Technical University of Crete

Department of Electrical and Computer Engineering

MEREBO2: A web mart analyzer for Multi-source Entity REsolution and Basket Optimization over Android Mobile Devices

Author:

Konstantinos Koulaouzidis

Committee:

Asst. Professor Nikos Giatrakos (supervisor)

Prof. Minos Garofalakis

Prof. Georgios Chalkiadakis

Abstract

In today's fast-paced world of e-commerce, where an ever-expanding array of products is available to the customers, the ability to efficiently and accurately identify similar products has become a critical component of online market analysis. In this context, image comparison techniques have emerged as a powerful tool for product similarity analysis. This thesis embarks on a comprehensive exploration of image comparison algorithms and methods specifically tailored for the identification of similar products to enhance the performance and credibility of MEREBO [1]. Furthermore, while the identification of similar products through image comparison is undeniably valuable, the scalability and execution time of such processes remain significant challenges. In the era of online retail, where vast product catalogs are common, it is imperative to develop algorithms and systems to not only provide accurate results but also do so in a swift and resource-efficient manner. Hence, the second focal point of this thesis is to address the pressing issue of execution time with the ultimate goal to make it appealing on real world applications of e-commerce. The combination of product similarity accuracy and time efficiency will contribute to a more seamless and productive online shopping experience, benefiting both the consumers and retailers alike. The last pivotal aspect of this thesis centers around the practical applicability of such systems in the mobile environment. This entails addressing the unique challenges posed by mobile platforms, such as limited computational resources, varying screen sizes and diverse network conditions. This third dimension, alongside accuracy and time efficiency, forms a holistic approach to advancing product similarity analysis in the context of e-commerce, aiming to improve the way consumers interact with and make informed choices in the digital marketplace.

Acknowledgements

First of all, I would like to express my sincere gratitude to my supervisor, Assistant Professor Nikos Giatrakos for giving me the opportunity to work on this thesis. He provided invaluable guidance and acted as a mentor throughout the research process and was there to answer all my questions.

I am also thankful to my committee, Professor Minos Garofalakis and Professor Georgios Chalkiadakis for participating in this step of my journey. I would also like to thank all the professors and the assistants of the Electrical and Computer Engineering department and their efforts throughout my years in the University. I would also like to thank Ioannis Misokalos for providing me with valuable help and information about MEREBO.

On a more personal note, I would like to thank my family for their unwavering love and support all these years. Without them none of my life's achievements, including this thesis, would have been possible. Special thanks goes to my good friend Manos for graciously opening their home to me amidst my last travels to Chania, helping me during a time of transition. Lastly, I would like to extend a thank you to my family's loyal dog, Roumi, who provided comfort and companionship during the long hours of research and writing.

Table of Contents

Table of Figures

List of Tables

1. Introduction

1.1 MEREBO: Departure point for MEREBO2

As is evident from the title, this thesis represents a logical continuation of its predecessor, *MEREBO* [1]. It is, therefore, prudent to provide a succinct overview of the existing framework and to identify the issues at hand.

MEREBO stands for Multi-source Entity REsolution and Basket Optimisation. It is a web application that is able to collect and process grocery store product data, compare and match the products between those stores, showcase the matches to the consumers and provide them with an optimization option in their basket so that they may have the best shopping experience.

The initial stage of the system involves the process of collecting data from three Greek grocery stores: *Sklavenitis*, *AB Vasilopoulos* and *Chalkiadakis*. This is achieved through the employment of the *Selenium* [2] web crawler tool. The result of the aforementioned process is a database that contains all important information about each product. This information is:

- Name (or description) of the product
- Name (or code) of the store
- Price per unit
- Supplementary price of product
- Image URL (for displaying purposes)

The next stage includes an algorithm that can efficiently recognize and match identical multi-source entities and provides aggregated data that is used to create the final database of the application. A more in depth analysis of the matching stage will be explored in Chapter 2 of this thesis. The outcome of this part of the system is a CSV file in the following format:

Every row contains information for up to 3 products, corresponding to each different potential store. The first product in the sequence represents the product that the algorithm

tries to find a match for. Consequently, the other two entries denote the products that the algorithm assessed as similar and found in a store other than the initial one in the row. This structure implies that in instances where a product is exclusive to its designated store, the corresponding row will contain solely a singular item.

This data is then stored in an *Apache Solr* [3] database, which is also used as a search engine, that is accessed via a dedicated server that handles the connection between the clients and the application. Consumers engage with the system through the user interface provided by the application's web page, enabling them to search for products of interest.

Lastly, in the server side of MEREBO's web application, we can find a basket optimization algorithm that finds and forms the combination of the user's basket that provides the best price while also meeting the lowest cost requirement for an online order. This optimized basket can be fulfilled by a singular store or multiple stores, provided that each selected shop has its lowest-cost order satisfied.

The optimization problem MEREBO solves can be formulated as follows:

Parameters:

i : items selected by the customer
j : the stores
q_i : the quantity of each item i requested by customer
x_{ij} : the quantity of item i purchased from store j
p_{ij} : the price of a singular item i when purchased from store j
L_j : the lowest cost order per store j

Objective:

$$\text{minimize cost} = \sum_{i,j} p_{ij} * x_{ij}$$

Constraints:

$$q_i = \sum_j x_{ij}$$

$$\forall\ j:\ \sum_i x_{ij} * p_{ij} \geq L_j \quad \text{OR} \quad \sum_i x_{ij} * p_{ij} = 0$$

1.1.1 MEREBO: Illustrated example

Using *Figure 1*, a more illustrative demonstration of the functionality of the MEREBO system can be presented. At this point, the system has already gathered the items along with their respective information from their stores and the matching algorithm has also already created this triple match. Then, when the user searched for the term 'μπισκότα παπαδοπούλου', this is the match that was evaluated as the best match. We can see that every product border is a different color, indicating the different stores where the item was found. The user can either 'ADD TO CART' the item from a specific store or 'AUTO CHOOSE' to let the optimization algorithm choose for them, one out of these 3 options.

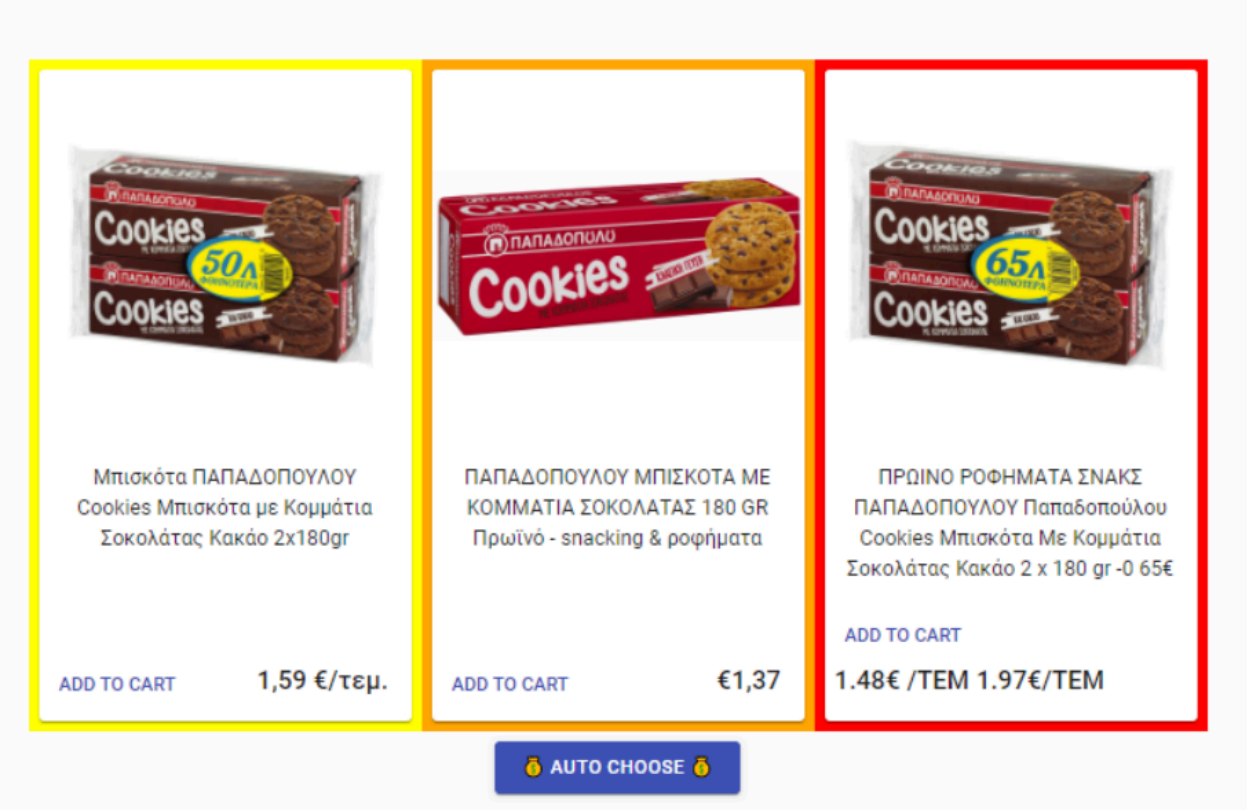


Figure 1: Merebo's web application interface

1.2 MEREBO2's contributions

This thesis' focal point is to enhance the precision of the pre-existing well-calibrated model. We will discuss the tools and methods we experimented with in our endeavor along with their respective strengths and weaknesses.

In practical aggregative web mart analysis and entity matching applications, it is also crucial to provide consumers with the most up to date data so they can make an informed decision. To tackle this problem, we added the functionality to auto perform web scraping and the follow up data matching once per day. While the ideal scenario would entail a real-time feed of web content changes, it is worth noting that the non-cooperative nature of the model we are working with dictates the compromise of a daily update schedule, balancing the need for up-to-date information with the system's inherent limitations.

Another aspect of the system that warrants enhancement is its time efficiency. In order for the application to be viable in real-world scenarios, we have to make sure that the entire process is completed within a 24-hour timeframe. The solution of this problem is quite complex and we find its roots in the first two stages of MEREBO's infrastructure, the web scraping and the matching algorithm. Several strategies and methods can help us tackle this objective, each with its own advantages and drawbacks. In the following discussion, we will present these methods and provide a comprehensive analysis of their respective merits and limitations with our goal being the improvement in time consumption by an order of magnitude.

Our final goal is the migration of the application onto mobile devices, given the prevailing trend of increased reliance on smartphones for access and interaction. This transition aims to align the system with evolving user preferences, as mobile platforms have become dominant in the digital landscape. We also aim to improve the user experience by making the application intuitive and providing the customer with more results to choose from.

1.3 Summary of thesis' goals

The final product will be configured to emulate real-world deployment within the competitive landscape of e-commerce, retaining a high degree of fidelity to practical implementation. The sole remaining constraint pertains to the non-cooperative nature of the system.

The contributions of this thesis and, thus, of MEREBO2 is that it enhances MEREBO along the following performance axes:

- It improves the model's accuracy by implementing image comparison.
- It automates web mart data scraping and categorization.
- It offers the desired functionality readily available to consumers via their Android Mobile Devices.

The aforementioned outline is how our thesis is organized for demonstration.

2. Accuracy

The first goal of this thesis is the enhancement of MEREBO's accuracy. In this chapter we discuss the methods we employed to achieve that goal and some methods that we experimented with but yielded less efficient results.

Before delving deeper into our experiments, we should briefly demonstrate how MEREBO is modeled to identify text similarities.

The input dataset, that contains the various products in our disposal, is preprocessed. Text preprocessing is a crucial step in Natural Language Processing (NLP), aiming to enhance the quality and suitability of textual data for analysis. It involves a series of operations that transform the text into a more refined and structured form. One reason text preprocessing is considered mandatory is that it helps in the removal of irrelevant or noisy information such as special character, punctuation and unnecessary whitespaces, thus improving the ability of detecting similar terms. Another important factor is that it performs a series of actions such as lowercasing, stemming and lemmatization of the text, which converts it to a consistent format. By clearing, normalizing and structuring data, text preprocessing facilitates a more effective text analysis and text similarity detection, common tasks in the context of similarity analysis.

After the preprocessing of product's descriptions, we calculate the *TF-IDF* values for each term in each product. *Term Frequency-Inverse Document Frequency or TF-IDF* is a numerical statistic used in NLP to evaluate the importance of a term in a document relative to the collection of documents. It consists of two components: *Term Frequency (or TF)* and *Inverse Document Frequency (IDF)*. TF measures how frequently a term appears in a document. On the other hand, IDF quantifies how special or important a term is across the entire collection of documents, giving higher weights to terms that are less common. The *TF-IDF* score as a whole is calculated by multiplying these two components, resulting in a value that ultimately marks terms that are both frequent within a document and at the same time relatively rare in the whole dataset. This method is established as one of the best tools for text analysis and information retrieval especially in machine learning and NLP tasks.

For a term **t** and a document **d** in a corpus of documents we calculate the the Term Frequency and the Inverse Document Frequency:

$$TF(t,d) = \frac{number\ of\ times\ t\ appears\ in\ d}{total\ number\ of\ terms\ in\ d}$$

$$IDF(t) = \frac{number\ of\ documents}{number\ of\ documents\ that\ contain\ t}$$

$$TFIDF(t,d) = TF(t,d) * log(IDF(t))$$

With the *TF-IDF* measured, the similarity between every combination of products is calculated by finding the *cosine similarity* between their *TF-IDF* scores. Every score that is higher than the established threshold is considered as the valid match until a higher score appears that replaces it.

To enhance this decision making, MEREBO trained a machine learning model based on the *Random Forest Classifier* [18] algorithm. Each combination of products is evaluated by the model and if it finds it (dis-)similar, the overall similarity score of that pair is (de-)creased. MEREBO also introduces a floor to how low cosine similarity could be before feeding a potential entity pair match in the Random Forest model. In other words, if the cosine similarity computed in the previous step is not high enough, the respective pair will be discarded anyway.

We now have a good understanding of the architecture behind MEREBO but the further explanation of it is out of the scope of this thesis. The interested reader is referred to [1].

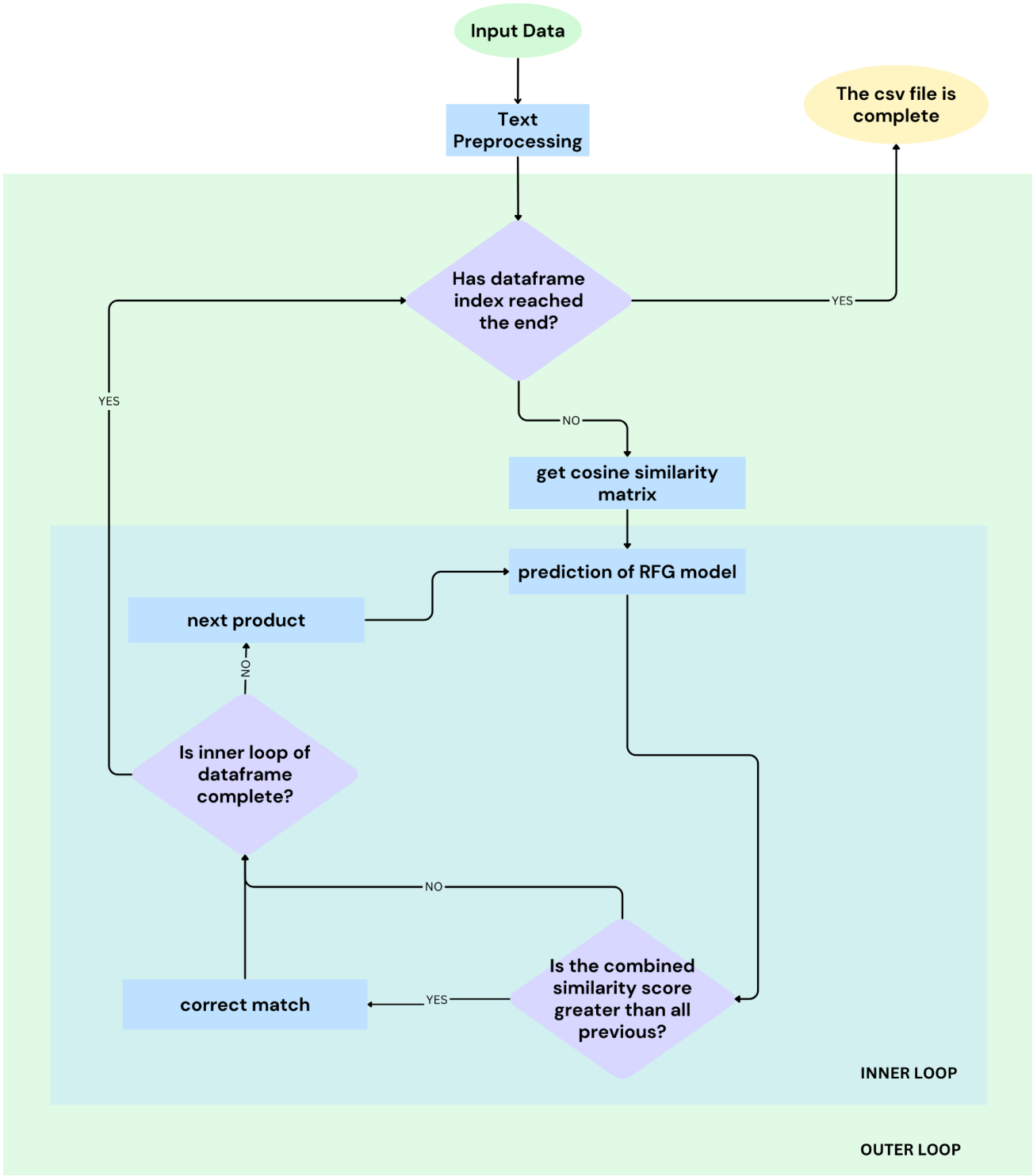


Figure 2: The matching algorithm of MEREBO

2.1 Specific categories enhancement through TF-IDF

Our initial strategy to enhance accuracy was to focus on specific product categories, particularly the groceries. During our analysis of the contents within this group of products, we identified a prominent recurring pattern. It became evident that the first word of every product description consistently represented the item being offered. As an example, a product description in this category could be the following:

‘Apples goldensmith 200gr’

All variations of apples are initiated with the word ‘apples’. As a result, we opted to exclude this category from the existing model and adopted a new approach. In this new method, we assigned greater weight to the first word of each document which led to a more efficient algorithm that bypassed the need to traverse the trained model. Through a series of experiments involving different weight values, it became apparent that giving additional weight to the second word was also crucial, as it typically denoted the brand of the product. This enhancement was particularly valuable in achieving higher accuracy, especially in cases where distinguishing between different types of apples was essential.

While the method we developed was tailored to certain categories and offered an improvement in addressing the execution time constraints of the algorithm, it was unable to surpass the performance of the pre-existing solution provided by MEREBO. In fact, our method yielded an F1 Score of 0.825 for the groceries category, but MEREBO achieved a slightly higher score of 0.834 within the same category.

The approach we experimented with had the potential for further improvement through exhaustive text manipulation, identification and removal or restructuring of problematic terms, but we opted not to pursue it any further and instead shifted our focus to the next method. It is noteworthy that this approach achieved a slightly lower score compared to its predecessor; however, it demonstrated a significant advantage in terms of processing time as it did not rely on MEREBO’s trained model. In summary, if the primary goal was to enhance the time efficiency of the system, this approach could warrant further exploration and experimentation.

2.2 Enhancement via image comparison

The fundamental process of image comparison, central to the context of this thesis, involves the systematic assessment and analysis of visual content to determine the degree of similarity or dissimilarity between product images. This process typically begins with the acquisition of image data. Then, various image processing techniques are applied to standardize and preprocess the images such as resizing, normalization and feature extraction. Feature vectors that represent essential characteristics of the images are then generated. These features are subsequently utilized to perform comparison between images, employing mathematical or statistical methods, similarity methods and machine learning algorithms. The outcome of this process is a quantitative measure of the degree of similarity.

2.2.1 VGG19

VGG19 [4], an integral component of the deep learning landscape, stands as a notable convolutional neural network architecture. It stands for **V**isual **G**eometry **G**roup, an academic group focused on computer vision at Oxford University. It is a *convolutional neural network* that consists of 16 convolutional layers and 3 fully connected layers, for a total of 19 layers. VGG19 employs max-pooling layers after several convolutional blocks to downsample the features and reduce computational complexity. These blocks contain multiple convolutional layers followed by a pooling layer and extract useful features from the input images. The network's 3 deeper layers that are referred to as fully connected layers, perform a high level decision making based on the features extracted by the previous layers and are especially responsible for classification tasks.

It is pre-trained on a large dataset of images, with the most popular of them being *ImageNet* consisting of 14 million images, to learn how to extract useful image features. With its 19 layers, VGG19 has approximately 143 million parameters which makes it computationally expensive to train. However, this depth and number of parameters contribute to its ability to learn features and identify patterns in the data, thus establishing it as a highly regarded tool in the machine learning community [5].

2.2.2 VGG19 Architecture

We can approach the architecture of this complex network as follows:

Input Layer
The network is trained and fine tuned to process images in standard shape. It has to be of a size of (224x224) and in RGB color format so the shape of the input image must be (224,224,3). This can be guaranteed in any case of image thanks to the preprocessing function that we discuss in the following chapter.

Convolutional Layers
VGG19 consists mainly of convolutional layers, where small 3x3 convolutional filters are used with a stride of 1. These layers are stacked on top of each other and each is followed by a rectified linear unit (ReLU) activation function that returns the input if positive and 0 if negative.

Max Pooling Layers
After a few convolutional layers, max pooling layers with 2x2 filters and a stride of 2 are used to downsample the spatial dimensions, reducing the computational load and controlling overfitting.

Fully Connected Layers
Towards the end of the network, fully connected (or dense) layers are used for high-level reasoning. The VGG19 architecture typically includes three fully connected layers which connect every neuron in one layer to every neuron in the next layer.

Softmax Layer
The final layer of the network (the last of the dense layers) is a softmax layer that produces probabilities for different classes. This layer is particularly useful in classification tasks to make the final predictions.

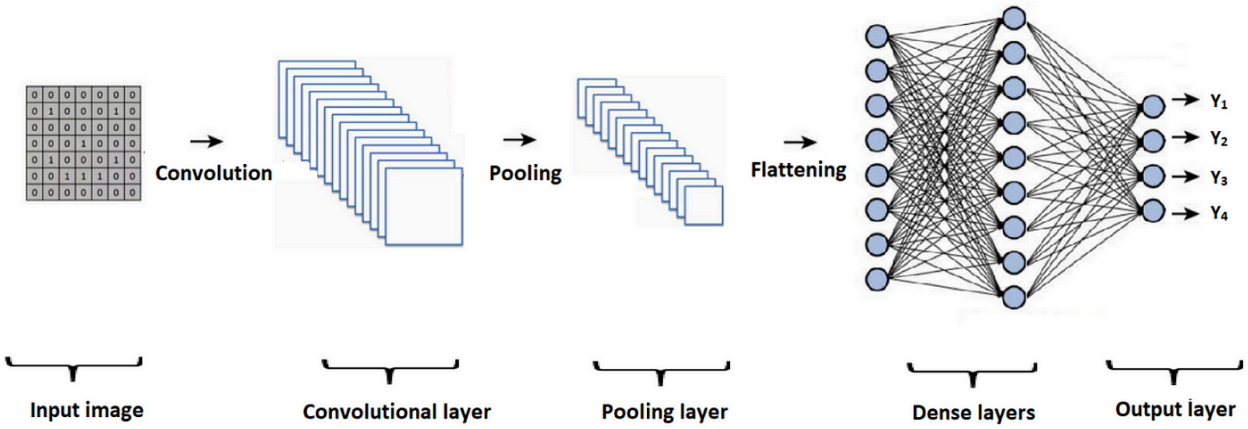


Figure 3: VGG19 architecture design

2.2.3 Implementation

The first step to use VGG19 is to load the base model provided by the keras package. It is important to note that the aim of this endeavor is feature extraction to perform some sort of similarity comparison. Because of that, we exclude the *Softmax* layer from the network. The reason for this is that Convolutional Neural Networks are typically designed to perform image classification. Their top layer, the Softmax, is responsible for this classification. So it is best practice to exclude it from our model since we only care about feature extraction. Doing so, we are significantly reducing the already intense computational load without compromising the accuracy of the network.

With the VGG19 CNN ready and loaded we then need to load and preprocess the images. Image preprocessing is a crucial step in computer vision and image analysis, as it involves a series of operations that enhance the quality of the image, reduce noise and make it more suitable for subsequent analysis. Some notable operations are resizing, normalization and edge detection. In the case of VGG19, keras also provides a *preprocess_input* function that handles all these tasks for us, thus satisfying the requirements of the input image shape.

Being known for its computational complexity, we performed image comparison cautiously. MEREBO already uses a random forest classifier on top of TF-IDF to determine similarity between the product descriptions. Every pair is represented by a

quantitative metric of similarity and by introducing a threshold we can determine which pairs are indeed alike. In our endeavor, we slightly lowered the threshold point so that more pairs can be considered for comparison. For each pair that has similarity greater than the threshold we perform image comparison and further adjust their similarity score based on the result of VGG19.

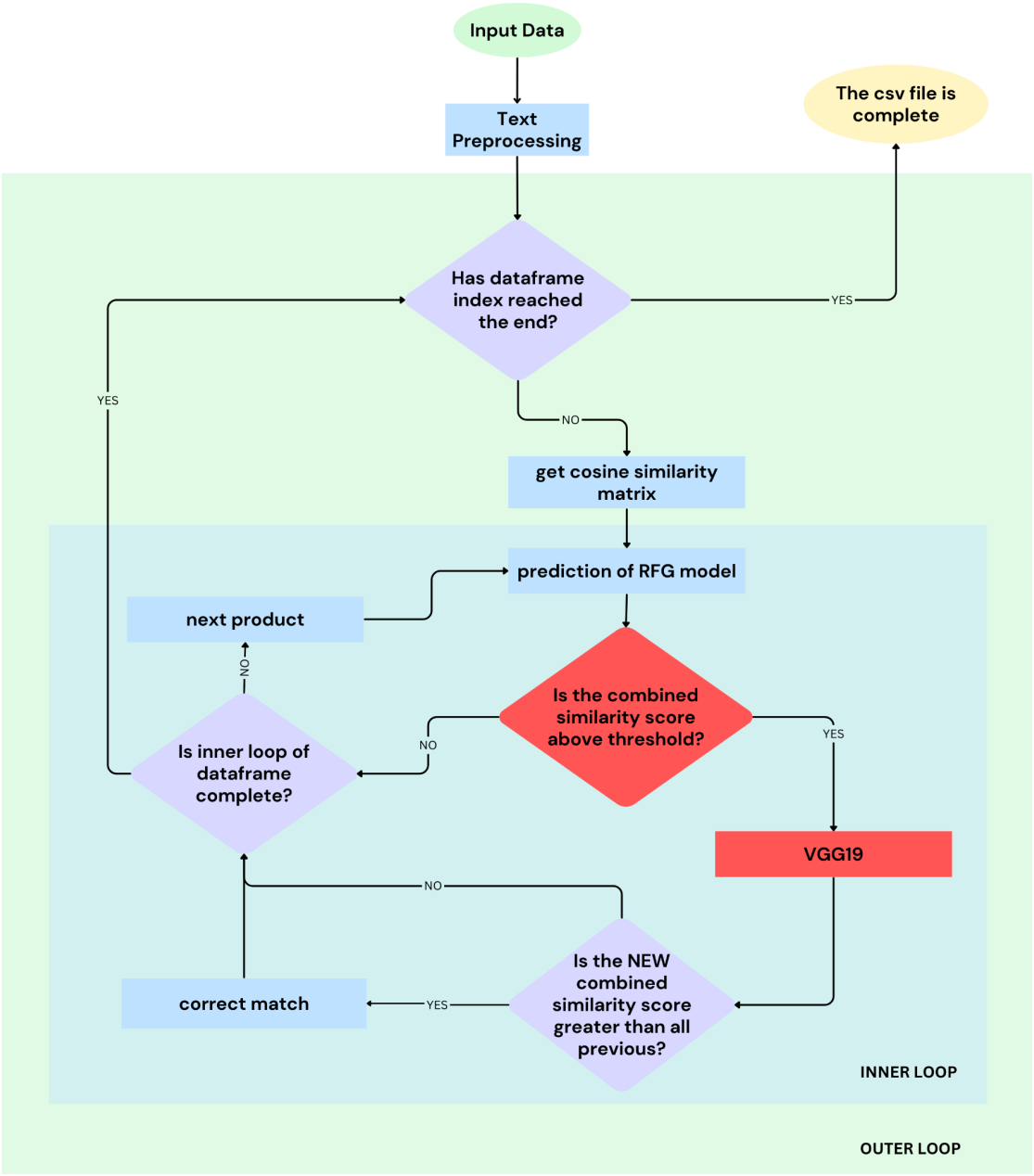
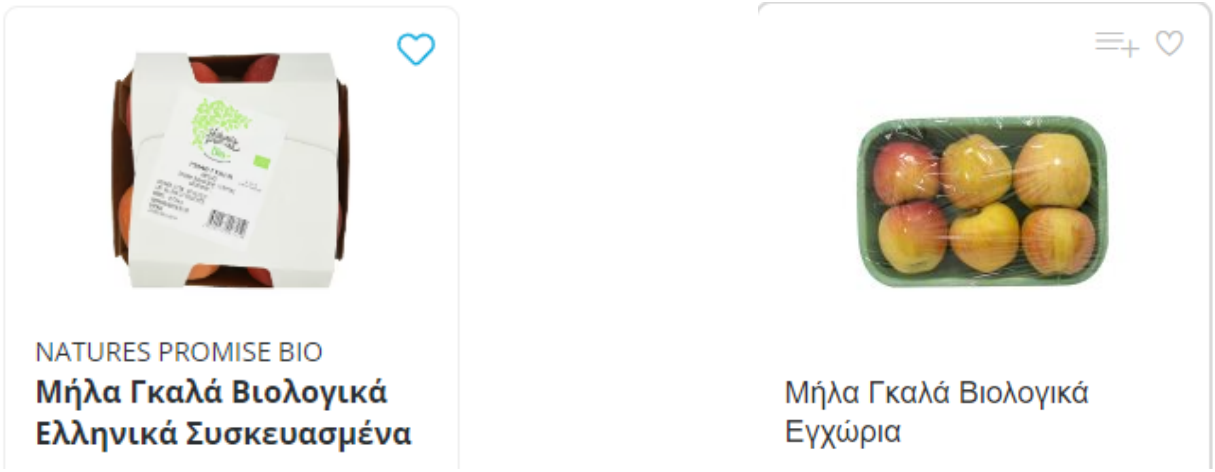


Figure 4: In red color is the addition of image comparison

2.2.4 Adjusting VGG19 to our needs

Many products are practically the same but have different images to represent them in their respective e-commerce pages. The best example for this problem is the vegetables and meat & dairy categories. A product in these sections can essentially be the same but have a very different image.



Figures 5 & 6: Identical products that are represented by vastly different images

On the other hand, a different challenge arises when certain products exhibit similar images despite representing different items. This scenario is frequently encountered within the groceries category because a group of products such as apples may look similar but they actually denote different varieties. As a result, in such cases it is important not to create matches between these visually similar but fundamentally different products to ensure a pleasant customer experience.

Figures 7 & 8: Similar images from different products

While VGG19 is powerful and trusty, these problems can undermine the accuracy of the network. To tackle them, we need to define 3 ranges on the VGG19 results spectrum: ‘pair’, ‘no-pair’ and ‘not sure’. We composed a test dataset consisting of 150 products. Each of these is either matched with another one or is not supposed to be. We then ran the algorithm using only VGG19 and found out the median of false matches’ scores and the median of correct ones. Although we use the cosine similarity metric and theoretically we could get a similarity score in the range of [-1,0], 0 suggests ‘no similarity’ and -1 complete dissimilarity. So in the following image we only show the positive range of the expected values, implying that lower than 0 values fall under the same category although we do not expect to encounter any.

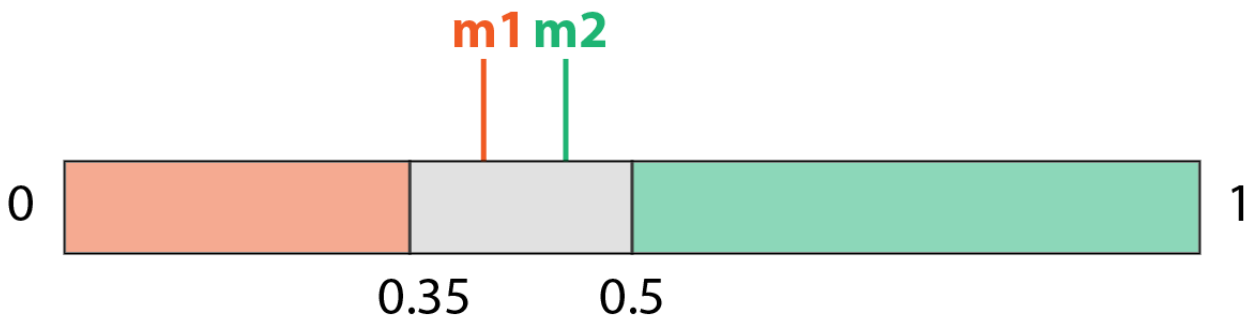


Figure 9: Similarity ranges of VGG19

The median image similarity score for correctly matched pairs is *m2=0.45*, whereas for false pairs, the median similarity score is *m1=0.4*. The range between ‘pair’ and ‘no-pair’ median is considered a buffer or a ‘not sure’ zone where if this is the case we do not alter the overall similarity of the pair. By expanding the threshold for the ‘not sure’ zone by *0.05* we adopted a more forgiving approach in our algorithm. This adjustment aimed to address the two issues we discussed at the beginning of Section 2.2.4.

In instances where the output of VGG19 falls below the ‘no-pair’ median, the pair is deemed incorrect, thereby prompting a reduction in their similarity score. Lastly, when the result surpasses the ‘pair’ median threshold, the pair is valued as correct and its similarity score is increased.

2.3 Structural Similarity Index (SSIM)

Structural Similarity Index or SSIM is a widely adopted metric utilized in image processing and computer vision for the purpose of quantifying the likeness between two images [7]. It assesses the structural resemblance by considering not only the pixel-wise differences but also the luminance, contrast and structural information in the images.

Luminance
Luminance comparison measures the similarity in terms of brightness. It uses the mean of pixel values and their variance to represent the luminance of the image.

Contrast
Contrast comparison considers the contrast which is related to the standard deviation of the pixel values. It Examines the quality of contrast in the images being compared.

Structure
It involves the comparison of the structural information present in the images. It uses a windowed approach, calculating the similarity of the local patterns in the images.

The computation process of the SSIM score between two images involves the following 4 steps:

- Dividing the images into smaller patches or windows
- Computing the *Luminance*, *Contrast* and *Structure* components for each patch
- These computed components are averaged over all patches.
- Lastly, the three components are combined to generate an overall index that represents the similarity score. This score ranges from -1 to 1, where 1 indicates total similarity and -1 complete dissimilarity.

2.3.1 Implementation

Utilizing SSIM is straightforward in comparison to VGG19. In order to compare two images there is a series of operations that need to be done, the preprocessing of the images. Unlike the VGG19 which provided its own preprocessing function, in SSIM we have to take care of it by ourselves. For the images to be able to be compared we need to make sure our images meet some prerequisites. All images need to be in the same color format, the same shape and size so we are performing color space conversion to make them all RGB, we change their shape to be the same and finally we resize. After all these operations, the images are ready to be compared using the *structural_similarity* [8] function that is provided to us by the scikit-image collection of algorithms. The result is handled the same way as we handled the VGG19 but with different ranges, as shown below.

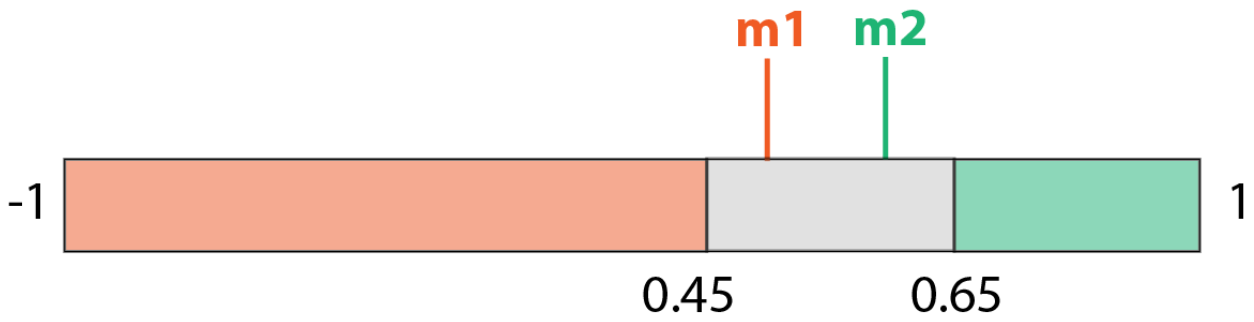


Figure 10: Similarity ranges of SSIM

2.4 Theoretical comparison of VGG19 and SSIM

VGG19 excels in capturing fine-grained features and exhibits versatility in various image analysis tasks. However, it is computationally heavy, which can pose performance challenges, particularly on lower-spec hardware systems. In contrast, SSIM offers a perceptual assessment aligning with human perception of image quality. Both methods are suitable for image comparison with one obvious difference between them being their computational demands. The selection of the right tool depends on the available resources and the goals of the task at hand.

2.5 Process of evaluation

To evaluate the performance of our models, we employ a set of metrics that goes beyond simple accuracy, providing a more comprehensive understanding of its effectiveness. Among these metrics, Precision, Recall and F1 Score play a pivotal role.

Precision quantifies the proportion of true positive instances out of all instances classified as positive

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

Recall, on the other hand, measures the ability of the model to identify all positive instances correctly.

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

The *F1 Score*, which is the harmonic mean of Precision and Recall, offers a balanced assessment, particularly useful when dealing with imbalanced datasets or when there is a need to strike a trade-off between false positives and false negatives.

$$F1\ Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

The final metric we use is the *Accuracy* score. It measures the number of correct predictions made by the model in relation to the total number of predictions made.

$$Accuracy = \frac{True\ Positives + True\ Negatives}{Total\ Predictions}$$

2.6 Results

	MEREBO	MEREBO 2	
	-	VGG19	SSIM
Precision	*0.78*	*0.86*	*0.84*
Recall	*0.88*	*0.84*	*0.85*
F1 Score	*0.83*	***0.85***	***0.84***
Accuracy	*0.76*	***0.8***	***0.79***
Execution Time	*180*	*232*	*192*

Table 1: Accuracy Results

To extract the results, we used a test dataset similar to the one we described earlier, consisting of 150 products from various categories that may or may not have a correct match among the rest of the products.

According to the results presented in the table, the addition of image comparison certainly improved the already high F1 score of MEREBO. Specificly, we observed a substantial 2.4% increase in F1 score when leveraging the powerful VGG19 and a 1.2% when utilizing the Structural Similarity Index (SSIM). However, the most significant and anticipated advancement lies in the Accuracy metric, which gained a remarkable 5.2 % enhancement with the help of VGG19 and a 4% utilizing SSIM. This surge in accuracy can be attributed to the ability of image comparison to identify previously False Positive (FP) matches as True Negatives (TN), whereas MEREBO predominantly excelled in recognizing identical or similar matches through text-based comparisons. The Accuracy metric, which takes into account the number of TN instances, was the metric most profoundly impacted by this enhancement, underlining the effectiveness of image comparison in rectifying False Positives and thereby augmenting the overall model accuracy.

The last thing to notice is the time results. As we see on the table, the introduction of the SSIM algorithm resulted in an approximate addition of 12 hours to the total processing time while the addition of VGG19 resulted in a staggering 52 hour increment. While this added execution time might not seem concerning in comparison to the already high estimated time of completion of MEREBO's infrastructure, it is crucial to contextualize

it. Someone could have expected at least a 100% increase in processing duration but this is definitely not the case. Specifically, the image comparison algorithms are only invoked in scenarios where a promising product pair is identified. Consequently, in cases where a word, such as ‘bread’, yields around 200 products containing the term in their description, the SSIM or VGG19 algorithms are exclusively executed for these 200 prospective pairs.

The main factor contributing to this extended estimated time is discussed in the forthcoming chapter.

3. Execution Time

3.1 Execution time of MEREBO

For an application to be useful and viable in real world usage, there are some issues to be addressed. This application is an online grocery basket that provides products from 3 different stores. It is possible that the prices and the products are being changed constantly so it is of paramount importance to update our database frequently. But when an item is added or removed we need to try to find its match from other markets if there is any. Since we don't know which product was changed we are forced to perform the whole process from the start and match every item. The problem is that scraping 3 stores takes about 6 hours and the matching algorithm takes approximately 180 hours with the addition of image comparison. Therefore, it is obvious we are far away from the 24-hour timeframe.

3.2 Multiprocessing in web scraping

Given the non-cooperative nature of the system, real-time updates are not to be considered. However a viable alternative lies in concurrent scraping of multiple online stores. To address this task, we used the multiprocessing package [9] provided by Python. This strategy involves the simultaneous scraping of data from all targeted websites, with the system effectively awaiting the completion of all these concurrent scrapping tasks. The critical path within this phase is determined by the longest duration required for scraping the most substantial webpage. This parallelized approach optimizes efficiency by capitalizing on available computational resources and effectively reducing by 66% the total time needed to update the database [10].

The implementation of parallel processing in the handling of diverse web pages constitutes an invaluable mechanism for facilitating vertical scalability in future scenarios. This strategy provides the capability to incorporate additional web stores into our system, all while not increasing the execution time. Such an approach presents a wider array of options to the users leading to an augmented overall quality of customer experience.

3.2.1 Headless browsers

Observing the web scraping process showed us that there was a time implication waiting for all the images of the site to load up that sometimes led to failure due to missed waiting time. Another issue was the computational power needed to handle 3 automated browsers in parallel processes, although with only 3 of them it is manageable. To overcome these issues and to further improve the performance of our system, we transitioned to *headless browsers*.

Headless browsers operate without a graphical user interface (GUI), which results in dramatically lower memory and CPU usage compared to the standard browser. This allows for more efficient web scraping without consuming excessive hardware resources, which further extends the vertical scalability we previously mentioned. Another benefit for using headless browsers is the faster execution of *Selenium* scripts that aim to collect data.

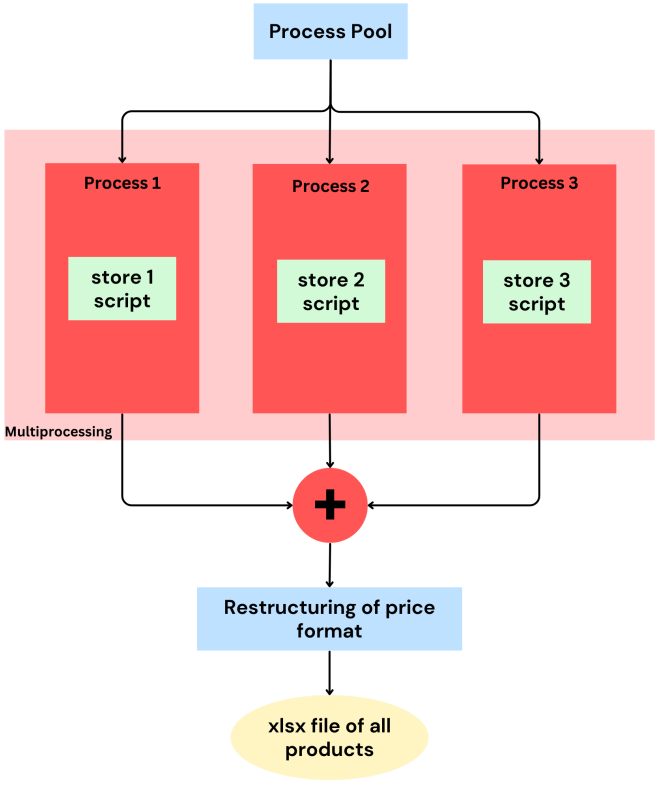


Figure 11: The multiprocessing design of web scraping algorithm

3.3 Sparse matrix handling

A sparse matrix is a matrix in which the majority of the elements are zero. In our case, we use a vector matrix that is essentially a sparse matrix to hold the similarity scores of each combination of a group of products. Our system is designed in a way that a repopulation of such a matrix is crucial for the Random Forest Classifier that came with MEREBO's infrastructure. We measured this process to be extremely time consuming and we introduced some simple and yet effective changes.

First, we transformed the dense matrix into a list of lists. This format that is also referred to as LIL, is particularly useful for constructing and modifying sparse matrices in Python as it allows for efficient row-based modifications. Then we used broadcasting to copy the indexed row to all rows in the LIL matrix, instead of iterating through all of them. Broadcasting refers to the implicit operations performed on arrays of different shapes by NumPy, a very popular library in Python for numerical operations. This change provided an improvement of 26.6 % as it helped to reduce the total time from 180 down to 132 hours. So the *Table 1* in regards to the Execution Time is now the following:

	MEREBO	**MEREBO2**	
	-	**SSIM**	**VGG19**
iteration	*180*	*192*	*232*
broadcasting	*132*	*144*	*177*

Table 2: Sparse matrix handling improvement

3.4 K-means clustering

K-means clustering is an unsupervised machine learning technique designed to partition a dataset into groups or clusters, based on the inherent similarity of data points. The main focus of K-means is to assign data points to clusters in such a way that data points within the same cluster are more similar to each other than to those in other clusters. This clustering algorithm operates by iteratively optimizing the partitioning of data into K clusters, where K represents the pre-defined number of clusters. The process begins with an initial selection of K cluster centers or centroids, either randomly or through a manual and strategic method. Each data point is then assigned to the cluster whose centroid is closest to it based on Euclidean distance. Following this assignment, the centroids are reevaluated as the mean of the data points assigned to each cluster.

$$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2 = \arg\min_{\mathbf{S}} \sum_{i=1}^{k} |S_i| \operatorname{Var} S_i$$

This iterative assignment and centroid recalculation process continues until the maximum number of prespecified iterations is reached. The final cluster assignments represent the outcome of the K-means algorithm.

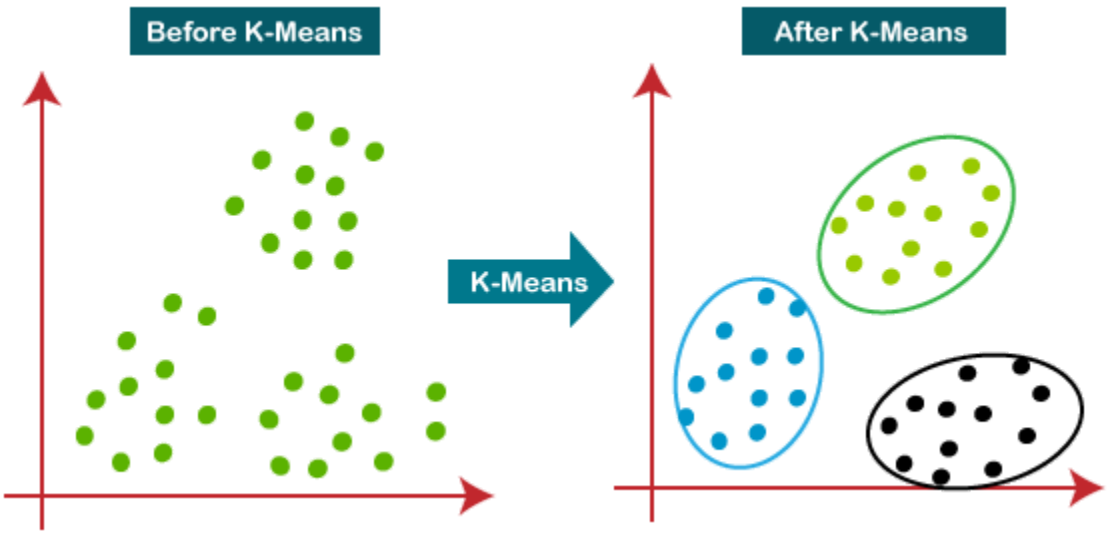


Figure 12: Visualization of clustering with K=3

3.4.1 Finding the perfect K

As previously mentioned, the parameter *K* in clustering represents the number of clusters to be created and it must be predetermined prior to the initialization of the clustering algorithm. Although there are methods that enable us to find the value of K dynamically based on the data, we chose the static approach. This decision was guided by the nature of our data input, characterized by a consistent number of products per category and a regularity in the terminology used within the product descriptions, given our focus is limited on grocery stores.

To find the optimal *K* we implemented a technique known as *the Elbow Method*. This method involves evaluating the variation in clustering performance across a range of *K* values and identifying a point at which the reduction in variation reaches a plateau, resembling an *elbow* in the graph. The *K* value at this point is considered optimal and serves as a constant value for every future clustering operation within the scope of this thesis.

In general, there can be one or more suitable values for the number of clusters. These values can be identified using the Elbow method but the final choice depends on the knowledge upon the data the engineer is working on and his experience.

```
For a range of K values:
        Create a kmeans model with K clusters
        Fit the data in that model
        Store the inertia of the fitted data

Plot the results and locate the 'elbow'
```

Where *inertia* is the Within-Cluster Sum of Squares. It represents the sum of the squared distances between each point in the cluster and the centroid.

$$\sum_{i=1}^{N}(x_i - C_k)^2$$

In our case, we estimated that the number of clusters should be in the range of 50 up to 100. There would be no reason to create less than 50 clusters because the categories of products on the stores are approximately 20 and we know that every category can be split to at least 3 or 4 sub groups. For example, we have 6000 items in the drinks&snacks group and for sure we can create a whole cluster that is centered around ‘water’ and another one around ‘cola’. So starting with the range [60,80], we get the following graph.

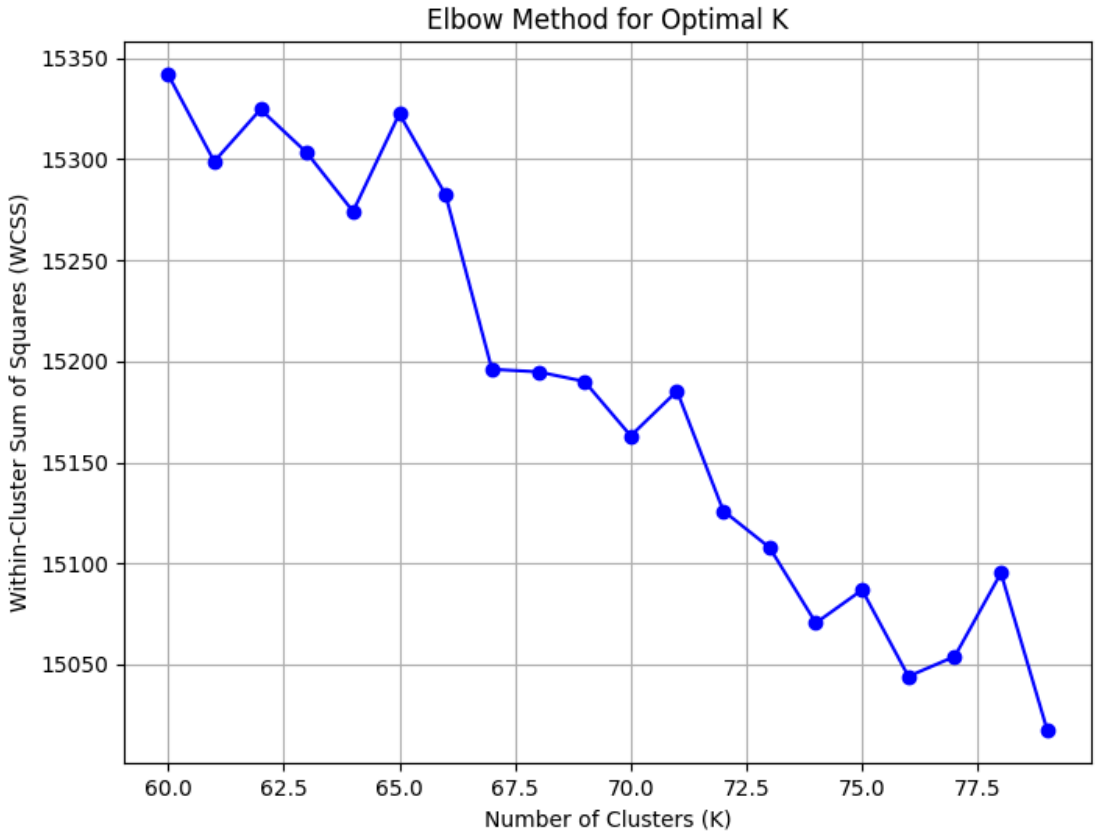


Figure 13: Elbow method in range [60,80]

We can see that we do not get a linear plot. This waviness in the plot arises when the rate of decrease in the distortion measure is not linear. This can occur due to the complexity of input data, where clusters are not distinctly separable, leading to overlaps and fluctuations. Additionally, the presence of noise and outliers can contribute to a non-linear decrease in distortion resulting in a wavy appearance. The nature of this plot introduces a challenge in deciding the perfect value of *K* but in this case, we can identify

a promising one. It is obvious that *K=67* can be an elbow point because it comes after a big decrease in distortion and is followed by a plateau.

To verify this is indeed a valid choice, we examined the plot in the range [20,50] that we considered not promising in order to compare them.

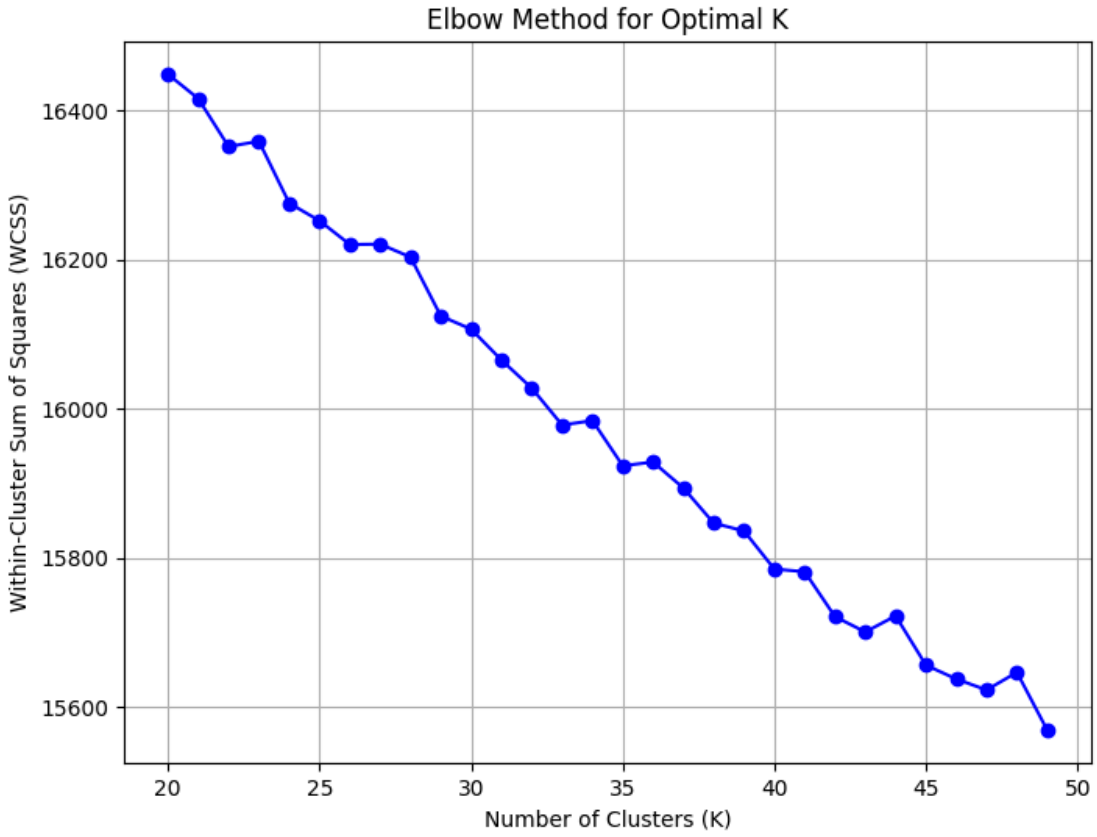


Figure 14: Elbow method in range [20,50]

In this plot we observe relatively less fluctuations in the rate of distortion but with no obvious elbow point. However, the absence of a distinctive elbow does not imply that selecting a K value within this range will form inaccurate clusters. Instead, the choice of the optimal number of clusters remains a subjective decision left to the experience of the data analyst. In this particular instance, we decided to settle with the initial value of *K=67* as it fell into our predicted range and is considered an elbow point, thus promising to deliver the desired results.

3.4.2 Utilizing K-means clustering

In our extensive database, housing approximately 25000 products, MEREBO's initial design involved performing exhaustive comparisons among all of these products. In this approach, a repetitive creation of a matrix, comprising 25000 rows was inevitable. This method, while manageable, presented computational challenges. The challenge escalated upon the implementation of image comparison algorithms, which was an anticipated consequence. However, the central bottleneck primarily stemmed from the frequent construction of this sizable matrix that our system requires.

To mitigate this computational burden and streamline the process, we employed K-means clustering. This technique enabled us to partition the products into subsets based on shared characteristics, resulting in more manageable cluster sizes. For each product, we only repopulate a matrix corresponding to the total number of possible matches within its respective cluster. For instance, one cluster might encompass a variety of water-related products, such as sparkling water, mineral water, water with lemon and so forth. From this point, the main algorithm of text and image comparison handles the rest.

The remarkable contribution of K-means clustering becomes evident in the reduction of processing time which is better demonstrated in *Table 3*. This table demonstrates the comparison of Clustering with an optimal K and a suboptimal one. We explore the values of K=67 and K=27 and observe the results in Execution Time. Notice that these improvements are on top of the enhancements yielded by the previous chapter and displayed in *Table 2*.

	w/o image	**SSIM**	**VGG19**
Pre-clustering	*132*	*144*	*177*
Clustering K =27	*8 (-93.33%)*	*14 (-90.27%)*	*30 (-83%)*
Clustering K =67	*6 (-95.45%)*	*9 (-93.75%)*	*26 (-85.31%)*

Table 3: clustering with K=27 & K = 67

Before any further discussion, it is important to clarify that the time values presented in the tables are approximations. These time calculations derive from the total sum of the average time taken for completion per cluster. The actual time required for the entire

process can vary based on the number of image comparisons executed within each cluster.

Table 3 highlights the critical significance of selecting an optimal K value in the clustering process. It is evident that a larger K value leads to a higher number of clusters, resulting in smaller, more partitioned dense matrices - that impose the biggest challenge - and generally expediting the overall process. However, an excessively large K value might significantly compromise the accuracy of the algorithm, especially when it far exceeds an optimal point. In the extreme scenario where K equals the total number of products, the process becomes exceptionally time-efficient. However, this situation renders each cluster to contain only a single item, essentially impeding the matching process because there would be no other products within each cluster for comparison.

On the contrary, a lower K value demonstrates negligible gains in comparison to the benefits derived from an optimal value, particularly noted in the VGG19 method. Choosing smaller values, the time required approaches the efficiency before clustering. When K equals 1, effectively, there are no clusters and each product is compared directly with every other product in the dataset.

Therefore, the selection of the K value in clustering significantly impacts both the efficiency and accuracy of the process. Striking a balance between computational efficiency and clustering quality is crucial and choosing the appropriate K value is pivotal in achieving this balance.

3.4.3 Further improvement of clustering via logistic regression

While the results of clustering did indeed help us overcome the time limitations, it was clear that clusters were not always able to cluster the products effectively, compromising the accuracy of the model because when there should be a match, it was falsely appointed to a different cluster resulting in a miss. To tackle this issue, we trained a model to categorize the products based on their description.

Initially, we made a temporary modification to the scrapping algorithm. This change involved the categorization or *classification* of products based on category names found on each store's webpage. This resulted in a 25000 training dataset that consisted of 2

columns: the description of the product and its category. Based on this new dataset, we trained a *Logistic Regression* model to classify products.

Logistic Regression excels in binary and multi-class classification tasks, offering a robust and interpretable approach to predicting outcomes. This method models the relationship between the input features and the probability of an instance belonging to a particular class.

$$P(y=1) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n)}}$$

In our multi-class identification task we employ the One-vs-Rest (OvR) strategy. Based on this method the model calculates the probability of the input being each different class and chooses as the final prediction the class with the biggest probability.

The logistic function ensures that the predicted probabilities fall within the [0,1] range which is essential for class probability estimates. On top of these factors, its simplicity warrants a reliable choice for training classification models.

To measure the accuracy of the classification process, we used the *train_test_split* function provided by sklearn. This function automatically divided our dataset into two subsets: the training dataset, encompassing 80% of the total items, and the remaining 20% allocated to the test or validation dataset. By utilizing the validation dataset and the *accuracy_score* also provided by the sklearn library, we measured the accuracy of our model to be *0.94* or *94%*. This level of accuracy is deemed substantial, suggesting the model's capacity to yield consistent and reliable outcomes, even in real-world applications.

This classification model was then utilized to predict and assign categories to each product, subsequently adding these predicted categories into the product descriptions. This augmented description now serves as a guideline for the clustering algorithm, enhancing its ability to cluster the products into more accurate and meaningful clusters. Remarkably, these improvements were achieved without compromising the time efficiency we had gained through the addition of clustering techniques, therefore the mistakes of plain clustering were rectified.

For example, the training dataset for the classification of products into categories looks like this:

	A	B
1	**TEXT**	**LABEL**
2	GORDON'S Τζιν 700ml	SNACKS - KABA
3	JOHNNIE WALKER Ουίσκι Red Label 700ml	SNACKS - KABA
4	SMIRNOFF Βότκα 700ml	SNACKS - KABA
5	SERKOVA Βότκα 700ml	SNACKS - KABA
6	TANQUERAY Τζιν 700ml	SNACKS - KABA
7	GORDON'S Space 275ml	SNACKS - KABA
8	ABSOLUT Βότκα 700ml	SNACKS - KABA
9	FAMOUS GROUSE Ουίσκι 700ml	SNACKS - KABA
10	APEROL Απεριτίφ 700ml	SNACKS - KABA
11	BACARDI Breezer Καρπούζι 275ml	SNACKS - KABA
12	GRANT'S Ουίσκι 700ml	SNACKS - KABA
13	BACARDI Ρούμι 700ml	SNACKS - KABA
14	HAIG Ουίσκι Gold Label 700ml	SNACKS - KABA
15	DEWAR'S Ουίσκι White Label 700ml	SNACKS - KABA
16	SMIRNOFF Ice 275ml	SNACKS - KABA
17	ΠΛΩΜΑΡΙ Ούζο 700ml	SNACKS - KABA

Figure 15: Snapshot of train dataset for our Logistic Regression Model

Consequently, the identification of a BACARDI product within the recently scrapped items leads to its classification within the SNACKS-KABA category. This predicted category is then concatenated temporarily to the product's description, transforming it into:

BACARDI Ρούμι 700ml SNACKS-KABA

The outcome of this addition is that a newly discovered BACARDI item is most likely going to be attached to the SNACKS-KABA class or category. As a result, when the k-means algorithm is activated, the term SNACKS-KABA is going to act as a guideline to more accurately distinguish similar category items and then try to find a comparable BACARDI entity within this refined subset.

In other words, this strategy aims to group products under a shared term, enabling k-means clustering to identify and form better and more accurate clusters guided by this uniform descriptor.

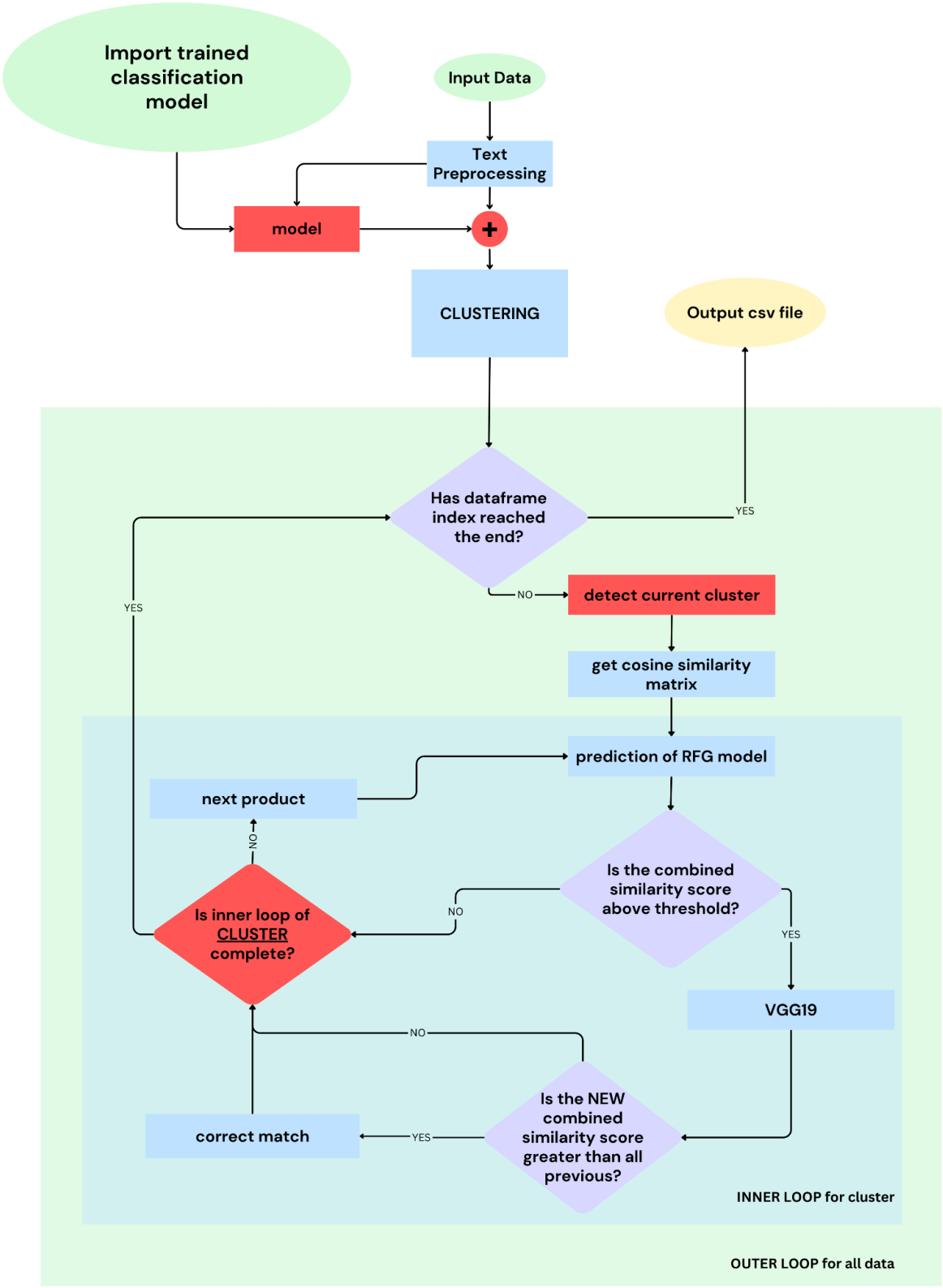


Figure 16: Addition of clustering in the matching algorithm

3.4.4 Why not use the categories on the stores instead?

During the scraping phase, we could hardcode this functionality instead of training a logistic regression model. Using some simple tagging between the product's category in each shop and a list of predetermined categories we could accomplish an even more accurate clustering.

While this approach may seem easier to implement and safer because we would not have to deal with a trained model's accuracy, it quickly became obvious that it was way too complex and tedious of a task to actually make it work that way.

The problem is that stores do not always display their products in the same way and their menu is almost never organized the same way as the others. Our first store might have the fresh meat and fresh fish under the same big category and then split them into sub categories, while the other shop could have them in two separate grand categories. Another example is the chips. One shop has them under the snacks category and the other under a grand category that also contains dried fruit and drinks.

The following images display the categorization of fish and meat in two separate shops. In one image (left), both fish and meat categories are grouped together, while in the other, these categories are distinctly separated into individual groupings.

Figures 17 & 18: (Left) the Fresh Meat & Fish category in AB.gr is one big category while (Right) the same categories are split into distinct groups in Sklavenitis.gr

Another possible issue is that a category name on any grocery store could change at any time. Even the slightest change in the category name could lead to serious trouble causing a chain reaction of in accuracy throughout the entire algorithm.

Essentially, the first time we would run the algorithm we would possibly get the most accurate results. But this is definitely not guaranteed to continue that way given the dynamic nature of the store pages and the ambiguity in their categories. In order for it to face the ambiguity issue, we would have to write exhaustive code to check every sub category of all three online stores and write a tagging script that compliments each one. But even if we did that, there is no way to work around the possibility of a category being changed by the supermarket's manager.

3.4.5 The trade-off of clustering

While clustering aims to group products of the same category or even more specific terms together, there is a potential issue where unique products may inadvertently be clustered with other unique items due to similarities in their volume or quantity descriptions that happen to be the same. To further understand the problem we present an example.

Consider a scenario where a cluster comprises 100 distinct products that, to human observers, exhibit marked dissimilarity and do not belong to any other cluster. In this collection, none of the products contains any common terms. However, two of these items share a common numerical value within their descriptions, denoting quantities, such as kilograms or liters. This shared numerical value serves as the sole commonality between these products yielding an unexpectedly high similarity score, surpassing the predefined threshold. In properly formed clusters this problem is mitigated by the fact that the rest of the terms appear in the description of other products within the same cluster so the *TF-IDF* combination of the document is reduced, thus the similarity score is reduced.

This issue could possibly result in false pair identification. A possible solution to this challenge involves increasing the matching threshold, which would prevent the matching of such different products that happen to be in the same cluster. However, due to the non-supervised nature of clustering algorithms, it becomes challenging to determine precisely which clusters are affected by this issue. Consequently, raising the threshold universally could lead to the exclusion of a significant number of accurate matches that would otherwise have been identified.

Considering that unique items constitute a relatively small proportion of the dataset, we have consciously accepted the trade-off of allowing for this risk in order to preserve the accuracy of clustering in all other cases.

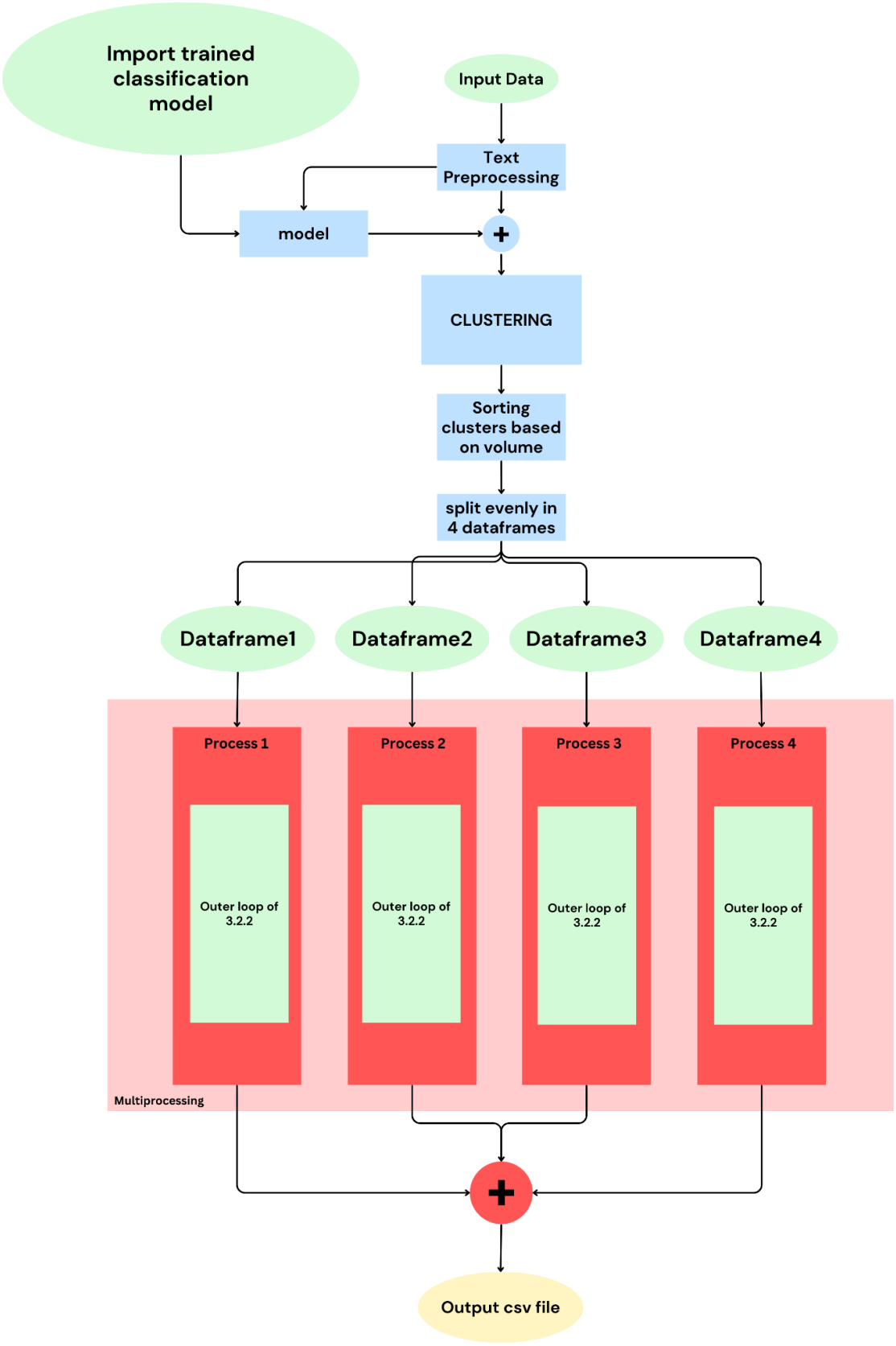
3.5 Balancing time efficiency with accuracy

Now that we have all the information regarding time and accuracy efficiency improvement methods we can make an informed decision about the combination of them to best suit our needs.

3.5.1 Multiprocessing approach with VGG19

We are presented with a dilemma, choose the SSIM algorithm and get slightly less accuracy but stay in the 24 hour timeframe or choose the VGG19 and be on the verge of the time limitation. Here comes the third approach which leverages the total computational power available by our hardware to take full advantage of the powerful and robust solution VGG19 offers. We split our clusters into 4 groups while making sure this assignment is equal by assigning the 4 biggest clusters as the first cluster for each group respectively and filling with the rest. Then we assigned each group to a different process and created a multiprocessing pool to handle this heavy task. By doing so, we were able to theoretically achieve the best result but the load on the system's hardware was so big that the temperatures on the CPU did not allow for thorough examination of this solution. In fact we were not able to run the script for more than 2 minutes so we only ran it for the first 2 products on each group or parallel process to verify its effectiveness.

To explore alternatives for mitigating the CPU load, we experimented with a reduced number of parallel processes, first employing three and subsequently reducing them to two. The latter, while alleviating the CPU burden, still did not achieve the timeframe demands as it was close to 13 hours. We concluded that multiprocessing in the matching phase under the VGG19 method is not a worthy choice as it leads to heavier hardware load, more processing time and only yields minor improvements in comparison to the SSIM method. Under different hardware or nature of a task, like identifying products as part of a bigger picture, VGG19 would possibly suggest a better method.


Import trained classification model
Input Data
Text Preprocessing
model
CLUSTERING
Sorting clusters based on volume
split evenly in 4 dataframes
Dataframe1
Dataframe2
Dataframe3
Dataframe4
Process 1
Process 2
Process 3
Process 4
Outer loop of 3.2.2
Outer loop of 3.2.2
Outer loop of 3.2.2
Outer loop of 3.2.2
Multiprocessing
Output csv file

3.5.2 Multiprocessing approach with SSIM

Another way we could furthermore enhance the speed of our algorithm involves the integration of multiprocessing in the SSIM version of our system. This integration would definitely lower drastically the computational time required for the matching phase, thus offering a pathway for scalability to accommodate potential future improvements. However, the incorporation of multiprocessing was not deemed an immediate priority, given that our existing solution, which operates within a 9-hour timeframe, already fulfills the time constraint of 24-hour work cycle within surplus time available.

3.6 Conclusion on accuracy and time efficiency

Given the limitations that our hardware systems impose, the selection of SSIM emerged as the only pragmatic choice. This decision was driven by the demand to meet the desired time threshold while maintaining commendable result quality. However, if a more powerful system was available, the multiprocessing approach using VGG19 would deliver superior results.

Consequently, MEREBO2's final version of matching algorithm consists of:

1. Classification into categories using a custom trained *Logistic Regression* model.
2. Clustering using *k-means* into K=67 clusters with the help of predicted categories.
3. Improved handling of matrices
4. Image comparison using the *Structural Similarity Index* (*SSIM*)

The following table demonstrates the total improvements of MEREBO 2 in comparison to MEREBO.

	MEREBO	**MEREBO 2**	**Improvement**
F1 score	*0.83*	***0.84***	*1.2 %*
Accuracy	*0.76*	***0.79***	*4 %*
Execution Time	*186 (hours)*	***11 (hours)***	*94 %*

Table 4: Total accuracy and execution time improvements of MEREBO2

4. Our server-side architecture

All the components we mentioned before, like the web scraping and matching algorithms, are connected and orchestrated by our *Node.js* server.

Node.js -or sometimes just *Node*- is an open-source backend framework built on *Javascript* that allows the execution of *JS* code outside a web browser. It is a crucial component of our system that handles user requests, interprets them into queries that our database understands and executes a plethora of other server-side logic. Node, with its non-blocking, event-driven architecture, is designed to manage multiple incoming requests at the same time without bottlenecks. This setup is particularly useful in scenarios where certain tasks, such as a database update event might require up to 14 hours. In the meantime, *Node.js* ensures that the application remains responsive and ready to handle incoming API requests.

4.1 API

An *Application Programming Interface (API)* serves as an intermediary that enables various software applications to communicate under a predefined set of rules. APIs can facilitate various functionalities such as sending and retrieving data or executing logic. The general rule or format to build an API is to follow the REST style.

Representational State Transfer or REST, refers to an architectural style for designing applications. A RESTful API is an API that follows the principles of REST. One key feature of REST design is that each client request contains all the necessary information for the server to be able to satisfy it. They use standard HTTP methods like GET, POST, PUT and DELETE to perform operations and return data in a specific format, most commonly JSON or XML form and are designed to be easily understood by the developers.
Developing RESTful APIs is particularly important for several reasons. Firstly, they offer a consistent way to interact with web services, promoting simplicity and intuition in how clients, servers and services communicate. Additionally, the robust nature of the request enables changes or updates to be made to the server of the client without affecting the communication, thus simplifying the development cycle and providing scalability and flexibility.

4.2 Our RESTful API

Our RESTful API is designed to handle all the necessary logic needed for the system to be viable. To connect with our local Sorl database we used the Javascript package *solr-node* [14] that provides all the needed functionality. There are 4 main endpoints available through which the application can communicate with the server in a very concise and strict form.

app.post('/search')

This endpoint serves as the channel through which interactions with the Solr database are carried out to locate a product. It utilizes the request body to encompass vital information necessary for executing the query. The included data entail the product description and category, allowing the system to generate a query in a format compatible with Solr. Upon the query;s execution, if the result proves to be empty, the response to the client is an error message that is then recognised by the application on the client-side. Conversely, if products are found, the subsequent steps involve sorting the returned products. The initial sorting prioritizes products based on completion percentage, meaning items available across all three stores, followed by those available in two stores and then in one store. Following this, a secondary sorting operation occurs within these subsets, based on the total image similarity index accompanying each product pair in the Solr database. This double sorting mechanism guarantees that the most promising and relevant products are presented foremost, enhancing the user experience.

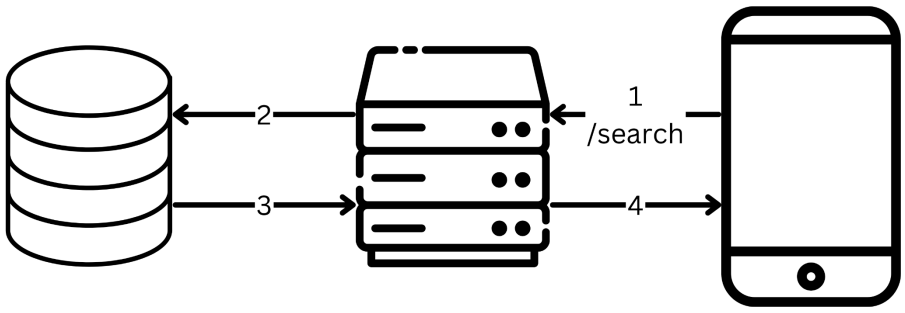


Figure 19: System flowchart of the /search endpoint

app.delete('clear')

As it is evident by its name, this endpoint is called when we are ready to update the Solr database. For the new data to be uploaded and accessed by the customers we first need to empty the outdated data. This is done by making a delete query to Solr with the prompt to delete everything.

app.get('update')

Because updating the database requires first the new data to be collected (scraped) and then to be analyzed to find potential pairs, this endpoint starts by invoking the python script that is responsible for the web scraping process. Then, inside this python script, we invoke the next script that is the matching algorithm. After approximately 12 hours, the time needed for the operation to complete its task, we use the AXIOS Javascript package to make a post request on our /clear endpoint to empty the database. Finally, after all this is completed, the new data is uploaded on the database and is ready to be accessed.

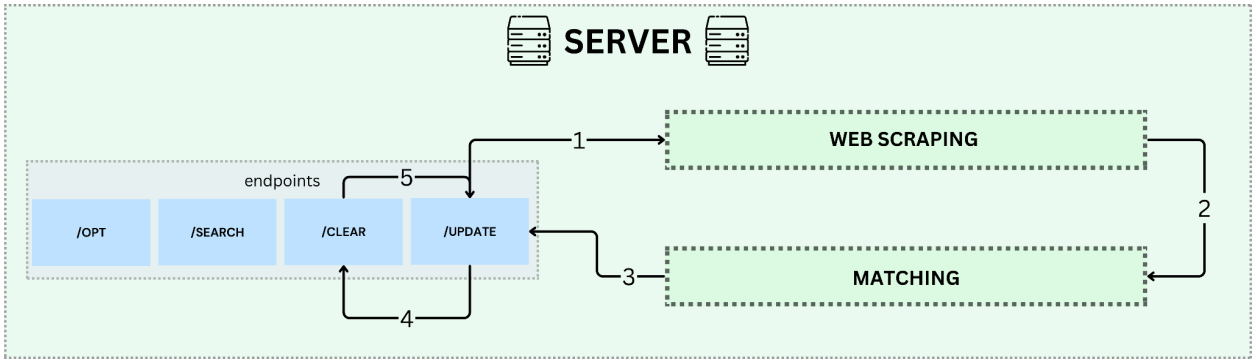


Figure 20: System flowchart of /update endpoint

app.post('/opt')

This is the endpoint that provides the optimization functionality. It should be invoked from the second page of our application containing a list of items with all the information they hold in the database, such as the alternative stores for that product, their prices, desired quantity etc. They are essentially copies of database documents with some extra information. Based on all this knowledge, the endpoint returns the optimized list of products in JSON format.

An optimization process based on the *Simplex Algorithm* [15] is executed that tries to find the best possible combination of products. This is done by splitting the items the consumers 'auto-selected' between the available supermarkets trying to minimize the total cost of the consumer's basket while satisfying the minimum order requirement for the stores and ensuring the inclusion of every item on the user's shopping list. Given that this optimization algorithm is inherited from the MEREBO system while MEREBO2 only focuses on its integration within the application rather than building it from scratch, we are not going to dive deeper into exhaustive analysis of the *Simplex Algorithm*.

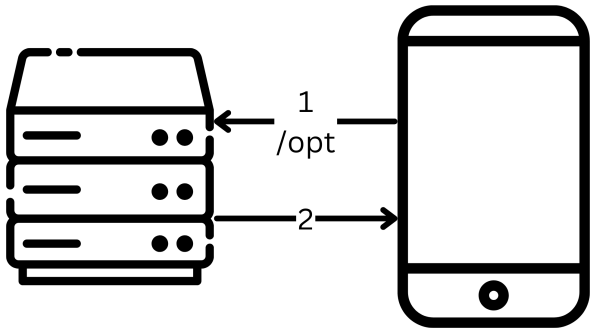


Figure 21: System flowchart of /opt endpoint

4.3 The scheduler and automation of the system

Now that all the work flow is organized under the *update* endpoint, we have to automate the execution of this single endpoint for the whole system to be independent of human interaction.

This challenge is addressed by implementing a *cron job*. Cron, short for ‘chronos’ that means time in Greek, is a background process that enables scheduling of tasks at specific times or intervals. The task that is scheduled to be executed is called a *cron job*.

We defined our cron job to be a function that using the *AXIOS* package makes a request on the update endpoint of our API. To run a cron job we need to set its schedule. Cron scheduler syntax may appear unconventional by with the following code snippet we set the scheduler to ‘every day at 8 am’

*cronSchedule = ‘0 8 * * *’;*

5. Migration to mobile devices

The increasing reliance of consumers on their mobile phones to fulfill a wide array of needs, including activities such as grocery store shopping, underscores the significance of developing a mobile application. There are many tools to build a mobile app such as java for Android, swift for iOS or react native for both of them. We decided to go with the 4th option on the table, the Flutter & Dart.

5.1 Flutter & Dart

Flutter [17] is an open-source framework developed by Google, designed to facilitate the creation of natively compiled applications for mobile, web and desktop from a single source code. The core programming language used in Flutter is Dart, an object-oriented language known for its efficiency and flexibility. Together, Flutter & Dart form a powerful combination, enabling developers to design and deploy cross-platform applications. Our decision was influenced by the increasing popularity of Flutter & Dart as prominent tools within the mobile application community. The appeal lies in the capability to write code that can be seamlessly deployed on both iOS and Android platforms. By writing a single codebase the process of development is streamlined while the compiler handles the technical intricacies and specific requirements imposed by different operating systems.

5.2 Our mobile application

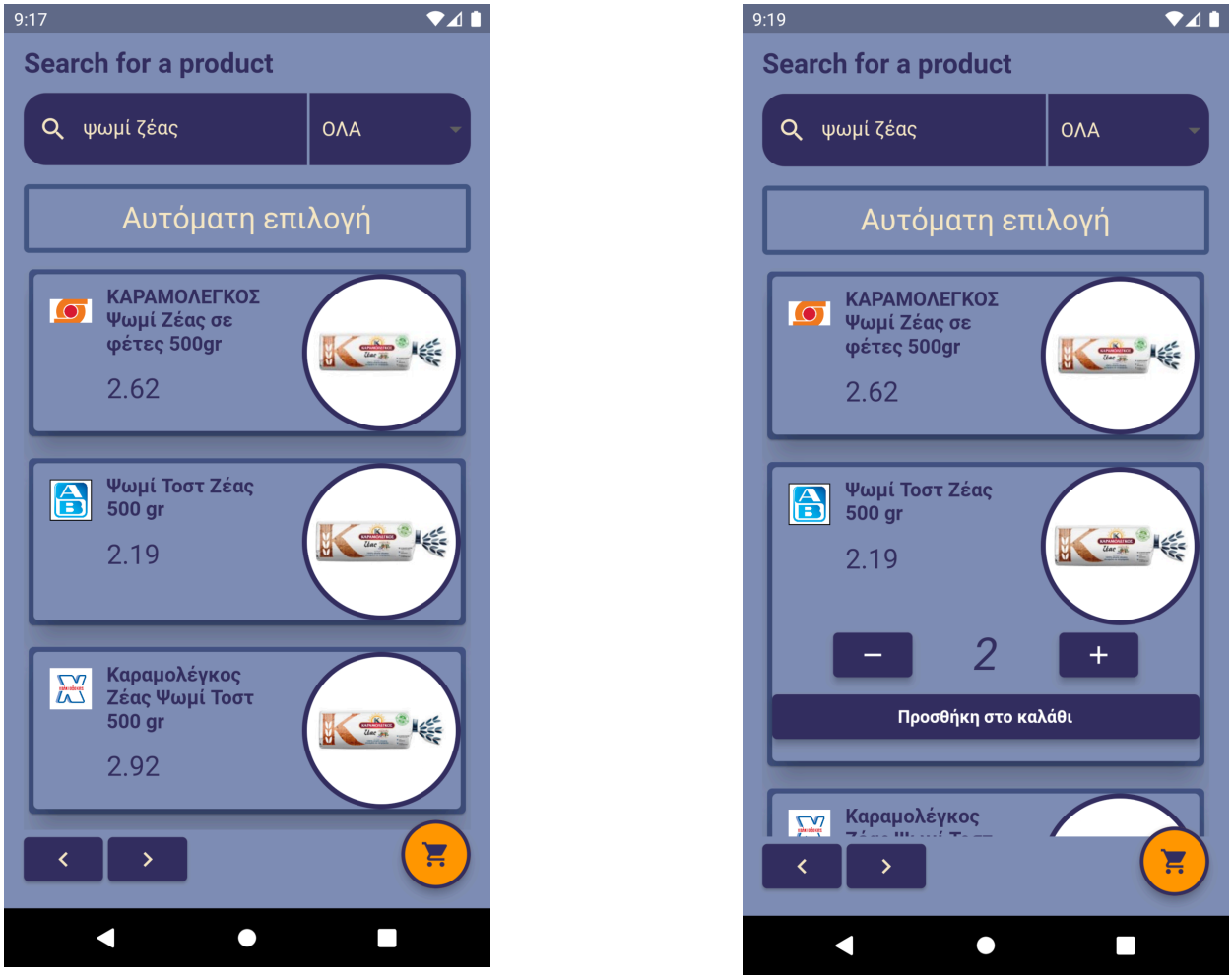
The application is designed into three distinct pages, each tailored to cater to specific user interactions.

5.2.1 First page

In the first page, users are greeted with a user-friendly interface featuring a prominent search bar and a category dropdown menu. This design facilitates a seamless search experience, enabling users to swiftly and precisely find the products they consider purchasing. Users have the flexibility to input the name of the desired product into the search bar and initiate a search, essentially creating a *POST* request on our server. Furthermore, they can opt to refine their search by selecting a specific product category from the dropdown menu, allowing them to narrow down their search to precisely match their requirements.

The process of category searching within the system relies on the predicted category determined by the custom logistic regression model we discussed earlier. Initially developed to aid and guide the clustering process, this model provided an extra opportunity. The system was enhanced by adding a new field in our Solr schema, dedicated to storing the output of the logistic regression model - the predicted category. By leveraging the predicted categories and integrating them into the database the users are allowed to conduct searches with enhanced precision and relevance.

Upon finding a product they wish to purchase, users are presented with two distinct options. The option provides an “auto-select” functionality, which lets them specify the desired quantity of the product but granting the algorithm freedom to select the most optimal product from the list. This strategic decision is integral to the subsequent stages of the process, enabling the algorithm to form the most optimized or the cheapest basket based on the users initial demands. This optimization is explained in detail in chapter 5.2.3. When the users are satisfied with the products they selected, they can navigate to the second page by tapping on the Floating Action Button at the bottom of the screen. This intuitive and accessible first page serves as the gateway to an efficient and tailored product search process.



Figures 23 & 24: Screenshots of the first page of our mobile app. In the right image, the user tapped on one product and was presented with some options.

Another option the user is presented with is setting the category to search in. At the right side of the search input field we can see a dropdown menu that the customer can interact with to refine their query to a particular category. This functionality serves as a mitigating measure against the challenge arising from product terms existing across multiple categories, like the term ‘chocolate’, which pertains to both chocolate ice cream in the freezer category and chocolate bars in the snacks category. Notably, users seeking to make a snack purchase without a specific product in mind have the option to select the snacks category while leaving the input field blank. This action results in the display of the entire snacks catalog, allowing users to seamlessly navigate through the list using the arrow buttons at the bottom of the screen and make spontaneous selections based on their preferences.

In the following screenshot, the user wants to buy an ALFA beer, but this exact beer was not found in the AB store. Choosing 'auto select' in this scenario, essentially means that the final order is forced to fulfill an order either from Sklavenitis or Chalkiadakis. However, in *Figure 24* they opted to buy a bread item directly from AB. Consequently, this means that in this example, the algorithm is obliged to meet the minimum order requirement of 25 € imposed by AB for the purchase of that specific bread, and simultaneously satisfy the minimum order constraints set by Chalkadakis or Sklavenitis in order to buy the beer. Now imagine this complex problem for a list of 60 or 70 products. The customer would almost certainly not be able to find the best combination.

Figure 25: Beer item is not found in all shops

5.2.2 Second page

The second page of the system serves as a straightforward interface presenting users with a comprehensive list of all the products currently added to their basket. Within this section, consumers are granted two primary actions. Firstly, they possess the autonomy to modify the quantity of an item or entirely remove it from the basket. This flexibility enables users to adjust their selection according to their preferences and needs.

Moreover, users have the option to navigate back to the first page, allowing them to supplement their list with additional items. Upon completing their selections and potential modifications, users can proceed by triggering the optimization function. Upon activating this function, the system automatically transitions to the third page. This seamless transition allows users to progress to the subsequent stage after finalizing their basket selections and adjustment, enhancing the overall user experience by facilitating a smooth and intuitive process.

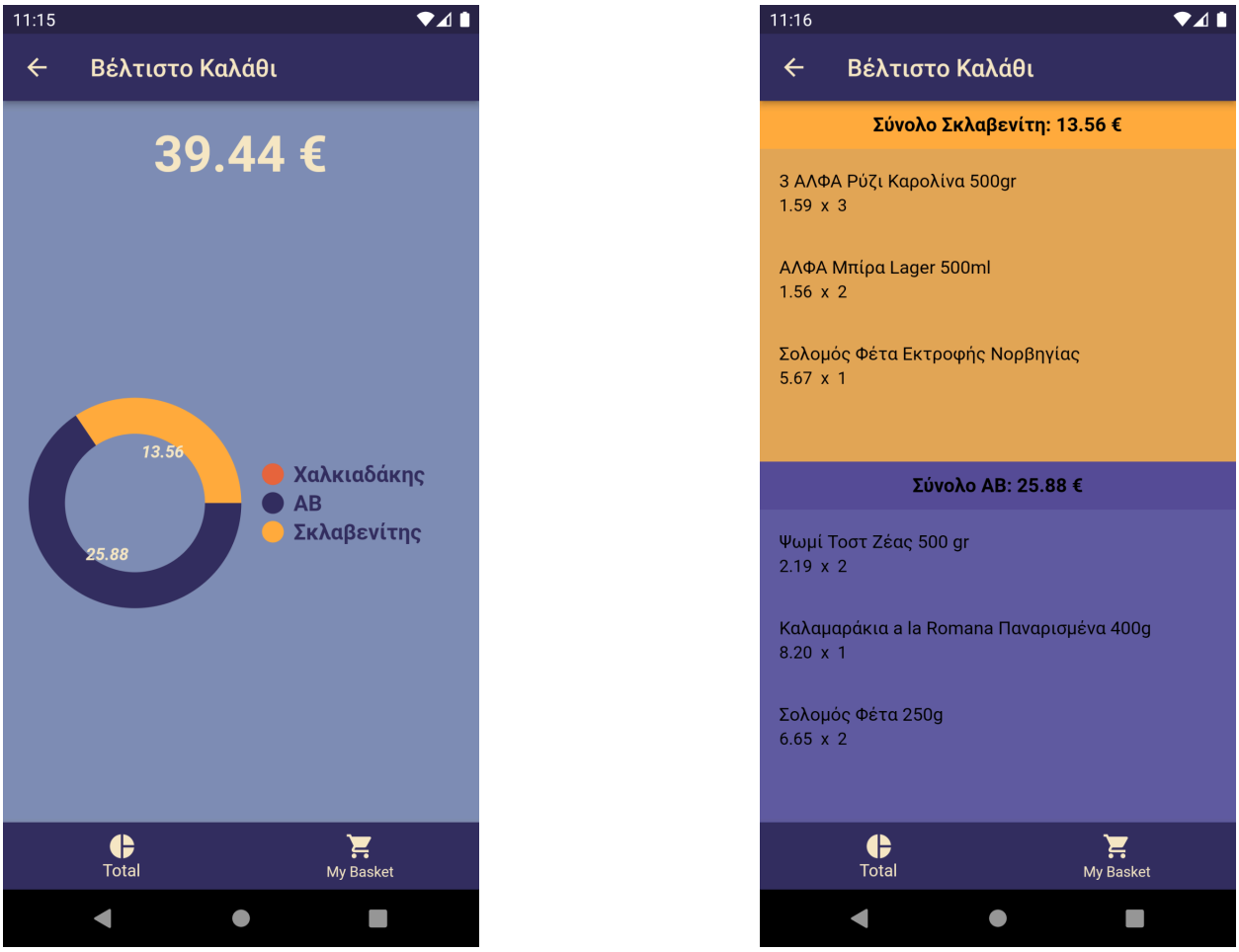
Figure 26: Second page of the app

In the previous screenshot of the second page, we can see that a user has picked 5 products of various quantities. The second column is particularly important as it denotes whether the product is autoselected or the user demands this specific store. Lastly, the Floating Action Button at the bottom of the screen performs the optimization of the basket. It sends a post request to the */opt* endpoint of our server and receives the optimized list which is then automatically passed on to the last page for displaying purposes.

5.2.3 Third page

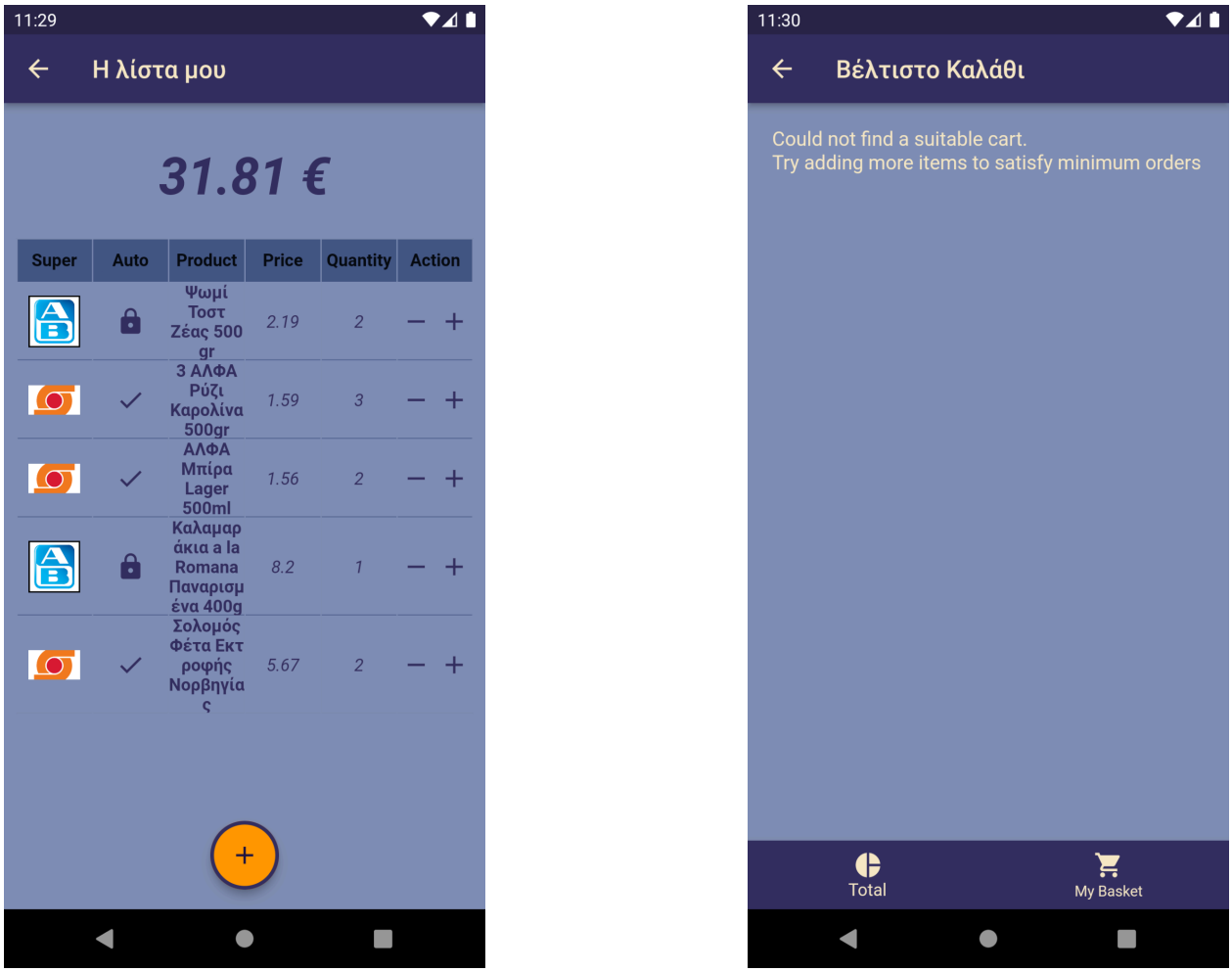
The third and final page of the application is the main selling point of the MEREBO system. Within this page, we display the results of the optimization algorithm. If the criteria are not met, a message appears that informs the customer that they need to add more items in order to complete an order. If there is a basket available, we display it in two distinct pages. The first one that appears, shows the new total cost of the optimized list along with a pie chart that gives a rough idea on how the order is proportioned among the stores. This addition is considered to be valuable for high cost orders. The second page the user can navigate to shows the complete list of products in groups based on the supermarket. Each group is associated with a color that matches the theme color of the online store it represents, so each list is clear as to what it represents.

In the example we followed for this demonstration, we can see that the total cost of the order as it is in the second page (pre-optimized) is lower than the third page's optimized cost. This is because in the second page the minimum order cost for the markets was not satisfied. The Simplex Algorithm was able to split the last product quantity between two stores instead of one because we marked it with auto select, thus satisfying the 25 € threshold for AB.



Figures 27 & 28: Left is the ring chart of the successful basket and right is the analysis of each shop's order

If we go back and reduce the quantity of the last product in the list by 1, we can see that the order is no longer possible. This circumstance arises due to the specific requirement outlined in *Chapter 5.2.1*, necessitating the acquisition of the beer item from Sklavenitis or Chalkiadakis. The decrease in the quantity of the last item in the list results in a failure to meet the minimum delivery cost criteria established for transactions with either of these two retailers.



Figures 29 & 30: Left is the same list of items shown in image 21 but with one less item and at the right image we see that the basket is no longer feasible

5.3 Multiple results

The final noteworthy contribution of this thesis involves the enhancement of the user experience by offering a diverse array of products to choose from.

5.3.1 Client side solution

When a *POST* request to find the products is initiated, the server constructs a query which is then sent to the Solr database for processing. Unlike a conventional approach that would typically return the most promising single result, our approach employs a series of actions to form a package of data in *JSON* format. This package is then sent back to the application as the result of the *POST* request. It encompasses every conceivable match that the query has returned, organized in descending order of similarity and completeness. However, this approach may impose limitations in scenarios where additional stores are introduced into the system. In such cases, the scope of results could expand significantly, potentially placing a heavier load on the mobile application posing a challenge for lower end phones that may struggle to handle the substantial volume of results. It is imperative to underscore that this concern is largely hypothetical within our current system state, as the volume of results is well-contained and manageable, thus posing no immediate threat. The theoretical nature of this issue alongside the simplicity of implementing this method prompted our selection of this approach.

5.3.2 Server side solution

An alternative approach involves generating a new request with each tap on the ‘next’ button to display subsequent items. This alternative, however, immediately posed various challenges. The primary obstacle was the necessity of developing a mechanism to retain the products retrieved in a query and subsequently recall the index of the last accessed product. Notably, given the context of this being a customer-oriented application intended for widespread use, it becomes imperative to store this information for each individual user. Furthermore, the implementation of this method raises concerns about potential server bottlenecks due to the heightened volume of requests to the server leading to higher operational costs for its efficient functioning. These economic implications coupled with the complexity of the solution led us to discard this approach.

6. Discussion & Future Work

In conclusion, this thesis has successfully achieved its goals in developing a robust and efficient MEREBO2 system on top of its predecessor. The system's performance, scalability and user-friendly interface align with the demands of the dynamic e-commerce landscape.

Moving forward, there are numerous avenues for further exploration and refinement. Specifically, one notable direction for improvement centers around enhancing the search functionality. Future work could focus on developing a more specialized and customizable search mechanism, thereby augmenting the user experience with a broader array of choices. A key aspect of this enhancement could involve refining the search parameters, such as enabling users to specify a maximum distance from a retailer. Doing so, the application could include not only the biggest supermarket chains but also the small local stores, thereby expanding its inclusivity within the retail landscape. Additionally, introducing filters related to estimated wait and delivery times would empower customers to set preferences based on their time constraints, further personalizing the search experience.

REFERENCES

[1] Ioannis Misokalos, "MEREBO: A web mart aggregator for multi-source entity resolution and basket optimization", Diploma Work, School of Electrical and Computer Engineering, Technical University of Crete, Chania, Greece, 2023

[2] Selenium v.4.8.2 https://www.selenium.dev/documentation/

[3] Apache Solr v.8.11.2 https://solr.apache.org/

[4] VGG19 https://keras.io/api/applications/vgg/

[5] Karen Simonyan, Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition, arXiv:1409.1556

[6] Jian Xiao et al 2020 J. Phys.: Conf. Ser. 1518 012041 Application of a Novel and Improved VGG-19 Network in the Detection of Workers Wearing Masks

[7] Sara, U., Akter, M. and Uddin, M.S. (2019) Image Quality Assessment through FSIM, SSIM, MSE and PSNR—A Comparative Study. Journal of Computer and Communications, 7, 8-18.

[8] SSIM https://scikit-image.org/docs/stable/auto_examples/transform/plot_ssim.html

[9] Python multiprocessing https://docs.python.org/3/library/multiprocessing.html

[10] Darmawan, Irfan & Maulana, Muhamad & Gunawan, Rohmat & Widiyasono, Nur. (2022). Evaluating Web Scraping Performance Using XPath, CSS Selector, Regular Expression, and HTML DOM With Multiprocessing Technical Applications. JOIV : International Journal on Informatics Visualization. 6. 904. 10.30630/joiv.6.4.1525.

[11] Ahmed, Mohiuddin, Raihan Seraj, and Syed Mohammed Shamsul Islam. 2020. "The k-means Algorithm: A Comprehensive Survey and Performance Evaluation" Electronics 9, no. 8: 1295.

[12] Shi, C., Wei, B., Wei, S. *et al.* A quantitative discriminant method of elbow point for the optimal number of clusters in clustering algorithm. *J Wireless Com Network* 2021, 31 (2021). https://doi.org/10.1186/s13638-021-01910-w

[13] Maalouf, Maher. (2011). Logistic regression in data analysis: An overview. International Journal of Data Analysis Techniques and Strategies. 3. 281-299. 10.1504/IJDATS.2011.041335.

[14] Solr-node package https://www.npmjs.com/package/solr-node

[15] J. A. Nelder and R. Mead. A simplex method for function minimization. The computer journal, 7(4):308–313, 1965.

[16] Singh, Navtej & Browne, Lisa-Marie & Butler, Ray. (2013). Parallel Astronomical Data Processing with Python: Recipes for multicore machines. Astronomy and Computing. 2. 1-10. 10.1016/j.ascom.2013.04.002.

[17] Flutter v.3.10.1 & Dart v.2.23.1 https://flutter.dev/

[18] L. Breiman. Random forests. Machine learning, 45:5–32, 2001.