Technical University of Crete
Department of Electrical and Computer Engineering

MEREBO: A web mart aggregator for multi-source entity resolution and basket optimization

Author: Ioannis Misokalos
Supervisor: Professor Minos Garofalakis

January 2023

Abstract

Today, more than ever, we can observe the facilitation of trading provided by the ability of the internet to connect - directly or indirectly - the distributor and the consumer. Many stores have taken to the electronic side of things, and have set up e-shops, to more easily provide their services to their buyers. As proven by the recent health crisis, the benefits of such stores are many, and the business is growing to a degree that has started to overshadow its physical counterpart, incentivising more and more businesses to set up their own electronic stores. This, however poses a problem for the consumer. While the decision of where to obtain our goods had been based on factors such as proximity to our homes, availability of certain products, etc., now we have at our discretion a plethora of equally convenient options that make our decision all the more difficult. To this end, a newer service model has been introduced. A medium that collects data from multiple stores, provides the user with the available options and helps them make the best decision based on their needs. In this thesis, we aim to create such a model of service. **MEREBO** stands for **M**ulti-source **E**ntity **RE**solution and **B**asket **O**ptimization. This app model attempts to collect and store grocery store product data in a database, compare and match the products between them, showcase those matches to the user of the app, and provide them with an optimization option in their basket, so that they may have the best shopping experience.

Acknowledgements

As the time to present my thesis approaches, I can't help but think back on these past years of studies, and think of all the people who made this possible. Like everything in life, my time in this university and this thesis has been a result of collective effort, and it would only be fair to use this chance to give my thanks to everyone who stood by me and took part in this journey of mine.

To start with, I would like to offer my thanks to my supervisor, Professor Minos Garofalakis, for providing me with this opportunity to work on this subject, and providing guidance throughout its completion. I would also like to thank Professor Georgios Chalkiadakis, who has also been closely involved with this project, and has offered his own assistance and expertise. I am grateful to all the professors of the department of electrical and computer engineering of the university for offering an abundance of knowledge in my years here.

I must express my gratitude to Professor Nikos Giatrakos, as well as congratulate him on his new, much deserved position. Throughout this project we have been in contact and he has answered all of my questions, guided me through the process and, overall, shown me how to be a better scientist.

I wish to thank my family for their unconditional love and support during my years in Crete. Without them and their patience and encouragement, none of this would be possible and I wouldn't be the person I am today. Lastly, I want to thank my friends and especially Dimitris, for being there and helping me through some of the tough times I had to face this past year.

Contents

Chapter 1

Introduction

1.1 Data Aggregation and its uses

The Information Age is characterized by a rapid shift from traditional industries, as established during the Industrial Revolution, to an economy centered on information technology. Data has arguably become the new currency of our century, because of the value that it provides for the successful operation of a business. Put simply, organizations with superior data management and the ability to use it more effectively, win in the market. This is also the reason why the amount of accessible data is the biggest it's ever been.

While the abundance of data available to us allows us to extract information and and use it in ways we otherwise couldn't, when examined in its most basic form, also known as atomic data, the process can be extremely time consuming and tedious. Data aggregation is any process whereby data is gathered and expressed in a summary form. When data is aggregated, atomic data rows – typically gathered from multiple sources – are replaced with totals or summary statistics. Aggregate data is typically found in a data warehouse, an enterprise system used for the analysis of data from multiple sources, as it can provide answers to analytical questions and also dramatically reduce the time to query large sets of data.

The collection of data needed to create those databases may be given by the websites that gather them, but when this is not the case, we often have to resort to ”crawling” or ”scrapping” those pages for their data. Crawling, is the process of accessing the databases of all necessary pages, collecting this data and storing it in our own database for processing. This can be achieved very easily on static pages, meaning web pages that are delivered to the user’s web browser exactly as stored, but in the case of dynamic web pages, that are generated by a web application, the process becomes a bit more complex.

1.2 The problem of Entity Resolution

In recent years, several databases have been built to enable large-scale knowledge sharing, but also an entity-centric Web search, mixing both structured data and text querying. These bases offer machine-readable descriptions of real-world entities published on the Web as Linked Data. It is often the case that a database of aggregated data includes tuples of a specific object. While entities are usually readable by algorithms, the same object may possess differently worded descriptions, that may seem obviously similar to humans, but indistinguishable to the algorithm. Depending on the field of study, specialized entity resolution algorithms can be used to identify such tuples, and allow for a more organized database.

Entity resolution is the problem of extracting, matching and resolving entity mentions in data. For long it has posed a challenge in database management, information retrieval, machine learning, natural language processing and statistics. Efficient entity resolution offers great benefits in a variety of commercial, scientific and security domains. Especially now, in the age of big data, the need for high quality entity resolution is more prevalent than ever, as we are overwhelmed with more and more data, the utility of which cannot be fully extracted until it has been integrated and matched.

In this thesis, the problem of entity resolution is rather complex. Our information is collected in a non-cooperative way between our app and the product providers. The scrapped information is semi-structured, in that it does not have a common schema between independent product providers. It is this lack of synergy that poses an especially significant challenge.

1.3 Apache Solr as a database and search engine

Solr is a search server built on top of Apache Lucene, an open source, Java-based, information retrieval library. It is designed to drive powerful document retrieval applications. Solr is highly reliable, scalable and fault tolerant, providing distributed indexing, replication and load-balanced querying, automated failover and recovery, centralized configuration and more. Solr powers the search and navigation features of many of the world's largest internet sites, like Macy's, EBay, and Zappo's. It can run both locally and using cloud technology, as a single Solr server can only support so much data, and so many queries.

While Solr's default UI is adequate for understanding its workings as a search engine, it is far from user-friendly. Throughout the years, there have been many interfaces created by developers to better achieve user satisfaction, although in this thesis we design our own interface for our web app that connects to Solr to better suits our needs.

1.4 Basket Optimization and its importance

When it comes to choosing a product that's available from various sources and for which we have already performed all necessary entity matching procedures, many factors have to be taken in mind. The obvious one that comes to mind is price. When faced with the decision between two identical objects, most of us would chose the cheaper one. However, there are other things to take into consideration. For example, when delivering products, most stores would require a minimum price of items in the basket, otherwise they may refuse to offer delivery services. While it is easy to distribute our items when they are few, it is often the case with grocery shopping, that we end up buying tens of items, each with their unique price. Calculating the optimal way of distribution of products into multiple baskets of separate stores in such cases might prove impossible for the average person. This is why some stores have begun to implement ”smart baskets”, algorithms that assist the user in choosing the most affordable combination of items, in a way that fulfills any requirements that might be in place.

1.5 Thesis Outline

This thesis has been organized into five chapters. This section outlines the description of each chapter:

- In Chapter 2, we collect the data of three Greek grocery electronic stores: Sklavenitis, AB Vasilopoulos, and Chalkiadakis. This will be achieved through the automated crawling of their web-pages, which manifest in both static and dynamic forms.
- In Chapter 3, we develop an algorithm that can efficiently recognize and match identical multi-source entities and provide us with aggregated data that can be used to create a database for our application.
- In Chapter 4, we use Apache Solr as a database and a search engine, that we will use to store our data and through queries obtain the products that we want. We will also examine Solr's UI, the existing, more user friendly alternatives, and finally develop our own UI's front-end that can be used in the application.
- In Chapter 5, we provide a basket optimization algorithm, that we use after adding products to our cart, allowing us to get the best price across all possible sources.
- In Chapter 6, we showcase our conclusions and review the accomplishments of this thesis.

Chapter 2

Data Aggregation

This chapter has the objective to introduce the electronic shop pages from the grocery stores which we chose in order to collect the data we used in this thesis, as well as the methods we used to do so. We will showcase the differences in the structure of the pages and present ways to collect item data most efficiently. Finally, we will show the database we created by the end of this step.

2.1 The Stores and their Pages

In order to collect the necessary data we wanted to make up our database, we first had to make a decision on which grocery stores to use as our ”supplier”. The stores we wanted had to at least have branches in Chania, offer delivery services, and most importantly, own electronic stores. We ended up with three candidates that fit these criteria: **Sklavenitis**, **AB Vasilopoulos**, and **Chalkiadakis**.

2.2 The process of Crawling

After deciding on which grocery stores to use, the first step we had to take was to collect the data we wanted. This included the whole catalogue of items from each store, regardless of category, including:

- Name and description
- Price before and after any sales that take place
- The image associated with the product
- The name of the grocery store

In the case the item was out of stock, we would still gather the appropriate data, but label it as such.

We examined several ways and methods of crawling [18] in this step, with the most notable being **HTTrack**, **Python BeautifulSoup**, and **Python Selenium**.

The crawling process had to be specific to each grocery store depending on its e-shop page, with the main concern being the model of the page, as static and dynamic pages have inherently different structures that must be tackled appropriately.

2.2.1 Static Pages

For static web pages when a server receives a request for a web page, the server sends the response to the client without doing any additional processing. Pages will remain the same until someone changes it manually. The main way this manifests in our case, is that once we have loaded a page, for example, the dairy category of the grocery store, the code of the page we can observe through our web browser includes all the necessary information we want. As such, it is a very simple and fast process to crawl a static page. Unfortunately, possibly because of its smaller size relatively to the other two stores, Chalkiadakis was the only one to have a static page.

The most obvious option was to use HTTrack. HTTrack [3] is a free offline browser utility. It allows you to download a World Wide Web site from the Internet to a local directory, recursively building all directories, getting HTML, images, and other files from the server to a computer. In about a day of waiting time, we had a complete, working mirror of the Chalkiadakis e-shop, including all the files we needed to fill our database. From there, it is a matter of sorting those files with the help of the HTML code we have downloaded.

To achieve this we used Python BeautifulSoup. BeautifulSoup [23] is a Python library for pulling data out of HTML and XML, helping us parse through the HTML file [25]. With the use of class names, we were able to isolate the data we wanted and store it.

2.2.2 Dynamic Pages

When the web server receives a user request for a dynamic page, it does not send the page directly to the requesting browser as it would do with a static page. Instead, it passes the page to the application server which then completes three activities:

- Read the code on the page
- Finish the page according to the code's instructions
- Remove the code from the page

This results in a static page that's passed back to the web server by the application server, and then to the requesting browser for display.

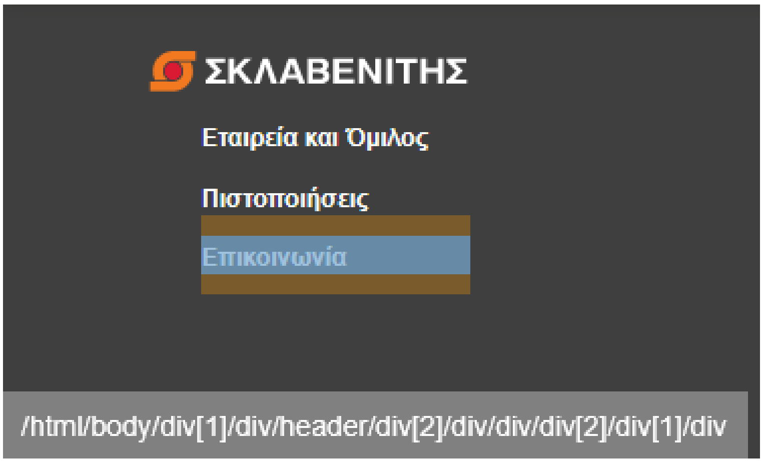
The other two grocery stores used dynamic pages for their e-shops, making the process slightly more complex. While in static pages we can simply download an HTML file and parse in order to isolate the data that are useful for our purposes, here each iteration of server-user communication produces a different static page, so for example the HTML code of a page showing the items of a category of products, would only include information on the items shown, and not other items, like those of different categories, or even those in different pages of the same category. This means that if we want to have a complete list of all the items included in an e-shop, we would have to parse a whole compilation of static pages across all of the stores' websites. In our case, this translates to each page of every item category included in either store's catalog.

In order to achieve this, we used Python Selenium [7]. Selenium is a library aimed at supporting browser automation [22], and the goal was to create a script that would successfully browse through the store's e-shop website and methodically collect the required data.

For compatibility reasons, our browser of choice was Mozilla Firefox, so we had to use GeckoDriver and create a path to it for Selenium. GeckoDriver serves as a proxy between WebDriver enabled clients and the Firefox browser. In other words, GeckoDriver is a link between Selenium tests and Firefox.

Because we can no longer find our goal from the get-go, we will have to show the script how to navigate through the e-shop, by using a simple algorithm that utilizes XPath expressions as anchor points. An XPath expression can be used to search through a document, and extract information from any part of the document, such as an element or attribute. For example, if we know the XPath used for navigating to the main page of the store, we can teach our script to return there after browsing each category.

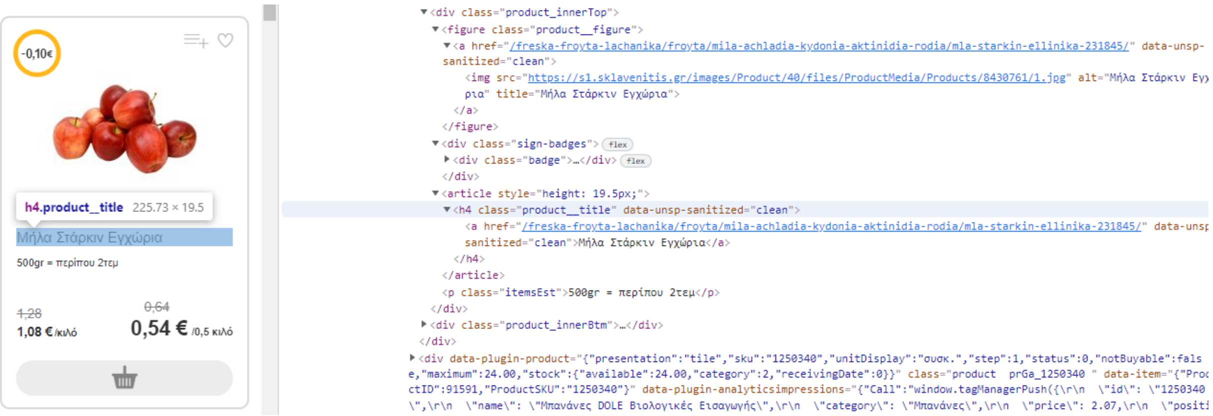
To be able to easily see the XPath expressions we need, we used the extension XPath Finder, which shows the XPath of an element with the click of the cursor. An example can be shown on the image below, where by clicking on the contact button of the grocery store, we are given its XPath expression.



```
/html/body/div[1]/div/header/div[2]/div/div/div[2]/div[1]/div
```

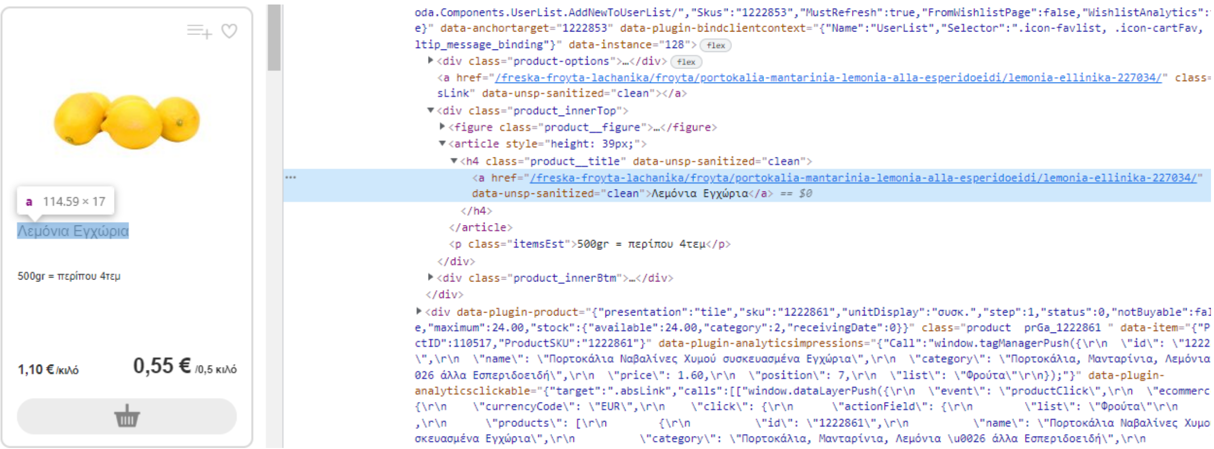
Clicking on the contact button shows the button's XPath in the gray box below

Unfortunately, in some cases the information we need isn't available through xPaths, or there is simply a better way of getting the data we want without having to acquire a whole package of elements. This was usually the case once we had navigated to the products themselves, and needed specific information. In those cases we had to use class names to direct our script to the goal. An example is shown in the images below.



```
▼<div class="product_innerTop">
  ▼<figure class="product__figure">
    ▼<a href="/freska-froyta-lachanika/froyta/mila-achladia-kydonia-aktinidia-rodia/mla-starkin-ellinika-231845/" data-unsp-
    sanitized="clean">
        <img src="https://s1.sklavenitis.gr/images/Product/40/files/ProductMedia/Products/8430761/1.jpg" alt="Μήλα Στάρκιν Εγχώ
        ρια" title="Μήλα Στάρκιν Εγχώρια">
      </a>
    </figure>
  ▼<div class="sign-badges"> flex
    ▶<div class="badge">…</div> flex
    </div>
  ▼<article style="height: 19.5px;">
    ▼<h4 class="product__title" data-unsp-sanitized="clean">
        <a href="/freska-froyta-lachanika/froyta/mila-achladia-kydonia-aktinidia-rodia/mla-starkin-ellinika-231845/" data-unsp
        sanitized="clean">Μήλα Στάρκιν Εγχώρια</a>
      </h4>
    </article>
    <p class="itemsEst">500gr = περίπου 2τεμ</p>
  </div>
 ▶<div class="product_innerBtm">…</div>
 </div>
▶<div data-plugin-product="{"presentation":"tile","sku":"1250340","unitDisplay":"συσκ.","step":1,"status":0,"notBuyable":fals
e,"maximum":24.00,"stock":{"available":24.00,"category":2,"receivingDate":0}}" class="product  prGa_1250340 " data-item="{"Prod
ctID":91591,"ProductSKU":"1250340"}" data-plugin-analyticsimpressions="{"Call":"window.tagManagerPush({\r\n  \"id\": \"1250340
\",\r\n  \"name\": \"Μπανάνες DOLE Βιολογικές Εισαγωγής\",\r\n  \"category\": \"Μπανάνες\",\r\n  \"price\": 2.07,\r\n  \"positi
```

We can find the product name under the class name "product__title"



```
oda.Components.UserList.AddNewToUserList/","Skus":"1222853","MustRefresh":true,"FromWishlistPage":false,"WishlistAnalytics":
e}" data-anchortarget="1222853" data-plugin-bindclientcontext="{"Name":"UserList","Selector":".icon-favlist, .icon-cartFav,
ltip_message_binding"}" data-instance="128"> flex
 ▶<div class="product-options">…</div> flex
  <a href="/freska-froyta-lachanika/froyta/portokalia-mantarinia-lemonia-alla-esperidoeidi/lemonia-ellinika-227034/" class=
  sLink" data-unsp-sanitized="clean"></a>
 ▼<div class="product_innerTop">
   ▶<figure class="product__figure">…</figure>
   ▼<article style="height: 39px;">
     ▼<h4 class="product__title" data-unsp-sanitized="clean">
…       <a href="/freska-froyta-lachanika/froyta/portokalia-mantarinia-lemonia-alla-esperidoeidi/lemonia-ellinika-227034/"
        data-unsp-sanitized="clean">Λεμόνια Εγχώρια</a> == $0
      </h4>
    </article>
    <p class="itemsEst">500gr = περίπου 4τεμ</p>
  </div>
 ▶<div class="product_innerBtm">…</div>
 </div>
▶<div data-plugin-product="{"presentation":"tile","sku":"1222861","unitDisplay":"συσκ.","step":1,"status":0,"notBuyable":fal
e,"maximum":24.00,"stock":{"available":24.00,"category":2,"receivingDate":0}}" class="product  prGa_1222861 " data-item="{"P
ctID":110517,"ProductSKU":"1222861"}" data-plugin-analyticsimpressions="{"Call":"window.tagManagerPush({\r\n  \"id\": \"1222
\",\r\n  \"name\": \"Πορτοκάλια Ναβαλίνες Χυμού συσκευασμένα Εγχώρια\",\r\n  \"category\": \"Πορτοκάλια, Μανταρίνια, Λεμόνια
026 άλλα Εσπεριδοειδή\",\r\n  \"price\": 1.60,\r\n  \"position\": 7,\r\n  \"list\": \"Φρούτα\"\r\n});"}" data-plugin-
analyticsclickable="{"target":".absLink","calls":[["window.dataLayerPush({\r\n  \"event\": \"productClick\",\r\n  \"ecommerc
{\r\n    \"currencyCode\": \"EUR\",\r\n    \"click\": {\r\n      \"actionField\": {\r\n        \"list\": \"Φρούτα\"\r\n
,\r\n      \"products\": [\r\n        {\r\n          \"id\": \"1222861\",\r\n          \"name\": \"Πορτοκάλια Ναβαλίνες Χυμο
σκευασμένα Εγχώρια\",\r\n          \"category\": \"Πορτοκάλια, Μανταρίνια, Λεμόνια \u0026 άλλα Εσπεριδοειδή\",\r\n
```

While a different product, the name is mentioned under the same class name

As we can see in both of these images, while the product names under examination are different, they are both stored under the class name **”product__title”**. Knowing these facts, we can explain how our selenium script works:

Once we are in the e-shop, we begin browsing through the categories one by one. Unlike static pages, where the products are sorted in pages, here we are faced with an Infinite Scroll mechanism, a JavaScript plugin that automatically adds the next page once the user scrolls down, saving them from a full page load. Because the page is dynamic, every time we reach the end of the page and we keep scrolling, a new HTML code is generated that includes all data on previous products, on top of the new data. Once we reach the end of the Infinite Scroll, we have an HTML code that includes all products of this particular category. We can then proceed to catalog all of this data using the class names of the information we require. We can then move on to another category using its xPath, and repeat this process until we have cleared all available categories.

2.3 The Aggregated Data

Once we finished with the process of crawling, our database was filled with data for over **25,000** products gathered from the three stores, along with their names, descriptions, prices, and any useful information we might need for our next step. After making sure that there were no duplicate objects from the same source, this data was initially stored in an Excel file for easy reviewing and editing.

2.4 Conclusions & Future Work

Having worked with multiple ways of collecting data, we will give our opinion on the most efficient ways of crawling, as well as any improvements that could be made on the existing algorithm.

To start with, when it comes to static pages, it's no surprise that HTTrack in combination with Python BeautifulSoup is much simpler and more efficient in data aggregation than trying to emulate user navigation on a web browser. For the sake of completeness, we tried utilizing Python Selenium to aggregate data from Chalkiadakis' static page, but the process was slower and Selenium itself is more demanding when running, having to keep a browser open for hours on end while collecting data. If, however, we wished to use Selenium for consistency with the other two stores, it should be noted that it is entirely feasible, as that the data collected was exactly the same as with the use of HTTrack and BeautifulSoup.

Another thing to take into consideration with Selenium is that the automation process has to be specifically programmed for each e-shop, as the xPath expressions are different, and the navigation process can be dissimilar enough to create issues. What's more, depending on the websites management, class names and xPath expressions can sometimes be changed, creating the need for editing of the code in those areas.

In the future, the script could be taught to search for these changes by itself and account for them, making it more functional without the programmer's involvement.

Chapter 3

Entity Resolution

In this chapter, we will review our system for matching the entities within our database. Our initial data has been in the form of isolated products, but in this step, we will try to create three-way matches as our basis for entity matching data tuples. To achieve this we must train our algorithm to differentiate between different products, and recognize similar ones. In our final version of the database, we will want to have a 1 to 1 counterpart to our current database, but along with each item's description, price, image, etc, we want to include the best matching item from each of the other two grocery stores, as well as its own relevant content.

3.1 The Data

As stated in the previous chapter, our database at this point contains over 25,000 entities from 3 grocery stores. Through the process of entity resolution we will keep a close eye to the quality of the results by manually verifying the matches in important nodes of the process, and showcase important information regarding the degree of success of the match itself.

3.2 TF-IDF

In order to determine whether two item descriptions accurately match each other, we need a basis for the comparison. The two texts in question might be nearly identical word for word, but still not refer to the same entity, or they might simply share a couple of common words and actually refer to the same product [13]. To tackle this issue, the first step was to decide on a way to determine word importance. We made the decision to use a standard process in information retrieval, i.e., the construction of the inverted index and the computation of **TF-IDF** [10] scores for each crawled product entity seeking for each top-1 (best) match in the rest of the suppliers.

3.2.1 Introduction

TF-IDF is by a wide margin the most popular term-weighting method today. A survey conducted in 2015 showed that 83% of text-based systems in digital libraries use TF-IDF.[11] Variations of the TF-IDF weighting scheme are often used by search engines as a central tool in scoring and ranking a document's relevance given a user query. TF-IDF can also be successfully used for stop-words (words irrelevant to the study) filtering in various subject fields, including text summarization and classification.

To properly understand the process of entity resolution applied to our problem, we first have to comprehend the TF-IDF method. **TF-IDF** stands for **T**erm **F**requency–**I**nverse **D**ocument **F**requency and it is a numerical statistic that is intended to reflect how important a word is to a document. It is usually applied to information retrieval, text mining, and user modeling to assign weights to words that accurately represent their importance. The TF-IDF value increases proportionally to the number of times a word appears in the document and inversely proportionallly to the number of documents in the collection that contain the word, which helps to adjust for the fact that some words appear more frequently in general. TF-IDF is compromised of two statistics: **Term Frequency** and **Inverse document frequency**.

3.2.2 Term Frequency

Term Frequency, according to Hans Peter Luhn, can be summarized as *”the weight of a term that occurs in a document is simply proportional to the term frequency”*.[19] In a collection of text documents, like a library of books, we wish to rank by relevancy to a query of our choice, let’s say *”the three musketeers”*, a simple way to begin is by eliminating documents that do not contain the words *”the”*, *”three”*, and *”musketeers”*. This, however, leaves many documents that still contain any of these terms. To further distinguish them, we might count the number of times each term occurs in each document; the number of times a term occurs in a document is called its term frequency. In the case where the length of documents varies greatly, adjustments are often made, although, in our case, this does not concern us, as product descriptions are always short.

In mathematical terms, Term frequency, **tf(t,d)**, is the relative frequency of term **t** within document **d**,

$$\mathrm{tf}(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}$$

where $f_{t,d}$ is the raw count of a term in a document, i.e., the number of times that term t occurs in document d. The denominator is the total number of terms in document d (counting each occurrence of the same term separately).

3.2.3 Inverse document frequency

In the previous example of *”the three musketeers”*, the term *”the”* is so common, term frequency will tend to overly emphasize documents which happen to use the word more frequently, without giving enough weight to the more meaningful terms ”three” and ”musketeers”. The term *”the”* is not a good keyword to distinguish relevant and non-relevant documents and terms, unlike the other two less-common words. Hence, an inverse document frequency factor is incorporated which diminishes the weight of terms that occur very frequently in the document set and increases the weight of terms that occur rarely. Karen Spärck Jones conceived a statistical interpretation of term-specificity called Inverse Document Frequency (IDF), which became a cornerstone of term weighting.[16]. According to her, *”The specificity of a term can be quantified as an inverse function of the number of documents in which it occurs”*.

The inverse document frequency is a measure of how much information the word provides, i.e., if it is common or rare across all documents. It is obtained by dividing the total number of documents by the number of documents containing the term, and then taking the logarithm of that quotient,

$$\mathrm{idf}(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

where

- N is the total number of documents in the collection of documents D
- The denominator is the number of documents where the term t appears

3.2.4 Term Frequency - Inverse Document Frequency

Knowing these equations, tf–idf can be calculated by multiplicating the two statistics:

$$\mathrm{tfidf}(t, d, D) = \mathrm{tf}(t, d) \cdot \mathrm{idf}(t, D)$$

A high weight in TF-IDF is reached by a high term frequency (in the given document) and a low document frequency of the term in the whole collection of documents; the weights hence tend to filter out common terms. So a term commonly used in a specific document but rarely used in other documents is deemed as improtant. As a term appears in more documents, the ratio inside the logarithm approaches 1, bringing the IDF and TF-IDF closer to 0. In our example, the term *”the”* would have a TF-IDF close to 0, as it appears in nearly every document, *”three”* would be of medium importance, as it is a term appearing relatively often, but probably not in every document, and the term *”musketeers”* would be deemed as high importance as it is a more uncommon word.

3.2.5 Other Vectorizers

While TF-IDF is the most popular vectorizer, there are other notable mentions that could be used, one in particular being the CLFD method and its variants, which provides better results for machine learning methods, compared to the results provided by traditional vectorization techniques for those methods. More specifically, comparing clfd-based vectorization to the next best performing traditional vectorization technique, there is an approximately 1.5% to 2% increase in performance across all metrics. [20]

3.2.6 Cosine Similarity

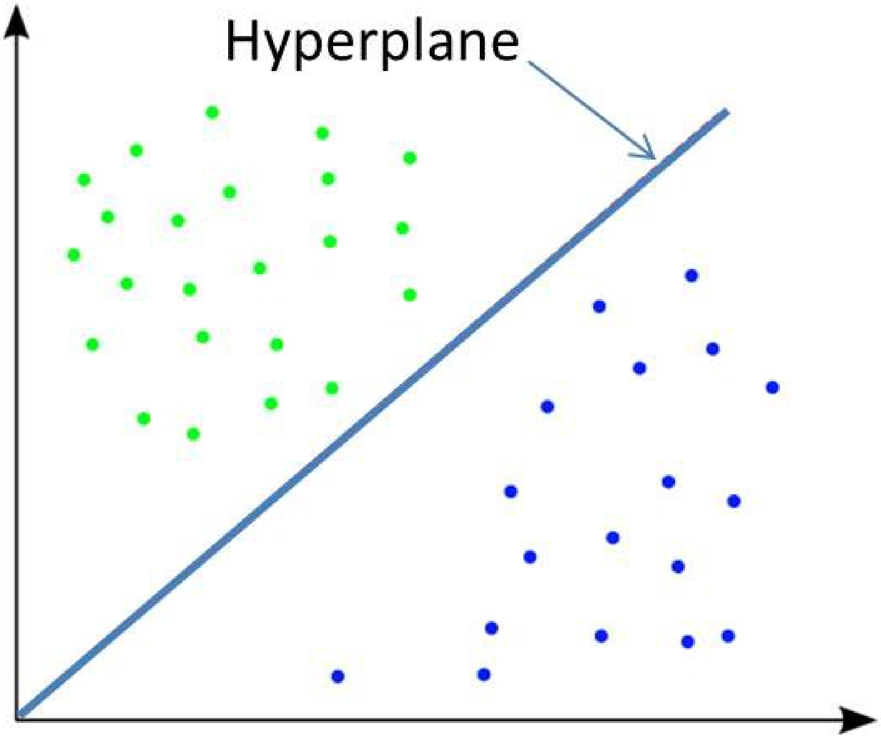
After assigning our data with TF-IDF vectors, we can start matching products to each other in a rough process. To do this, we used cosine similarity as a metric for the degree to which different entities’ TF-IDF vectors match with each other. The match with the highest Cosine Similarity [17] would be considered a match.

3.3 Machine Learning Algorithms

Using TF-IDF we are able to match pairs of entities depending on how similar their feature vectors are, but that similarity does not guarantee that the pair is matched correctly. There might be a mistake, and a different item than the one that should have been matched has been picked, or there might just not be a match between these two grocery stores, as the product might only exist in one of them. To distinguish between true and false matches made by TF-IDF vector similarity, we utilized some machine learning algorithms. In this section we will examine them, and see each one's contributions to the results.

3.3.1 Support Vector Machine

Originally developed by Vladimir Vapnik, Support Vector Machines [14] [9], or SVMs, are supervised learning models that analyze data for classification. In order to train an SVM algorithm we have to give it a dataset of training examples, each belonging to one of two categories. After being trained, the algorithm will attempt classifying any examples given as part of the testing dataset to either of these two classes, making it a non-probabilistic binary linear classifier. To accomplish this, SVM will create a plane and assign each example to a point in space. Its ideal plane of choice, is one that will maximize the length of the distance, called margin width, between the two distinct categories, each taking up part of the space with no common points. In case of multiple features SVMs will create hyperplanes of N dimensions, N being equal to the number of features.



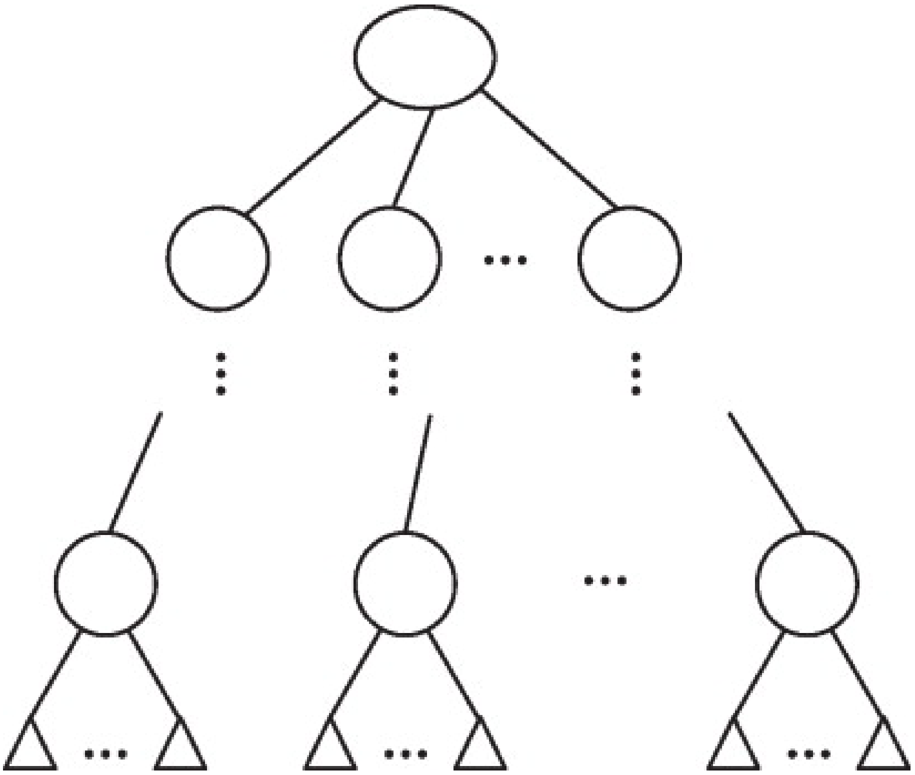
We can see the different classes in distinct colors being seperated by the hyperplane

When given a dataset including training samples, each tagged as belonging to one of two categories, an SVM training algorithm builds a model that assigns new samples included in a testing dataset to one category or the other. SVM maps training examples to points in space so as to maximise the margin width between the two categories by using the hinge loss function. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap splitting the categories they fall.

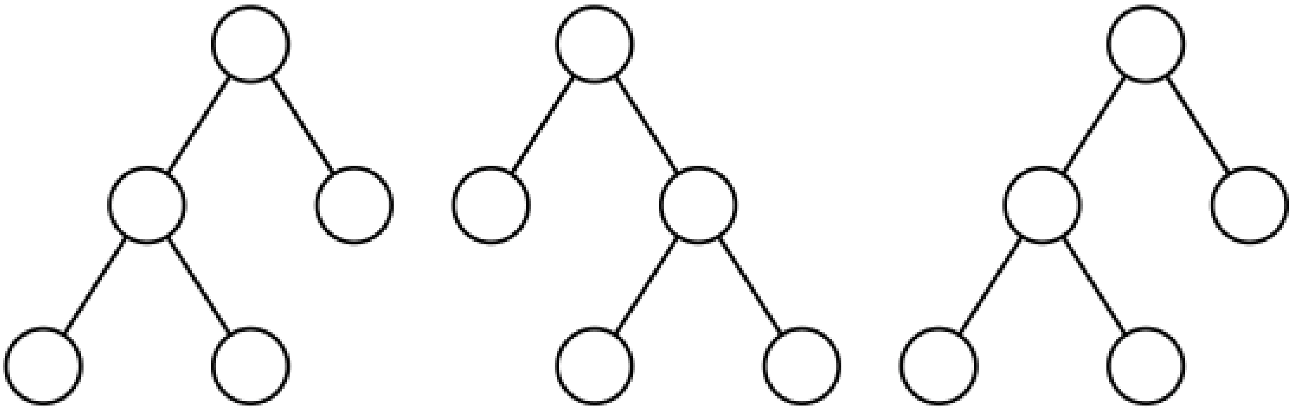
3.3.2 Random Forest

While SVM significantly improved the match results when compared to simply matching entities by their TF-IDF vector similarity, we still didn't have the level of entity resolution that could be used in a market environment. To improve our results even further, we tried using the random forest method.

In a decision tree classifier [24] each node in the tree corresponds to a test on a feature, branching out to the possible values of this feature. Data is continuously tested until it reaches its class, in one of the final nodes of the tree.



A Random Forest [12] [5] then, is a classification method where a multitude of decision trees are created during training, which is why it's categorised as an ensemble learning algorithm. Depending on their depth, decision trees tend to overfit their data, often making them inaccurate in their classifications. Random forest provide a welcome upgrade in the quality of the results, by introducing some additional mechanisms in the training process.



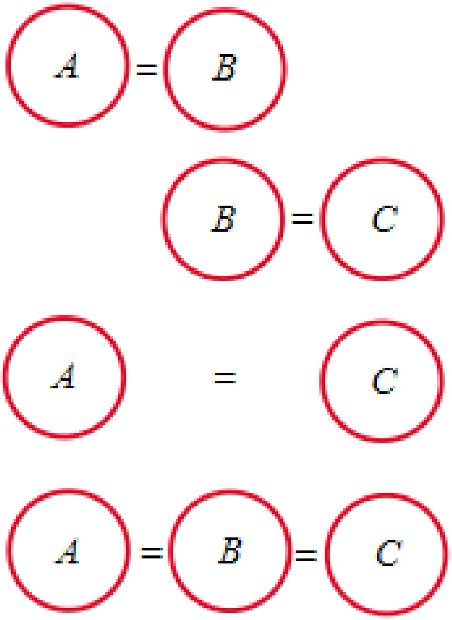
Random forests usually apply Bootstrap Aggregation, or Bagging. Bootstrapping is a test method that uses random sampling with replacement. Without bootstrap, each decision tree would include the whole dataset. With bootstrap, instead of training on the whole dataset, each tree of the forest is trained on a subset of data, called the Bag. Multiple trees are trained on different bags, and later the results from all the trees are aggregated.

In our thesis, we tested random forests with and without bootstrapping to see which version of the algorithm yields the better results.

3.4 Transitive Closure

Because we possess in our database entitites from three different sources, we will have to perform two different matches for each entity, one for each of the two remaining grocery stores, excluding the one the item in question comes from. So assuming an item **A** from the first grocery store, we will have to find its most successful match from the second grocery store, let's call this item **B**, and the most successful match from the third grocery store, **C**. It is then an interesting question to ask ourselves, whether entities **B** and **C** should also match with each other. If $\mathbf{A} = \mathbf{B}$ and $\mathbf{A} = \mathbf{C}$, it is logical to assume that $\mathbf{B} = \mathbf{C}$ as well. On the other hand, if our algorithm decides that **B** and **C** do not match, we can take this as an indication that one or both of the matches with **A** might be false, and use this knowledge to find matches that are more likely to be true.

This property is called Transitive Closure, and is a transitive relation. A relation R on a set X is transitive if, for all A, B, C in X, whenever ARB and BRC then ARC. In our example, the relation would be an equation between matching entities.



We tested this by inserting a requirement to the matching process, that the two matches of our initial entity must match with each other to a certain degree, otherwise we would consider the matches false and try to find the next most likely match.

3.5 Precision and Recall

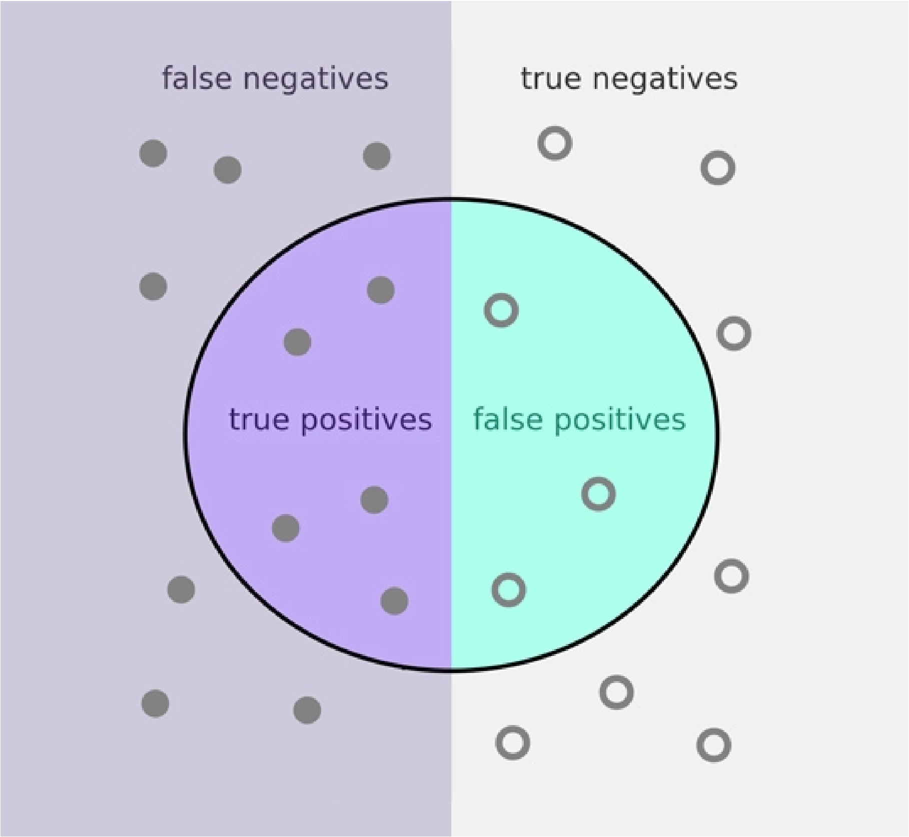
Ideally, we would be able to manually confirm our matches correctness by going over them and tagging them as true or false, then calculating the success percentage. While this was an apporach that we tried during the training stages – we had to be sure about the data we feed our models, after all – the testing part could prove to be extremely time consuming and tedious, as we would repeatedly try different approaches, each with different outputs, and thousands of matches to go over in total. This is why we ultimately chose a different approach, the Precision and Recall method. Precision and Recall are metrics for performance of data classification, and through them, a plethora of other important helpful metrics can be expressed.

3.5.1 Basic Terms

Before we can start delving into the metrics that Precision and Recall offer us, we need to define a few basic terms that will serve as the building blocks for the equations to follow:

- **Positive**: An entity pair that has been estimated to be positive for a match.
- **True Positive**: A positive estimation that has been confirmed as correct.
- **False Positive**: A positive estimation that has been confirmed as incorrect.
- **Negative**: An entity pair that has been estimated to be negative for a match.
- **True Negative**: A negative estimation that has been confirmed as correct.
- **False Negative**: A negative estimation that has been confirmed as incorrect.

A graphic representation can be shown bellow, where the filled in dots on the left half of the image represent the Positives, and the hollow dots on the right represent the Negatives.



3.5.2 Metrics

Now that we understand the basic terminology, we can start to focus on the metrics that the Precision and Recall method offers for performance:

- **Precision or Positive Predicted Value**: Of the sum of matches that have been estimated to be Positive, Precision refers to the subset of matches that are True Positives.

  $$PPV = \frac{TP}{TP + FP}$$

  It is a good indication of what percentage of positives have been successfully classified.

- **Recall or True Positive Rate**: Of all of the matches that should have been classified as positive, recall accounts for the ones that have been correctly classified.

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

  It is a good indication of the success we have in making correct matches at all, and as such is an important metric in this thesis.

- **Negative Predictive Value**: Opposite to Precision, NPV predicts the percentage of negative matches that have been correctly classified.

  $$NPV = \frac{TN}{TN + FN}$$

- **Specificity or True Negative Rate**: Opposite to Recall, TNR is used to calculate our success in classifying negative matches.

  $$TNR = \frac{TN}{TPN + FP}$$

- **Prevalence**: The ratio of Positive matches to the sum of all matches.

$$\frac{P}{P+N}$$

- **Accuracy**: A very important metric as to the degree of success of our classification model, it calculates the accuracy with which data is classified correctly

$$ACC = \frac{TP+TN}{P+N}$$

- **Balanced Accuracy**: In imbalanced sets, accuracy can be misleading. A set of 99 negatives and 1 positive can be classified as all negatives and give an accuracy score of 99%. This is where Balanced Accuracy is prefered.

$$BACC = \frac{TPR+TNR}{2}$$

  In the previous example, TPR would be 0%, while TNR would be 100%, giving us a 50% balanced accuracy.

- **F1-Score**: Also known as F-Score, or F-Measure, is the harmonic mean of Precision and Recall, and is a metric that can be used to evaluate the overall performance of our classifier.

$$F_1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} = 2 \times \frac{PPV \times TPR}{PPV + TPR}$$

There are more metrics that emerge from these values, but the above are enough to get an accurate idea of this thesis' performance.

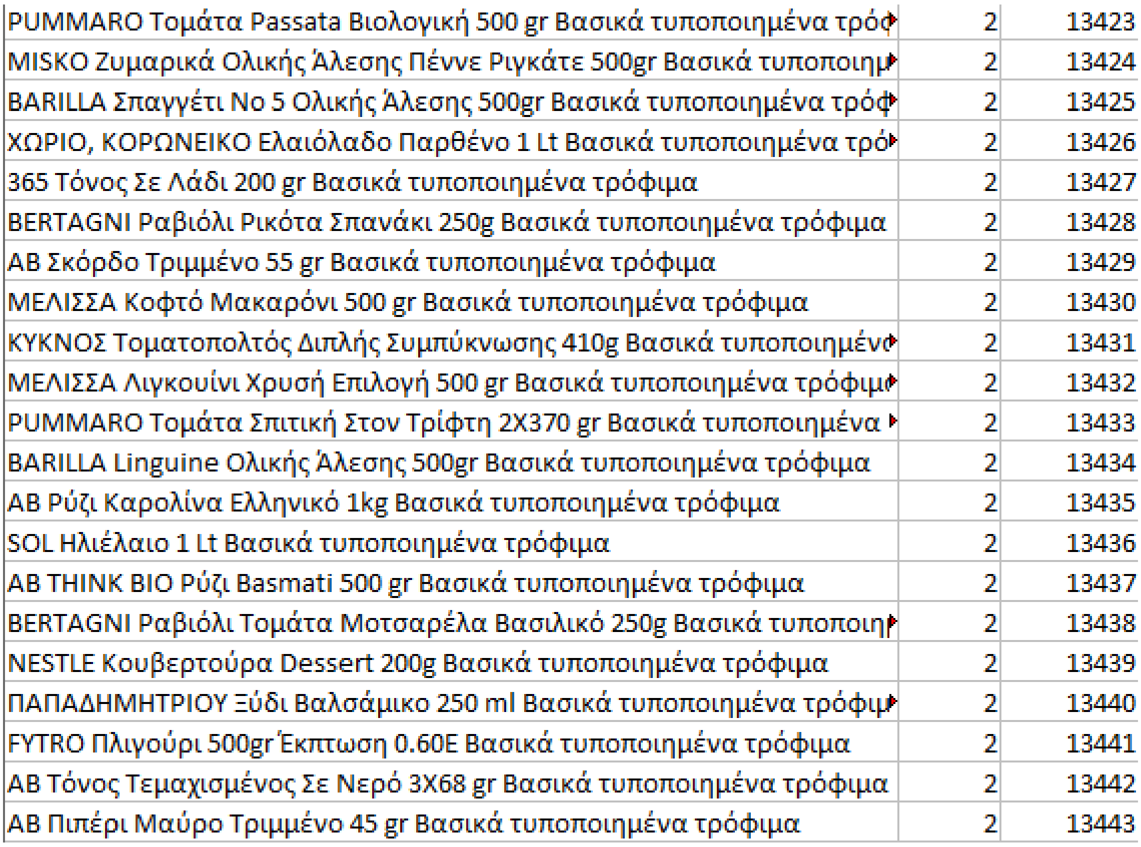
3.6 Dataset Balance

Before proceeding to the part of the chapter where we showcase our baselines for entity resolution, there is some merit in examining one last parameter that will prove to be an important factor in the efficiency of our algorithm.

When it comes to training machine learning models to recognize identical entities, the balance of the training datasets plays a major role. A balanced dataset is one where an equal amount of correct and incorrect matches occur, with the intention of avoiding any biases that could appear in the model. Another school of thought is that an imbalanced training dataset might be more effective, as it will show a more realistic image of what would really happen in a real-life scenario. We approached this by working on a scale of balancing ratios, until we could decide on the one that is optimal.

3.7 Implementations

Now that we have explained the terminology and methods used in this part of the thesis, we can start to showcase the implementations and results of our entity resolution algorithm. To briefly recap the previous chapter, at this point we have in our possession a database of about 25,000 products, each from one of three distinct grocery stores. Along with its description, each entity has been given a unique ID in our database to facilitate searching and a grocery store ID that references its source. We have other information on each product, but for this chapter these are all we are going to need. So for our first step, we edit our database to only include this data. We can find more information about each item later by cross-referencing the ID of each entity in the initial database.



PUMMARO Τομάτα Passata Βιολογική 500 gr Βασικά τυποποιημένα τρόφ	2	13423
MISKO Ζυμαρικά Ολικής Άλεσης Πέννε Ριγκάτε 500gr Βασικά τυποποιημ	2	13424
BARILLA Σπαγγέτι No 5 Ολικής Άλεσης 500gr Βασικά τυποποιημένα τρόφ	2	13425
ΧΩΡΙΟ, ΚΟΡΩΝΕΙΚΟ Ελαιόλαδο Παρθένο 1 Lt Βασικά τυποποιημένα τρό	2	13426
365 Τόνος Σε Λάδι 200 gr Βασικά τυποποιημένα τρόφιμα	2	13427
BERTAGNI Ραβιόλι Ρικότα Σπανάκι 250g Βασικά τυποποιημένα τρόφιμα	2	13428
AB Σκόρδο Τριμμένο 55 gr Βασικά τυποποιημένα τρόφιμα	2	13429
ΜΕΛΙΣΣΑ Κοφτό Μακαρόνι 500 gr Βασικά τυποποιημένα τρόφιμα	2	13430
ΚΥΚΝΟΣ Τοματοπολτός Διπλής Συμπύκνωσης 410g Βασικά τυποποιημένα	2	13431
ΜΕΛΙΣΣΑ Λιγκουίνι Χρυσή Επιλογή 500 gr Βασικά τυποποιημένα τρόφιμ	2	13432
PUMMARO Τομάτα Σπιτική Στον Τρίφτη 2X370 gr Βασικά τυποποιημένα	2	13433
BARILLA Linguine Ολικής Άλεσης 500gr Βασικά τυποποιημένα τρόφιμα	2	13434
AB Ρύζι Καρολίνα Ελληνικό 1kg Βασικά τυποποιημένα τρόφιμα	2	13435
SOL Ηλιέλαιο 1 Lt Βασικά τυποποιημένα τρόφιμα	2	13436
AB THINK BIO Ρύζι Basmati 500 gr Βασικά τυποποιημένα τρόφιμα	2	13437
BERTAGNI Ραβιόλι Τομάτα Μοτσαρέλα Βασιλικό 250g Βασικά τυποποιη	2	13438
NESTLE Κουβερτούρα Dessert 200g Βασικά τυποποιημένα τρόφιμα	2	13439
ΠΑΠΑΔΗΜΗΤΡΙΟΥ Ξύδι Βαλσάμικο 250 ml Βασικά τυποποιημένα τρόφιμ	2	13440
FYTRO Πλιγούρι 500gr Έκπτωση 0.60E Βασικά τυποποιημένα τρόφιμα	2	13441
AB Τόνος Τεμαχισμένος Σε Νερό 3X68 gr Βασικά τυποποιημένα τρόφιμα	2	13442
AB Πιπέρι Μαύρο Τριμμένο 45 gr Βασικά τυποποιημένα τρόφιμα	2	13443

In the example above, the columns represent from left to right:

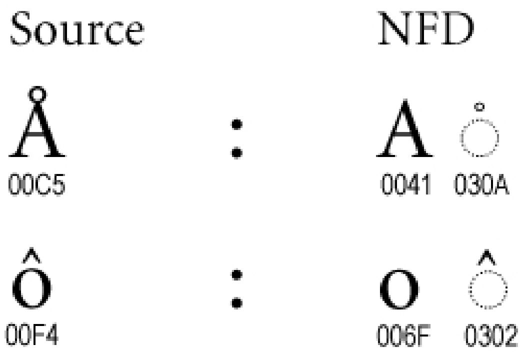
1. Product name and description

2. Source Grocery Store ID, with 1 being Sklavenitis, 2 being AB Vasilopoulos, and 3 being Chalkiadakis

3. the Unique entity ID

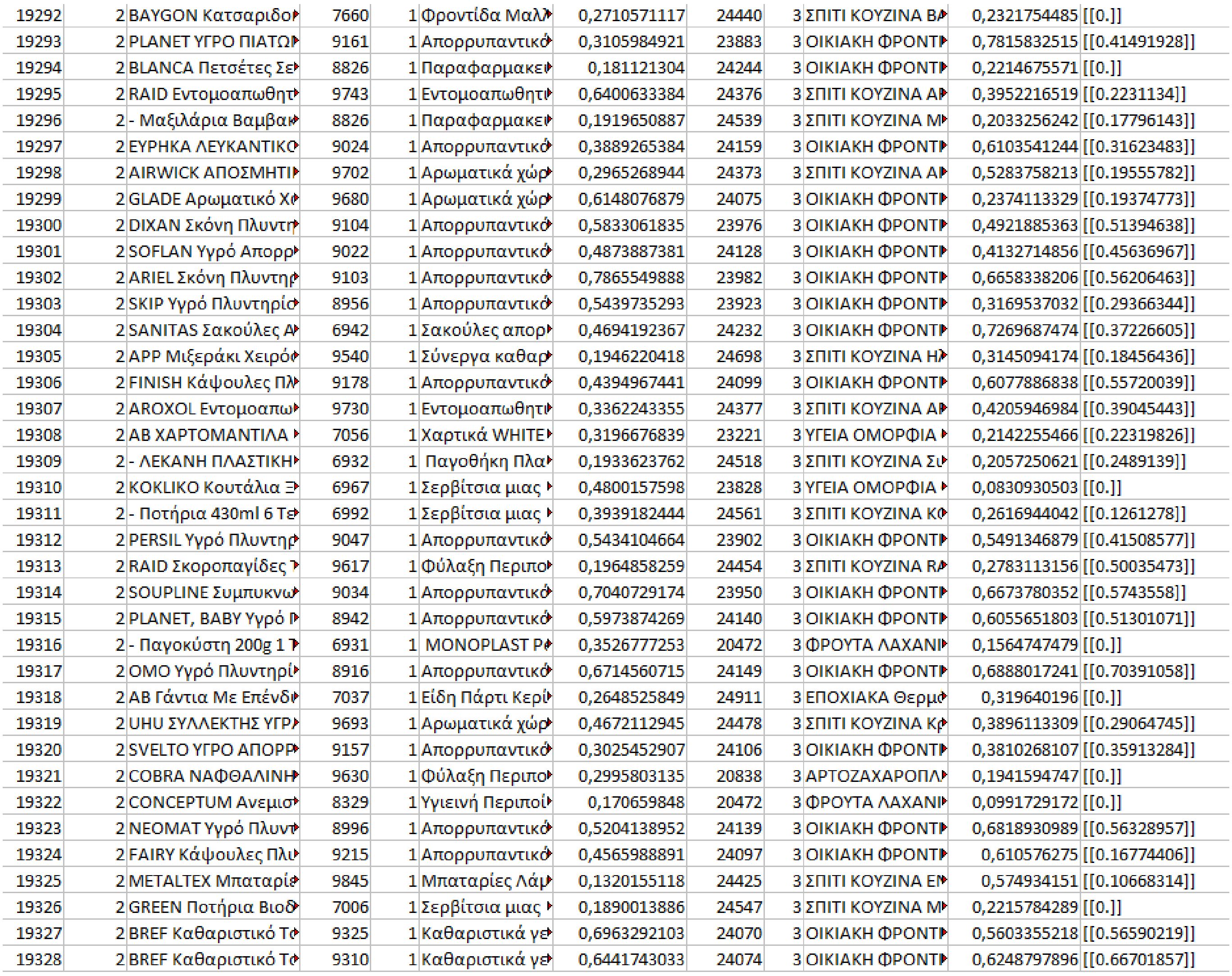
3.7.1 Baseline 1

In the first iteration of our algorithm, we focused on achieving the basic requirements of entity resolution. We started by applying a TF-IDF algorithm on the product descriptions. To do this, we first had to transform our descriptions in a compatible form. We made each description consist only of lower case letters, and used Canonical Decomposition, to adjust for the fact that the original descriptions were in Greek. Canonical Decomposition, or NFD for short, is a way of decomposing letters that are not in the default English language into smaller parts with the intention of composing them again later. An example can be shown in the image below.



After this, we created a bag of words, and split each description into words by using the space character ' ' as an indication of where each new word starts, then added the words into the bag. When we had placed all of our words in the bag, we created a TF-IDF vector for each description, and assigned it two variables with the intention of giving them cosine similarity values of 0 to 1, 0 being a 0% match, and 1 being a 100% match, and initialized them at 0. Then, for each vector we accessed every other vector in the whole list that wasn't from the same grocery store. In every loop we would check if the cosine similarity of the two vectors was higher than the assigned value of the corresponding grocery store, which at first was 0, but would then be replaced with the first positive value. When we found a match with higher similarity than the one we had, we would keep it in a temporary array, along with its other information.

After going through the whole database, for each vector, we would have zero to two matches for each entity, depending on whether we found matching products on the other grocery stores. In the case there were zero matches, it was estimated by the algorithm that there were no other similar products in the other two grocery stores, and in the case there were two matches, we had found similar products in both other sources. In this case, we would also calculate those products' relative similarity, in order to have an idea of how well transitive closure would be accounted for with this approach. An example of our new dataset can be seen below.



19292	2	BAYGON Κατσαριδο	7660	1	Φροντίδα Μαλλ	0,2710571117	24440	3	ΣΠΙΤΙ ΚΟΥΖΙΝΑ ΒΑ	0,2321754485	[[0.]]
19293	2	PLANET ΥΓΡΟ ΠΙΑΤΩ	9161	1	Απορρυπαντικό	0,3105984921	23883	3	ΟΙΚΙΑΚΗ ΦΡΟΝΤΙ	0,7815832515	[[0.41491928]]
19294	2	BLANCA Πετσέτες Σε	8826	1	Παραφαρμακει	0,181121304	24244	3	ΟΙΚΙΑΚΗ ΦΡΟΝΤΙ	0,2214675571	[[0.]]
19295	2	RAID Εντομοαπωθητ	9743	1	Εντομοαπωθητι	0,6400633384	24376	3	ΣΠΙΤΙ ΚΟΥΖΙΝΑ ΑΙ	0,3952216519	[[0.2231134]]
19296	2	- Μαξιλάρια Βαμβακ	8826	1	Παραφαρμακει	0,1919650887	24539	3	ΣΠΙΤΙ ΚΟΥΖΙΝΑ Μ	0,2033256242	[[0.17796143]]
19297	2	ΕΥΡΗΚΑ ΛΕΥΚΑΝΤΙΚΟ	9024	1	Απορρυπαντικό	0,3889265384	24159	3	ΟΙΚΙΑΚΗ ΦΡΟΝΤΙ	0,6103541244	[[0.31623483]]
19298	2	AIRWICK ΑΠΟΣΜΗΤΙ	9702	1	Αρωματικά χώρ	0,2965268944	24373	3	ΣΠΙΤΙ ΚΟΥΖΙΝΑ ΑΙ	0,5283758213	[[0.19555782]]
19299	2	GLADE Αρωματικό Χ	9680	1	Αρωματικά χώρ	0,6148076879	24075	3	ΟΙΚΙΑΚΗ ΦΡΟΝΤΙ	0,2374113329	[[0.19374773]]
19300	2	DIXAN Σκόνη Πλυντη	9104	1	Απορρυπαντικό	0,5833061835	23976	3	ΟΙΚΙΑΚΗ ΦΡΟΝΤΙ	0,4921885363	[[0.51394638]]
19301	2	SOFLAN Υγρό Απορρ	9022	1	Απορρυπαντικό	0,4873887381	24128	3	ΟΙΚΙΑΚΗ ΦΡΟΝΤΙ	0,4132714856	[[0.45636967]]
19302	2	ARIEL Σκόνη Πλυντηρ	9103	1	Απορρυπαντικό	0,7865549888	23982	3	ΟΙΚΙΑΚΗ ΦΡΟΝΤΙ	0,6658338206	[[0.56206463]]
19303	2	SKIP Υγρό Πλυντηρίο	8956	1	Απορρυπαντικό	0,5439735293	23923	3	ΟΙΚΙΑΚΗ ΦΡΟΝΤΙ	0,3169537032	[[0.29366344]]
19304	2	SANITAS Σακούλες Α	6942	1	Σακούλες απορ	0,4694192367	24232	3	ΟΙΚΙΑΚΗ ΦΡΟΝΤΙ	0,7269687474	[[0.37226605]]
19305	2	APP Μιξεράκι Χειρό	9540	1	Σύνεργα καθαρ	0,1946220418	24698	3	ΣΠΙΤΙ ΚΟΥΖΙΝΑ Η	0,3145094174	[[0.18456436]]
19306	2	FINISH Κάψουλες Πλ	9178	1	Απορρυπαντικό	0,4394967441	24099	3	ΟΙΚΙΑΚΗ ΦΡΟΝΤΙ	0,6077886838	[[0.55720039]]
19307	2	AROXOL Εντομοαπω	9730	1	Εντομοαπωθητι	0,3362243355	24377	3	ΣΠΙΤΙ ΚΟΥΖΙΝΑ ΑΙ	0,4205946984	[[0.39045443]]
19308	2	AB ΧΑΡΤΟΜΑΝΤΙΛΑ	7056	1	Χαρτικά WHITE	0,3196676839	23221	3	ΥΓΕΙΑ ΟΜΟΡΦΙΑ	0,2142255466	[[0.22319826]]
19309	2	- ΛΕΚΑΝΗ ΠΛΑΣΤΙΚΗ	6932	1	Παγοθήκη Πλα	0,1933623762	24518	3	ΣΠΙΤΙ ΚΟΥΖΙΝΑ Συ	0,2057250621	[[0.2489139]]
19310	2	KOKLIKO Κουτάλια Ξ	6967	1	Σερβίτσια μιας	0,4800157598	23828	3	ΥΓΕΙΑ ΟΜΟΡΦΙΑ	0,0830930503	[[0.]]
19311	2	- Ποτήρια 430ml 6 Τε	6992	1	Σερβίτσια μιας	0,3939182444	24561	3	ΣΠΙΤΙ ΚΟΥΖΙΝΑ Κ	0,2616944042	[[0.1261278]]
19312	2	PERSIL Υγρό Πλυντηρ	9047	1	Απορρυπαντικό	0,5434104664	23902	3	ΟΙΚΙΑΚΗ ΦΡΟΝΤΙ	0,5491346879	[[0.41508577]]
19313	2	RAID Σκοροπαγίδες Τ	9617	1	Φύλαξη Περιπο	0,1964858259	24454	3	ΣΠΙΤΙ ΚΟΥΖΙΝΑ RA	0,2783113156	[[0.50035473]]
19314	2	SOUPLINE Συμπυκνω	9034	1	Απορρυπαντικό	0,7040729174	23950	3	ΟΙΚΙΑΚΗ ΦΡΟΝΤΙ	0,6673780352	[[0.5743558]]
19315	2	PLANET, BABY Υγρό Ι	8942	1	Απορρυπαντικό	0,5973874269	24140	3	ΟΙΚΙΑΚΗ ΦΡΟΝΤΙ	0,6055651803	[[0.51301071]]
19316	2	- Παγοκύστη 200g 1 Τ	6931	1	MONOPLAST Ρ	0,3526777253	20472	3	ΦΡΟΥΤΑ ΛΑΧΑΝΙ	0,1564747479	[[0.]]
19317	2	OMO Υγρό Πλυντηρί	8916	1	Απορρυπαντικό	0,6714560715	24149	3	ΟΙΚΙΑΚΗ ΦΡΟΝΤΙ	0,6888017241	[[0.70391058]]
19318	2	AB Γάντια Με Επένδι	7037	1	Είδη Πάρτι Κερί	0,2648525849	24911	3	ΕΠΟΧΙΑΚΑ Θερμό	0,319640196	[[0.]]
19319	2	UHU ΣΥΛΛΕΚΤΗΣ ΥΓΡ	9693	1	Αρωματικά χώρ	0,4672112945	24478	3	ΣΠΙΤΙ ΚΟΥΖΙΝΑ Κρ	0,3896113309	[[0.29064745]]
19320	2	SVELTO ΥΓΡΟ ΑΠΟΡΡ	9157	1	Απορρυπαντικό	0,3025452907	24106	3	ΟΙΚΙΑΚΗ ΦΡΟΝΤΙ	0,3810268107	[[0.35913284]]
19321	2	COBRA ΝΑΦΘΑΛΙΝΗ	9630	1	Φύλαξη Περιπο	0,2995803135	20838	3	ΑΡΤΟΖΑΧΑΡΟΠΛ	0,1941594747	[[0.]]
19322	2	CONCEPTUM Ανεμιστ	8329	1	Υγιεινή Περιποί	0,170659848	20472	3	ΦΡΟΥΤΑ ΛΑΧΑΝΙ	0,0991729172	[[0.]]
19323	2	NEOMAT Υγρό Πλυντ	8996	1	Απορρυπαντικό	0,5204138952	24139	3	ΟΙΚΙΑΚΗ ΦΡΟΝΤΙ	0,6818930989	[[0.56328957]]
19324	2	FAIRY Κάψουλες Πλυ	9215	1	Απορρυπαντικό	0,4565988891	24097	3	ΟΙΚΙΑΚΗ ΦΡΟΝΤΙ	0,610576275	[[0.16774406]]
19325	2	METALTEX Μπαταρίε	9845	1	Μπαταρίες Λάμ	0,1320155118	24425	3	ΣΠΙΤΙ ΚΟΥΖΙΝΑ ΕΙ	0,574934151	[[0.10668314]]
19326	2	GREEN Ποτήρια Βιοδ	7006	1	Σερβίτσια μιας	0,1890013886	24547	3	ΣΠΙΤΙ ΚΟΥΖΙΝΑ Μ	0,2215784289	[[0.]]
19327	2	BREF Καθαριστικό Το	9325	1	Καθαριστικά γε	0,6963292103	24070	3	ΟΙΚΙΑΚΗ ΦΡΟΝΤΙ	0,5603355218	[[0.56590219]]
19328	2	BREF Καθαριστικό Το	9310	1	Καθαριστικά γε	0,6441743033	24074	3	ΟΙΚΙΑΚΗ ΦΡΟΝΤΙ	0,6248797896	[[0.66701857]]

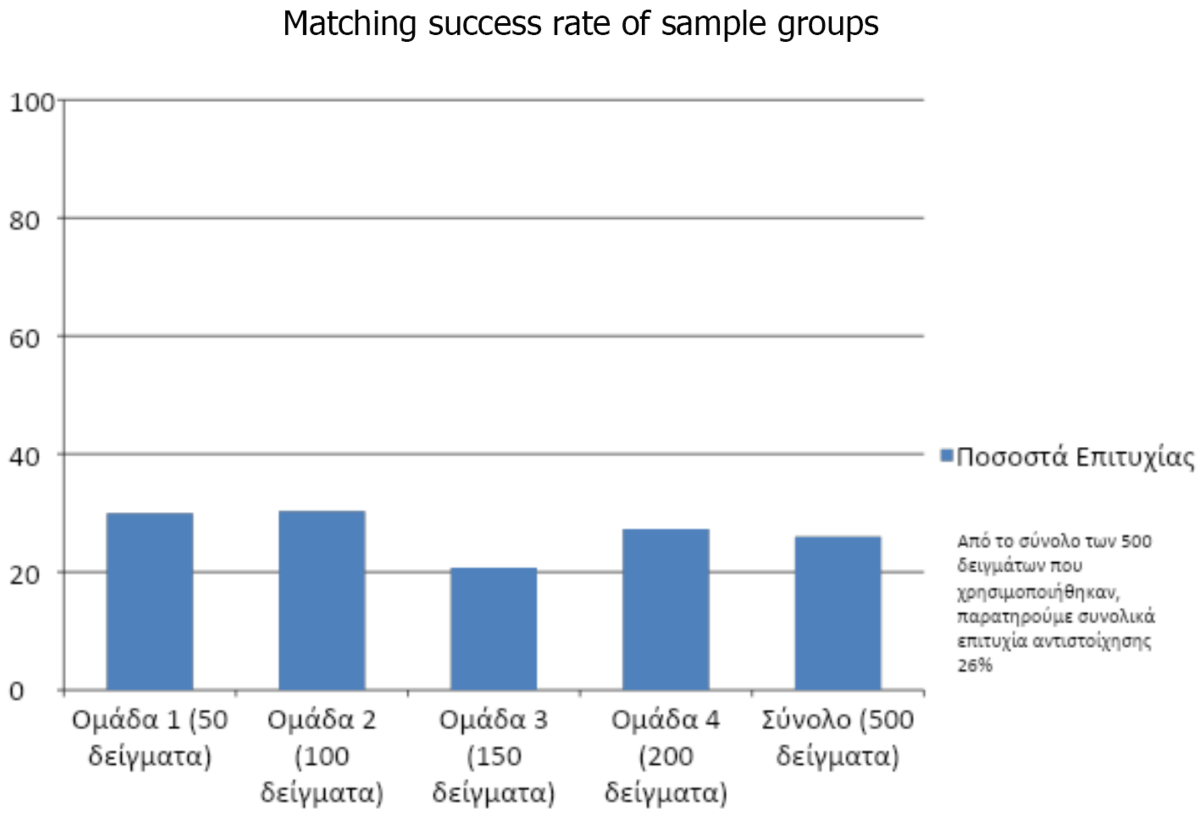
Where assuming the initial entity is A, the match from one of the other stores is B and the match from the last store is C, each column represents in order of left to right:

1. the Unique entity ID of A
2. grocery store ID of A
3. Product name and description of A
4. the Unique entity ID of B
5. grocery store ID of B
6. Product name and description of B
7. Cosine Similarity of A and B
8. the Unique entity ID of C
9. grocery store ID of C
10. Product name and description of C
11. Cosine Similarity of A and C
12. Cosine Similarity of B and C

When examining our new dataset we can see that some matches have really good cosine similarity, some have really bad one, and the cases where no matches are found are particularly rare. In general, the results are very inconsistent. While some good matches are made, they are lost in a sea of irrelevant data that makes this first iteration not able to be used in a commercial environment, as the user will only want to see relevant matches.

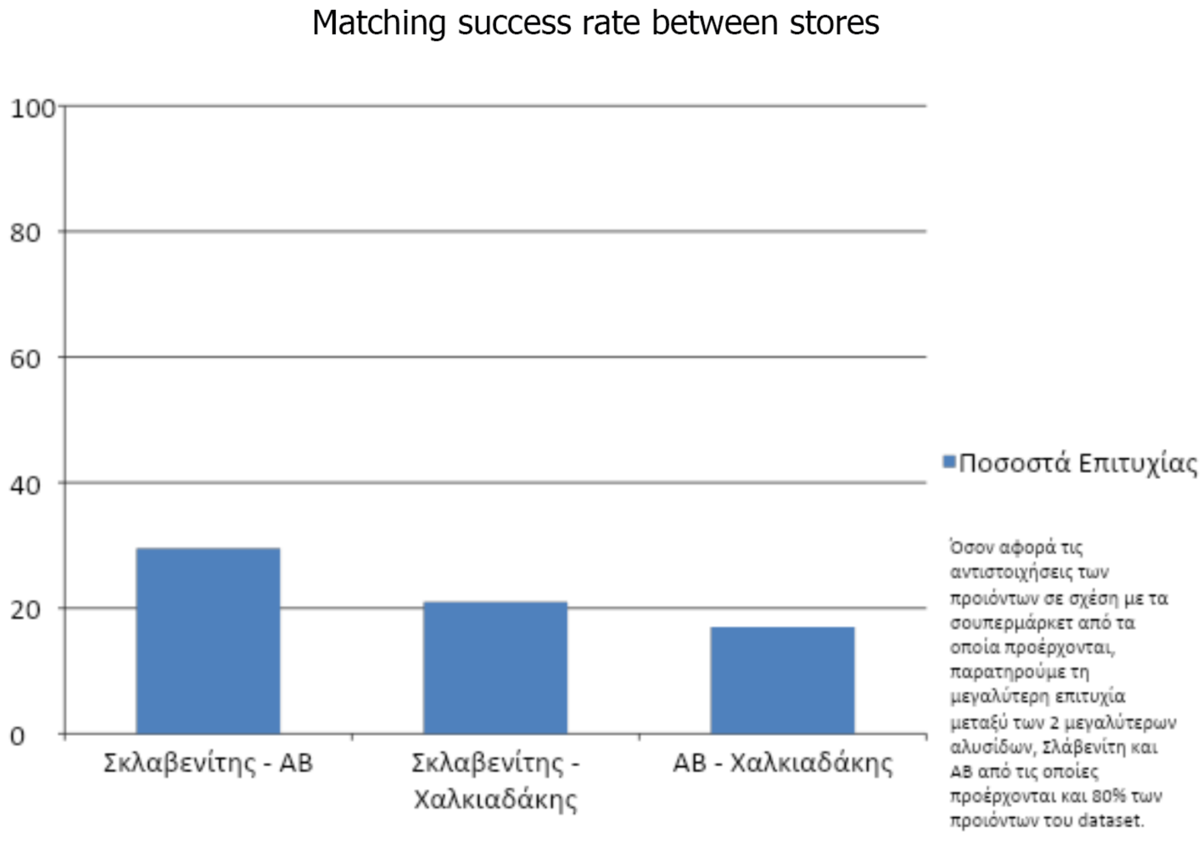
Before proceeding to the next step, we took a more careful look into this dataset in order to confirm the degree of success that the TF-IDF and cosine similarity method had, as well as maybe take notice of any interesting facts that come up. To do this, we created four groups of randomly chosen samples, with each sample containing a random entity and its matches from the previous process. The groups differed in size and consisted of 50, 100, 150, and 200 samples, making up for a total of 500 samples, or 1000 matches. We manually went over each match and tagged it as True, in case the match was correct, or False, in case it wasn't.

After tagging all of the randomly selected samples, we run a calculation on the tags we manually input and found these results:



Matching success rate of sample groups in increasing order of size, ending with the sum of all groups

The graph above shows us that the sample groups had matching success of varying degrees, from 20%, up to 31%. When put together, all 500 samples had an average matching success of 26%, far too low for it to be considered reliable. Another interesting statistic that came up through this process was the correlation between the degree of matching success and the sources of the entities.



Όσον αφορά τις αντιστοιχήσεις των προιόντων σε σχέση με τα σουπερμάρκετ από τα οποία προέρχονται, παρατηρούμε τη μεγαλύτερη επιτυχία μεταξύ των 2 μεγαλύτερων αλυσίδων, Σλάβενίτη και AB από τις οποίες προέρχονται και 80% των προιόντων του dataset.

Matching success rates between Sklavenitis-AB, Sklavenitis Chalkiadakis, AB-Chalkiadakis

A shown above, matches between Sklavenitis and AB Vasilopoulos, where 80% of our total dataset originates, were considerably more successful than matches of either of those grocery stores with Chalkiadakis. We attribute this to the fact that these two bigger grocery stores have clearer and more machine-readable descriptions for their products.

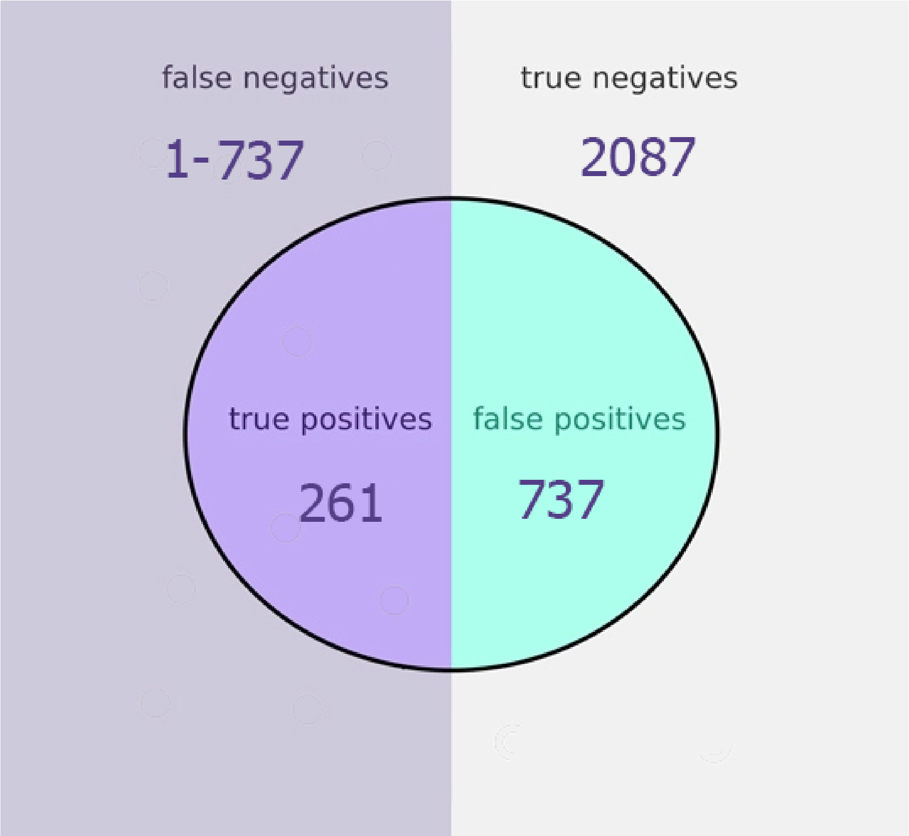
Because manually verifying whether each of the matches is correct or not would become extremely time consuming as we continued testing more and more methods, some of them multiple times, we decided to create an ”artificial” dataset, which would comprise of 3086 random matches, making up 7172 entities. These entities were made up from the original 500 checked samples, about 500 matches of some of the falsely matched products with 9 other candidates that weren't picked in the first phase that had the highest cosine similarity, and an additional set of samples that we'll address in the next paragraphs.

After manually tagging each sample match, we ended up with 262 true matches, and 2824 false matches. This dataset would allow us to have a common testing ground for all following tests made with different methods and algorithms, and have comparable results, while also saving us the time of manually going over each one individually. To automate the process of testing, we used the precision and recall method. When an algorithm makes a match between those items, we will know whether this match is True or False, and classify it in one of 4 categories: True Positive, False Positive, True Negative and False Negative.

There are some caveats in this method that should be taken into account. Mainly, while we are aware that there are 262 true matches, meaning that there are 524 items with confirmed matches in the dataset, the rest would not be confirmed true negatives. While we know that the match they were part of wasn't correct, there is no guarantee that there isn't another item in the dataset that would have matched correctly. So when a match is made that wasn't in the original dataset, we would only be able to classify it as true negative if we know that one of its items was part of a true positive match, as an item can only match with one other product from each store correctly. In any other case, we had to assume that a negative match was either a true negative, or a false negative. And a positive match that wasn't part of the original true negatives, was either false positive or true positive. This would normally put us in a position where we would be aware of true positive values but all the others would be considered relatively unknown.

To account for this, we inserted 2000 confirmed true negative matches. We did this by using our true positive pairs, and because each entity can only match with one other entity from a separate store, we can assume that if we know what that match is, then all other matches with those entities within those two stores must be wrong. What this allowed us to do, was have a baseline for true positive and true negative matches, being 262 and 2000 each, with the other 737 matches being either false or true negatives.

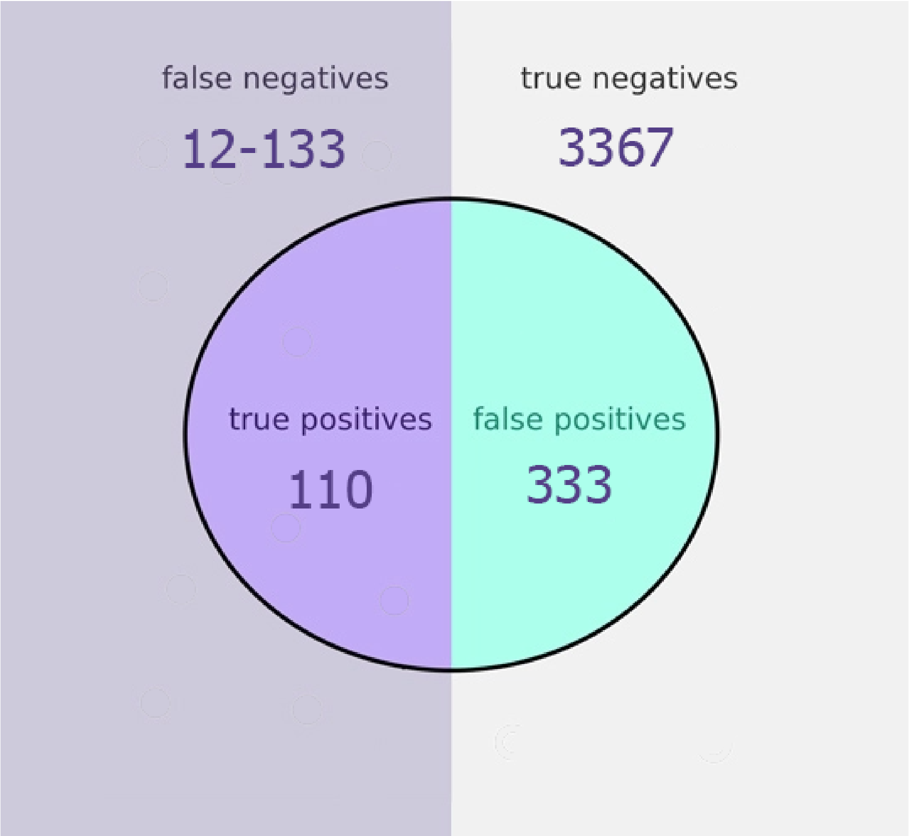
An example of what a classification with this new method would look like for the simple TF-IDF algorithm is shown in the image below.



Where the algorithm correctly recognised all but one the true positive samples and matched them, didn't find matches for 2087 of the true negative samples, and found matches for 737 of the entities that weren't correct. We then assume that the number of false negatives is between 1 (the one true postiive sample that the algorithm didn't recognise) and 737, the number of false positives, because of how the wrongly matched false positives might have been correctly matched otherwise.

The next thing we wanted to introduce to the algorithm was the transitive closure apporach. The idea was to use the transitive properties in a way that would help us improve the quality of the matches. If we are examining product A, and if it can only find one match through cosine similarity, B, then if we can find a match for product B from the grocery store that's left, through transitive property that product, let's call it C, should also match with A. If, however, A finds matches with both stores with items B and C then we check which match has the higher cosine similarity, if it's B, we find its match from the store that's left, let's say item C2, and use that product as a match with A as well. Again, because of the transitive property if A=B and B=C2, then A=C2.

Unfortunately, this apporach proved to be inefficient compared to its counterpart, and reduced the quality of the results as shown below, so we did not examine it further.



We also had an additional 105 samples that we weren't able to categorize correctly because of how vast the initial dataset that we had to work with was. Still, it was clear that transitive closure provided a downgrade.

3.7.2 Baseline 2

In the latter part of our entity resolution algorithm we mainly experimented with machine learning in an attempt to see how it would affect the results from the TF-IDF and cosine similarity matching. We will still use TF-IDF to create vectors of term importance, but using the new artificial set we have created we will train and test machine learning algorithms. When it comes to the TF-IDF algorithm, we will still use a fit transform function on the initial dataset that includes all 25,000 entities, but now we will use a transform function for the testing dataset, which comprises of a subset of the artificial dataset, in order to assign predefined weights to the terms.

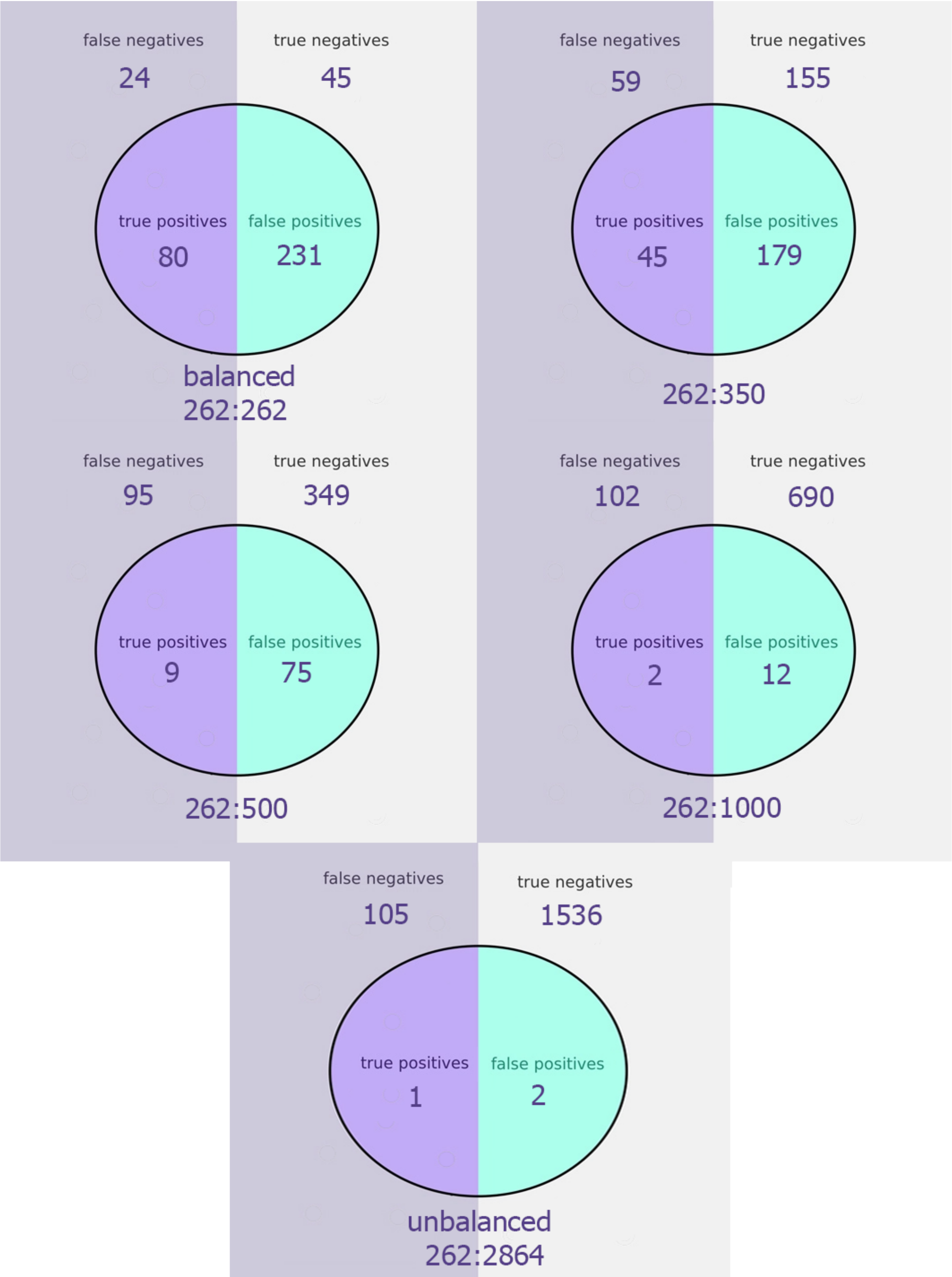
The first machine learning model we examined was an One vs. One SVM model, but we then followed it up with a random forest model, as well as a random forest model without bootstrap.

Training these models provided an interesting challenge in the form of the data we feed it. A training function has two parameters, a vector of features of the data, and the label we classify it as. When we train a model with multiple samples, we must provide it with lists of equal length in both of those parameters. In our case, the labels would be a list of our tags, those being either true or false. The model would then encode them as a binary decision, eg. 0 for True and 1 for False. The features however were a bit more complex. Normally, we feed the sample parameter with a feature vector, but in our case we possess a pair of entities as a match. Because adding an additional parameter for the second entity would prove problematic, we used a different approach. By concatenating the two feature vectors of the matched items, we now had a singular vector that we would be able to feed into the model for training.

Another factor we had to take into account was the ratio of the sizes if the training set and the testing set. We decided to use a 20/80 split so that the testing set was still sizeable enough to provide realistic results. The samples were split in a way that this ratio was also applied to both the classes, so for example our 262 confirmed True matches were split into 2 groups of 52 and 210 for training and testing respectively.

The last thing we tested was the most efficient ratio of true to false matches used in the training process. In a balanced iteration we used 262 true and 262 false matches, so as to avoid over fitting. On the opposite side, when going for an unbalanced approach we used the whole dataset, that being 262 true matches and 2864 false matches. We experimented with various values in between in an attempt to demonstrate a trend in this ratio's optimacy.

Our first test was an SVM model that was trained with a subset of the artificial test and then tested on a subset of the remaining artificial test. These two subsets were always distinct so as to avoid bias.


false negatives
24
true negatives
45
true positives
80
false positives
231
balanced
262:262
false negatives
59
true negatives
155
true positives
45
false positives
179
262:350
false negatives
95
true negatives
349
true positives
9
false positives
75
262:500
false negatives
102
true negatives
690
true positives
2
false positives
12
262:1000
false negatives
105
true negatives
1536
true positives
1
false positives
2
unbalanced
262:2864

- Precision:

    1. 262:262 : 0.26
    2. 262:350 : 0.20
    3. 262:500 : 0.10
    4. 262:1000 : 0.14
    5. 262:2864 : 0.33

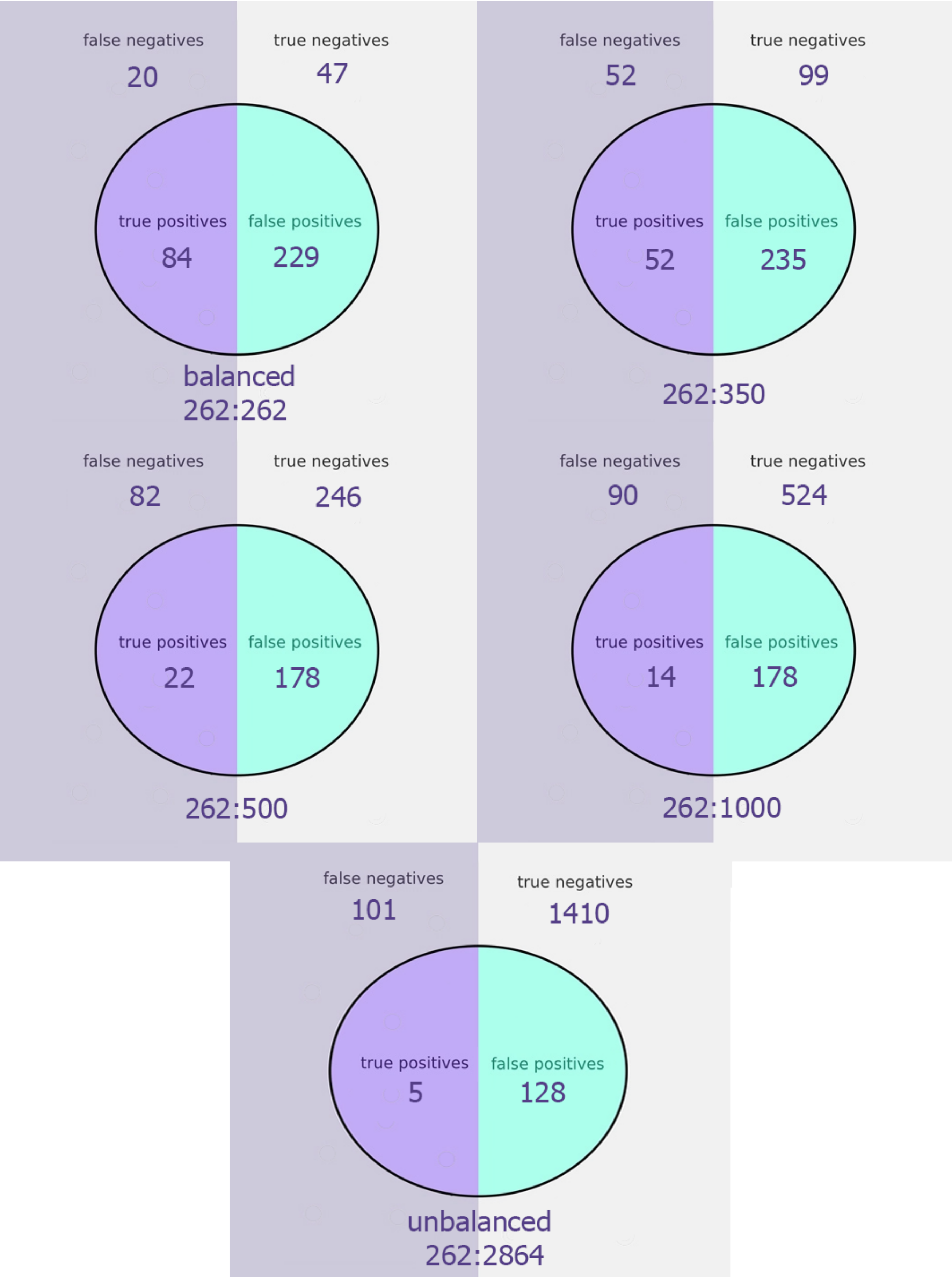
- Recall:

    1. 262:262 : 0.77
    2. 262:350 : 0.43
    3. 262:500 : 0.09
    4. 262:1000 : 0.02
    5. 262:2864 : 0.01

- F1-Score:

    1. 262:262 : 0.39
    2. 262:350 : 0.27
    3. 262:500 : 0.09
    4. 262:1000 : 0.04
    5. 262:2864 : 0.02

Then we tested a Random Forest model using the same training and testing datasets.


false negatives
20
true negatives
47
true positives
84
false positives
229
balanced
262:262
false negatives
52
true negatives
99
true positives
52
false positives
235
262:350
false negatives
82
true negatives
246
true positives
22
false positives
178
262:500
false negatives
90
true negatives
524
true positives
14
false positives
178
262:1000
false negatives
101
true negatives
1410
true positives
5
false positives
128
unbalanced
262:2864

- Precision:

    1. 262:262 : 0.27
    2. 262:350 : 0.18
    3. 262:500 : 0.11
    4. 262:1000 : 0.07
    5. 262:2864 : 0.03

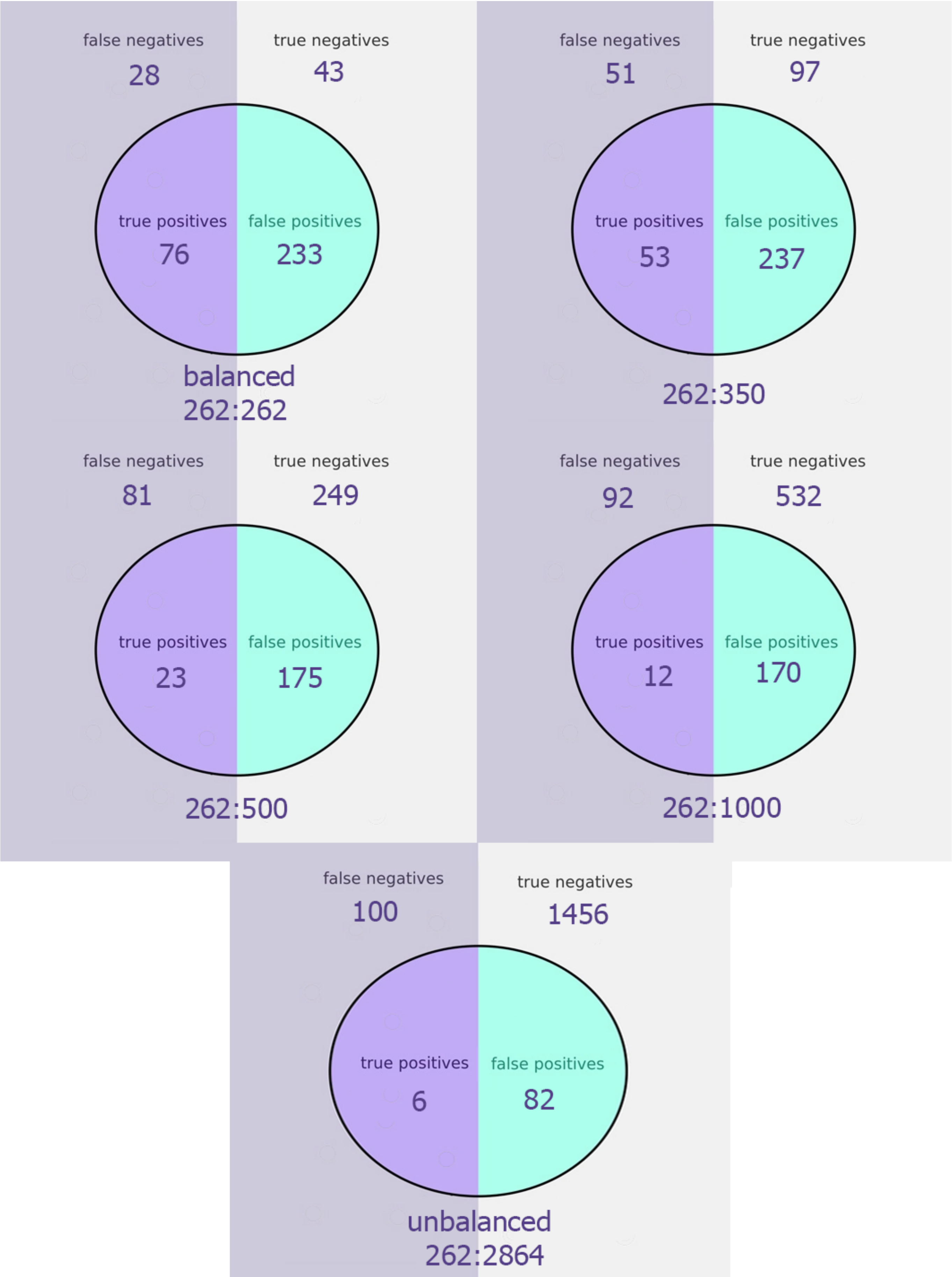
- Recall:

    1. 262:262 : 0.80
    2. 262:350 : 0.50
    3. 262:500 : 0.21
    4. 262:1000 : 0.13
    5. 262:2864 : 0.04

- F1-Score:

    1. 262:262 : 0.40
    2. 262:350 : 0.26
    3. 262:500 : 0.14
    4. 262:1000 : 0.09
    5. 262:2864 : 0.03

And lastly we modified that Random Forest model to not include the bootstrap mechanic.


false negatives
28
true negatives
43
true positives
76
false positives
233
balanced
262:262
false negatives
51
true negatives
97
true positives
53
false positives
237
262:350
false negatives
81
true negatives
249
true positives
23
false positives
175
262:500
false negatives
92
true negatives
532
true positives
12
false positives
170
262:1000
false negatives
100
true negatives
1456
true positives
6
false positives
82
unbalanced
262:2864

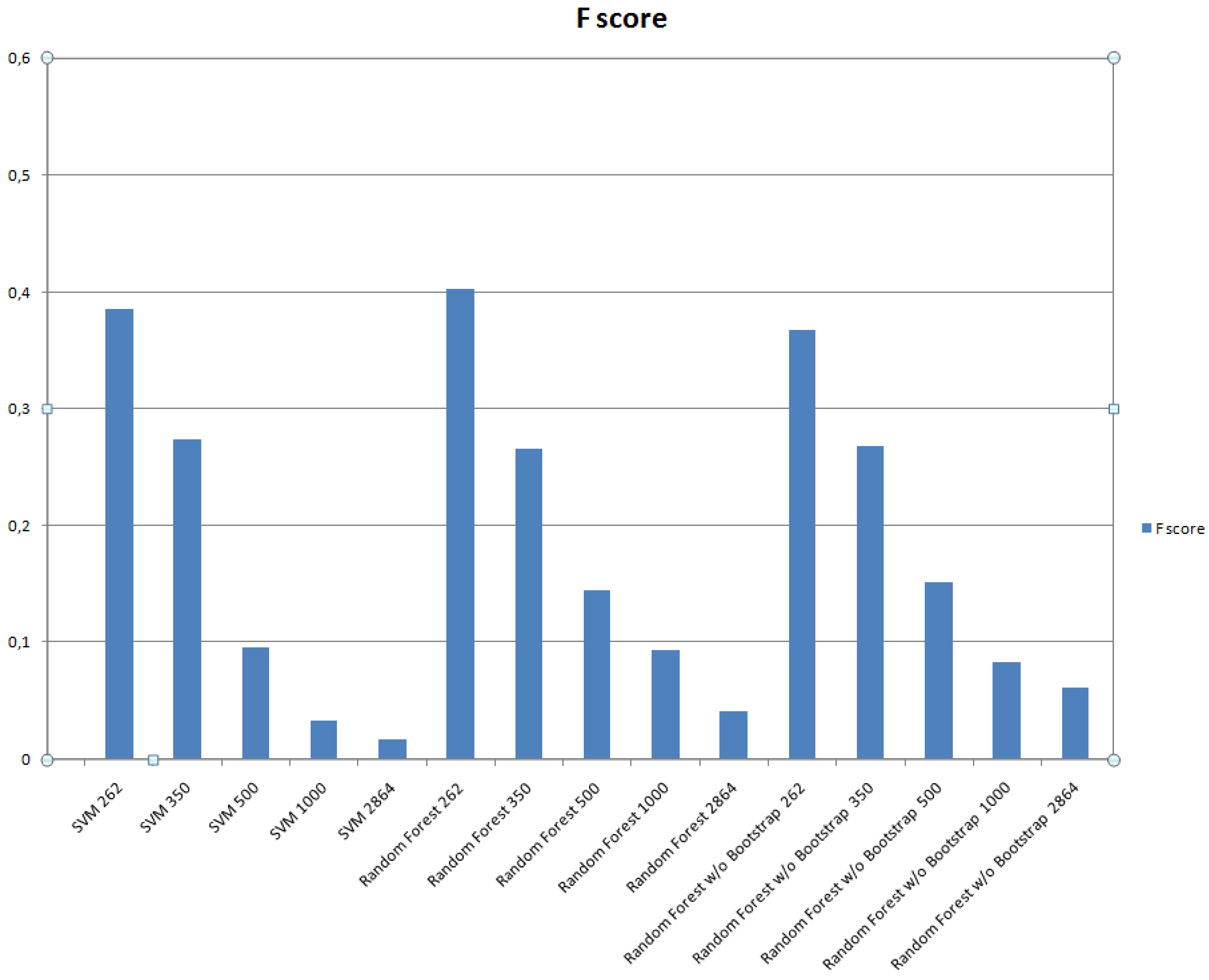
- Precision:
  1. 262:262 : 0.25
  2. 262:350 : 0.18
  3. 262:500 : 0.11
  4. 262:1000 : 0.06
  5. 262:2864 : 0.07
- Recall:
  1. 262:262 : 0.73
  2. 262:350 : 0.51
  3. 262:500 : 0.22
  4. 262:1000 : 0.12
  5. 262:2864 : 0.06
- F1-Score:
  1. 262:262 : 0.37
  2. 262:350 : 0.27
  3. 262:500 : 0.15
  4. 262:1000 : 0.08
  5. 262:2864 : 0.06

From the statistics and graphs above we see some trends:

1. The more balanced the training set, the better the model becomes at recognizing matches, but the worse it becomes at recognizing non matches
2. The more unbalanced the training set, the better it becomes at recognizing non matches, but the worse it becomes in recognizing matches because of overfitting

3. Balanced datasets tend to do better at F1-Score values than unbalanced datasets
4. Random Forest with Bootstrap fares a little better than its counterparts when a balanced dataset is used for training
5. Random Forest without Bootstrap is the worst of the three in its F1-Score when a balanced dataset is used, but maintains the relative best F1-Score when an unbalanced dataset is used

If we were to put all of these models in a single graph that compares their F1-score, it would look like the following image:



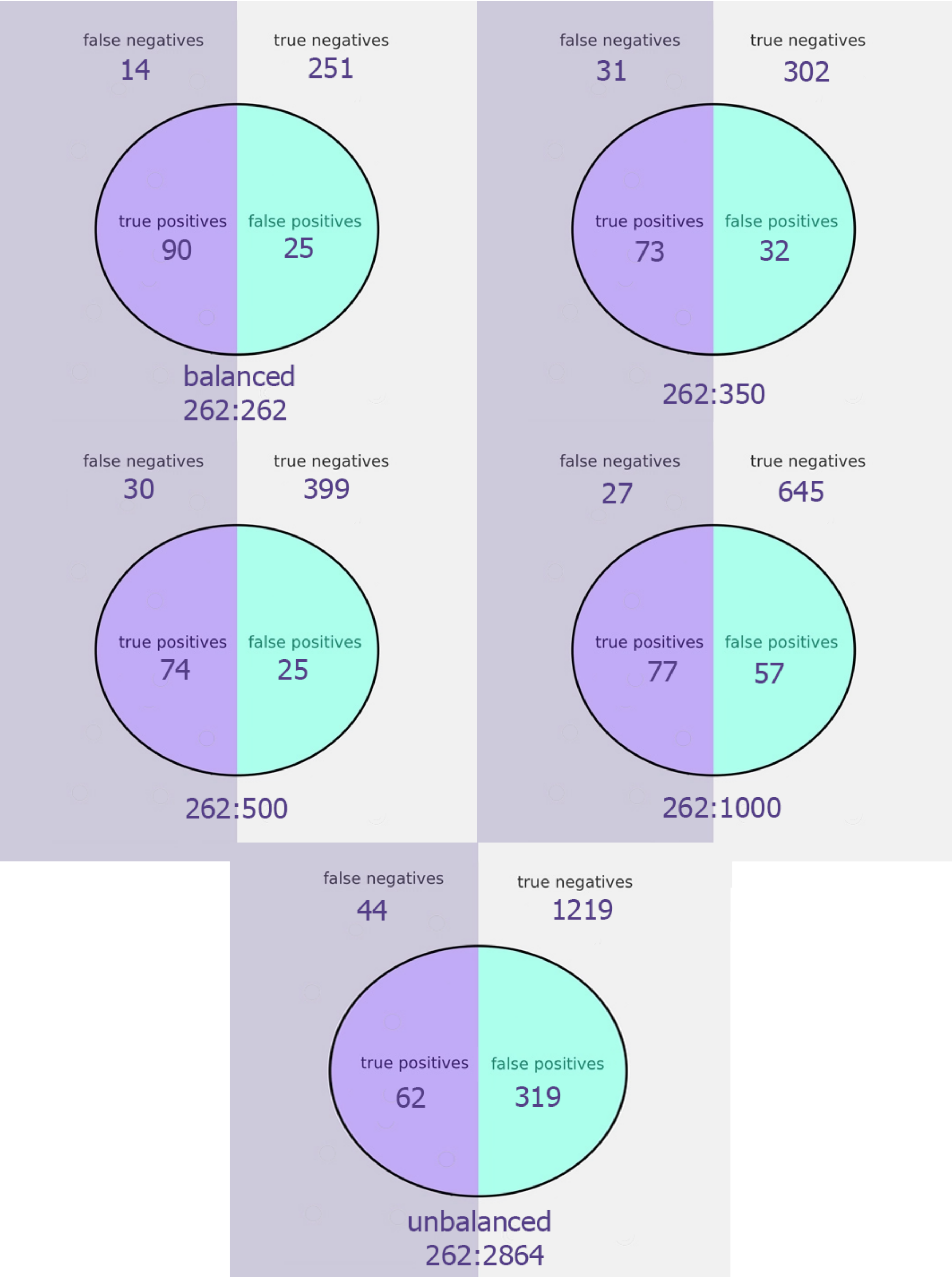
F-scores of all combinations of machine learning methods and dataset balances, results remain inconsistent

At this point we have managed to create an algorithm that is somewhat competent in entity resolution, but not enough for commercial use. This is why we had to introduce two additional mechanisms that worked in tandem in the algorithm in order to improve its performance.

The first mechanism was a cutoff threshold in the cosine similarity. Before now, it has been possible for the machine learning model to categorize a match with very little cosine similarity as true. That will result in two drastically different items being matched together. To avoid this we introduced a floor to how low cosine similarity could be before we decide that it's not even worth to test it on a machine learning model, and instead conclude that the entity has no match. After various tests, we concluded that a threshold of 0.5 cosine similarity is optimal. So for example, if an item has many candidates for a match above 0.5 similarity this change would not affect it. But if all of an item's candidates were below 0.5 similarity, this change would make it so we would instead determine that this item doesn't have a match.

The second mechanism was a penalty to the cosine similarity of matches when the machine learning algorithm determined that the match was false. The machine learning models would sometimes falsely determine that very similar items were different. We needed to introduce a penalty to the similarity value that would be significant enough to alter the order of the most probable candidates for matching with our product, but also forgiving enough to not exclude a candidate if it were the only good one. After multiple tests, the penalty was fixed at 10% of the similarity value. So for example a match with similarity of 0.9 that was categorised as false by the Random Forest model would have its similarity altered to 0.81. If there were any candidates above that similarity that the model categorized as true, they would take its place as the prime candidate, otherwise the match would remain as the most probable one.

The results of this new and final iteration of Baseline 2 can be seen below:


false negatives
14
true negatives
251
true positives
90
false positives
25
balanced
262:262
false negatives
31
true negatives
302
true positives
73
false positives
32
262:350
false negatives
30
true negatives
399
true positives
74
false positives
25
262:500
false negatives
27
true negatives
645
true positives
77
false positives
57
262:1000
false negatives
44
true negatives
1219
true positives
62
false positives
319
unbalanced
262:2864

- Precision:

    1. 262:262 : 0.78
    2. 262:350 : 0.70
    3. 262:500 : 0.75
    4. 262:1000 : 0.57
    5. 262:2864 : 0.16

- Recall:

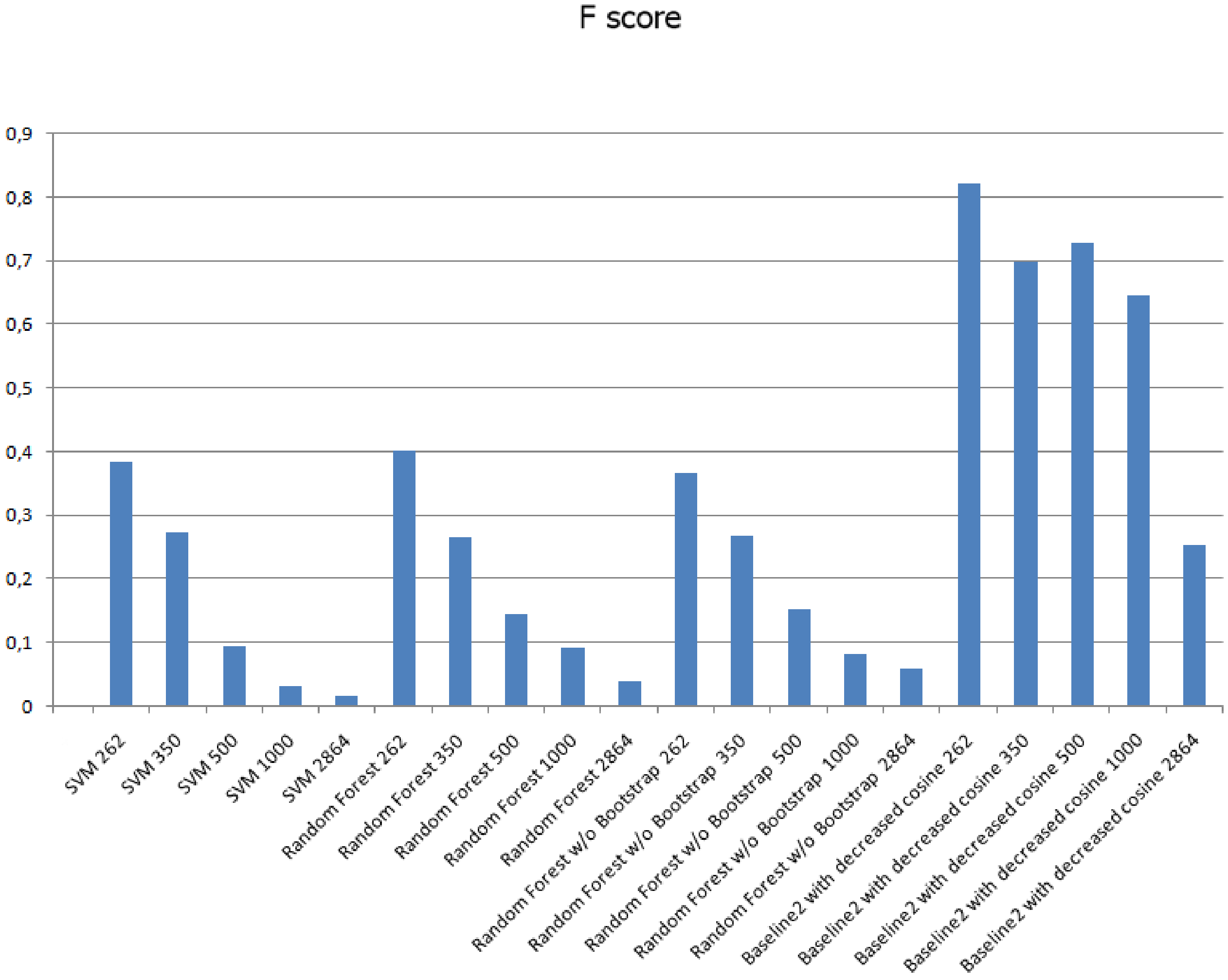
    1. 262:262 : 0.87
    2. 262:350 : 0.70
    3. 262:500 : 0.71
    4. 262:1000 : 0.74
    5. 262:2864 : 0.58

- F1-Score:

    1. 262:262 : 0.82
    2. 262:350 : 0.70
    3. 262:500 : 0.73
    4. 262:1000 : 0.65
    5. 262:2864 : 0.25

It is apparent from these figures that this new iteration is not only significantly better than its predecessors, but also competent enough to be used in a commercial environment. Unlike previous iterations, we do not have to chose between a good True Positive Rate and a good True Negative Rate, as we can achieve both at the same time. The error rate is relatively small, and even when two items are falsely matched, they are very similar in nature, such as two version of the same product that only differ in quantity, or size.

To showcase the results better, below we show the previous graph including all tested methods, this time including the new baseline 2 with decreased cosine and similarity cutoff.



F-scores of our new method compared to the previous scores, showing success rate of over 80%

3.8 Code Presentation

In this section we will present the entity resolution algorithm we developed in pseudo code, so as to better explain the process.

```
function randomforestapproves(item, dataset, model)
        flag_list = predict match between item and each item in dataset

        return flag_list
//this function predicts match correctness between an item and other
//items in a database based on training

cutoff = 0.5
//below this point of similarity we decide that there is no match

decrease_margin = 0.9
//the factor we multiply similarity as a penalty if the machine
//learning model makes a negative prediction

for each item in full_dataset
        add description_words in total_bag_of_words

total_matrix = tfidf_fit_transform(total_bag_of_words)
// we fit transform our tfidf vectorizer on the total dataset

for each item in training_dataset
        add description_words in training_bag_of_words
        tags = match_tags
 //these are the tags we manually assigned to matches

training_matrix = tfidf_transform(training_bag_of_words)
//we then transform on the training dataset
```

```
model = RandomForestClassifierWithBootstrap
//the model with the best resutls was a random
//forest classifier with bootstrap

model.fit(training_matrix ,tags)

for each item in testing_dataset

          flag_list = randomforestapproves(item, testing_dataset, model )
          calculate cosine similarity with all items in testing_dataset

          if flag = false
                    similarity = similarity*decrease_margin

          for each grocery store other than the item source

                    find product with highest cosine similarity with item
                    if similarity < cutoff
                              no match with that store
                    else
                              found match with this store

          create a match of item with its matches from the other stores

save matches in database
```

3.9 Conclusions & Future Work

In conclusion, with a combination of information retrieval and machine learning we have created an entity resolution algorithm that we can use for our application. We have seen which models work best in which cases, what changes have to be made for them to adapt to our specific problem, and we have showcased all of these results using metrics that determine not only surface level performance, but also account for all aspects of a stable classification algorithm.

In future implementations of this work, there could be some alternative options that might be chosen and tested for results. In this implementation we concluded that the prices of the pairs even when the match was false, were similar enough to not warrant much afterthought, but an additional filtering of the products' prices may yield some improvement to the resutls. Moreover, because of restrictions of the testing hardware we were not able to capitalize on the fact that we had the product images in our possession. With better equipment or more time to spare, an algorithm for image matching might also significantly improve the results.

Chapter 4

Using Apache Solr

In this chapter we examine Apache Solr [2] in its use as a database and search engine. We take a look in the existing UIs that are available for users online, and, finally, our own implementation of the Merebo Web App, an application that allows the user to interact with the results of our work.

4.1 Using Solr as a Database

So far we have been storing our entities in excel sheets. We chose this method because while in the experimenting phase multiple rounds of editing and interaction with the data was neccessary. However, after finishing our entity resolution algorithm and using it on a sizeable sample dataset, we finally have the final form of our dataset. We have each item comprising of 1-3 products depending on how many matches were found, along with their descriptions, prices, ID of the store they originate from, and finally a local path to their images. An example of one sample of data in our database is the following:

- Store 1 ID: 1
- Product 1 Name: PAPADOPOULOU Cookies with Chocolate pieces Cocoa 2x180gr

- Price 1: 1,59 €

- Image 1:
- Store 2 ID: 2
- Product 2 Name: PAPADOPOULOU COOKIES WITH CHOCOLATE PIECES 180 GR BREAKFAST - snacking and refreshments
- Price 2: €1,37

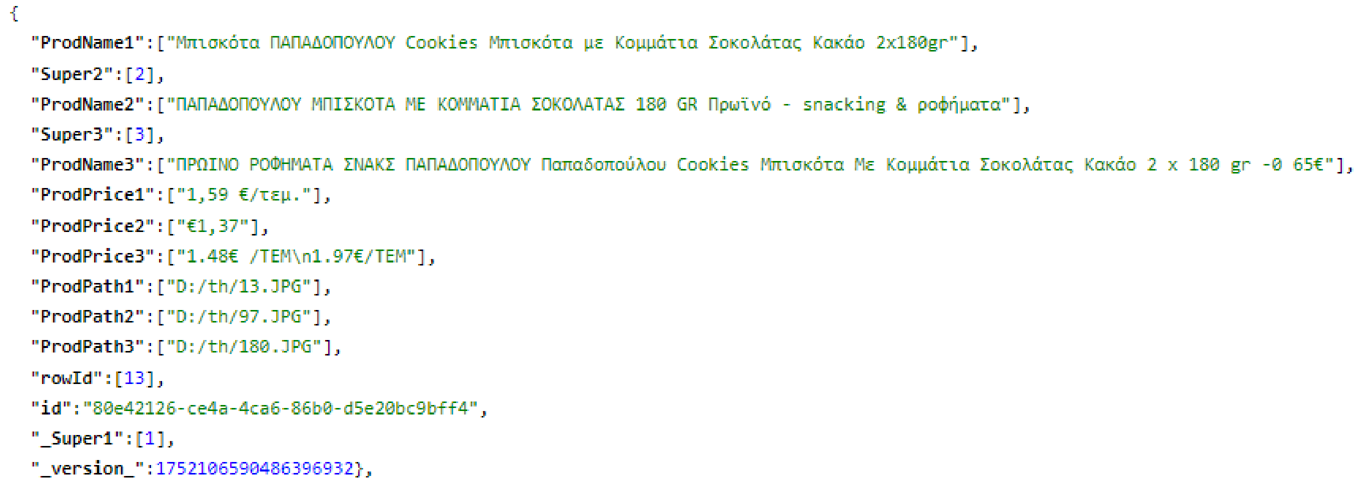
- Image 2:
- Store 3 ID: 2
- Product 3 Name: BREAKFAST REFRESHMENTS SNACKS PAPADOPOULOU Papadopoulou Cookies Cookies with Chocolate Pieces Cocoa 2 x 180 gr -0 65€
- Price 3: 1.48€

- Image 3:

In this example, we can see a case where an item has a correct match from one store and an incorrect match with the other one, although even then the products were relatively similar.

Now that we have the finished form of our data, we can input it in a more appropriate database, for which purpose we decided to use Apache Solr. Although Solr is predominately a search engine, it can also be utilized as a database. To use it as such, we need to download a version of Solr, run it through command line or terminal, create a core (or collection if running in cloud mode), and use the post command to upload a compatible file, like Json, CSV and others. In our case, we chose to use the CSV format to upload to our database. After this, more files can be added or even removed from a specific core, allowing us to keep our database up to date.

The form in which the previous example will be logged into our Solr Database can be shown in the image below:

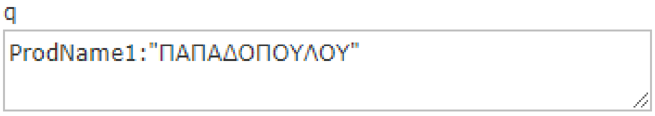


```
{
  "ProdName1":["Μπισκότα ΠΑΠΑΔΟΠΟΥΛΟΥ Cookies Μπισκότα με Κομμάτια Σοκολάτας Κακάο 2x180gr"],
  "Super2":[2],
  "ProdName2":["ΠΑΠΑΔΟΠΟΥΛΟΥ ΜΠΙΣΚΟΤΑ ΜΕ ΚΟΜΜΑΤΙΑ ΣΟΚΟΛΑΤΑΣ 180 GR Πρωϊνό - snacking & ροφήματα"],
  "Super3":[3],
  "ProdName3":["ΠΡΩΙΝΟ ΡΟΦΗΜΑΤΑ ΣΝΑΚΣ ΠΑΠΑΔΟΠΟΥΛΟΥ Παπαδοπούλου Cookies Μπισκότα Με Κομμάτια Σοκολάτας Κακάο 2 x 180 gr -0 65€"],
  "ProdPrice1":["1,59 €/τεμ."],
  "ProdPrice2":["€1,37"],
  "ProdPrice3":["1.48€ /TEM\n1.97€/TEM"],
  "ProdPath1":["D:/th/13.JPG"],
  "ProdPath2":["D:/th/97.JPG"],
  "ProdPath3":["D:/th/180.JPG"],
  "rowId":[13],
  "id":"80e42126-ce4a-4ca6-86b0-d5e20bc9bff4",
  "_Super1":[1],
  "_version_":1752106590486396932},
```

Once we have run Solr, a locally hosted server will be created that will support our database, by using references to local files. That is to say, we can't upload an image to our Solr database directly, but we can give it a path to the image from a local directory, like we did in the example above.

4.2 Using Solr as a Search Engine

After successfully uploading our database to a Solr core, we can use its search engine algorithm to query the data. The querying mechanism is usable for a person who understands basic database querying, but very unintuitive for the average user. Where most of us are used to simply typing terms we are searching for in the search bar, the Solr seach engine also requires us to define the source of the term. So for example, if we are looking for a specific category of products, like products from the brand PAPADOPOULOU, we would have to clarify that we are looking for them in the product name field of the data. So in this example our query would have to be:

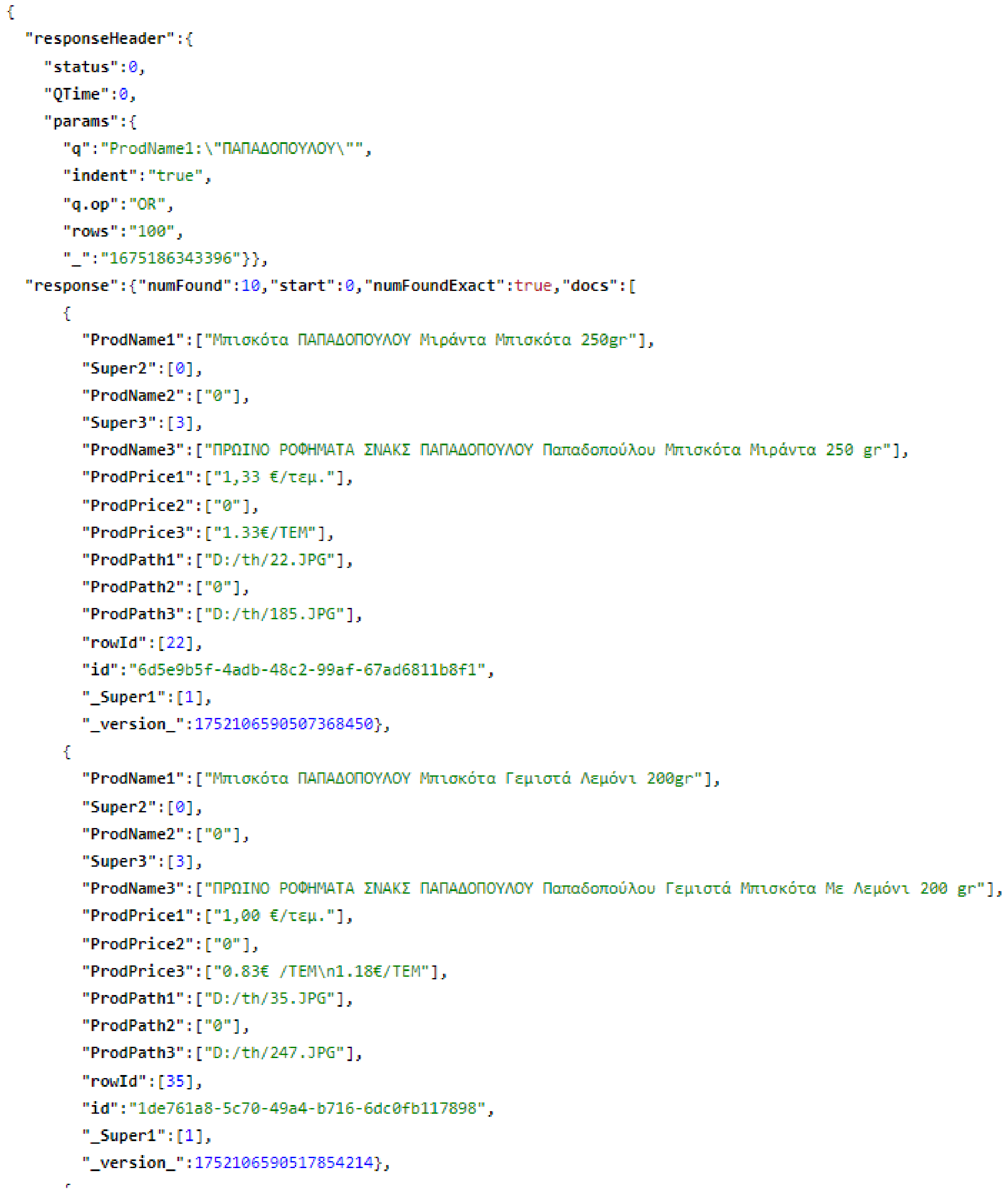


q

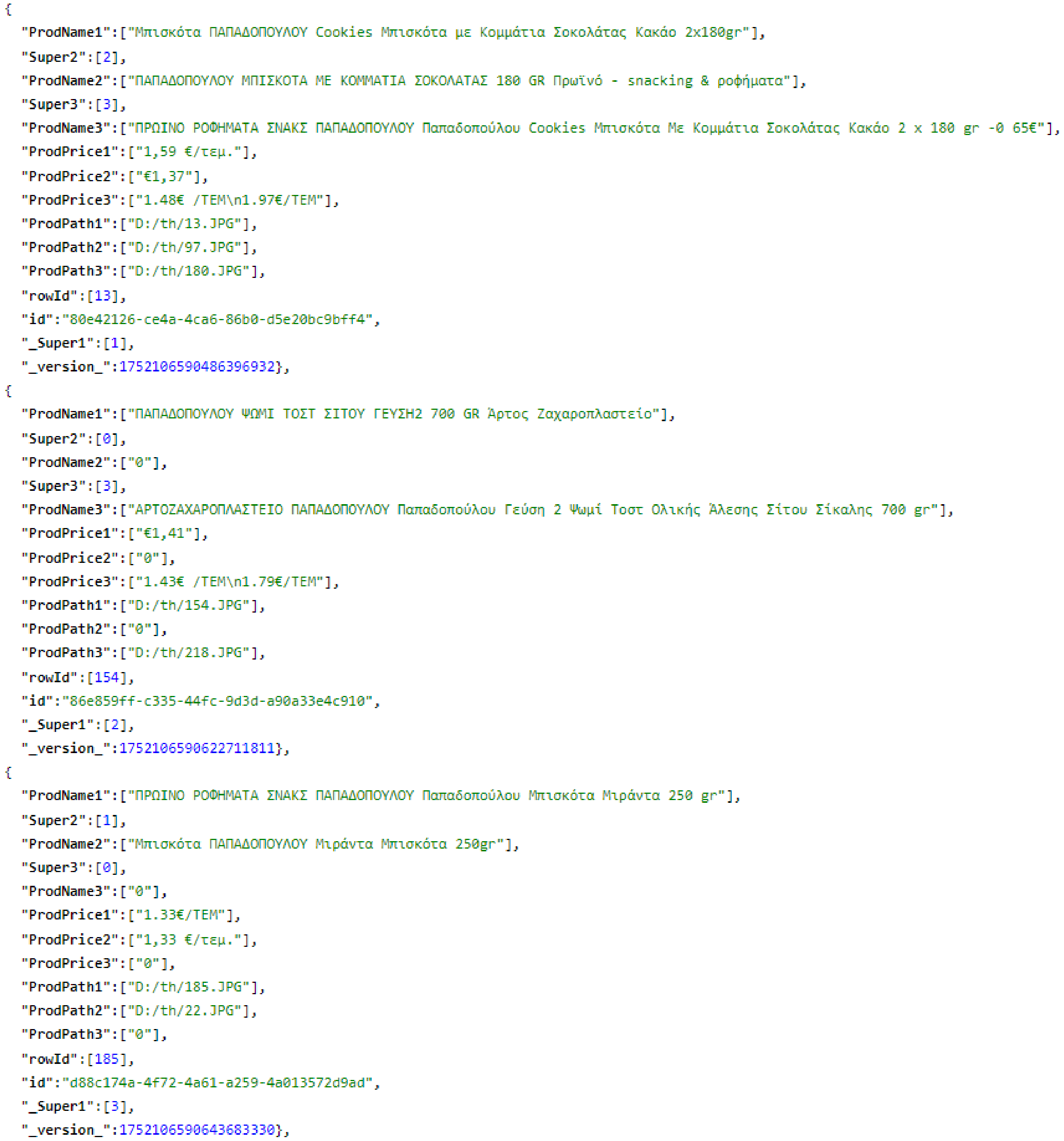
```
ProdName1:"ΠΑΠΑΔΟΠΟΥΛΟΥ"
```

Where ProdName1 is the field of the data for the item which is matched with the other two from different stores.

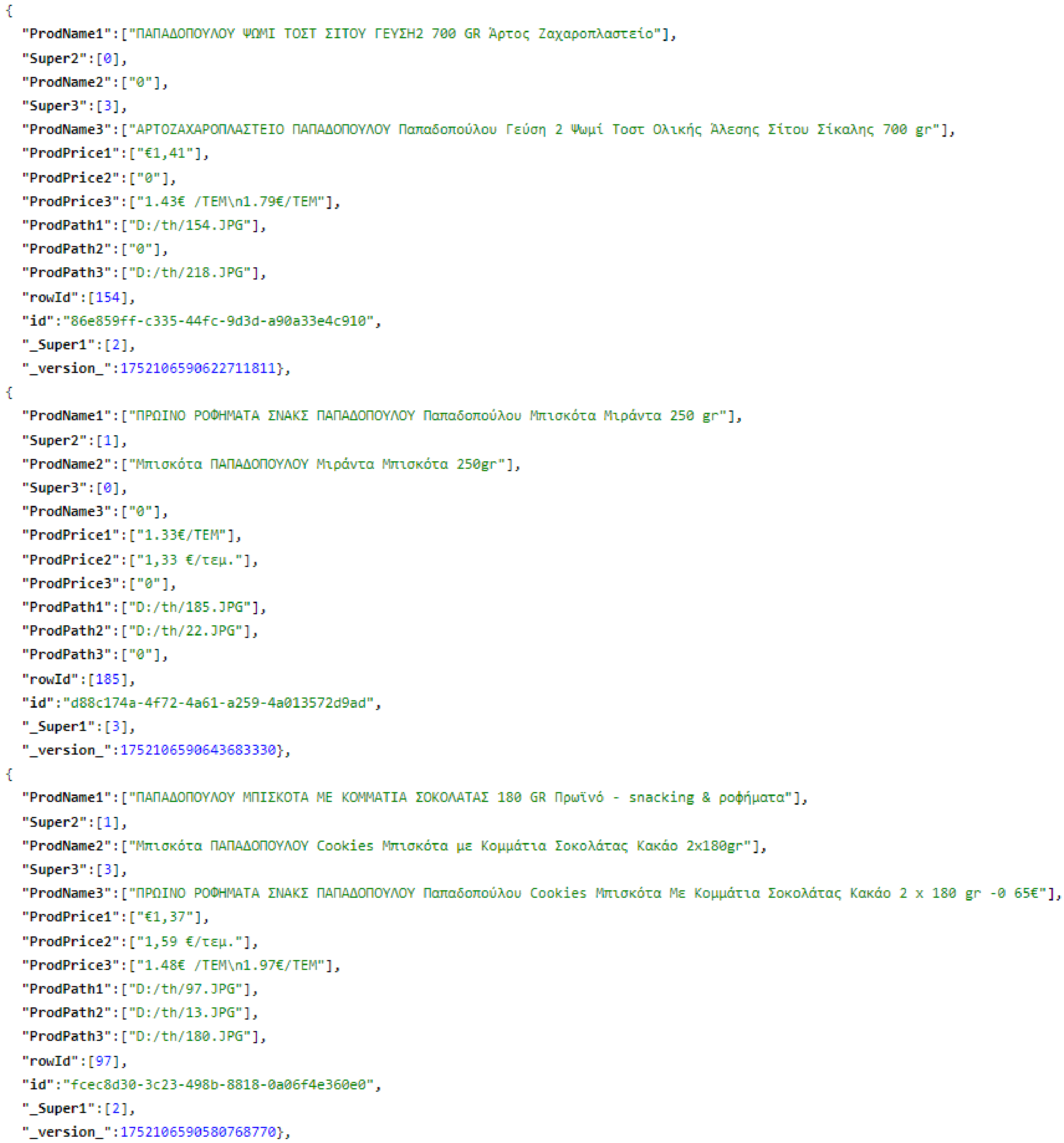
This query will give us for search results every primary item that has in its description the term we searched for. To show a few results:



```
{
  "responseHeader":{
    "status":0,
    "QTime":0,
    "params":{
      "q":"ProdName1:\"ΠΑΠΑΔΟΠΟΥΛΟΥ\"",
      "indent":"true",
      "q.op":"OR",
      "rows":"100",
      "_":"1675186343396"}},
  "response":{"numFound":10,"start":0,"numFoundExact":true,"docs":[
      {
        "ProdName1":["Μπισκότα ΠΑΠΑΔΟΠΟΥΛΟΥ Μιράντα Μπισκότα 250gr"],
        "Super2":[0],
        "ProdName2":["0"],
        "Super3":[3],
        "ProdName3":["ΠΡΩΙΝΟ ΡΟΦΗΜΑΤΑ ΣΝΑΚΣ ΠΑΠΑΔΟΠΟΥΛΟΥ Παπαδοπούλου Μπισκότα Μιράντα 250 gr"],
        "ProdPrice1":["1,33 €/τεμ."],
        "ProdPrice2":["0"],
        "ProdPrice3":["1.33€/TEM"],
        "ProdPath1":["D:/th/22.JPG"],
        "ProdPath2":["0"],
        "ProdPath3":["D:/th/185.JPG"],
        "rowId":[22],
        "id":"6d5e9b5f-4adb-48c2-99af-67ad6811b8f1",
        "_Super1":[1],
        "_version_":1752106590507368450},
      {
        "ProdName1":["Μπισκότα ΠΑΠΑΔΟΠΟΥΛΟΥ Μπισκότα Γεμιστά Λεμόνι 200gr"],
        "Super2":[0],
        "ProdName2":["0"],
        "Super3":[3],
        "ProdName3":["ΠΡΩΙΝΟ ΡΟΦΗΜΑΤΑ ΣΝΑΚΣ ΠΑΠΑΔΟΠΟΥΛΟΥ Παπαδοπούλου Γεμιστά Μπισκότα Με Λεμόνι 200 gr"],
        "ProdPrice1":["1,00 €/τεμ."],
        "ProdPrice2":["0"],
        "ProdPrice3":["0.83€ /TEM\n1.18€/TEM"],
        "ProdPath1":["D:/th/35.JPG"],
        "ProdPath2":["0"],
        "ProdPath3":["D:/th/247.JPG"],
        "rowId":[35],
        "id":"1de761a8-5c70-49a4-b716-6dc0fb117898",
        "_Super1":[1],
        "_version_":1752106590517854214},
      {
```



```
{
  "ProdName1":["Μπισκότα ΠΑΠΑΔΟΠΟΥΛΟΥ Cookies Μπισκότα με Κομμάτια Σοκολάτας Κακάο 2x180gr"],
  "Super2":[2],
  "ProdName2":["ΠΑΠΑΔΟΠΟΥΛΟΥ ΜΠΙΣΚΟΤΑ ΜΕ ΚΟΜΜΑΤΙΑ ΣΟΚΟΛΑΤΑΣ 180 GR Πρωϊνό - snacking & ροφήματα"],
  "Super3":[3],
  "ProdName3":["ΠΡΩΙΝΟ ΡΟΦΗΜΑΤΑ ΣΝΑΚΣ ΠΑΠΑΔΟΠΟΥΛΟΥ Παπαδοπούλου Cookies Μπισκότα Με Κομμάτια Σοκολάτας Κακάο 2 x 180 gr -0 65€"],
  "ProdPrice1":["1,59 €/τεμ."],
  "ProdPrice2":["€1,37"],
  "ProdPrice3":["1.48€ /TEM\n1.97€/TEM"],
  "ProdPath1":["D:/th/13.JPG"],
  "ProdPath2":["D:/th/97.JPG"],
  "ProdPath3":["D:/th/180.JPG"],
  "rowId":[13],
  "id":"80e42126-ce4a-4ca6-86b0-d5e20bc9bff4",
  "_Super1":[1],
  "_version_":1752106590486396932},
{
  "ProdName1":["ΠΑΠΑΔΟΠΟΥΛΟΥ ΨΩΜΙ ΤΟΣΤ ΣΙΤΟΥ ΓΕΥΣΗ2 700 GR Άρτος Ζαχαροπλαστείο"],
  "Super2":[0],
  "ProdName2":["0"],
  "Super3":[3],
  "ProdName3":["ΑΡΤΟΖΑΧΑΡΟΠΛΑΣΤΕΙΟ ΠΑΠΑΔΟΠΟΥΛΟΥ Παπαδοπούλου Γεύση 2 Ψωμί Τοστ Ολικής Άλεσης Σίτου Σίκαλης 700 gr"],
  "ProdPrice1":["€1,41"],
  "ProdPrice2":["0"],
  "ProdPrice3":["1.43€ /TEM\n1.79€/TEM"],
  "ProdPath1":["D:/th/154.JPG"],
  "ProdPath2":["0"],
  "ProdPath3":["D:/th/218.JPG"],
  "rowId":[154],
  "id":"86e859ff-c335-44fc-9d3d-a90a33e4c910",
  "_Super1":[2],
  "_version_":1752106590622711811},
{
  "ProdName1":["ΠΡΩΙΝΟ ΡΟΦΗΜΑΤΑ ΣΝΑΚΣ ΠΑΠΑΔΟΠΟΥΛΟΥ Παπαδοπούλου Μπισκότα Μιράντα 250 gr"],
  "Super2":[1],
  "ProdName2":["Μπισκότα ΠΑΠΑΔΟΠΟΥΛΟΥ Μιράντα Μπισκότα 250gr"],
  "Super3":[0],
  "ProdName3":["0"],
  "ProdPrice1":["1.33€/TEM"],
  "ProdPrice2":["1,33 €/τεμ."],
  "ProdPrice3":["0"],
  "ProdPath1":["D:/th/185.JPG"],
  "ProdPath2":["D:/th/22.JPG"],
  "ProdPath3":["0"],
  "rowId":[185],
  "id":"d88c174a-4f72-4a61-a259-4a013572d9ad",
  "_Super1":[3],
  "_version_":1752106590643683330},
```



```
{
  "ProdName1":["ΠΑΠΑΔΟΠΟΥΛΟΥ ΨΩΜΙ ΤΟΣΤ ΣΙΤΟΥ ΓΕΥΣΗ2 700 GR Άρτος Ζαχαροπλαστείο"],
  "Super2":[0],
  "ProdName2":["0"],
  "Super3":[3],
  "ProdName3":["ΑΡΤΟΖΑΧΑΡΟΠΛΑΣΤΕΙΟ ΠΑΠΑΔΟΠΟΥΛΟΥ Παπαδοπούλου Γεύση 2 Ψωμί Τοστ Ολικής Άλεσης Σίτου Σίκαλης 700 gr"],
  "ProdPrice1":["€1,41"],
  "ProdPrice2":["0"],
  "ProdPrice3":["1.43€ /TEM\n1.79€/TEM"],
  "ProdPath1":["D:/th/154.JPG"],
  "ProdPath2":["0"],
  "ProdPath3":["D:/th/218.JPG"],
  "rowId":[154],
  "id":"86e859ff-c335-44fc-9d3d-a90a33e4c910",
  "_Super1":[2],
  "_version_":1752106590622711811},
{
  "ProdName1":["ΠΡΩΙΝΟ ΡΟΦΗΜΑΤΑ ΣΝΑΚΣ ΠΑΠΑΔΟΠΟΥΛΟΥ Παπαδοπούλου Μπισκότα Μιράντα 250 gr"],
  "Super2":[1],
  "ProdName2":["Μπισκότα ΠΑΠΑΔΟΠΟΥΛΟΥ Μιράντα Μπισκότα 250gr"],
  "Super3":[0],
  "ProdName3":["0"],
  "ProdPrice1":["1.33€/TEM"],
  "ProdPrice2":["1,33 €/τεμ."],
  "ProdPrice3":["0"],
  "ProdPath1":["D:/th/185.JPG"],
  "ProdPath2":["D:/th/22.JPG"],
  "ProdPath3":["0"],
  "rowId":[185],
  "id":"d88c174a-4f72-4a61-a259-4a013572d9ad",
  "_Super1":[3],
  "_version_":1752106590643683330},
{
  "ProdName1":["ΠΑΠΑΔΟΠΟΥΛΟΥ ΜΠΙΣΚΟΤΑ ΜΕ ΚΟΜΜΑΤΙΑ ΣΟΚΟΛΑΤΑΣ 180 GR Πρωϊνό - snacking & ροφήματα"],
  "Super2":[1],
  "ProdName2":["Μπισκότα ΠΑΠΑΔΟΠΟΥΛΟΥ Cookies Μπισκότα με Κομμάτια Σοκολάτας Κακάο 2x180gr"],
  "Super3":[3],
  "ProdName3":["ΠΡΩΙΝΟ ΡΟΦΗΜΑΤΑ ΣΝΑΚΣ ΠΑΠΑΔΟΠΟΥΛΟΥ Παπαδοπούλου Cookies Μπισκότα Με Κομμάτια Σοκολάτας Κακάο 2 x 180 gr -0 65€"],
  "ProdPrice1":["€1,37"],
  "ProdPrice2":["1,59 €/τεμ."],
  "ProdPrice3":["1.48€ /TEM\n1.97€/TEM"],
  "ProdPath1":["D:/th/97.JPG"],
  "ProdPath2":["D:/th/13.JPG"],
  "ProdPath3":["D:/th/180.JPG"],
  "rowId":[97],
  "id":"fcec8d30-3c23-498b-8818-0a06f4e360e0",
  "_Super1":[2],
  "_version_":1752106590580768770},
```

A thing that quickly becomes apparent in this example is the number of shortcomings in the Solr search engine. We notice that Solr is sensitive to factors like accentuation and capitalization. All of the items shown in the images include only the term we searched for letter for letter, and exclude search results that have alternative versions of the word, either with accentuation, or lowercase use.

4.3 Solr UIs

While Solr does provide a default UI to use its functions without having to resort to using the command line/terminal every time we want to do anything, its UI is not user friendly enough for us to use as is. Especially its search engine, as shown in the previous section can be a bit confusing if the user isn't familiar with queries. Because our goal is for the MEREBO App to be useable by everyone, we wanted to refine it and make it more approachable and user friendly.

As we wern't satisfied with the default interface, we tried to browse the web for any existing UIs for Solr created from users for similar applications. Some of the more noteable ones, which can also be found in the Anant Github repository [1], were the following:

1. Blacklight
2. AJAX Solr
3. Apache Solr Search Drupal Plugin
4. Solr PHP UI
5. Spyglass
6. Splainer

Among many others. Unfortunately, we decided against using any of these options for several reasons. Some repositories were deprecated, others were not ideal for our particular case, and some we just couldn't support because of software limitations.

Therefore, we concluded that the best way to give MEREBO a deserving interface would be by creating one from scratch so that it matches our needs for the application.

4.4 MEREBO Web App

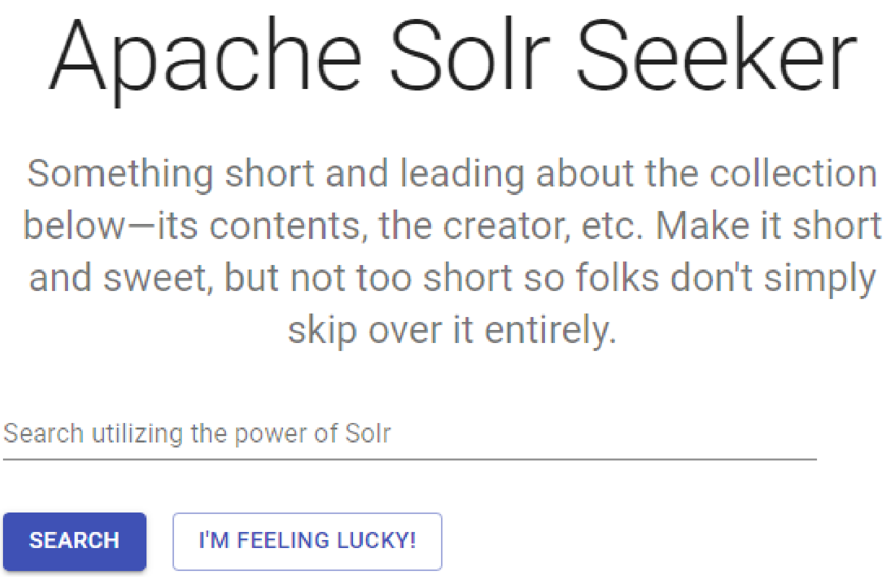
The MEREBO Webb App is built mainly using React [6], a JavaScript library for building user interfaces. It offers the user the basic features required of an e-commerce website, while maintaining a simple structure.

4.4.1 Solr Connection

The app is connected to the Solr Core where we uploaded our database and for all intents and purposes is a more user friendly version of its default UI, with some additional features.

4.4.2 Navigation Bar

The first thing that we observe when browsing the app is the navigation bar. Typing anything in the bar, for example the term ”water” will translate the search into a query of **ProdName1:”water”** and use the Solr search engine to find matching terms in the respective fields of the database we have uploaded. This way anything typed in the navigation bar will be searched in the catalogue of item descriptions of every first item in a match. Because every match has up to 3 different versions in the database, depending on how many similar products were found where each item takes turns in being the first item, there is no risk of leaving any items out.



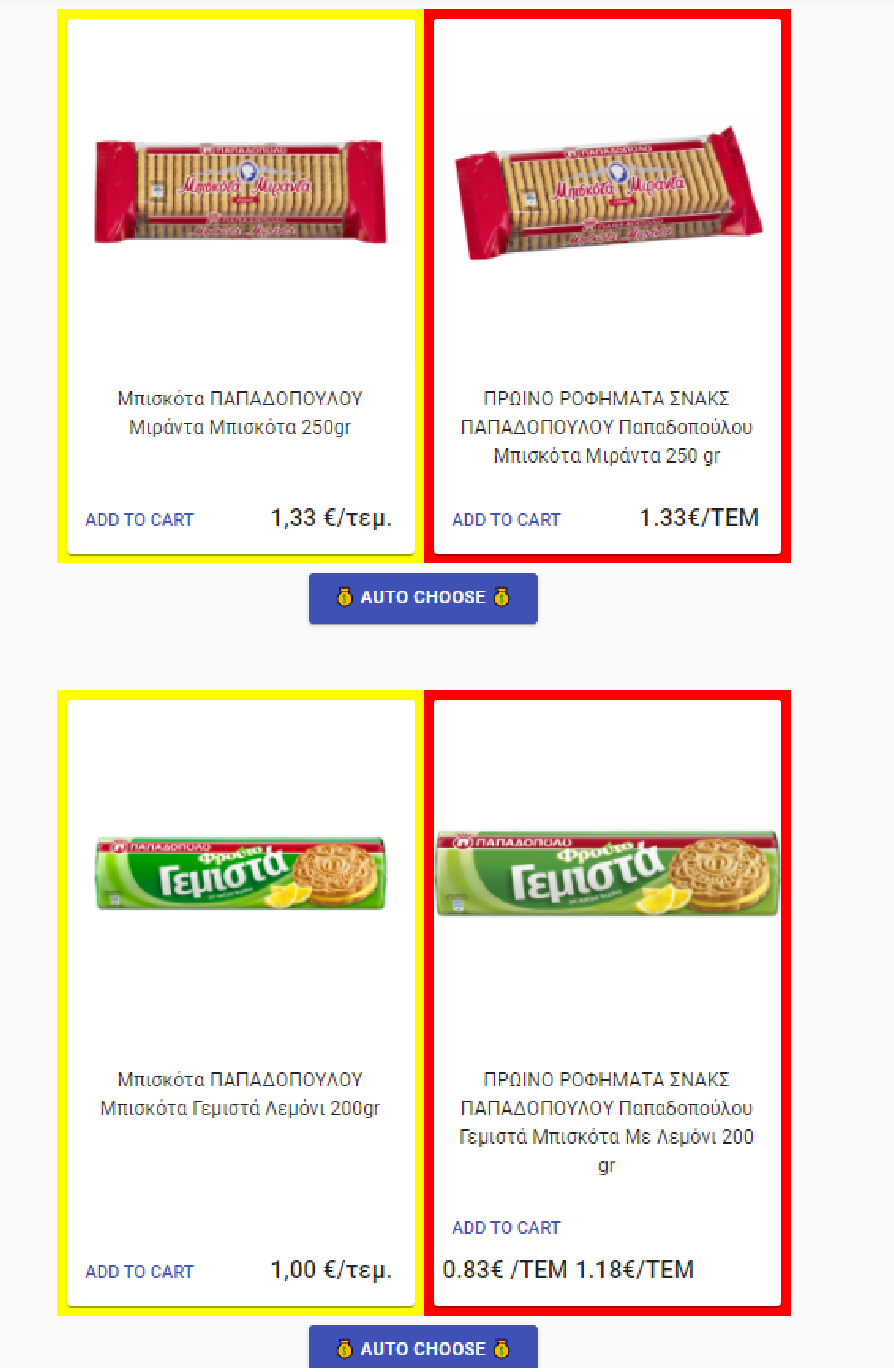
Apache Solr Seeker

Something short and leading about the collection below—its contents, the creator, etc. Make it short and sweet, but not too short so folks don't simply skip over it entirely.

4.4.3 Item List

Once we have typed a term in the navigation bar, a search takes place and a list of items found is presented to the user. The items are color coded depending on their store of origin, with Yellow for Sklavenitis, Orange for AB and red for Chalkiadakis. With each item we also are presented with a price, image, description and buying options. An important thing to note, is that as previously stated each item match can be presented up to 3 times, which is not desirable. To avoid this, we placed each item match in a javascript Set, a list made up of unique items, so when a duplicate match is found, just in shuffled order, it is recognized and excluded.

A search of the term ”cookies” gives us the following results:



Μπισκότα ΠΑΠΑΔΟΠΟΥΛΟΥ Μιράντα Μπισκότα 250gr

1,33 €/τεμ.

ΠΡΩΙΝΟ ΡΟΦΗΜΑΤΑ ΣΝΑΚΣ ΠΑΠΑΔΟΠΟΥΛΟΥ Παπαδοπούλου Μπισκότα Μιράντα 250 gr

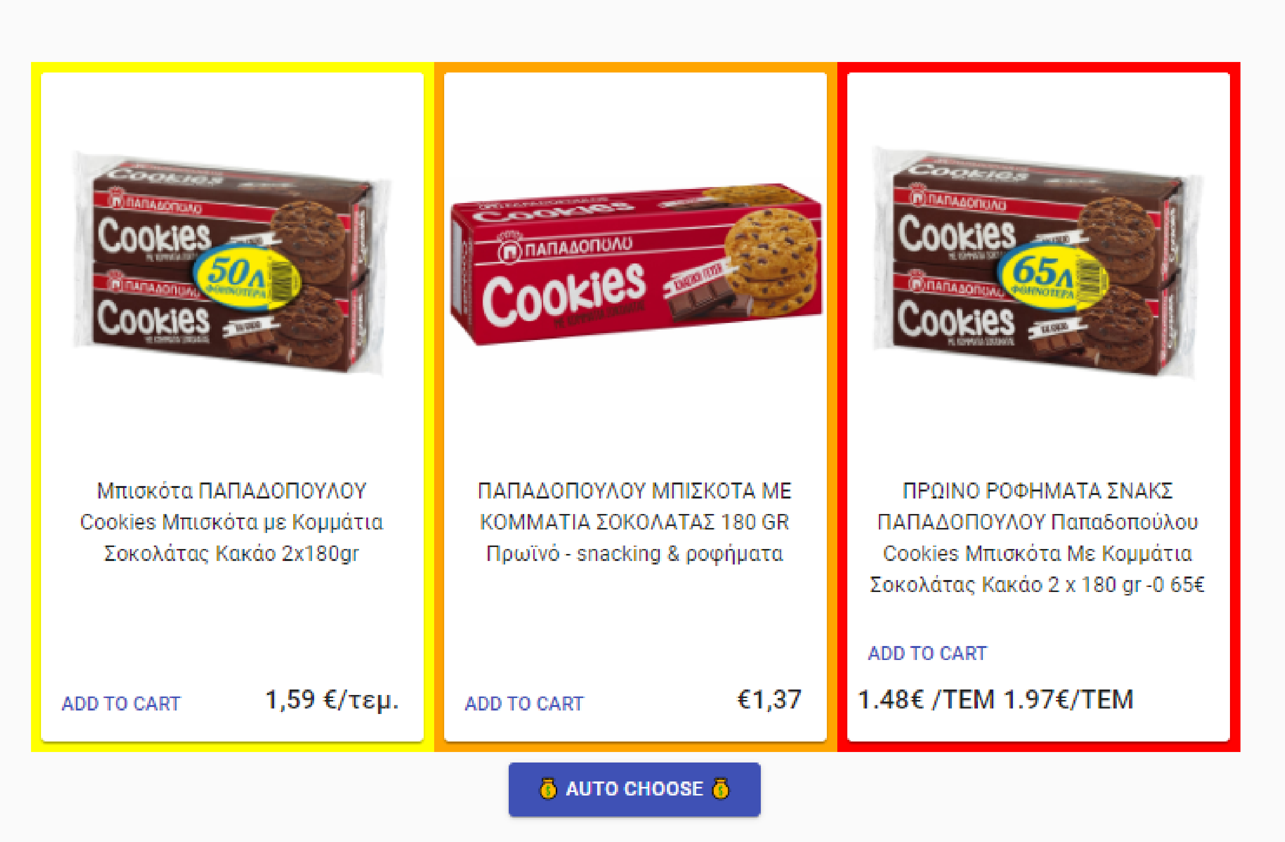
1.33€/ΤΕΜ

Μπισκότα ΠΑΠΑΔΟΠΟΥΛΟΥ Μπισκότα Γεμιστά Λεμόνι 200gr

1,00 €/τεμ.

ΠΡΩΙΝΟ ΡΟΦΗΜΑΤΑ ΣΝΑΚΣ ΠΑΠΑΔΟΠΟΥΛΟΥ Παπαδοπούλου Γεμιστά Μπισκότα Με Λεμόνι 200 gr

0.83€ /ΤΕΜ 1.18€/ΤΕΜ



Μπισκότα ΠΑΠΑΔΟΠΟΥΛΟΥ Cookies Μπισκότα με Κομμάτια Σοκολάτας Κακάο 2x180gr

1,59 €/τεμ.

ΠΑΠΑΔΟΠΟΥΛΟΥ ΜΠΙΣΚΟΤΑ ΜΕ ΚΟΜΜΑΤΙΑ ΣΟΚΟΛΑΤΑΣ 180 GR Πρωϊνό - snacking & ροφήματα

€1,37

ΠΡΩΙΝΟ ΡΟΦΗΜΑΤΑ ΣΝΑΚΣ ΠΑΠΑΔΟΠΟΥΛΟΥ Παπαδοπούλου Cookies Μπισκότα Με Κομμάτια Σοκολάτας Κακάο 2 x 180 gr -0 65€

1.48€ /TEM 1.97€/TEM

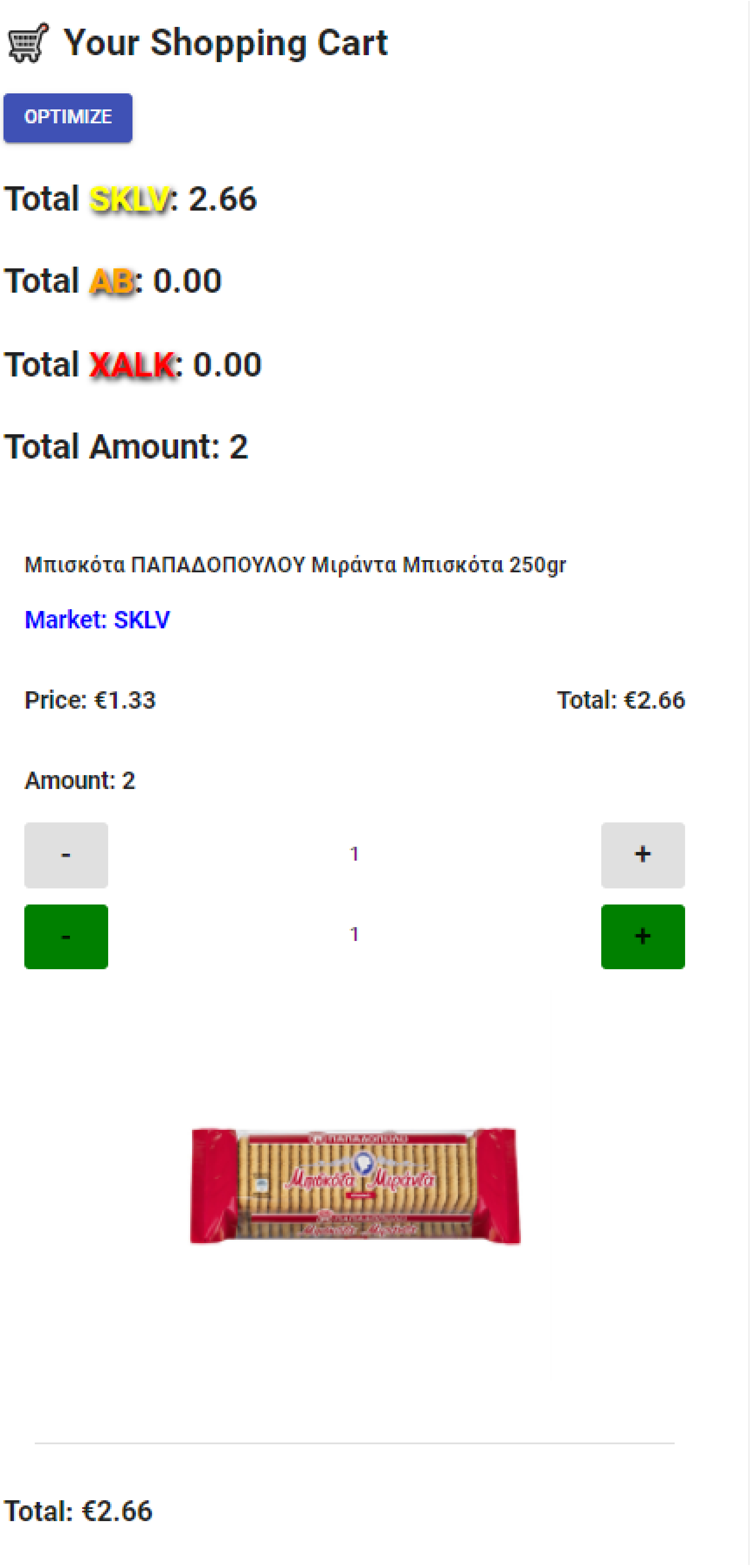
4.4.4 Buying Options

Each item is accompanied by two buying options, **"Add to Cart"** and **"Auto Choose"**.

The former places an item in the basket in a way that prohibits the optimization algorithm from changing its source. This option is useful in the case where a match is false, and only one of the items matches the user's search. In that case, the user can choose to place that specific item in his basket.

The latter is the option that allows the algorithm to optimize the products placed in the basket. When first clicked, the algorithm will initially place the cheapest option in the basket. Items placed this way are kept in a different list than items placed in the previous way.

4.4.5 Basket



Your Shopping Cart

Total SKLV: 2.66

Total AB: 0.00

Total XALK: 0.00

Total Amount: 2

Μπισκότα ΠΑΠΑΔΟΠΟΥΛΟΥ Μιράντα Μπισκότα 250gr

Market: SKLV

Price: €1.33 Total: €2.66

Amount: 2

Total: €2.66

The Basket is the final part of the Webb App interface, and provides a list with all bought items for review and editing. First off, the user is shown the total sum of the amount he has spent, both in each individual store, and in total. This allows him to keep an eye on his expended amount while being able to take note of the minimum delivery costs for each store.

Bellow these values, we are met with a list of all items placed in the basket. Each item has two values, each with a different set of buttons attached to it. The **Green** buttons are for the amount of the item placed in the basket using the **Auto Choose** option, while the **Gray** ones are for the amount placed with the **Add to Cart** option. For example, an item that was placed once with Auto Choose and once with Add to Cart, will have a value of two in total, but two different instances in the cart, a ”1” that can be edited with the Green buttons and a ”1” that can be edited with the Gray buttons. It should be noted that reducing an items price to 0 using a button will remove that items from the basket from this specific option only. In the above example, if we pressed the Gray **-** button, the item would still have one instance in the basket, as if it were only placed with the Auto Choose option.

Lastly, there is the Optimize button. This button uses an optimization algorithm to find the optimal combination of products with the restrictions placed indirectly by the user, to find the minimum price while taking into account the minimum delivery costs of each store. This process will freely change the ammount of items placed with the Auto Choose option, but will not make any changes to the items placed with the Add to Cart option. When this option is chosen, one of three outcomes will occur:

1. in the case where the minimum delivery costs cannot be met to fulfil the delivery, an error message will pop up.
2. if the items placed in the basket are already in the optimal combination the user will be let to know
3. if optimization can take place, the items will be placed in the optimal combination, the total sums will be changed for each grocery store, and in total, and the new item list will be shown

4.5 CORS

Before concluding this section of the thesis we must address an important detail that makes this application useable. Cross-origin resource sharing (CORS) is a mechanism that allows restricted resources on a web page to be requested from another domain outside the domain from which the first resource was served. In its current form, the application will refuse to return information to the user's requests unless CORS is enabled. To do this we downloaded the Moesif CORS extention for Chrome, which allows for CORS to be enabled in the click of a button.

4.6 Conclusions & Future Work

In conclusion, we have used Solr Apache as a Database to store our entity data, and a search engine to navigate through our matches. We have examined the preexisting UIs, and developed our own web application that best fits our needs. We feel that the end result is a decent application for commercial use.

In the future, the application could be further refined, and made to to work for android and IOS, as it currently only works for web browsing. It can also be made to work together with other applications that provide grocery shopping delivery services, in order to lessen delivery times.

Chapter 5

Basket Optimization

In this final chapter, we will look into our method for basket optimization. We have examined the basic features of our web application, and have made it functional enough to be used by a user. We will now introduce a mechanism wherein we will assist the user in finding the best combination when it comes to the source of the products he's looking for, in a way that we will optimize both the price and the feasibility of their orders.

5.1 The Basket

To recap what we have already shown about the basket, our working features are:

1. The sets of + and - buttons for our hardset and softset items
2. Total price and amount
3. Sum for each individual Store
4. Optimization button

In this chapter we will focus on the Optimization option and the algorithm we use to optimize our basket.

With the rest of the features in place we can utilize the hardset (placed with Add to Cart option) items in our basket, but we still don't take advantage of the options that the Auto Choose button offers us with the softset items. Ideally, we would like to offer an optimization algorithm to the user that will take into account the items they have noted they want as-is, and optimize the other part of the bought items, the ones the user is willing to have optimized.

The former can be ignored for the most part, as the only contribution to the final basket is their impact on the minimum delivery costs. We still need to take them into account however, as in the case where even one item is hardset from a store source we must fulfill the minimum delivery cost for that store, otherwise the order will not go through.

The latter is the part of the basket where most of the optimization will happen.

5.2 The Simplex Algorithm

The algorithm we used to for basket optimization is the Simplex algorithm[15] [21] [4], an algorithm used for linear programming problems. It has three inputs:

1. The target for optimization, along with optimization type
2. The variables
3. Restrictions, or constraints

In other words, the algorithm checks which values should be used for the variables in order to optimize the target, which making sure that those values are within the constraints we have set. In our case, the target for optimization would be the cost of the sum of items the user purchases and the optimization type would be minimization. The variables would be the amount of each item in the basket as well as the amount of alternative options. Finally, the constraints would include two parts:

1. The amount of items must be equal to the number the user has defined for each type of item. If the user has chosen the auto choose option for 3 of the same coockies, we are allowed to try combinations between the items that have matched that search, but only up to a quantity of 3, and not outside of those options.
2. Each store must have a sum of bought item costs equal to or larger than its minimum delivery cost. These costs are adjusted by taking into account the prices of the items the user has manually put in the basket with the "Add to Cart" option.

While not technically a constraint, another thing that should be taken into account is that if the user has placed one item manually from any store in his basket, we are obligated to try and fulfill that store's delivery cost, even if it means increasing the price. If, on the other hand, a store doesn't have any items placed manually in the basket from it, we are free to even completely exclude it from the final selection of items if it means reducing the cost.

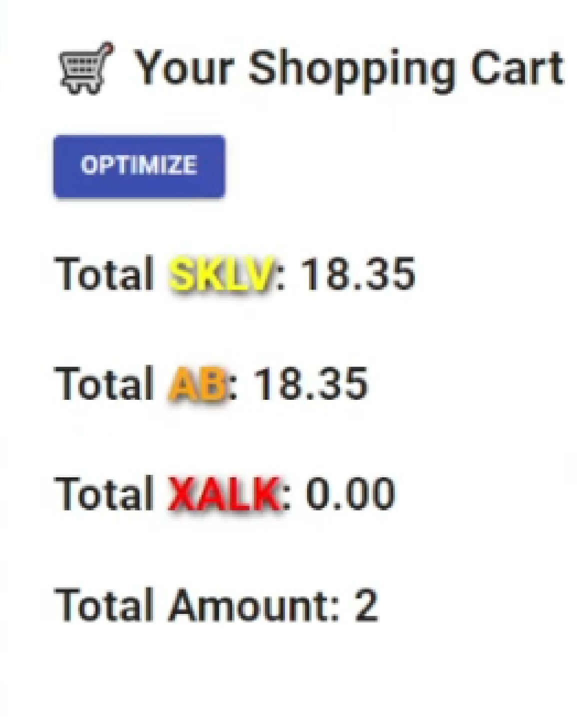
5.3 Results

Below we show examples of the algorithm working as intended, and we review the process, but first a reminder that minimum costs for the grocery stores are:

- 15 euros for Sklavenitis
- 30 euros for AB Vasilopoulos
- 45 euros for Chalkiadakis

5.3.1 Example 1

In the first example we have purchased 2 matching products. The products are exactly the same with the exceptions of their store of origin, and the way they were added to the cart. The first product was added manually with the "Add to cart" option, and was from AB, while the second one was purchased automatically using the "Auto Choose" option, and was from Sklavenitis.



Your Shopping Cart

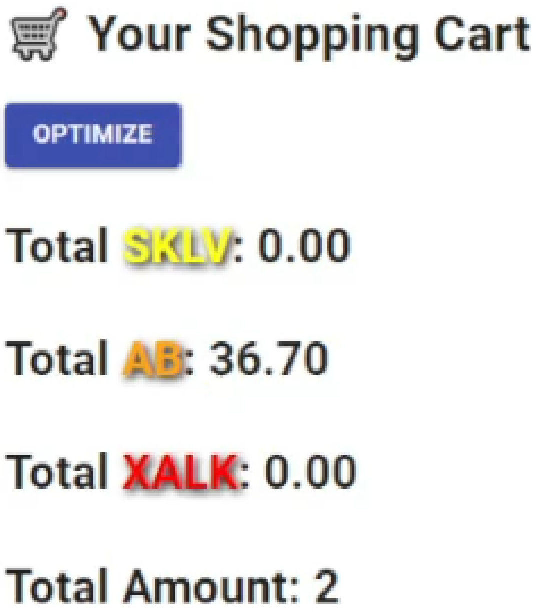
Total SKLV: 18.35

Total AB: 18.35

Total XALK: 0.00

Total Amount: 2

Because we do not meet the minimum delivery costs for both stores, delivery would be impossible as is. Based on what we have explained so far, using the optimization button should reorganize our basket in a way that will fulfill any requirements if possible. Because we have manually added the product from AB, we expect the algorithm to transfer the product from Sklavenitis to AB, which would take us over the minimum delivery cost. Indeed after we optimize the basket, this is the result:



🛒 Your Shopping Cart

Total SKLV: 0.00

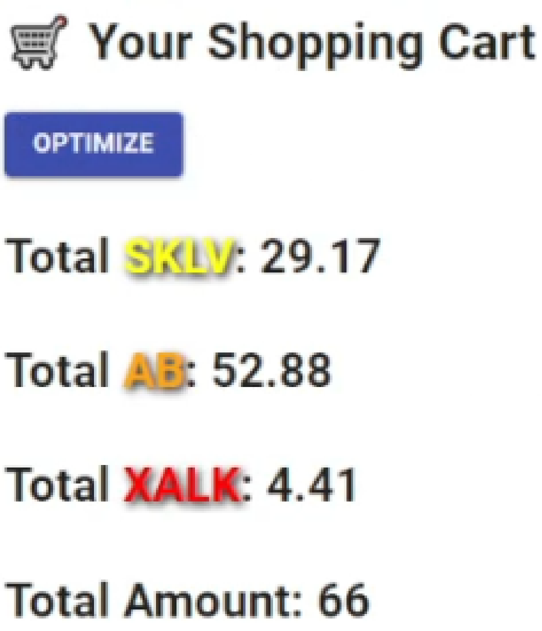
Total AB: 36.70

Total XALK: 0.00

Total Amount: 2

5.3.2 Example 2

In the second example we present a more realistic scenario, that more accurately represents a shopping cart. We have randomly added 66 products with the ”Auto Choose” button, most multiple times, and added a twist: one of those products was only available from Chalkiadakis.



Your Shopping Cart

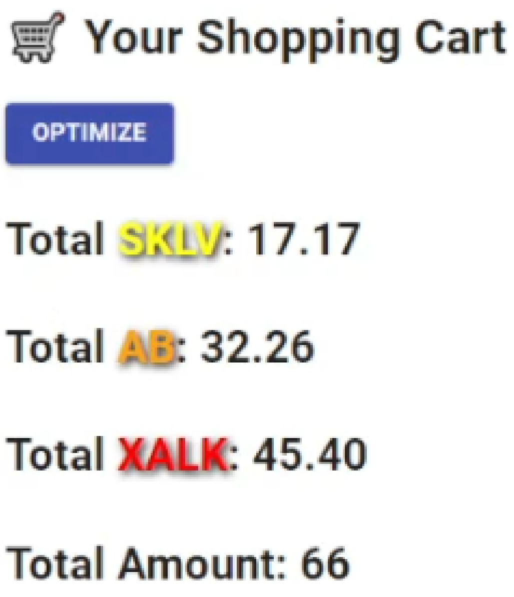
Total SKLV: 29.17

Total AB: 52.88

Total XALK: 4.41

Total Amount: 66

We can see the total price for these items comes to 86.46 euros. Before optimization, the ”Auto Choose” option finds the cheapest option for each product, so this is the best price we can get for each individual product. We can also see, however, that we do not meet the requirement for delivery for Chalkiadakis, so we have to use the optimization option. Because one item has to be purchased from Chalkiadakis, we expect the basket to be reorganized in a way that items are taken from the other two stores and added there. These are the optimization results:



🛒 Your Shopping Cart

Total SKLV: 17.17

Total AB: 32.26

Total XALK: 45.40

Total Amount: 66

We can see that the new price adds up to 94.83 euros, which is 8.37 euros more than the previous one, but this time all requirements for delivery are met. The algorithm found cheaper option in the first two stores, so it tries to maintain as many options from them, while also fulfilling the requirements for Chalkiadakis. With 66 products in their basket, it is most likely that the user would not have been able to reach this combination without assistance.

5.4 Conclusions

To conclude, using the simplex algorithm we are able to find the cheapest combination of items to offer to the user, while also taking into account any constraints we may have. It is efficient in both accuracy and time, as the user needs to quickly access their basket and optimize it, maybe even multiple times. All in all, it offers a great way for assisting the use in their shopping.

Chapter 6

Conclusion

This chapter presents the conclusions of this thesis, along with a brief review of what we have shown, some facts about the thesis process, and related work that we found during its development.

6.1 Summary of Thesis Achievements

In this thesis, we collected data from 3 different e-shops. We experimented with both static crawling with beautiful soup, and using an automated browser crawler with selenium.

We used entity resolution to find multiple instances of the same objects from different sources and match them together using a combination of natural language processing and machine learning, until we reached results that were satisfying in their accuracy.

We then utilized Apache Solr as a database to store our newly edited data, and a search engine to navigate through all our matched entities.

We Created a user interface using React Javascript to offer a user friendly environment and add options to facilitate the user in their shopping.

Finally, we introduced a basket optimization algorithm with the simplex method to find the optimal combination of products for the user's needs while also meeting the stores' requirements for delivery.

Although the final product is much more compact, this thesis took about 8,000 lines of code in total during its development.

6.2 Related Work

During our research and development of this project we came across other similar ideas that have been implemented commercially [8].

Bibliography

[1] Anant. https://github.com/Anant/awesome-solr. [accessed 13/2/2023].

[2] Apache Solr v.8.11.2. https://solr.apache.org/. [accessed 13/2/2023].

[3] HTTrack v.3.49-2. https://www.httrack.com/. [accessed 13/2/2023].

[4] javascript lp solver. https://www.npmjs.com/package/javascript-lp-solver. [accessed 13/2/2023].

[5] Random Forest Classifier. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html. [accessed 13/2/2023].

[6] React JS v.18.2.0. https://reactjs.org/. [accessed 13/2/2023].

[7] Selenium v.4.8.0. https://www.selenium.dev/. [accessed 13/2/2023].

[8] Skroutz Smart Basket. https://www.skroutz.gr/blog/posts/408-to-exypno-kalathi-tou-skroutz-irthe-kai-sou-lynei-ta-cheria. [accessed 13/2/2023].

[9] Support Vector Machines. https://scikit-learn.org/stable/modules/svm.html. [accessed 13/2/2023].

[10] TfidfVectorizer. https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html. [accessed 13/2/2023].

[11] J. Beel, B. Gipp, S. Langer, and C. Breitinger. Paper recommender systems: a literature survey. *International Journal on Digital Libraries*, 17(4):305–338, 2016.

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

[13] V. Christophides, V. Efthymiou, and K. Stefanidis. Entity resolution in the web of data. *Synthesis Lectures on the Semantic Web*, 5(3):1–122, 2015.

[14] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[15] J. Dongarra and F. Sullivan. Guest editors introduction to the top 10 algorithms. *Computing in Science amp; Engineering*, 2(01):22–23, jan 2000.

[16] K. S. Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 1972.

[17] A. R. Lahitani, A. E. Permanasari, and N. A. Setiawan. Cosine similarity to determine similarity measure: Study case in online essay assessment. In *2016 4th International Conference on Cyber and IT Service Management*, pages 1–6. IEEE, 2016.

[18] R. Lawson. *Web scraping with Python*. Packt Publishing Ltd, 2015.

[19] H. P. Luhn. A statistical approach to mechanized encoding and searching of literary information. *IBM Journal of research and development*, 1(4):309–317, 1957.

[20] M. Mersinias, S. Afantenos, and G. Chalkiadakis. CLFD: A novel vectorization technique and its application in fake news detection. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 3475–3483, Marseille, France, May 2020. European Language Resources Association.

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

[22] S. Nyamathulla, P. Ratnababu, N. S. Shaik, et al. A review on selenium web driver with python. *Annals of the Romanian Society for Cell Biology*, pages 16760–16768, 2021.

[23] L. Richardson. Beautiful soup documentation, 2007.

[24] S. R. Safavian and D. Landgrebe. A survey of decision tree classifier methodology. *IEEE transactions on systems, man, and cybernetics*, 21(3):660–674, 1991.

[25] C. Zheng, G. He, and Z. Peng. A study of web information extraction technology based on beautiful soup. *J. Comput.*, 10(6):381–387, 2015.