 E-01
1 F-09 F-08 F-07 F-06 F-05 F-04 F-03 F-02 F-01
П01-1
П01-2
П01-3
П02-3
П02-2
П02-1

Figure 5.1: sorted location clusters

5.2.3 SHAP based sorting

As explained in the Related works and Background Information section, SHAP values represent the contribution of individual features to the final outcome of the model. Intuition behind them is they show how individual features shift the predicted value from the average prediction. In our case, the feature is a number of products required to take from individual locations, so we have 156 features corresponding to locations on map and SHAP values related to them. We have 2 Random Forest models trained on different data splits, so we have different SHAP values for them. Each of this models has different average prediction value and different magnitude of SHAP values prediction, so the SHAP value vectors need to be normalized (transformed to a unit length) so that they can be compared or combined.

Below is an example visualization of SHAP values for Random Forest model, trained of second data split (>=3 orders). They are ordered by their effect on prediction. SHAP values were calculated using TreeExplainer class provided by shap python library. In this example feature 75 corresponds to number of products in place 84, feature 98 - number of products in place A-3, feature 26 - number of products in place 33.

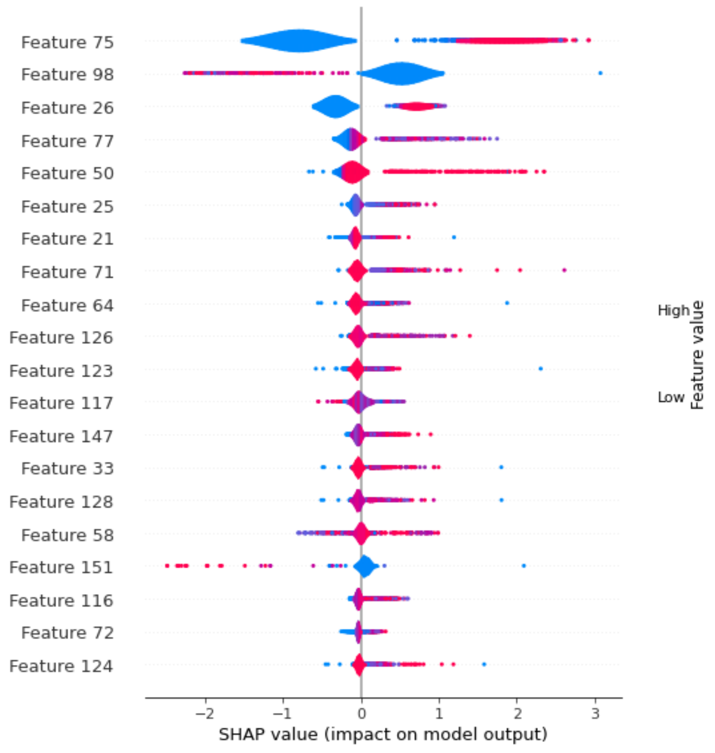

Figure 5.2: SHAP example

Every little dot represents an observation. The horizontal axis represents the SHAP value, while the color of the point shows us if that observation has a higher (red) or a lower (blue) value, when compared to other observations.[5]
In this example, higher values for feature 98 have a negative impact on the prediction, while lower values have a positive impact. Intuitively it means that it takes less time to take products from place A-3.
The suggested algorithm using SHAP values looks like this:

1. Calculate SHAP values for every feature for both Random Forest models (for 2 data splits)
2. Normalize both SHAP vectors to unit norm
3. Create a combined SHAP vector by weighting 2 original vectors proportionally to data split size on which their models were trained. The final SHAP vector is calculated SHAP_vect1 * 0.4 + SHAP_vect2 * 0.6.
4. Sort locations by their related SHAP values
5. Assign products by popularity, placing more popular products to locations with lower SHAP value.

5.2.4 Optimization of order picking time results

Below are the results of the described experiments sorted by average order picking time improvement.

Approach	Average OPT improvement (minutes)	Average relative OPT improvement (percentage)
SHAP based sorting	5.3	0.23
Rows locations clustering	3.2	0.14
Alphabetical sorting	0.005	0.0

Table 5.1: Optimization of order picking time results

Average order picking time predicted in the original setup is 22.8 minutes. The best average order picking time predicted in a new setup achieved by SHAP based sorting is 17.5 minutes. It gave us **5.3** minutes of absolute improvement and **23**% of relative improvement.

Chapter 6

Summary

A new products locations assignment was created for our warehouse, that is expected to give **23**% improvement in order picking time, according to our model. It takes into account constraints requested by warehouse owners such as location capacity limitations and special dedicated places for some products.

The project was divided into 2 parts: predicting order picking time and solving a storage location assignment problem. Evaluation of improvement by part 2 was done using a model created during part 1.

We tried predicting order picking time from 6 suggested features using Linear Regression, Random Forest with different target preprocessing and data splitting approaches. The best validation metrics were demonstrated by Random Forest predicting cubic root of order picking time using the number of products to pick up from cells of a particular class as a feature, trained on 2 separate data splits. However, for evaluation of SLAP solutions in part 2 we used a different model. The model used was Random Forests predicting the cubic root of the number of products required to take from locations as a feature trained on 2 separate data splits. The reason for using a different model is that it does not have a big difference in validation metrics from the best approach, but that is more important, it uses features that can be optimized using SLAP.

The best average order picking time improvement, according to our model, was achieved by using SHAP based sorting approach for solving SLAP. It gave us **5.3** minutes of absolute improvement and **23**% of relative improvement compared to the original warehouse setup. This means a proportional decrease of costs and the corresponding potential increase of revenue.

The new storage locations satisfy all the constraints discussed with domain experts and can be implemented in real life. This approach can be also used for optimizing other fulfillment warehouses that have similar work processes set up.

Bibliography

[1] Afroz Chakure. Random Forest Regression - The Startup. Mar. 2022. url: https://medium.com/swlh/random-forest-and-its-implementation-71824ced454f.

[2] Monika Kofler et al. "Robust Storage Assignment in Warehouses with Correlated Demand". In: Studies in Computational Intelligence (2015), pp. 415–428. doi: 10.1007/978-3-319-15720-7\{_}29.

[3] René de Koster, Tho Le-Duc, and Kees Jan Roodbergen. "Design and control of warehouse order picking: A literature review". In: European Journal of Operational Research 182.2 (2007), pp. 481–501. doi: 10.1016/j.ejor.2006.07.009.

[4] Mukund Sundararajan. The many Shapley values for model explanation. Aug. 2019. url: https://arxiv.org/abs/1908.08474.

[5] Vinícius Trevisan. Using SHAP Values to Explain How Your Machine Learning Model Works. Mar. 2022. url: https://towardsdatascience.com/using-shap-values-to-explain-how-your-machine-learning-model-works-732b3f40e137.

[6] Hoyt G. Wilson. "Order Quantity, Product Popularity, and The Location of Stock in a Warehouse". In: A I I E Transactions 9.3 (1977), pp. 230–237. doi: 10.1080/05695557708975151.

[7] D. Daryl Wyckoff and Ronald H. Ballou. "Business Logistics Management". In: Journal of Marketing 37.4 (1973), p. 119. doi: 10.2307/1250368.