Heuristic Approach for Content Based Recommendation System Based on Feature Weighting and LSH

Beleveslis Dimosthenis

SID: 3308170002

SCHOOL OF SCIENCE & TECHNOLOGY

A thesis submitted for the degree of

Master of Science (MSc) in Data Science

DECEMBER 2019

THESSALONIKI – GREECE

INTERNATIONAL
HELLENIC
UNIVERSITY

Heuristic Approach for Content Based Recommendation System Based on Feature Weighting and LSH

Beleveslis Dimosthenis

SID: 3308170002

Supervisor:	Prof. Christos Tjortjis
Supervising Committee Members:	Dr C. Berberidis
	Dr M. Gatzianas

SCHOOL OF SCIENCE & TECHNOLOGY

A thesis submitted for the degree of

Master of Science (MSc) in Data Science

DECEMBER 2019

THESSALONIKI – GREECE

Abstract

This dissertation was written as a part of the MSc in Data Science at the International Hellenic University. The aim of this research is to design an efficient content-based product recommendation system in the field of e-commerce. A heuristic content-based approach that incorporates feature weighting and locality-sensitive hashing (LSH) is proposed. The recommender system uses the TF-IDF method and incorporates a functionality of tuning the importance of product features in order to adjust its logic to the requirements and needs of different e-commerce sites. The problem of efficiently producing recommendations, without compromising similarity, is addressed by approximating product similarities via the LSH technique.

The system is implemented and evaluated based on two datasets that include real e-commerce data. Specifically, product details and customer actions from two e-commerce sites are used in order to conduct various scenarios and tests. The results and the evaluation of the proposed methodology show that the recommendations can improve the user experience and increase the product sales. Finally, it turns out that the system incorporates recommendation diversity which can be adjusted by tuning the appropriate feature weights.

Beleveslis Dimosthenis

02/12/2019

Acknowledgements

I would first like to thank my supervisor Dr. Christos Tjortjis for his excellent guidance and his important help during the implementation of this dissertation. I would also like to thank the International Hellenic University for giving me the opportunity to undertake this dissertation. Lastly, I am grateful to my family and friends who have supported me along the way.

Contents

1 Introduction

A Recommendation or Recommender System (RS) mainly refers to an intelligent system that produces suggestions about items to users. In particular, its role is to predict items that are likely to be of interest to the user. They have turned out to be essential due to the big increase of options that users have nowadays while using every aspect of the web. With the appearance of big data, the problem of selecting the right information arose. Recommendation systems are used to solve this problem and improve the online experience. They provide personalized information by learning the user's interests through his/her interaction with items.

Over the past years, Recommendation Systems have turned out to be very popular as we come across them almost everywhere in the web (e-commerce, movie sites, music sites etc.). The recommended items can potentially be anything (products, movies, songs, services etc.) that a user is looking for. There are plenty of applications and domains that such systems are used in and produce significant results. For example, it is important for an e-shop to recommend the right products to each customer and for an insurance company to recommend the appropriate plan to each client.

The way that these recommendations are produced is of great interest. It certainly depends on the specific domain and the desired results. In general, they are based on item similarities, user preferences, past purchases and other actions related to the items and the users. Over the past decades, recommender systems have become an important research area and much work has been done on developing new approaches. In order to predict good recommendations, the system needs to know some important information about the items and the users. In other words, these refers to the 'item profile' and 'user profile' respectively. The item profile corresponds to the characteristics of the item. These characteristics may be a description, specific attributes, keywords etc. The user profile mainly refers to the information that summarize the preferences of the user.

1.1 Problem

E-commerce is one of the areas that recommendation systems are being used extensively. The role of a recommender system in an e-shop is similar to the role of a salesperson in a physical store. Online stores need to offer to customers, a similar or even better shopping experience than that of

an offline store. A recommendation system helps in that direction by providing customers with online product recommendations that help them find what they need easily. This will result in customer satisfaction and engagement with the products and the e-shop. Consequently, upselling, cross selling, less cart abandonments and overall improvement in other KPIs, such as daily sales, are accomplished.

A customer can see product recommendations in various pages of an e-commerce site. One of them is the ‘product page’ which is the page where users visit to see a specific product along with its detailed description and features. The user can add the product to his/her cart or visit an alternative product page that better matches his/her preferences. The main aim of the system on such a page is to display relevant items and help customers to continue browsing the site by providing them with the necessary “next step”. The recommended products have to be similar to the one of the product pages, by taking into consideration the importance of their characteristics and features. However, it is important to achieve diversity in the recommendations because the customer doesn’t need to see a list of almost the same products. For example, assuming a user that is seeing a laptop of a specific brand and all the recommendations correspond to similar laptops of the same brand. In this case, the user will not be able to visit the product page of a laptop of an alternative brand by navigating through the recommendations.

Furthermore, the logic of such a recommendation system relies on the idea of finding similar products by calculating similarities among them. This is a computationally heavy and time-consuming procedure when an e-commerce site has thousands of products. Calculating the similarity of between a product and all the rest products of an e-shop, in order to find the most similar products, is usually inefficient. So, it is important to apply more sophisticated methods that will reduce the complexity and computational cost without compromising the accuracy of the system.

1.2 Scope and contribution

The scope of this research is to implement a content-based recommendation system suitable to the 'product page' of an e-commerce site. Specifically, we address the problem of product representation by incorporating a weighted method that offers the functionality to customize the importance of product features. In that way, the logic of the system is adjusted to the needs of the e-commerce site. In addition, diversity of the recommendations is accomplished without compromising the similarity. Furthermore, this study is trying to address the problem of efficient product similarity calculations for e-commerce sites that dispose thousands of products. This problem is tackled by incorporating locality sensitive hashing method. Finally, our implemented heuristic approach is being evaluated based on real data from two e-commerce sites.

1.3 Dissertation outline

In the first chapter, the concept of recommendation systems was introduced. We also defined the problem that the current research addresses and we presented the scope of the dissertation. Finally, we provide its structure by mentioning the key points of each chapter.

In the second chapter, we introduce the core concepts of recommendation systems and we provide the background knowledge that is necessary in order to follow the dissertation. We present the basic types of recommender systems and their appliance in e-commerce. Furthermore, we refer to the concept of similarity calculations and locality-sensitive hashing. Finally, we discuss about weighted methods that are applied in similar systems and are used in the proposed approach.

In the third chapter, we briefly review some previous research efforts related to recommender systems and locality sensitive hashing. We also present previous works that incorporated weighted methods in the field of RS.

In the fourth chapter, we describe in detail the two datasets that are used in the current research. We provide information about their content and present the preprocessing steps that were followed. We also describe the way that they were initially created.

In the fifth chapter, we provide details of the methodology that was followed in order to build the proposed recommendation system. The weighted method that was applied is presented in detail. We also present the process that was followed in order to incorporate the LSH method.

In the sixth chapter, we present the experiments that were conducted in order to test the proposed approach based on real data. The results of the experiments are demonstrated and explained.

In the seventh chapter, we evaluate the proposed recommendation system by further analyzing the results of the experiments that were conducted. We also examine the diversity of the generated recommendations and we perform a session-based analysis. In addition, we present the conclusions that have been drawn and we discuss about our plans to improve the system.

2 Background

In this chapter, we introduce the core concepts of recommendation systems and we provide the background knowledge that is necessary in order to follow the dissertation. We present the basic types of recommender systems and their appliance in e-commerce. Furthermore, we refer to the concept of similarity calculations and locality-sensitive hashing. Finally, we discuss about weighted methods that are applied in similar systems and are used in the proposed approach.

2.1 Types of Recommendation Systems

The rapid increase of the need to apply recommendation systems almost everywhere in the web is combined with the deep research of the issue. The interest in this area still remains high due to the growing demand on practical applications and different approaches have been developed. However, all the new approaches are based on three major types of recommendation systems. The recommendation techniques are usually classified into the following types, based on the logic that the recommendations are generated [1], [2], [3], [31].

2.1.1 Content Based Filtering

This type of recommendations is mainly based on the content of the items. The system recommends items that are similar to items that a user interacted with (viewed, liked, purchased) in the past. In content-based filtering, the user is represented by his/her profile. This profile represents the interests of the user based on previous interactions with items. The user may express his/her interests through implicit or explicit actions. Each item is also represented by a profile that consists of the important information that summarizes it and characterizes it [38]. So, the system tries to find the item profiles that best match the user profile. These item profiles correspond to the items that the user would like to see or buy. For example, assuming that the attributes of product A and product B are very similar and a user liked product A. Then the system will recommend product B to the user.

Advantages [1]

- *User independence* - Considering the difficulty in gathering ratings about items, content-based filtering can be applied easier than collaborative filtering as it is independent from other users. The profile of the user that the recommendations are made for, is enough.
- *New item* - Content-based approach can be easily adapted to new items. Even if a new item with no ratings comes in, the system can recommend it to a user. This happens because the system just relies on the content of the item, so based on its characteristics it will match it with older items.
- *Transparency* - Content-based filtering offers transparency to the evaluation process of the system. The reason that an item is recommended to a user can clearly be explained based on the content of the item and the specific user profile. While in collaborative filtering, the explanation of recommendations is hidden behind ratings of many unknown users.
- *Text mining* - Considering the fact that the content of an item can be represented as text, this approach opens up the options to use various text mining methods.

Disadvantages [1]

- *Over specialization* – Content-based approach recommends items that are very similar to items that the user has already interacted with (liked, purchased etc.) in the past. Hence, it provides a limit degree of novelty and diversity, since it has to match up the user and the item profiles.
- *Limited content analysis* – This type of recommendation systems requires representative information about the items. The less information about the content of the product, the harder it is for the system to discriminate the items precisely. As a result, it is difficult for the system to discriminate items that the user likes from items that the user does not like. Furthermore, for the best exploitation of the information, the domain knowledge is often necessary.
- *New user* – A problem arises when a new user comes in. The system needs information about the user preferences in order to create the corresponding user profile. So, the user needs to start interacting with items in order to express his/her preferences and help the system produce personalized recommendations.

2.1.2 Collaborative Filtering

This type of recommendations is mainly based on users' opinions and ratings (implicit or explicit) about items. The system recommends items that similar users like. So, in order to produce accurate results, it is necessary to have ratings of items. The ratings can be presented as a user-item matrix that shows the items that each user likes or dislikes. Based on this matrix, similarities between users are calculated. The basic premise of CF is that if two users have the same opinion about a bunch of products, then they are likely to have similar opinions about other products too [37]. For example, two users that have highly rated the same movies will be considered as similar. This method produces recommendations with higher diversity than content-based filtering.

Advantages [2]

- *Diversity* – Collaborative filtering relies on the idea that a user will be recommended items, that people with similar tastes and preferences to him/her, liked in the past. Based on that, the user might be recommended items that he never liked before, but other similar users did. So, there is much more diversity in recommendations than in content-based approaches. This gives to the user the opportunity to discover new preferences.
- *Adaptability* – This type of RS can be applied in any type of items and domain without having to deal with the different content of the products. It can even produce very good results in domains where there is not much content associated with items and where the content is difficult for a computer system to be analyzed.

Disadvantages [2]

- *Cold-start problem* – This problem refers to the situation where the system does not have adequate information about an item or a user. When a new item comes in, there are no past raters for it. So, it cannot be recommended to new users. Also, similar problem occurs when a new user comes in as the system doesn't have information about his/her preferences.
- *Data sparsity problem* – Almost always, each user rates only a small percentage of the available items. This leads to difficulty in finding users that have rated the same items. In addition, the data sparsity decreases the percentage of the available items that can be recommended.

2.1.3 Hybrid methods

This type of recommendation systems combines the content based and collaborative filtering that were described previously. The idea behind hybrid techniques is that a combination of both approaches can solve the major problems that each technique has. A basic problem of collaborative filtering approach is the cold start problem. This problem refers to the difficulty in recommending an item that has not been rated in the past. Content-based filtering that doesn't need ratings could be a solution to the cold start problem. So, hybrid methods have been developed in order to overcome certain limitations that the two main approaches have [3]. Specifically, there are three main approaches of building hybrid methods, based on the way that the two main types are combined:

a) implementing collaborative and content-based methods separately and combining their predictions,
b) incorporating some content-based characteristics into a collaborative approach,
c) incorporating some collaborative characteristics into a content-based approach

2.2 Recommendation Systems in e-commerce

E-commerce is one of the areas that recommendation systems are being used extensively. Marketers are trying to create a great customer experience by taking advantage of such intelligent systems. The role of a recommender system in an e-shop is similar to the role of a salesperson in a physical store. In a bricks and mortar store, a sales assistant is there to help customers find products that match their needs and wishes. Respectively, an e-shop needs a RS to offer to customers an online shopping experience similar to the one that they would have in a traditional store. In particular, the system exploits data about the products and the way that customers interacted with them. Based on that, more personalized product recommendations are presented to the users. Furthermore, the appearance of big data in e-commerce made recommendation systems essential, as they offer an efficient way to take advantage of all this data in order to improve the customers shopping experience and increase revenues.

2.2.1 Benefits

Personalization is becoming increasingly important in online marketing and the benefits of a recommender system are both for the customers and the e-shops. E-commerce sites use recommender systems with the aim to produce personalized suggestions to the customers about products and services [4], [31].

User Experience - An intelligent product recommender system produces the right suggestions to the right customers. A customer prefers to see products that he/she likes or needs. In that way his/her shopping journey becomes easier and more pleasant. So, instead of leaving the customer in the chaos of thousands of products, a recommendation system helps him/her to find what he/she is looking for fast and efficiently. Hence, the user feels more comfortable while purchasing products online.

Loyalty – With the numerous options that are available in e-commerce, loyalty is very important for sites. An e-shop needs to offer to the customers an online experience that will make them return in the future. Loyalty is improved through recommendation systems by creating a value-added relationship between the site and the customer. By making personalized recommendations that match customer needs, the customer feels valued and finds the website user-friendly. A satisfied customer will keep returning to the e-shop and the engagement will be enhanced.

Sales – Incorporating and using a recommender system can result in an increase in profits. Being closer to customers by making personalized recommendations leads to higher revenue. Many times, visitors leave the site before finding products that are appropriate to their needs. A RS solves this problem as it can lead the visitor to specific products that are of his/her interest fast. Furthermore, product suggestions can be additional to already selected products. In that way, cross-selling is improved, and the average order size is increased.

2.2.2 Use cases

The main role of a recommender system in an e-commerce site relies on the idea that the visitors should see relevant products that will help them find what they need fast and in a pleasant way. These products can be relevant to their tastes or can help them discover new preferences. Some sites use non personalized approaches to make product suggestions that do not change according to each item or customer profile. Other sites use more sophisticated systems that produce personalized recommendations that focus on the characteristics of the customers and products. Basic use cases of such systems are described below. [32]

Non-personalized: Product recommendations are based on specific filters, without taking advantage of the content of the products and the way that each customer interacts with them. This kind of recommendations are inevitably in cases that there is no information about the customer profiles or purchase history. In addition, this logic can be applied when products with specific characteristics must be promoted.

- *Popular products* – The suggestions correspond to popular products, incorporating basic filters such as product categories. In fact, purchase history is important here as it holds the information about the best-selling products among each product category.
- *New products* – Another way to take advantage of a recommendation system is to promote new items that are in stock. It is often difficult to start boosting new products and a RS can offer a valuable solution.
- *Slow moving inventory* – A usual situation in e-shops is having products that are not selling as much as others. A simple way to boost the sales of such products is to present them more often to the customers via a recommender system.

- *Special offers and seasonality* – In the same logic, a recommender system can have the role of a campaign for products that are in discount or should be high selling during specific seasons. For example, presenting to customers sun care creams that are in discount during summer can attract their attention.

Personalized: A more sophisticated and intelligent recommender system produces personalized suggestions that focus on the needs and preferences of the customers. It takes into account both the characteristics of the products and the customer profile that is created based on his/her online habits. This kind of recommendations are clearly more effective and produce more accurate results. In this case, it is essential to have product and customer data in order to present relevant suggestions that will increase engagement and conversions.

- *Recently viewed* – This type of recommendations doesn't require any in depth data about the customer. The idea relies on the fact that a customer may express his/her interest about an item while browsing the site but get distracted from other products. So, based on that idea, it is reasonable to remind him of the products that he may be interested in.
- *Customers who viewed this item also viewed* – Associations between products are created based on data that are generated while customers are browsing through the product pages. These associations can be used in order to suggest to a customer the next available product that he should see.
- *Customers who bought this item also bought* – Respectively, product association rules can be generated based on purchase history. This kind of suggestions aim to improve the cross-selling. It tries to increase the average order value by offering products that go along with the customer current purchase.
- *View this item next* – Based on the product profiles that consists of their important characteristics and features, the system suggests similar products with this that the customer is currently seeing. The idea is to help customer find other similar products that may match to their preferences and needs. The current research focuses on this kind of logic.

The above kinds of recommendations can be displayed at plenty of places on an e-commerce site. Two common places are the product and the cart page of the site [33]. The first corresponds to the recommends that are produced while the customer is still searching for items. While the second

corresponds to the page where the customer is completing his/her purchase. These two kinds of RS and their logic are described in the following chapters.

2.2.3 Product Page RS

The product page corresponds to the page where the customer is viewing a product and can find all the available information of it. Detailed description, product pictures and price are some of the products features that a customer expects to see in the product page. Assuming that the customer doesn't leave the site, he has two options there. The first is to add the product in his/her cart and the second is to look for another product. Most of the times the second option is selected as the customer is searching for that product that best matches his/her needs. So, it is important to provide the customers with an efficient way to choose their 'next step' and keep them browsing the site. This can be accomplished through a RS that is able to make searching a lot easier for the users. Instead of clicking through many pages to find the right product, the aim of the product suggestions is to exclude the irrelevant ones so that those displayed are the most appropriate for the user.

In particular, the main objective is to produce specialized product recommendations for each specific product page, taking into consideration the user's profile. The main target is to help users find what they are looking for easily and make the navigation through the product pages efficient. This helps customers stay more time in the e-shop and finally increases the probability to purchase products.

Many e-shops make recommendations based on simple aspects. A commonly adopted approach is to recommend popular products by applying basic filters about the product category, color, brand etc. But this naïve approach leads to undesirable results as many times the user gets disturbed by products that are irrelevant to the ones that he has already seen. However, great results can be produced by combining such simple methods with more complexed exploitation of product metadata. Taking into consideration the fact that the customer wishes to visit a product page that better matches his/her preferences, the system must recommend several options of similar products to the one that corresponds to the current product page. Therefore, the logic of the system should be based on the content of the products and not on past purchases or ratings. Hence, content-based filtering is more appropriate in product page recommendation system. This dissertation aims to propose such an approach.

2.2.4 Cart Page RS

The cart page corresponds to the place where the customer has already selected the items that he/she wants to buy and completes his/her purchase. That step is a brilliant opportunity for the marketeers to increase cross-selling by recommending products that can be purchased along with the already selected ones. So, the idea behind those recommendations is completely different to that in a product page. By recommending just similar products with the cart products, the possibility that the customer will replace the already chosen products with the recommended ones arises. A RS in this specific place of the e-commerce site tries to lure the customer into increasing the average order value by offering products that go along with his/her current purchase. A recommender system works like a brilliant salesman who is well trained in cross selling and upselling.

Since, the main objective is to recommend products that could potentially be purchased along with the already chosen products, it is important to take into consideration the historic data about purchases and user preferences. Such data can reveal associations between products and customers. A common type of association is 'Customers that buy products A and B, also buy product C'. The more personalized the recommendations are the more sophisticated and accurate the system is considered to be. Significant results are produced when RS combine collaborative filtering approaches with other methods that take into consideration purchase history and past sales.

2.3 Similarities & LSH

Information explosion has led to an increasing number of applications that need to deal with large volumes of data. Although traditional algorithm analysis considers that the data fits in main memory, this is not possible when dealing with massive data sets, such as multimedia data, web page repositories, etc. The same issue appears in the domain of recommendation systems, that need to handle large amounts of data [5]. Usually, the main objective of a recommendation system is to calculate similarities among a set of items and/or users. Such items can be products, documents, songs etc. A RS taking into consideration the representation of the items should be able to identify which of them are similar in order to make appropriate recommendations. A similarity search problem includes a big group of items (products, documents, etc.) that are represented by a set of characteristics, as points in a high-dimensional attribute space. A particularly interesting and well-studied instance is d-dimensional Euclidean space. This important issue appears in various applications, including data mining, information retrieval and more specifically RS. Usually, the items of interest (products, documents, etc.) are represented as points and a distance metric is used to calculate the similarity among them. The number of product attributes corresponds to the dimensionality of the problem and ranges widely. Many accurate solutions have been developed for low-dimensional cases. However, for high-dimensional cases the issue still remains, as improvement can be accomplished.

Approximation has been proposed in several researches as a solution to the problem of scalability. An interesting approach in order to handle the issue of high dimensionality are approximate nearest neighbor (ANN) (Definition 2) algorithms. Such algorithms improve dramatically the search speed and are necessary in high-dimensional cases. They rely on the idea that calculating similarities by applying the approximate nearest neighbor is almost as accurate as using the common procedure of nearest neighbor (NN) (Definition 1). This is a reliable approach in cases where small errors, caused by approximation, are not significant [6]. Locality sensitive hashing (LSH) and its variations have been proposed as techniques for approximate similarity search in cases of high dimensionality. This method uses a family of locality-sensitive hash functions to hash nearby objects in the high-dimensional space into the same bucket. A large number of hash functions is needed in order to achieve good search quality [7].

Definition 1 (Nearest Neighbor or NN) [8]. Given a query object q, the goal is to find an object NN(q), called nearest neighbor, from a set of objects $P = \{p_1, p_2, \cdots, p_N\}$ so that $NN(q) = \arg \min_{p \in P} dist(q, p)$, where dist(q, p) is a distance between q and p.

Definition 2 (c-Approximate Near Neighbor or ANN) [8]. Given a query object q, the distance threshold $R > 0$, $0 < \delta < 1$ and $c \geq 1$, the goal is to report some cR-near neighbor of q with the probability of $1-\delta$ if there exists an R-near neighbor of q from a set of objects $P = \{p_1, p_2, \cdots, p_N\}$.

There are various techniques in order to calculate the similarities between items such as cosine and Jaccard similarity. Traditional RS have been designed to handle this issue by incorporating simple similarity methods where each pair of items is a candidate. But the problem of scalability arises due to big data. Nowadays, it is common to face the problem of dimensionality which is often called "curse of dimensionality". The exponential growth of data brought those traditional approaches across the problem of scalability. For example, in a RS of an e-shop with hundreds of thousands of products, calculating similarities between all of them is inefficient. Specifically, it is time consuming and computationally heavy. Hence, LSH is a method that can be applied in RS, aiming to tackle the scalability problem.

2.3.1 Jaccard Similarity

Many applications, including RS, adopt the bag-of-words model [9] as representation of textual data. The Jaccard similarity has been widely applied in order to calculate similarities among items that are represented in the bag-of-words model. The Jaccard similarity is statistically a measure of comparing the similarity of two binary sets. Jaccard index is often used for comparing similarity, dissimilarity, and distance of the data set. Calculating the Jaccard similarity between two data sets is the result of division between the number of common elements and the number of all elements in the sets.

Definition 3 (Jaccard Similarity) [8]. Given two sets S and T, the Jaccard similarity is defined as

$$J(S,T) = \frac{|S \cap T|}{|S \cup T|}$$

The above definition assumes that all elements are equally important. But in many cases, some of the elements are more important than others and this is represented by assigning weights to them. For the case of such weighted sets, we need to define the generalized Jaccard similarity as follows.

Definition 4 (Generalized Jaccard Similarity) [8]**.** Given two sets $S = [S_1, \cdots, S_n]$ and $T = [T_1, \cdots, T_n]$ with all real weights S_k, $T_k \geq 0$ for $k \in \{1, \ldots, n\}$, the generalized Jaccard similarity is defined as

$$J_{gen}(S, T) = \frac{\sum_k \min(S_K, T_k)}{\sum_k \max(S_K, T_k)}$$

The weighted Jaccard similarity is a natural generalization of Jaccard similarity. It will become Jaccard similarity if all token weights are set as 1.0.

Usually, such representations lead to large dimensionality and calculating similarities among items turn to be computationally heavy. In the case of a RS, there may be thousands of items and each of the item profile may consist of hundreds of keywords or features. Therefore, a large dictionary and a sparse matrix is produced. So, a more efficient way to represent items and users' profiles should be used that will make the whole process faster. Such an approach is the representation of items by their signatures that are created by hash functions. The Minhash algorithm, which is a well-known LSH algorithm, is used to estimate the Jaccard similarity and is described in the following paragraphs.

2.3.2 Random permutations

Assuming a universal set U and its subset $S \subseteq U$, we can define S as a binary or weighted set as follows. The subset S is a binary one if each element of the set S ($k \in S$) has weight equal to 1 ($S_k = 1$), while any other element ($k \in U - S$) has weight equal to 0 ($S_k = 0$). Respectively, the subset S is a weighted one if each element of the set S ($k \in S$) has weight greater than 0 ($S_k > 0$), while any other element ($k \in U - S$) has weight equal to 0 ($S_k = 0$). The set or every subset can be represented as vectors with length equal to the number of the elements of the universal set. For example, $U = [U_1, U_2, \cdots, U_n]$ and $S = [S_1, S_2, \cdots, S_n]$ [8].

A random permutation can be performed on such a set, but this can be very complex. Therefore, an approximation of the random permutation can be accomplished by uniformly and injectively mapping each element of the set into the real axis. Specifically, each element is assigned with a unique hash value $v \in R$. Uniform mapping on binary sets can also be extended into weighted sets. A random permutation or uniform mapping can be applied on the universal set U or to a subset S and the most significant element will be the first one in the leftmost. Taking the first element can be considered as a hash function, i.e., $h(S) = \min(\pi(S))$ where π represents the random permutation. A fingerprint that consists of hash values is obtained if we repeat the above process. The number of hash values is the same with the number of repetitions of the process [8].

In the case of a RS, the universal set may consist of all the words that appear in the titles of the items. While a subset can be a specific item with a number of elements equal to the number of unique words of its title. So, random permutations map each word from the dictionary to a different number. Hash functions basically does the same as the permutation, by mapping a word to a number. But in the case of hash functions, the whole dictionary is not needed in advance. So, when a new item or user profile appears, its signature can be calculated fast. Random permutations with hash functions as described above are a significant step in order to create the representation of each item as a signature. Then the signatures can easier be compared with each other than comparing the initial sets of each item. The signature representation can be accomplished through the Minhash algorithm.

2.3.3 Minhash

Two sets are near duplicate if their similarity score is above a predefined threshold. Calculating the similarity scores among sets through simple methods, such as Jaccard similarity, is usually inefficient due to high dimensionality. Minhash is an efficient LSH algorithm that have been successively proposed to approximate such similarity calculations. In particular, Minhash is used to approximate the Jaccard similarity of two sets. It is proved that the Jaccard similarity of two sets is equal to the probability that the two sets will generate the same Minhash value (hash collision). The Minhash method has been applied in many domains such as news recommendation [10], near duplicate web page detection [11] and image search [12].

In order to estimate how similar two sets are, a number of independent Minhash functions are applied to the sets. These functions correspond to the random permutations of their elements. A Minhash is a single number having the property that two sets have the same value of Minhash with probability equal to their similarity [13].

Assuming a universal set U and a set of hash functions we need to estimate similarity between different subsets S1, S2 ⊆ U. So, the hash functions (random permutations) are applied to U. As a result, the Minhashes for a subset S1 are the elements in it which have the minimum hash value in each hash function. In other words, the Minhashes for a subset S1 are the elements which are placed in the first position of each permutation [8]. In order to make the retrieval efficient, the values of the Minhash functions are grouped into n-tuples called sketches. Identical sketches are then efficiently found using a hash table. Sets with high similarity tend to have common values of the Minhash signature for many random permutations. Consequently, there is a high probability of having the same sketches. In contrast, sets that are not similar have low probability of having the same sketches.

2.3.4 Locality-sensitive hashing

This technique was originally introduced by Indyk and Motwani [14] for the purposes of devising main memory algorithms for nearest neighbor search. Afterwards, Aristides Gionis in 1999 [15] used LSH in an attempt to introduce a more effective indexing method for approximate nearest neighbor. Locality-sensitive hashing (LSH) has since been proven to be an effective way for approaching the approximate nearest neighbor (ANN) search.

The main idea that LSH relies on is to hash items several times, in such a way that similar items are more likely to be hashed to the same bucket than dissimilar items are. Then, each pair of items that hashed in the same bucket is considered to be a candidate pair. Finally, the similarity of only the candidate pairs is calculated. In that way, the calculations are being reduced dramatically and the process becomes more efficient. There are cases of similar pairs that don’t hash in the same bucket (false negatives) and dissimilar pairs that hash to the same bucket (false positives). The less such cases are the more accurate the method is [8].

Hence, LSH technique relies on the idea that the probability of collision of two points p and q is closely related to the distance between them. In particular, the larger the distance, the smaller the collision probability. Respectively, LSH function families have the ability to have higher probability of collision between objects that are close to each other than objects that are far apart. Specifically, let S be the domain of objects, and D be the distance measure between objects.

Definition 5 [16]. A function family $H = \{h : S \rightarrow U\}$ is defined as (r, cr, p_1, p_2)-sensitive for D if for any $q, p \in S$

- If $D(q, p) \leq r$ then $P_{rH}[h(q) = h(p)] \geq p1$ (1)
- If $D(q, p) > cr$ then $P_{rH}[h(q) = h(p)] \leq p2$ (2)

Therefore, the main idea that LSH follows is selecting a hashing function (or a hashing function family) such that if (1) applies, there is a high probability that two neighboring objects will also be neighbors after hashing. Contrariwise, if (2) applies, there is a high probability that two non-neighboring objects will also be non-neighbors after hashing. If a hashing function satisfies the above two conditions, then it is called LSH function [17].

2.4 Weighted methods

Content based RS are mainly based on the characteristics and features of items. Each item is represented by a profile that holds its information. So, it is of great importance to represent this information in a way that is suitable for the process of finding item similarities. The process of extracting features highly depends on the specific domain. In a product RS, the title and the description include some of the most important features that need to be considered. In addition, depending on the type of products, the category and the brand of them can be very informative. Such textual information needs to be preprocessed in order to take advantage out of it. Furthermore, different features carry different amount of information. A word in a product title may be more informative than the rest and this fact must be taken into consideration. Incorporating token weights can have a major impact on the computed similarity and the quality of the recommendations.

The "importance" (or "informativeness") of word k_j in document d_j is determined with some weighting measure w_{ij} that can be defined in several different ways. One of the best-known measures for specifying keyword weights is the Term Frequency-Inverse Document Frequency (TF-IDF) that is presented below. The idea behind TF-IDF is that terms with high occurrence in a document but rare in the rest are more important for that document.

2.4.1 TF–IDF

Term frequency – Inverted Document Frequency (TF-IDF) is a simple and effective weighing scheme that is used in order to calculate the importance of terms in a set of documents. In the field of RS, TF-IDF can be used in order to represent the textual information of items so that more informative terms will be more important in the similarity calculations. Considering each item as a document, TF-IDF is based on the frequency of words in a specific document compared to the inverse proportion of that word over the entire document corpus. Through this process, a weight is assigned to each unique term for each document. This weight corresponds to the significance of the term in the specific document. The more documents a term appears in, the less informative is, thus it gets a smaller weight. Respectively, the significance of a term in a document increases with the occurrences of it in that specific document [18].

In a mathematical perception, TF-IDF weight ($w_{t,d}$) consists of two parts. The first is the Term Frequency (TF) that refers to the frequency of the term in a specific document ($f_{t,d}$). The second part is the Inverted Term Frequency (IDF) that consists of the number of documents in the collection (N) and the number ($d_{f,t}$) of documents that contain the term.

$$w_{t,d} = f_{t,d} * \log\left({}^{N}/_{d_{f,t}}\right)$$

2.4.2 Weighted Minhash

Minhash algorithm is a way to estimate the Jaccard similarity (Definition 5) between two sets. Jaccard similarity considers all items of a set to be of the same importance as all of them have equal weights. Respectively, Minhash approach treats all elements in a set equally and each element can be mapped to the minimum hash value with equal probability. This leads to information loss in cases where different elements carry different amount of information. In reality, we need to treat elements as having different weights. For example, in a product RS specific words in a product title that correspond to specific characteristics (category, brand etc.) are more important than other words. In such cases, sets are weighted, and weights correspond to the significance of each element in the set. Incorporating weights can have a major impact on the computed similarity. Hence, a way to approximate the Generalized Jaccard Similarity (Definition 6) is needed. Assuming a representation of sets in which elements have different weights, the challenge is to incorporate these weights into min-hash algorithm so that similar sets have high probabilities to be mapped into the same bucket. Over the last years, significant research has been done in order to create Minhash schemes that can handle weighted sets. These researches of weighted Minhash algorithms can be split into the following two categories: “quantization-based” and “sampling-based” approaches [19], [8].

By applying quantization based weighted Minhash algorithms, a weighted set is converted into a binary set. This is accomplished by quantizing each weighted element into a number of distinct and equal-sized sub-elements. In particular, elements with large weights should be assigned more sub-elements, while elements with small weights should be assigned less sub-elements. Weights that are produced by weighting methods such as TF-IDF are usually small. So, these weights should first be multiplied by a large constant. The size of the constant affects the accuracy and the

complexity of the method. The larger the constant, the more accurate the results are. Although, increasing the constant leads to the increase of time complexity. Finally, all the sub-elements of the binary set are considered to be independent elements and are treated equally. So, the computation of the hash values for all the sub-elements is needed. This makes quantization-based methods inefficient for large sets.

The above method leads to large sets that are difficult to be handled in terms of Minhash computation. In order to avoid computing the hash value for every sub-element, researchers proposed a sample-based approach. The main idea of the sampling-based algorithms is to compute the Minhash value only for special sub-elements in order to decrease time complexity. Based on this idea, much work has been done in order to address the problem of the quantization-based approach. Specifically, important research was focused on the idea of "active index" [20] in order to improve the algorithms by sampling several "active indices" and then computing the hash values for them. Gollapudi and Panigraphy proposed an improved integer-value weighted Min-Hash algorithm that decreases the number of sub-elements that need to be taken into consideration for Minhash computation. This method requires the real weights to be multiplied by a big constant. This makes the method inefficient for real value weights and leads to an expanded set of sub-elements that cannot be handled easily.

Afterwards, the Consistent Weighted Sampling Scheme (CWS) [21] was introduced as an attempt to address the issues of the previous approaches. The most important advantage of this approach is that it can handle real weights without first converting them to integers. Later, Ioffe proposed the Improved Consistent Weighted Sampling (ICWS) algorithm [22] which is the fastest known exact weighted minwise sampling scheme. ICWS is considered to be an efficient and unbiased estimator of the generalized Jaccard similarity and is used in the current research. Its performance does not depend on the weights as long as the universe of all possible elements is known.

3 Previous work

During the last years, interesting research has been done in the field of Recommendation Systems. Many researches were held and led to techniques that have practical use in various domains. Researchers have also tried to address problems that have arisen due to the data explosion. In this section, we briefly review some previous research efforts related to recommender systems and locality sensitive hashing. We also present previous works that incorporated weighted methods in the field of RS.

Interesting research was implemented in [23]**,** where the problem of high dimensionality, that traditional RS face, was addressed. E-commerce platforms with very big number of items and users need to incorporate large volume of data in the recommendation process. This makes the generation of real time recommendations inefficient. The paper incorporates LSH techniques in a collaborative filtering (CF) approach in order to reduce the time complexity. Minhash hashing method is used for binary data and simhash for real-valued data. Similar candidate pair identification is performed through LSH in order to increase the efficiency of similarity computing, which is the most time-consuming task for traditional collaborative filtering recommender systems. By conducting experiments on synthetic and real-world datasets, it is shown that LSH can approximately preserve similarities of data while significantly reducing data dimensions.

In [24], in attempt to introduce a RS that can scale with the increasing amount of data, the authors use LSH approach. Specifically, they provide novel improvements to the already proposed LSH based recommender algorithms and make a systematic evaluation of LSH in neighborhood-based CF. By making extensive experiments on real-life datasets, they present algorithms that have better running time performance than the standard LSH-based applications while preserving the prediction accuracy in reasonable limits. These algorithms also produce recommendations with diversity which is an important aspect of RS.

In [25], a hybridization of content based and collaborative filtering-based recommendation, which incorporates product attribute weighting, is proposed. It is argued that human judgment of similarity between two items often gives different weights to different attributes and that

recommendations systems need to consider this aspect. The weights refer to the importance of each product attribute to customers and are estimated from a set of linear regression equations obtained from a social network graph, which captures human judgment about similarity of items. The proposed system is compared with content-based methods that consider the importance of different products features as equal. The evaluation is based on IMDB recommendations which are considered as benchmark. Finally, it turns out that the proposed method outperforms simple methods. Hence, the effectiveness of feature weighting is demonstrated.

In [26], a feature weighting method is proposed with the aim to improve the content-based filtering in cases of multi-valued item features. The authors argue that a user considers some specific features as more important than other, when selecting an item. This consideration represents an implicit feature weighting which is subjective and different for each user. Their feature weighting method is based on entropy and coefficients of correlation and contingency. In particular, the weight of each feature is computed according to (i) the entropy or amount of information provided by itself (the more entropy the more weighting should have), and (ii) the correlation between items chosen by the user in the past and the values of some features of the set of items.

4 Data

In this dissertation, two separate datasets have been used in order to build and evaluate the recommendation system. The first dataset was created by scraping a real e-commerce site and consists of information about thousands of products. The second dataset was created by preprocessing an initial ready-made dataset that has already been used for similar purposes in the past. These two datasets are described in detail in the following chapters.

4.1 Bestprice dataset

Bestprice dataset is a completely new dataset that was created for the purposes of this research. The data was gathered from BestPrice.gr which is a commercial site where a customer can compare the price of products across different e-shops. A huge variety of product categories and subcategories is available. In this research, only a set of them has been used. Specifically, data about technological products that belong to 6 main categories has been gathered. The gathering procedure, an overview of the dataset and the preprocessing steps are described below.

4.1.1 Gathering procedure

The dataset was initially created by scraping the 'Bestprice.gr' e-commerce site. Web scraping refers to the process of automatically collecting data and information from web pages by using a programming language. In our case, a scraper based on python language was created. Information of thousands of products was gathered by scraping 22 product categories. Each product category consisted of thousands of pages. So, around 1.200 page requests were performed by the scraper. The main issue that we faced were the sites anti-scraping measures. In particular, the site is blocking scrapers and crawlers that perform too many requests. We overcame this issue by adding a functionality to the scraper so that it can change IP every five requests. This was accomplished by scraping two different sites that offer free proxies. In that way, the scraper was able to request and scrape thousands of pages without getting blocked as a bot. The three scripts (scrape_bestprice.py, bestprice.py and general.py) that were created for the scraping process are available in the appendix of this dissertation.

4.1.2 Overview

Bestprice dataset was created by gathering data from Bestprice.gr as described in the previous paragraph. Particularly, it consists of data regarding 29.541 products that belong to 6 main categories in the domain of technology. Each of those main categories has smaller categories that are called subcategories. Specifically, the products are split in 22 subcategories. Additionally, there are 515 different product brands. A summarization of the category tree and the number of products and brands in each subcategory is presented in table 4.1.

Besides the information regarding the categories and the brand of each product, additional information is available. Particularly, there is a unique id, a title and a price for each product. The available information about 5 random products are presented in table 4.2. Lastly, a set of 10 product recommendations is available for each product. These recommendations are those that were provided to the users in each product page by the e-commerce site. This information is available for a subset of 1.182 products and is used in the evaluation procedure that is presented in paragraph 7.1.2.

Table 4.1: Overview of Bestprice dataset.

ID	Category	Subcategory	Products	Brands
0	Desktop_pc	Desktop	709	7
1	Desktop_pc	Desktop_monitors	2193	25
2	Desktop_pc	Desktop_rams	2770	25
3	Laptop_pc	Laptop	2718	19
4	Laptop_pc	Laptop_bases	172	19
5	Laptop_pc	Laptop_battery	6642	23
6	Laptop_pc	Laptop_cases	1834	92
7	Mobiles	Bluetooth	656	47
8	Mobiles	Handsfree	943	77
9	Mobiles	Mobile_phone	1529	60
10	Mobiles	Portable_speaker	2394	135
11	Mobiles	Power_bank	1731	74
12	Photograph	Analog	34	4
13	Photograph	Compact	158	9
14	Photograph	DSLR	183	3
15	Photograph	Photograph_battery	1666	23
16	Photograph	Photograph_cases	2132	49
17	Tablets_other	Tablet_bases	173	19

18	Tablets_other	Tablet_cases	2495	93
19	Tablets_other	Tablet_chargers	180	11
20	Wearables	Smartwatch	433	21
21	Wearables	Transmitter	45	3

Table 4.2: Details of 5 random products in Bestprice dataset.

Product ID	Title	Price (€)	Category	Subcategory	Brand
2155531584	Samsung Galaxy S10+ 128GB Dual	595.00 €	Mobiles	Mobile_phone	Samsung
2155112026	Sony SBH56	42.04 €	Mobiles	Bluetooth	Sony
2154004522	Omega Ice Box	8.22 €	Laptop_pc	Laptop_bases	Omega
2155081727	Lenovo Thinkvision T24I	147.49 €	Desktop_pc	Desktop_monitors	Lenovo
2155432313	Huawei Watch GT Graphite Black	122.00 €	Wearables	Smartwatch	Huawei

4.1.3 Preprocessing

The preprocessing of the initial dataset that is described above was a significant step. The target was to transform the textual data so that it can be used appropriately in the next steps and accomplish the best results. Specifically, each product should be assigned a text that holds the information that characterizes it. This text is produced by concatenating the title, category, subcategory and brand of each product. However, each of those features had to be preprocessed accordingly before adding them to the final text. The most important steps are described below. The script (preprocess_bestprice.py) that was created for this reason can be found in the appendix of this dissertation.

Firstly, the textual data of each feature was transformed to lowercase. The second step was to discriminate the categories, subcategories and brands from other words by adding the suffixes '_cat', '_subcat' and '_brand' respectively. This will help us assign different weights to those specific words. Furthermore, specific symbols that don't add any value were removed from the title. Finally, a text that consists of the above textual features was created for each product. This

set of texts will be used in order to calculate the TF-IDF matrix. The corresponding final text for each product that was presented in table 4.2, is presented in table 4.3.

Table 4.3: The final textual representation of 5 products in Bestprice dataset.

Product ID	Text
2155531584	samsung_brand galaxy s10+ 128gb dual mobiles_cat mobile_phone_subcat
2155112026	sony_brand sbh56 mobiles_cat bluetooth_subcat
2154004522	omega_brand ice box laptop_pc_cat laptop_bases_subcat
2155081727	lenovo_brand thinkvision t24i desktop_pc_cat desktop_monitors_subcat
2155432313	huawei_brand watch gt graphite black wearables_cat smartwatch_subcat

4.2 Retailrocket dataset

Retailrocket dataset is a ready-made dataset that has been used in various researches in the field of RS. It was published by a company named Retail Rocket which offers e-commerce solutions to personalize the online shopping experience. The data has been collected from a real-world e-commerce website. It is raw data as there are no content transformations. However, most of the values are hashed due to confidential issues. The purpose of publishing this dataset was to motivate researches in the field of recommender systems with implicit feedback and has already been used in other researches [35]. Although, in the current research, a subset of it has been used for the evaluation of our content-based RS.

4.2.1 Overview

The dataset consists of three different files. The first file contains the customer behavioral data and represents interactions that were collected over a period of 4.5 months. These interactions of customers with products are called events and are of three types. The first is the 'view' event that refers to the action of viewing a product page. The second is the 'addtocart' event that means that the customer added a product to his/her cart. The third is the 'transaction' event that corresponds to the action of purchasing a product. In total there are 2.756.101 events including 2.664 312 views, 69.332 add to carts and 22.457 transactions produced by 1.407.580 unique visitors.

The second file holds the product details. Each product has a number of properties but only three of them have been used in the current research. Specifically, the title, category and subcategory properties have been used. Lastly, there is a third file that contains the relationships between the categories and subcategories. Every row in the file specifies a child categoryId and the corresponding parent category. This file is used to find the main category that each subcategory belongs to. Furthermore, it is important to mention that a subset of 28.241 products was selected. These products belong to 6 main categories which are split in 37 subcategories.

4.2.2 Preprocessing

The first step of the preprocessing phase was to create a dataset with the sessions that were produced over the period of 4.5 months. A session refers to a group of user interactions with an e-shop that take place within a given time frame. For example, a session may contain the information that a customer first viewed a number of different products, added some of them in his/her cart

and finally purchased them. Alternatively, a session may refer to a user that visited an e-shop, viewed a number of products and then left the e-shop without buying anything. So, based on the dataset that represents such events that 1.407.580 unique visitors made, a new dataset of 1.650.654 sessions was created.

Subsequently, this session-based dataset was used in order to determine the product pages that users visit after viewing each single product. This helps us understand the relationships between products. These relationships are produced based on real actions that users perform while searching for products that meet their preferences. An important aspect that was taken into consideration is the number of consecutively events that are related with each other. We call this window size. Window size equal to 1 means that a product that a user viewed is linked only with the product that he visited next. Respectively, window size equal to 2 means that a product that a user viewed is linked with the next two products that he visited.

Furthermore, a representative text has been assigned to each product. Similar to the logic that was described for the Bestprice dataset, this text consists of the title, category and subcategory of each product. This set of texts will be used in order to calculate the TF-IDF matrix.

5 Methodology

The heuristic approach for content-based Recommendation Systems that is proposed in the current research is based on feature weighting and Locality Sensitive Hashing (LSH). The design of the system consists of three parts. The first refers to the method that is used in order to represent the set of products as a weighted matrix. The second is the weighted Minhash method that is used to approximate the Jaccard similarity of two sets. The last part is the efficient production of the recommendations based on LSH. The implementation of the system based on the Bestprice dataset is presented in the following three paragraphs and in figure 5.1.

Figure 5.1 Methodology parts diagram.

5.1 Weighted method

The first phase of our methodology is to create a TF-IDF matrix for the set of 29.541 products. Usually, different features carry different amount of information. A word in a product title may be more informative than the rest and this fact must be taken into consideration during the similarity calculations. In our case, each product is represented by a text that is a concatenation of the product title and three specific features as described in paragraph 4.1.3. The terms that correspond to the three specific product features are initially considered to be simple terms. These features are the category, subcategory and brand of each product. The total number of products is 29.541 and the respective corpus consists of 35.405 unique terms. The target is to create a matrix that represents how important each term is for each product. A weight is assigned to each unique term for each product text. This weight corresponds to the significance of the term for each specific product. The more texts a term appears in, the less informative is considered to be, thus it gets a smaller weight. Respectively, the significance of a term in a product text increases with the occurrences of it in that specific text. In particular, the TF-IDF matrix has been created by using the corresponding functions of the ‘scikit-learn’ python library [34].

Furthermore, specific terms of the corpus have been given extra weight. These terms correspond to the three product features that are parts of each product text. The target is to have a functionality with which we can easily adjust the importance of each feature in the calculation of product similarities. For example, by increasing the weights of the terms that correspond to the product brands, we force the system to consider this specific feature as more important during the calculation of product similarities. Alternatively, by decreasing the weights of the terms that refer to the product categories, we force the system to consider this specific feature as less important. In other words, this functionality offers the opportunity to adjust the logic of the recommendation system to the requirements and the aims of an e-commerce site. It also offers diversity in the recommendations without compromising similarity or efficiency. This was accomplished by multiplying the TF-IDF weights with small constants. For example, in order to increase the significance of the brand feature, the weight of the corresponding terms must be multiplied by a constant larger than 1. Respectively, a positive smaller than 1 constant is necessary in order to decrease the significance of a feature. Multiplying the TF-IDF weight of a term by 1 means that its importance is not changed. The multiple scenarios that were implemented are presented in paragraph 8. The script (weighted_scheme.py) that was created for this reason can be found in the appendix of this dissertation.

5.2 Weighted Minhash

Two sets are near duplicates if their similarity score is above a predefined threshold. Calculating the similarity scores among sets through simple methods, such as Jaccard similarity, is usually inefficient due to high dimensionality. Minhash is an efficient LSH algorithm that has been successively proposed to approximate such similarity calculations. In particular, Minhash is used to approximate the Jaccard similarity of two sets. Minhash can also be used to compress unweighted set and estimate the unweighted Jaccard similarity. This simple Minhash approach can be applied in weighted sets by expanding each item based on its weight. However, this approach does not support real number weights.

In our approach, the set of products is represented by a TF-IDF matrix. So, a method that incorporates real number weights has been used. Specifically, the weighted Minhash algorithm that is available in ‘datasketch’ python library is used [36]. Weighted Minhash was created by

Sergey Ioffe [22], and its performance does not depend on the weights as long as the universe of all possible items is known. In practice, a Minhash signature has been created for each product based on the corresponding TF-IDF array. Hence, each product is represented by a much smaller array than before. Specifically, while the TF-IDF array consists of 35.405 elements, the Minhash signature consists of only 128 elements. The length of the signature corresponds to the 'sample_size' parameter that can be adjusted accordingly as by increasing the number of samples, a better accuracy is accomplished, at the expense of slower speed. These Minhash signatures are used in order to approximate the Jaccard similarity between products by applying the LSH approach that is presented below. The script (minhash_lsh.py) that was created for this reason can be found in the appendix of this dissertation.

5.3 LSH forest & Recommendations

Having a large collection of sets and a query set, the aim is to find those sets that have Jaccard similarities above a certain threshold. By creating a Minhash signature for every set as described in 5.2, when a query comes, the Jaccard similarities between the query Minhash and all the Minhash of the collection needs to be calculated. This makes the procedure more efficient since the Minhash signature is a compressed representation of the initial set. However, this approach is still an O(n) algorithm and the query cost increases linearly with respect to the number of sets. A popular alternative is to use Locality Sensitive Hashing (LSH) which is an effective way for approximating Jaccard similarity between sets. LSH incorporates the representation of items by their Minhash signature. The details of the algorithm can be found in Chapter 3 of [27] and in paragraph 2.3.4 of the current dissertation. LSH assures that sets with higher Jaccard similarities always have higher probabilities to get returned than sets with lower similarities.

In the current research, the LSH approach is used in order to find the most similar products to each product based on their Minhash signature. In particular, we search for the top 10 similar products that correspond to 10 recommendations. For this reason, a variation of LSH that is known as LSH Forest is used. LSH Forest was proposed by Bawa et al. [28] and is a general LSH data structure that makes top-k query possible for many different types of LSH indexes, which include Minhash LSH. Minhash LSH Forest, uses the Minhash representation of the query product and returns the top-k matching products that have the approximately highest Jaccard similarities with the query

product. In that way, it is not necessary to pre-define a specific threshold for the Jaccard similarity score. In this way, we produced 10 recommendations for each of the 29.541 products in the Bestprice dataset. The script (minhash_lsh.py) that was created for this reason can be found in the appendix of this dissertation.

6 Experiments & Results

Based on the aforementioned methodology, we have conducted experiments with the aim examine the effect that different feature weights have in the final recommendations. Specifically, scenarios with different combinations of weights regarding the 3 basic product features (brand, category & subcategory) have been tested for the Bestprice dataset. Each scenario is presented below along with the results. The results correspond to statistics about the number of brands, categories and subcategories that are present in the set of ten recommendations of each product. The results of each scenario are presented in two parts. The first corresponds to statistics about all the products regardless the product category. The second examines each product category separately.

6.1 Keeping the initial TF-IDF weights

- *Scenario 1*: w_brand = 1, w_category=1, w_subcategory=1

This is the base scenario in which the three product features have the weights that were produced by TF-IDF method. So, we don't consider any of the features as more or less important during the similarity calculations. In table 6.1, we see that there is an average of 2.54 different brands present in each set of 10 recommendations. There are also cases with more than 4 different brands, reaching a maximum of 10. The number of different categories and subcategories is much less, having a mean of 1.16 and 1.30 respectively. The cases that more than two categories or subcategories are present in the set of 10 recommendations are really few. In table 6.2, we see that the statistics are different in each product category. The category with the biggest diversity concerning product brands is 'mobiles'. On the contrary, 'wearables' category has the lowest number of brands at average.

Table 6.1: Statistics for the recommendations regardless the product category (scenario 1).

	Brands	Categories	Subcategories
mean	2.54	1.16	1.30
min	1.00	1.00	1.00
25%	1.00	1.00	1.00
50%	2.00	1.00	1.00
75%	4.00	1.00	1.00
max	10.00	6.00	7.00

Table 6.2: Average number of brands, categories and subcategories per category (scenario 1).

Category	Brands	Categories	Subcategories
desktop_pc	2.34	1.12	1.14
laptop_pc	2.39	1.17	1.28
mobiles	3.08	1.16	1.46
photograph	2.26	1.16	1.33
tablets_other	2.78	1.21	1.33
wearables	2.14	1.22	1.27

6.2 Tuning the brand weight

- *Scenario 2*: w_brand = 0.5, w_category=1, w_subcategory=1
- *Scenario 3*: w_brand = 1.5, w_category=1, w_subcategory=1

In these two scenarios, we consider the product brand feature to be less important (Scenario 2) or more important (Scenario 3) than the rest. This is accomplished by multiplying the TF-IDF weights of the terms that correspond to this specific feature by small constants. These constants are 0.5 and 1.5 respectively. By observing table 6.3, it is obvious that the more important the brand is considered to be, the less brands at average are present in each set of 10 recommendations. On the contrary, the number of different categories and subcategories remains almost the same. The same conclusions are extracted by observing table 6.4 where the average numbers are presented per product category. As in the base scenario 1, 'mobiles' and 'wearables' have the largest and smallest average number of brands in each set of 10 recommended products.

Table 6.3: Statistics for the recommendations regardless the product category (scenarios 2&3).

	Scenario 2			**Scenario 3**		
	Brands	**Categories**	**Subcategories**	**Brands**	**Categories**	**Subcategories**
mean	3.07	1.16	1.30	2.16	1.16	1.31
min	1.00	1.00	1.00	1.00	1.00	1.00
25%	1.00	1.00	1.00	1.00	1.00	1.00
50%	3.00	1.00	1.00	1.00	1.00	1.00
75%	5.00	1.00	1.00	3.00	1.00	1.00
max	10.00	5.00	7.00	10.00	5.00	7.00

Table 6.4: Average number of brands, categories and subcategories per category (scenarios 2&3).

	Scenario 2			Scenario 3		
Category	**Brands**	**Categories**	**Subcategories**	**Brands**	**Categories**	**Subcategories**
desktop_pc_cat	2.84	1.11	1.14	1.92	1.10	1.14
laptop_pc_cat	2.76	1.17	1.27	2.13	1.17	1.27
mobiles_cat	4.02	1.15	1.43	2.45	1.16	1.50
photograph_cat	2.64	1.18	1.36	1.96	1.16	1.34
tablets_other_cat	3.27	1.20	1.31	2.37	1.23	1.35
wearables_cat	2.40	1.26	1.31	1.91	1.22	1.27

6.3 Tuning the subcategory weight

- *Scenario 4*: w_brand = 1, w_subcategory=0.5, w_category=1
- *Scenario 5*: w_brand = 1, w_subcategory=1.5, w_category=1

In these scenarios, we consider the product subcategory feature to be less important (Scenario 4) or more important (Scenario 5) than the rest. This is accomplished by multiplying the TF-IDF weights of the terms that correspond to this specific feature, by small constants. These constants are 0.5 and 1.5 respectively. We can see in table 6.5 that the more important the subcategory is, the more reduced the average number of different subcategories (in each set of 10 recommendations) is. Reduction is also observed in the average number of different categories. On the contrary, the average number of different brands is increased by increasing the importance of subcategory feature. Similar conclusions are extracted by observing table 6.6 where the average numbers are presented per product category.

Table 6.5 Statistics for the recommendations regardless the product category (scenarios 4 &5).

	Scenario 4			Scenario 5		
	Brands	**Categories**	**Subcategories**	**Brands**	**Categories**	**Subcategories**
mean	2.47	1.21	1.40	2.58	1.14	1.25
min	1.00	1.00	1.00	1.00	1.00	1.00
25%	1.00	1.00	1.00	1.00	1.00	1.00
50%	2.00	1.00	1.00	2.00	1.00	1.00
75%	4.00	1.00	2.00	4.00	1.00	1.00
max	10.00	6.00	7.00	10.00	6.00	7.00

Table 6.6: Average number of brands, categories and subcategories per category (scenarios 4 &5).

	Scenario 4			Scenario 5		
	Brands	**Categories**	**Subcategories**	**Brands**	**Categories**	**Subcategories**
desktop_pc_cat	2.07	1.19	1.28	2.35	1.09	1.10
laptop_pc_cat	2.38	1.19	1.31	2.45	1.15	1.23
mobiles_cat	2.98	1.22	1.65	3.10	1.13	1.37
photograph_cat	2.27	1.19	1.39	2.29	1.13	1.25
tablets_other_cat	2.74	1.28	1.45	2.81	1.18	1.28
wearables_cat	1.99	1.27	1.32	2.10	1.19	1.22

6.4 Tuning the category weight

- *Scenario 6*: w_brand = 1, w_subcategory=1, w_category=0.5
- *Scenario 7*: w_brand = 1, w_subcategory=1, w_category=1.5

In these scenarios, we consider the product category feature to be less important (Scenario 6) or more important (Scenario 7) than the rest. This is accomplished by multiplying the TF-IDF weights of the terms that correspond to this specific feature, by small constants. These constants are 0.5 and 1.5 respectively. We can see in table 6.7 that the more important the category is considered to be, the less different categories are present in each set of 10 recommendations. Reduction is also observed in the average number of different subcategories. On the contrary, the number of different brands is increased by increasing the importance of category feature. Similar conclusions are extracted by observing table 6.8 where the average numbers are presented per product category.

Table 6.7: Statistics for the recommendations regardless the product category (scenarios 6&7).

	Scenario 6			Scenario 7		
	Brands	**Categories**	**Subcategories**	**Brands**	**Categories**	**Subcategories**
mean	2.51	1.2	1.33	2.63	1.12	1.29
min	1.00	1.0	1.00	1.00	1.00	1.00
25%	1.00	1.0	1.00	1.00	1.00	1.00
50%	2.00	1.0	1.00	2.00	1.00	1.00
75%	4.00	1.0	1.00	4.00	1.00	1.00
max	10.00	6.0	7.00	10.00	6.00	8.00

Table 6.8: Average number of brands, categories and subcategories per category (scenarios 6&7).

	Scenario 6			Scenario 7		
	Brands	**Categories**	**Subcategories**	**Brands**	**Categories**	**Subcategories**
desktop_pc_cat	2.33	1.12	1.15	2.25	1.09	1.11
laptop_pc_cat	2.38	1.19	1.29	2.40	1.14	1.26
mobiles_cat	2.97	1.25	1.53	3.49	1.05	1.48
photograph_cat	2.23	1.21	1.36	2.37	1.11	1.31
tablets_other_cat	2.75	1.27	1.37	2.81	1.19	1.31
wearables_cat	2.22	1.27	1.30	2.05	1.17	1.22

6.5 Tuning the brand and subcategory weights

- *Scenario 8*: w_brand = 0.5, w_subcategory=0.5, w_category=1
- *Scenario 9*: w_brand = 1.5, w_subcategory=1.5, w_category=1

In these scenarios, we tune the weights of both the brand and the subcategory features. In particular, we consider both features to be less important (Scenario 8) or more important (Scenario 9) by multiplying the TF-IDF weights of the corresponding terms by 0.5 and 1.5 respectively. By observing table 6.9, we conclude that decrease in the average number of all three features is accomplished by increasing the importance of brand and subcategory simultaneously. Similar changes are observed in all the products categories (table 6.10).

Table 6.9: Statistics for the recommendations regardless the product category (scenarios 8&9).

	Scenario 8			Scenario 9		
	Brands	**Categories**	**Subcategories**	**Brands**	**Categories**	**Subcategories**
mean	2.94	1.20	1.40	2.19	1.14	1.26
min	1.00	1.00	1.00	1.00	1.00	1.00
25%	1.00	1.00	1.00	1.00	1.00	1.00
50%	2.00	1.00	1.00	1.00	1.00	1.00
75%	4.00	1.00	2.00	3.00	1.00	1.00
max	10.00	5.00	7.00	10.00	5.00	7.00

Table 6.10: Average number of brands, categories and subcategories per category (scenarios 8&9).

	Scenario 8			Scenario 9		
	Brands	**Categories**	**Subcategories**	**Brands**	**Categories**	**Subcategories**
desktop_pc_cat	2.27	1.20	1.27	1.90	1.09	1.10
laptop_pc_cat	2.76	1.18	1.30	2.19	1.15	1.24
mobiles_cat	3.90	1.21	1.64	2.48	1.14	1.41
photograph_cat	2.67	1.20	1.43	1.99	1.13	1.26
tablets_other_cat	3.23	1.26	1.43	2.39	1.20	1.30
wearables_cat	2.28	1.30	1.40	1.93	1.20	1.24

6.6 Tuning the brand and category weights

- *Scenario 10*: w_brand = 0.5, w_subcategory=1, w_category=0.5
- *Scenario 11*: w_brand = 1.5, w_subcategory=1, w_category=1.5

In these scenarios, we tune the weights of both the brand and the category features. In particular, we consider both features to be less important (Scenario 10) or more important (Scenario 11) by multiplying the TF-IDF weights of their terms by 0.5 and 1.5 respectively. By observing table 6.11, we conclude that decrease in the average number of all three features is accomplished by increasing the importance of brand and category simultaneously. Similar changes are observed in all the products categories (table 6.12).

Table 6.11: Statistics for the recommendations regardless the product category (scenarios 10&11).

	Scenario 10			Scenario 11		
	Brands	**Categories**	**Subcategories**	**Brands**	**Categories**	**Subcategories**
mean	3.01	1.2	1.32	2.23	1.13	1.29
min	1.00	1.0	1.00	1.00	1.00	1.00
25%	1.00	1.0	1.00	1.00	1.00	1.00
50%	2.00	1.0	1.00	1.00	1.00	1.00
75%	4.00	1.0	1.00	3.00	1.00	1.00
max	10.00	6.0	7.00	10.00	5.00	7.00

Table 6.12: Average number of brands, categories and subcategories per category (scenarios 10&11).

	Scenario 10			Scenario 11		
	Brands	**Categories**	**Subcategories**	**Brands**	**Categories**	**Subcategories**
desktop_pc_cat	2.72	1.12	1.14	1.94	1.09	1.13
laptop_pc_cat	2.73	1.18	1.27	2.14	1.15	1.26
mobiles_cat	3.89	1.24	1.49	2.73	1.08	1.46
photograph_cat	2.63	1.23	1.38	1.96	1.13	1.32
tablets_other_cat	3.27	1.25	1.35	2.38	1.21	1.34
wearables_cat	2.42	1.29	1.33	1.87	1.21	1.26

6.7 All scenarios

All the above scenarios (1-11) are presented in table 6.13. Specifically, the average numbers of different brands, categories and subcategories (in each set of 10 recommendations) are presented for each scenario. In general, we observe that the incorporation of the feature weighting functionality affects the characteristics of the produced recommendations. In that way we can control and adjust the recommendation systems logic according to the needs of each e-commerce site. In particular, through this weighted scheme, we are able to increase or decrease the recommendation diversity and novelty concerning specific product attributes. This can also be applied to only specific product categories or subcategories. By observing table 6.13, we can see that by increasing the weight of a product feature, the corresponding average number is reduced. For example, by increasing the weight of the product brand, the average number of different brands in each set of 10 recommendations is reduced. The increase of the weight corresponds to the increase of the importance of this specific feature in the similarity calculations. Hence, the system considers the brand similarity between two products as more important than the similarity of their categories or subcategories.

Table 6.13 Average number of brands, categories and subcategories for scenarios 1-11

	W_brand	W_category	W_subcategory	Brands	Categories	Subcategories
Scenario 1	1	1	1	2.54	1.16	1.30
Scenario 2	0.5	1	1	3.07	1.16	1.30
Scenario 3	1.5	1	1	2.16	1.16	1.31
Scenario 4	1	1	0.5	2.47	1.21	1.40
Scenario 5	1	1	1.5	2.58	1.14	1.25
Scenario 6	1	0.5	1	2.51	1.20	1.33
Scenario 7	1	1.5	1	2.63	1.12	1.29
Scenario 8	0.5	0.5	1	2.94	1.20	1.40
Scenario 9	1.5	1.5	1	2.19	1.14	1.26
Scenario 10	0.5	1	0.5	3.01	1.20	1.32
Scenario 11	1.5	1	1.5	2.23	1.13	1.29

7 Evaluation and future work

In this chapter, we evaluate the proposed content-based recommendation system. The evaluation process is split in two different parts. The first one refers to a session-based analysis and examines its effectiveness. In the second part the recommendation diversity is examined. We also present the most important conclusions and insights that have been extracted from the conducted experiments and the evaluation process. Finally, we discuss about the additions and improvements that we believe could result in even better results.

7.1 Evaluation

The evaluation of the recommendation system consists of two parts. The first part is based on the Retailrocket dataset that consists of real sessions as described in paragraph 4.2. The aim is to examine whether the recommendations produced by our system would help a user navigate in the e-shop and prevent them from leaving the site without finding products that meet their preferences. The second part is based on the Bestprice dataset that was created by scraping a real e-commerce site and is presented in paragraph 4.1. Having proven that our recommendations help customers finding products that satisfy their preferences, the aim is to compare the recommendations of our system with those that were available on the site, concerning certain aspects. Specifically, we compare the diversity of the recommendations and we examine whether we can achieve different results by tuning the weights of the product features.

7.1.1 Session based

In this part of the evaluation process, the recommendations that are produced for Retailrocket dataset are compared with the real customer sessions. This dataset consists of real sessions that were generated in a real e-commerce site as described in paragraph 4.2. A session is a group of customer interactions with the e-commerce site that take place within a given time frame. In this dataset, a single session can contain multiple product page views, adds to cart and transactions. A product view refers to the action of visiting a product page in order to see its details. An ‘addtocart’ action corresponds to the action of adding a product in the cart, while a transaction refers to finally buying at least one product.

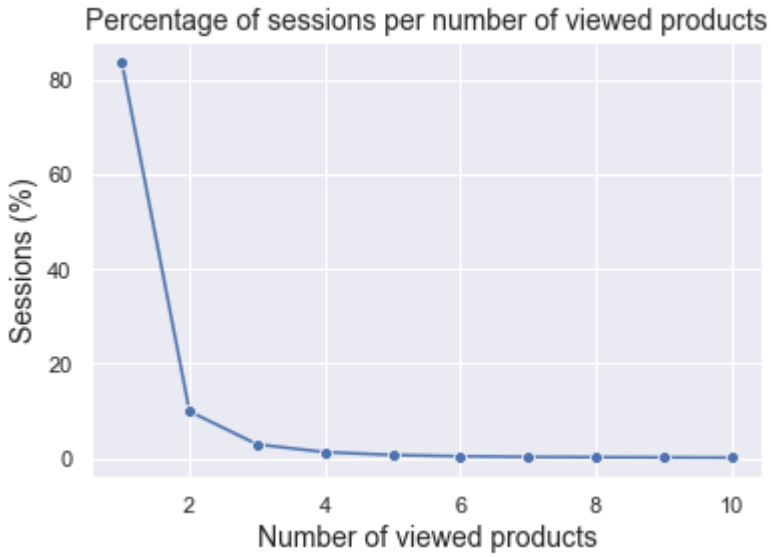
The analysis of those sessions is summarized in the table 7.1 and figures (7.1-7.3). Analyzing sessions that contain up to ten product views showed that the more products a customer views, the more possible it is to add a product in his/her cart and finally complete a purchase. In particular, 83.9% of the customers view only one product and only 1.48% of them add it to their cart. In addition, only 0.46% of the customers that see only one product, finally buy it. In other words, around 99,54% of the users that view only one product, leave the e-shop without buying anything. The percentage of sessions that ends up with at least one 'addtocart' and finally 'transaction' is increased in the cases that the customer views two products. However, these cases are much less (9.89% of the total sessions) than those in the first case (one product view). Furthermore, 23,61% of the sessions with ten product views has at least one 'addtocart' action and 7.71% ends up purchasing. However, only 0.09% of the sessions refers to cases that the customer views ten products.

Table 7.1 Percentage of sessions, of sessions with 'AddtoCart' and of sessions with 'Transaction' per number of viewed products.

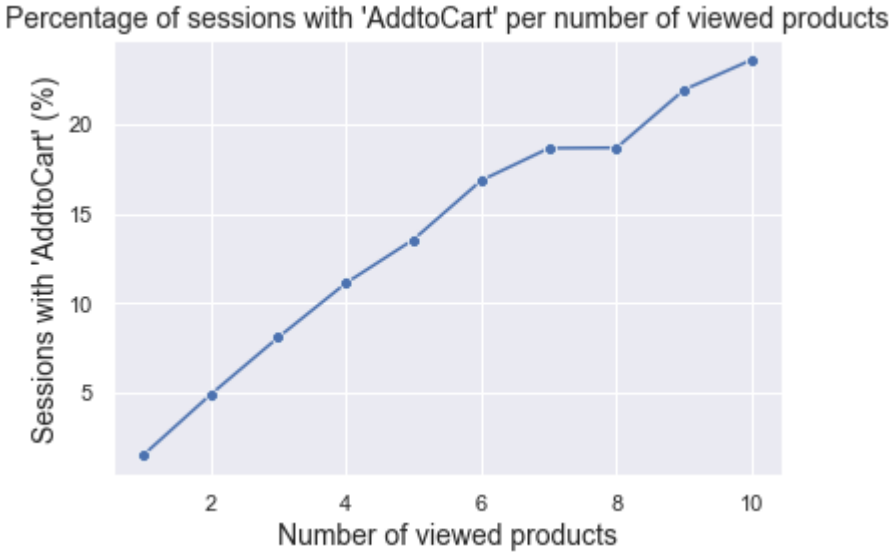
Viewed products	Sessions (%)	AddtoCart (%)	Transaction (%)
1	83.9 %	1.48 %	0.46 %
2	9.89 %	4.89 %	1.35 %
3	2.89 %	8.08 %	2.4 %
4	1.25 %	11.12 %	3.3 %
5	0.65 %	13.56 %	4.5 %
6	0.38 %	16.86 %	6.15 %
7	0.24 %	18.66 %	5.38 %
8	0.17 %	18.69 %	6.36 %
9	0.12 %	21.93 %	6.76 %
10	0.09 %	23.61 %	7.71 %

In figure 7.1, we can see that the number of sessions decreases as the number of product views increases. Users usually leave the site after seeing one or only a few products. In addition, figure 7.2 shows that the more products a user views, the more possible it is to add a product to his/her cart. Consequently, in figure 7.3, we can see that the more products a user views, the more likely they are to purchase a product. Hence, we conclude that helping customers stay longer in the e-shop and view more products, helps both the e-shop to increase its sales and customers to find products that they need to buy. This can be accomplished by providing customers with

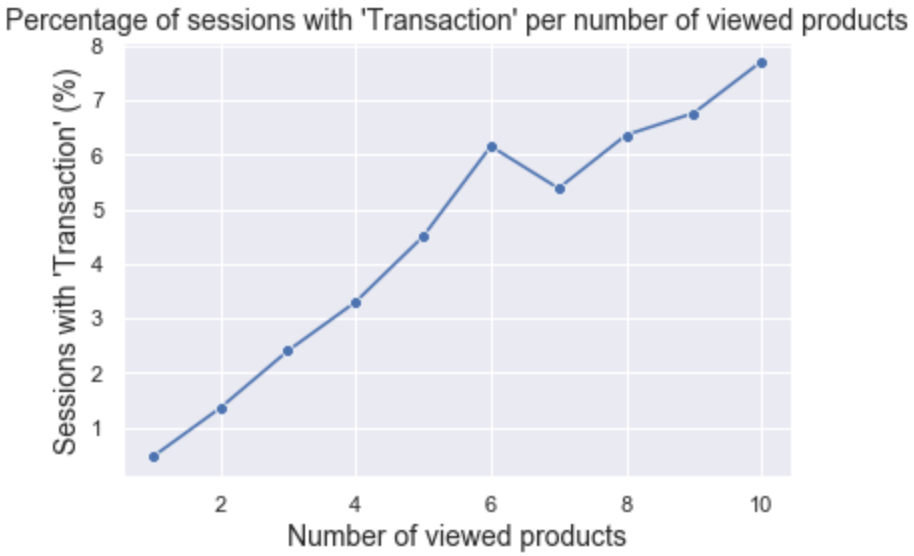
recommendations that help them navigate through the site and finally find the products that meet their preferences before leaving.



Picture 7.1 Percentage of sessions per number of viewed products



Picture 7.2 Percentage of sessions with 'AddtoCart' per number of viewed products



Picture 7.3 Percentage of sessions with 'Transaction' per number of viewed products

Having applied the methodology that is presented in detail in chapter 5, we have produced a set of ten recommendations for each product page in the Retailrocket dataset. The dataset consists of 28.241 products that belong to 6 main categories which are split in 37 subcategories. We have used a subset of 4.000 products for the evaluation process. The aim is to determine whether the produced recommendations would help customers navigate through the e-commerce site, by offering them the required next step in order to move from a product page to another. For this purpose, we compare the produced recommendations with the product pages that users visited after viewing each single product. These product views are described in paragraph 4.2.2 and are based on real actions that users performed while searching for products that meet their preferences.

Specifically, table 7.2 presents the percentage of cases that at least one recommended product matches with one product view. The test was conducted for multiple scenarios in which the importance of specific product features was tuned. These product features are the category and subcategory of each product. In addition, each scenario was tested with a window size up to 3. The window size refers to the number of consecutively events that are related with each other. Window size equal to 1 means that a product that a user viewed is linked only to the product that he/she visited next. Respectively, window size equal to 2 means that a product that a user viewed is linked to the next two products that he/she visited. It is obvious that the percentage of cases that at least one recommended product matches with the real product views is high, regardless the feature

importance. The percentage increases further with the increase of the window size. Hence, in most of the cases, the recommendations that our system produces would indeed help users move to a similar product page instead of leaving the e-commerce site.

Table 7.2: Percentage of cases that at least one recommended product matches with the product views.

	Subcategory Weight				Category Weight			
Window Size	0.5	1	1.5	2.0	0.5	1	1.5	2.0
1	94.74 %	94.79 %	94.84 %	94.86	94.71 %	94.79 %	94.79 %	94.79 %
2	96.90 %	97.01 %	96.98 %	97.03 %	96.95 %	97.01 %	97.01 %	97.03 %
3	97.68 %	97.73 %	97.68 %	97.73 %	97.71 %	97.73 %	97.76 %	97.79 %

7.1.2 Recommendation diversity

In this part of the evaluation process, the recommendation diversity of our methodology is examined. We evaluate the recommendations that our system produced for the Bestprice dataset. Finally, we compare the recommendations that our system produced with those that were scraped from the respective site. The dataset is presented in detail in paragraph 4.1. It consists of 29.541 products that belong to 22 subcategories of 6 main categories in the domain of technology. Additionally, there are 515 different product brands. A subset of 1.182 products has been used in the evaluation process. Having applied the methodology that is presented in detail in chapter 5, we have produced a set of ten recommendations for each of those product pages. The experiment was conducted for the 11 scenarios that are presented in chapter 6. In each scenario we consider the importance of the product features to be different, by tuning the corresponding weights. The average number of different brands, categories and subcategories for those sets of 10 recommendations is presented in table 7.3.

In the first 3 scenarios we see that the number of different brands decreases by increasing the corresponding weight. This means that considering the product brand as more important in the similarity calculations, results in recommendations that include more of the same brand for each product page. Respectively, in scenarios 4 and 5, the number of different subcategories decreases by the increase of the corresponding weight. The same happens for the product category feature in scenarios 6 and 7. Furthermore, in the rest of the scenarios, the weights of different combinations of the product features are tuned. The characteristics of the recommended products are affected by the weights that are assigned to each product feature.

In general, by observing table 7.3, we see that the system incorporates diversity as it does not recommend products that are almost the same with each other. Specifically, products from more than one brand are present in each set of 10 recommendations. Also, by assigning the appropriate weights, we have product recommendations from more than one subcategory. There are also cases in which the recommended products belong to more than one category. The recommendation diversity is very important for the quality of the system [29]. In content-based recommendation systems, diversity can be as important as similarity [30]. Similarity assures that the recommended products are similar to the target product. Diversity means that the recommended products are not very similar to each other. The importance of the recommendation diversity can be explained through the following example. Assume that the target product is a Dell laptop and the system recommends 10 different Dell laptops. This might probably mean that the recommended products are very similar to the target product. However, the user will not have the option to move to a laptop of a different brand. Similar problem will occur in a case that all the recommended products are of the same subcategory or category.

In the last row of table 7.3, we see the respective statistics in the product recommendations that were scraped from the site (BestPrice.gr). It is obvious that there is diversity in the recommendations concerning the brand feature. There is an average of more than 5 different brands in each set of 10 product recommendations. However, there is no diversity in the cases of subcategory and category. The system recommends products that belong only to the category and subcategory of the target product. As a result, a user does not have the option to move to a product of a similar subcategory through clicking one of the recommendations.

Table 7.3: Average number of different brands, categories, subcategories in the sets of 10 recommendations

	W_brand	W_category	W_subcategory	Brands	Categories	Subcategories
Scenario 1	1	1	1	2.53	1.13	1.27
Scenario 2	0.5	1	1	3.25	1.12	1.24
Scenario 3	1.5	1	1	2.07	1.13	1.28
Scenario 4	1	1	0.5	2.40	1.20	1.41
Scenario 5	1	1	1.5	2.57	1.10	1.19
Scenario 6	1	0.5	1	2.57	1.17	1.31
Scenario 7	1	1.5	1	2.63	1.09	1.25
Scenario 8	0.5	0.5	1	2.98	1.18	1.37
Scenario 9	1.5	1.5	1	2.10	1.11	1.22
Scenario 10	0.5	1	0.5	3.17	1.14	1.26

Scenario 11	1.5	1	1.5	2.18	1.10	1.24
Bestprice	-	-	-	5.16	1.0	1.0

7.2 Conclusions

Based on the evaluation of our proposed recommendation system and the results that were produced after conducting several experiments, some interesting conclusions and insights have been extracted. Our insights mainly refer to the way that customers behave while navigating in an e-commerce site and the value of a recommendation system that can improve the user buying experience. In addition, interesting conclusions have been drawn regarding our proposed approach of content-based recommendation system and we believe that they can help in the development of an even more efficient and customizable system.

Regarding the customer behavior, the session-based analysis that was performed showed that the number of products that a customer views while navigating through an e-shop is linked with the possibility of finally buying something. In particular, we saw that users usually leave the site after seeing one or only few products and this leads to reduced sales. The more products a user views, the more possible it is to add a product to his/her cart and finally purchase it. Hence, we conclude that helping customers stay longer in the e-shop and view more products, helps both the e-shop to increase its sales and the customers to find products that are appropriate to their preferences and needs. A recommendation system can play a significant role in this direction. Specifically, our proposed approach was proven to offer recommendations that indeed helped users move to a similar product page instead of leaving the e-commerce site.

Furthermore, interesting conclusions have been drawn regarding the characteristics of our proposed hybrid method. We saw that the characteristics of the recommendations can easily be adjusted to the desired results by tuning the product features appropriately. The system incorporates recommendation diversity without compromising similarity. This aspect turns out to be very important because most of the content-based approaches lack diversity. Considering the product brand as less important in the similarity calculations, results in recommendations that include products of different brands. Similar results can be achieved by tuning other product features. In that way, customers see recommendations that are similar to the target product but are not all very similar to each other.

In addition, our approach incorporates a hashing method that makes the product similarity calculations much faster and efficient than traditional systems. The representations of product profiles as compressed signatures, by applying the Minhash method, turns out to be effective. Finally, the approximation of product similarities by applying the LSH method results in a system that can handle thousands of products efficiently without compromising similarity.

7.3 Future work

The evaluation part has shown that the proposed recommendation system approach is a very promising content-based approach, based on feature weighting and LSH. However, additions and improvements are necessary in order to achieve even better results. Our plans to improve the system are presented in this chapter.

Firstly, we plan to incorporate more product meta data in the feature weighting method. In the current research, three product features (brand, category and subcategory) are present in the two datasets. The corresponding feature weights were tuned according to their importance in the similarity calculations. However, there are various features that can be taken into consideration. Some of them are the color, size and seasonality of products. The selection of the product features highly depends on the kind of the products. So, we could also consider testing our approach in another field besides technology.

Secondly, we plan to test the approach with alternative weighting schemes and product profile representations. In the current approach, we adjust the importance of the product features by multiplying the corresponding TF-IDF weights by small constants [39]. We believe that through different product profile representations and alternative ways of assigning weights, even better results may be achieved. For example, the textual representation of products by ngrams instead of single terms, may improve the similarity calculation results.

Furthermore, we plan to make the system more personalized by adjusting the recommendations of each product page to the actions that a user does while navigating through the e-commerce site. In the current approach, the recommended products are completely independent to the user actions. We believe that when a user visits a recommended product page, the system should take into consideration his/her previous actions. In particular, the characteristics of the previously viewed products should affect the recommendations of the product pages that the customer views next.

We have conducted an experiment in order to test the simplest case for a subset of 10.000 products in the Bestprice dataset. In this case, a product view affects the recommendations of only the next product page that the customer visits. Specifically, assuming there is a set of 30 candidate recommendations, we choose to select the 10 most similar to the previous product page recommendations. The results of the experiment seem to be promising. Indicatively we mention that each set of 10 recommendations had an average of 2.4 brands and 1.22 subcategories.

References

[1] Lops P., de Gemmis M., Semeraro G. (2011) Content-based Recommender Systems: State of the Art and Trends. In: Ricci F., Rokach L., Shapira B., Kantor P. (eds) Recommender Systems Handbook. Springer, Boston, MA

[2] Adomavicius, Gediminas & Tuzhilin, Alexander. (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. Knowledge and Data Engineering, IEEE Transactions on. 17. 734-749. 10.1109/TKDE.2005.99.

[3] Çano, Erion & Morisio, Maurizio. (2017). Hybrid Recommender Systems: A Systematic Literature Review. Intelligent Data Analysis. 21. 1487-1524. 10.3233/IDA-163209.

[4] Schafer, Ben & Konstan, Joseph & Riedl, John. (1999). Recommender Systems in E-Commerce. 1st ACM Conference on Electronic Commerce, Denver, Colorado, United States. 10.1145/336992.337035.

[5] Charikar, M.S. (2002). Similarity estimation techniques from rounding algorithms. Conference Proceedings of the Annual ACM Symposium on Theory of Computing. 380-388.

[6] Datar, Mayur & Indyk, Piotr & Immorlica, Nicole & S. Mirrokni, Vahab. (2004). Locality-sensitive hashing scheme based on p-stable distributions. Proceedings of the Annual Symposium on Computational Geometry. 10.1145/997817.997857.

[7] Lv, Qin & Josephson, William & Wang, Zhe & Charikar, Moses & Li, Kai. (2007). Multi-Probe LSH: Efficient Indexing for High-Dimensional Similarity Search .. VLDB. 950-961.

[8] Wu, Wei & Li, Bin & Chen, Ling & Gao, Junbin & Zhang, Chengqi. (2018). A Review for Weighted Minhash Algorithms.

[9] Zhang, Yin & Jin, Rong & Zhou, Zhi-Hua. (2010). Understanding bag-of-words model: A statistical framework. International Journal of Machine Learning and Cybernetics. 1. 43-52. 10.1007/s13042-010-0001-0.

[10] Das, Abhinandan & Datar, Mayur & Garg, Ashutosh & Rajaram, ShyamSundar. (2007). Google news personalization: Scalable online collaborative filtering. 16th International World Wide Web Conference, WWW2007. 271-280. 10.1145/1242572.1242610.

[11] Rajaraman, Anand & Leskovec, Jure & Ullman, Jeffrey. (2014). Mining of Massive Datasets. 10.1017/CBO9781139058452.

[12] Chum, O. & Perdoch, Michal & Matas, Jiri. (2009). Geometric min-Hashing: Finding a (thick) needle in a haystack. 2012 IEEE Conference on Computer Vision and Pattern Recognition. 17-24. 10.1109/CVPRW.2009.5206531.

[13] Chum, Ondrej & Philbin, James & Zisserman, Andrew. (2008). Near Duplicate Image Detection: min-Hash and tf-idf Weighting. BMVC 2008 - Proceedings of the British Machine Vision Conference 2008. 10.5244/C.22.50.

[14] Indyk, Piotr & Motwani, Rajeev. (2000). Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. Conference Proceedings of the Annual ACM Symposium on Theory of Computing. 604-613. 10.1145/276698.276876.

[15] Gionis, Aristides & Indyk, Piotr & Motwani, Rajeev. (2000). Similarity Search in High Dimensions via Hashing. Proceeding VLDB '99 Proceedings of the 25th International Conference on Very Large Data Bases. 99.

[16] Lv, Qin & Josephson, William & Wang, Zhe & Charikar, Moses & Li, Kai. (2007). Multi-Probe LSH: Efficient Indexing for High-Dimensional Similarity Search .. VLDB. 950-961.

[17] Qi, Lianyong & Zhang, Xuyun & Dou, Wanchun & Ni, Qiang. (2017). A Distributed Locality-Sensitive Hashing-Based Approach for Cloud Service Recommendation From Multi-Source Data. IEEE Journal on Selected Areas in Communications. PP. 1-1. 10.1109/JSAC.2017.2760458.

[18] Ramos, Juan. (2003). Using TF-IDF to determine word relevance in document queries.

[19] Wu, Wei & Li, Bin & Chen, Ling & Zhang, Chengqi. (2017). Consistent Weighted Sampling Made More Practical. 1035-1043. 10.1145/3038912.3052598.

[20] Gollapudi, Sreenivas & Panigrahy, Rina. (2006). Exploiting asymmetry in hierarchical topic extraction. 475-482. 10.1145/1183614.1183683.

[21] Manasse, Mark & Mcsherry, Frank & Talwar, Kunal. (2007). Consistent weighted sampling.

[22] Ioffe, Sergey. (2010). Improved Consistent Sampling, Weighted Minhash and L1 Sketching. Proceedings - IEEE International Conference on Data Mining, ICDM. 246-255. 10.1109/ICDM.2010.80.

[23] Zhang, Kunpeng & Fan, Shaokun & Wang, Harry. (2018). An Efficient Recommender System Using Locality Sensitive Hashing. 10.24251/HICSS.2018.098.

[24] Aytekin, Ahmet & Aytekin, Tevfik. (2019). Real-time recommendation with locality sensitive hashing. Journal of Intelligent Information Systems. 53. 1-26. 10.1007/s10844-019-00552-1.

[25] Debnath, Souvik & Ganguly, Niloy & Mitra, Pabitra. (2008). Feature weighting in content based recommendation system using social network analysis. Proceeding of the 17th International Conference on World Wide Web 2008, WWW'08. 1041-1042. 10.1145/1367497.1367646.

[26] Barranco, Manuel & Martinez, Luis. (2010). A Method for Weighting Multi-valued Features in Content-Based Filtering. 6098. 409-418. 10.1007/978-3-642-13033-5_42.

[27] Rajaraman, Anand & Leskovec, Jure & Ullman, Jeffrey. (2014). Mining of Massive Datasets. 10.1017/CBO9781139058452.

[28] Bawa, Mayank & Condie, T. & Ganesan, Prasanna. (2005). LSH forest: Self-tuning indexes for similarity search. Proceedings of the 14th International Conference on World Wide Web. 651-660.

[29] Bradley, Keith. (2001). Improving Recommendation Diversity. Proc. AICS '01.

[30] Smyth, Barry & McClave, Paul. (2001). Similarity vs. Diversity. Lecture Notes in Computer Science. 347-361. 10.1007/3-540-44593-5_25.

[31] Karimova, Farida. (2016). A Survey of e-Commerce Recommender Systems. European Scientific Journal. 12. 10.19044/esj.2016.v12n34p75.

[32] Schafer, Ben & Konstan, Joseph & Riedl, John. (2000). E-Commerce Recommendation Applications. 5. 10.1023/A:1009804230409.

[33] Han, Eui-Hong & Karypis, George. (2005). Feature-based recommendation system. 446-452. 10.1145/1099554.1099683.

[34] scikit-learn, Machine Learning in Python, scikit-learn.org

[35] I. Schoinas, C. Tjortjis, “MuSIF: A Product Recommendation System Based on Multi-source Im-plicit Feedback”, Proc. 15th Int’l Conf. on Artificial Intelligence Applications and Innovations (AIAI 19), IFIP AICT 559, pp. 660–672, Springer, 2019.

[36] datasketch 1.5.0, http://ekzhu.com/datasketch/

[37] Nalmpantis O. and Tjortjis C., “The 50/50 Recommender: a Method Incorporating Personality into Movie Recommender Systems”, Proc. 8th Int’l Conf. on Engineering Applications of Neural Net-works (EANN 17), Communications in Computer and Information Science (CCIS) Vol. 744, pp. 498–507, Springer-Verlag, 2017.

[38] Gerogiannis V.C., Karageorgos A., Liu L., and Tjortjis C., “Personalised Fuzzy Recommendation for High Involvement Products”, IEEE Int’l Conf. Systems, Man, and Cybernetics (SMC 2013), pp. 4884-4890, 2013.

[39] D. Beleveslis, C. Tjortjis, “Promoting Diversity in Content Based Recommendation using Feature Weighting and LSH”, to appear at the 16th Int’l Conf. on Artificial Intelligence Applications and Innovations (AIAI 20), 2020.

Appendix

preprocess_bestprice.py

```
"""
This script is used to preprocess the initial 'Bestprice' dataset
"""

# import libraries
import pandas as pd
import re
from nltk.tokenize import RegexpTokenizer

# read the initial 'bestprice' dataset
df = pd.read_pickle('data/df_init.pkl')

# extract the id number from each product url
df['product_id'] = df['url'].apply(lambda x: re.search(r"item\/([^']*)\/", x).group(1))

# convert category to lowercase
df['Category_lc'] = df['Category'].apply(lambda text: text.lower())
# convert SubCategory to lowercase
df['SubCategory_lc'] = df['SubCategory'].apply(lambda text: text.lower())
# convert brand_name to lowercase
df['brand_name_lc'] = df['brand_name'].apply(lambda text: text.lower())
# convert title to lowercase
df['Title_lc'] = df['Title'].apply(lambda text: text.lower())

# create a new column for 'Category' and add the string '_cat' after the category
df['Category2'] = df['Category_lc'].apply(lambda x: x+'_cat')
# create a new column for 'SubCategory' and add the string '_subcat' after the SubCategory
df['SubCategory2'] = df['SubCategory_lc'].apply(lambda x: x+'_subcat')
# create a new column for 'brand' and add the string '_brand' after the brand_name
df['brand_name2'] = df['brand_name_lc'].apply(lambda x: x+'_brand')
```

```
def replace_brand(row):
    """
    This function adds (or replaces) the brand_name2 to the product title
    :row: the available information of the product
    :return: the product title with the brand_name2
    """
    title = row['Title_lc']
    brand = row['brand_name_lc']
    brand2 = row['brand_name2']

    if brand in title:
        title2 = re.sub(brand, brand2, title)
    else:
        title2 = title +' '+ brand2

    return title2

# add brand in the product title by calling 'replace_brand' function
df['Title_lc2'] = df[['Title_lc', 'brand_name_lc', 'brand_name2']].apply(lambda x:
replace_brand(x), axis=1)

# add category in the end of the product title
df['Title_lc2'] = df[['Title_lc2', 'Category2']].apply(lambda x: x['Title_lc2']+' '+x['Category2'],
axis=1)

# add sub-category in the end of the product title
df['Title_lc2'] = df[['Title_lc2', 'SubCategory2']].apply(lambda x: x['Title_lc2']+'
'+x['SubCategory2'], axis=1)

# keep only useful columns
df = df[['product_id', 'url', 'Title', 'Category2', 'SubCategory2', 'brand_name2', 'Title_lc2']]

# keep each product only once by deleting duplicates
df = df.drop_duplicates(subset='product_id', keep='first')

# save the dataframe as pickle file
# this file will be used in order to create the TF-IDF matrix
df.to_pickle('data/product_details/df_preproc.pkl')
```

weighted_scheme.py

```
"""
This script is used to create the TF-IDF matrix and assign feature weights
"""

# import libraries
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer

# read the dataset with the preprocessed product details of bestprice.gr
df = pd.read_pickle('data/product_details/df_preproc.pkl')

# create a list (corpus) of all preprocessed product titles (product profiles)
corpus = df.Title_lc2.values.tolist()

# convert the above corpus to a matrix of TF-IDF features
tf = TfidfVectorizer()
tfidf_matrix = tf.fit_transform(corpus)


# get a list of the unique terms in the corpus
feature_names = tf.get_feature_names()

# transform the sparce matrix to a list of dicts
# each dict corresponds to each term of a product title
tfidf_list = []
for doc in range(0,len(corpus)):
    feature_index = tfidf_matrix[doc,:].nonzero()[1]
    tfidf_scores = zip(feature_index, [tfidf_matrix[doc, x] for x in feature_index])

    for i, w, s in [(i, feature_names[i], s) for (i, s) in tfidf_scores]:
        doc_dict = {'doc_id':doc, 'term_id':i, 'term':w, 'tfidf':s}
        tfidf_list.append(doc_dict)

# transform the list of dicts to a pandas dataframe
df_tfidf = pd.DataFrame(tfidf_list)

# set the weights for the 3 features (brand, category, subcategory)
# here we increase the significance of only the brand feature
brand_weight = 1.5
category_weight = 1.0
subcategory_weight = 1.0

# create a dataframe with the terms of each feature (brand, category, subcategory) along with the
weights
```

```
df_brand_weights = pd.DataFrame({'term':list(df.brand_name2.unique()),
                        'weight':brand_weight})

df_category_weights = pd.DataFrame({'term':list(df.Category2.unique()),
                        'weight':category_weight})

df_subcategory_weights = pd.DataFrame({'term':list(df.SubCategory2.unique()),
                        'weight':subcategory_weight})

# concatenate the above 3 dataframes
df_weights = pd.concat([df_brand_weights, df_category_weights, df_subcategory_weights],
axis=0)

# merge the main 'df_tfidf' with the 'df_weights' dataframe
df_tfidf = df_tfidf.merge(df_weights, left_on='term', right_on='term', how='left')
# set the weight of all the other terms to 1
df_tfidf = df_tfidf.fillna(1)

# create a new column with the final weight of each term
df_tfidf['tfidf'] = df_tfidf['tfidf'] * df_tfidf['weight']

# groupby each product(doc_id) to a row and convert the rest of the columns to lists
df_tfidf2 = df_tfidf.groupby(['doc_id'], as_index=False)['term','term_id','tfidf'].agg(lambda x:
list(x))

# add a column with the product id
df_tfidf2['product_id'] = df['product_id'].values.tolist()
# add a column with the product brand
df_tfidf2['brand_name2'] = df['brand_name2'].values.tolist()
# add a column with the product category
df_tfidf2['Category2'] = df['Category2'].values.tolist()
# add a column with the product subcategory
df_tfidf2['SubCategory2'] = df['SubCategory2'].values.tolist()

# save the dataframe as pickle file
# this file will be used in order to calculate the Minhash signature of each product
df_tfidf2.to_pickle('data/tfidfs/df_tfidf_brand_1-5.pkl')
```

minhash_lsh.py

```
"""
This script is used to create the weighted Minhash signatures and the recommendations for each product
"""

# import libraries
import pandas as pd
from datasketch import WeightedMinHashGenerator
from datasketch import MinHashLSHForest

# read the pickle file that was created by 'tfidf.py' script
# this file contains the weighted representation of all products in 'Bestprice' dataset
df_tfidf = pd.read_pickle('data/tfidfs/df_tfidf_brand_1-0.pkl')

# create an extra column with the minhash id (m1, m2 etc)
df_tfidf['Minhash_id'] = df_tfidf['doc_id'].apply(lambda x: 'm'+str(x))

# create a WeightedMinHashGenerator object with the appropriate arguments
# dim: dimension - the number of unique terms
# sample_size: number of samples (similar to number of permutation functions in MinHash)
mg = WeightedMinHashGenerator(dim=35405, sample_size=128)

def create_minhash(doc):
    """
    This function takes the weighted representation of a product and returns its Minhash signature.
    :param doc: The weighted representation of the product
    :return: The Minhash signature of the product as a Minhash object
    """
    term_ids = doc['term_id']
    tfidfs = doc['tfidf']
    tfidf_list = [0]*35405

    i = 0
    for term_id in term_ids:
        tfidf_list[term_id] = tfidfs[i]
        i += 1

    m1 = mg.minhash(tfidf_list)

    return m1

# create a minhash for each row(product) by calling the 'create_minhash' function
df_tfidf['Minhash'] = df_tfidf[0:].apply(lambda x: create_minhash(x), axis=1)
```

```
# create a list with all the Minhash signatures
minhash_list = df_tfidf['Minhash']

# create a MinHashLSHForest object with num_perm parameter equal to sample_size(=128)
# num_perm: the number of permutation functions
forest = MinHashLSHForest(num_perm=128)

# add each Minhash signature into the index
i = 0
for minhash in minhash_list:
    # Add minhash into the index
    forest.add("m"+str(i), minhash)
    i += 1

# call index() in order to make the keys searchable
forest.index()

# create the recommendations by retrieving top 10 keys that have the higest Jaccard for each
product

def make_recs(doc_id, n_recs):
    """
    This function takes the id of the target product and returns the top n_recs(=10) keys that have
the higest Jaccard
    :param doc_id: the id of the target product
    :param n_recs: the number of similar products to be returned
    :return: top n_recs keys that have the higest Jaccard for each product
    """
    query = minhash_list[doc_id]

    # Using m1 as the query, retrieve top 10 keys that have the higest Jaccard
    results = forest.query(query, n_recs)

    return results

# for each product find the top 10 most similar products by calling the 'make_recs' function
df_tfidf['recs'] = df_tfidf['doc_id'].apply(lambda x: make_recs(x, 10))

# finalize the dataset

# create a df with only the recs of each product
df_recs = df_tfidf[['product_id', 'recs']]
# expand each row to as many rows as the length of the recs list
df_recs = df_recs.set_index('product_id').recs.apply(pd.Series).stack().reset_index(level=-1,
drop=True).astype(str).reset_index()
# rename the columns
```

```
df_recs.columns = ['product_id', 'rec_m_id']

# add the brand, category, subcategory of each recommended product
df_recs = df_recs.merge(df_tfidf[['Minhash_id', 'brand_name2', 'Category2', 'SubCategory2']],
left_on='rec_m_id', right_on='Minhash_id', how='left')

# groupby each product and convert to lists
df_recs = df_recs.groupby(['product_id'], as_index=False)['brand_name2', 'Category2',
'SubCategory2'].agg(lambda x: list(x))
# rename columns
df_recs.columns = ['product_id', 'Brands', 'Categories', 'Subcategories']

# add the above information to the main dataset
df_recs2 = df_tfidf.merge(df_recs, left_on='product_id', right_on='product_id', how='left')

# create 3 columns with the number of uniique brands, categories, subcategories for the
evaluation process
df_recs2['N_Brands'] = df_recs2['Brands'].apply(lambda x: len(set(x)))
df_recs2['N_Categories'] = df_recs2['Categories'].apply(lambda x: len(set(x)))
df_recs2['N_Subcategories'] = df_recs2['Subcategories'].apply(lambda x: len(set(x)))

# save the dataframe as pickle file
df_recs2.to_pickle('data/recommendations/df_recos_brand_1-5.pkl')
```

scrape_bestprice.py

```
"""
This is the main script of the process that scrapes 'BestPrice' e-commerce site
"""

# import other scripts and classes
from bestprice import *
from general import *

# a list with a sample of the product subcategories that were scraped
# 'N' refers to the number of products that will be scraped from each subcategory
categories = [
{'Category':'Mobiles', 'SubCategory':'Mobile_phone',
'url':'https://www.bestprice.gr/cat/806/mobile-phones.html?pg={}', 'N':1600},
{'Category':'Mobiles', 'SubCategory':'Bluetooth',
'url':'https://www.bestprice.gr/cat/813/bluetooth.html?v=r&pg={}', 'N':1900},
{'Category':'Mobiles', 'SubCategory':'Handsfree', 'url':'https://www.bestprice.gr/cat/811/hands-
free.html?v=r&pg={}', 'N':2400},
]

# create an object of the bestprice class
sk = bestprice()

# for each category call the 'scrape_bestprice' function to scrape the corresponding pages
for category in categories[0:]:

    # define the path of the .csv file in which the scraped data will be saved
    csvFileName = '/home/desktop/dissertation/results/' + category['Category'] + '-' +
category['SubCategory']+'.csv'
    # initialize the .csv file
    initializeCsv(filename=csvFileName)

    print('- Going to scrape the Subcategory - ', category['SubCategory'], ' - ',
category['Category'])
    # call the function to start the scraping process
    sk.scrape_bestprice(category)
```

bestprice.py

```
"""
This is the script with the class and the functions that are used to scrape 'BestPrice' e-commerce site
"""

# import 'general' script
from general import *

# import libraries
import requests
from bs4 import BeautifulSoup
import re
from collections import OrderedDict
from itertools import cycle
import traceback
from datetime import datetime

# set counters
cnt_pages = 0
cnt_error = 0
# call 'update_proxy_pool' function in order to get a list of proxies
proxy_pool = update_proxy_pool(1)
# set the site that will be scraped in order to gather new proxies later
n_site = 0

class bestprice:
    # the class that is used to scrape the 'BestPrice' e-commerce site

    def __init__(self):
        # create the 'bestprice' object

        print('Object of class bestprice has been created.')


    def scrape_bestprice_page(self, category, brand_info, ith):
        # scrapes all the products of a page and saves the data in a .csv file

        # define global variables
        global cnt_pages
        global proxy_pool
        global n_site

        category_ = category['Category']
        subcategory = category['SubCategory']
```

```
        basic_url = category['url']
        brand_name = brand_info['brand_name']
        brand_url = brand_info['brand_url']
        next_page = brand_url.format(ith)

        # define the path of the csv file in which the scraped data will be saved
        csvFileName = '/home/desktop/dissertation/results/' + category['Category'] + '-' +
category['SubCategory']+'.csv'

        try:
            # increase the counter each time a page is scraped
            cnt_pages += 1

            # change the site from which we scrape proxies every 5 pages
            if cnt_pages%5==True:
                if n_site%2==0:
                    site = 1
                else:
                    site = 2

                # scrape proxies from the corresponding site
                proxy_pool = update_proxy_pool(site)
                print('proxy_pool is updated.')
                n_site += 1

            # try to request the page with one of the available proxies
            # try up to 20 times
            for i in range(20):
                try:
                    # get a proxy from the pool and try to request the page
                    proxy = next(proxy_pool)

                    # perform the page request and wait up to 10 seconds
                    page = requests.get(next_page, headers=headers, proxies={"http": proxy, "https":
proxy}, timeout=10)

                    # convert the page to a BeautifulSoup object
                    soup = BeautifulSoup(page.content, 'html.parser')

                    # find the html tags with the product details
                    products_grid = soup.find('div', class_='grid products products--row')
                    product_divs1 = products_grid.find_all('div', class_="product__wrapper g-1 g-xsm-
2 g-lg-3 g-xl-4 g-xxl-4 product__wrapper--even")
                    product_divs2 = products_grid.find_all('div', class_="product__wrapper g-1 g-xsm-
2 g-lg-3 g-xl-4 g-xxl-4 product__wrapper--odd")
                    product_divs = product_divs1 + product_divs2
```

```
            break
        except Exception as e:
            # move to the next available proxy if an error occurs
            pass


    # iterate through products of this page in order to extract their details
    for product_div in product_divs[0:]:
        try:
            # find the html tags of the specific product
            product_info = product_div.find('div', class_="product__main")
            product_title_div = product_info.find('h2', class_='product__title')

            # extract the url of the product page
            try:
                product_url = product_title_div.find("a")['href']
                product_page = 'https://www.bestprice.gr' + product_url
            except:
                product_page = None
            # extract the product title
            try:
                product_title = product_title_div.find("a").text
            except:
                product_title = None
            # extract the product description
            try:
                description = product_info.find('div', class_='product__description').text
            except:
                description = None
            # exctract the product price
            try:
                price = product_info.find('div', class_='product__cost-price').text
            except:
                price = None

            # create a dictionary to store the scraped product details
            row = OrderedDict()
            row['Category']= category_
            row['SubCategory']= subcategory
            row['url']= product_page
            row['N']= category['N']
            row['Title']= product_title
            row['Price']= price
            row['Description']= description
            row['Page'] = ith
```

```
                row['brand_name'] = brand_name
                row['brand_url'] = brand_url

                # write the dictionary to the csv file
                appendDictToCsv(filename=csvFileName, data=row)

            except Exception as e3:
                # continue to the next product if an error occurs
                print('Error in a product: ', e3, ' - ', product_page)

        except Exception as e4:
            # continue to the next page if an error occurs
            print('Error #4: ', e4)


    def scrape_bestprice(self, category):
        # scrapes the product brands of the category and calls the function to scrape each product page

        # define a global variable
        global proxy_pool

        # the url of the page that will be scraped
        url = category['url']
        # the number of products that will be scraped
        n = category['N']

        # first scrape the available brands from the main page
        # try to request the page with one of the available proxies
        # try up to 30 times
        for i in range(30):
            try:
                # get a proxy from the pool and try to request the page
                proxy = next(proxy_pool)
                # perform the page request and wait up to 10 seconds
                page = requests.get(url, headers=headers, proxies={"http": proxy, "https": proxy}, timeout=10)
                # convert the page to a BeautifulSoup object
                soup = BeautifulSoup(page.content, 'html.parser')
                # find the html tags with the product brands
                filters_div = soup.find('div', id='filters')
                brand_filter_div = filters_div.find('div', class_='filter-brand default-list')
                brand_lis = brand_filter_div.find_all('li')
                break
            except Exception as e:
                # move to the next available proxy if an error occurs
```

```
            pass

        # create a list to store the information about each brand
        brand_info_list = []
        # extarct the details of each brand from the corresponding html tag
        for brand_li in brand_lis:
            # extract the url of page that refers to the brand
            brand_a = brand_li.find('a')
            brand_url = brand_a['href']
            brand_url = 'https://www.bestprice.gr' + brand_url + '&pg={}'
            # extract the name of the brand
            brand_name = brand_a.text
            brand_cnt = brand_a['data-c']

            # create a dictionary with the details of the brand
            brand_info = {'brand_name':brand_name, 'brand_url':brand_url, 'brand_cnt':brand_cnt}
            # append the above dictionary to the list
            brand_info_list.append(brand_info)

        # convert the list of dictionaries to a pandas dataframe
        df_tmp = pd.DataFrame(brand_info_list)
        # convert the datatype of  'brand_cnt' column to integer
        df_tmp['brand_cnt'] = df_tmp.brand_cnt.astype(int)

        # set counters
        total_N = 0
        i_brand = 0

        # for each page of each brand call the 'scrape_bestprice_page' to scrape it
        for brand_info in brand_info_list[0:]:
            i_brand += 1
            print('- Scraping category:', category['Category'], ',subcategory:',
category['SubCategory'], ',brand:', brand_info['brand_name'], ' - ', i_brand, 'of',
len(brand_info_list))

            brand_cnt = int(brand_info['brand_cnt'])
            total_N += brand_cnt

            # fix the pagination according to the site
            pages = int(brand_cnt/25)+2
            # scrape each page
            for i in range(1,pages):
                next_page = brand_info['brand_url'].format(i)
                print('-- Scraping page: ', next_page)
                self.scrape_bestprice_page(category, brand_info, i)
```

```
# stop the process if the defined number of products have already been scraped
if total_N>n:
    break
```

general.py

```
"""
This is the script with the functions that are used from 'bestprice.py' file
"""
# import libraries
import csv
import random
from collections import OrderedDict
from openpyxl.styles import PatternFill
from itertools import cycle
import traceback
import requests
from bs4 import BeautifulSoup
import pandas as pd


#gives different user agent randomly for each request.
agent_version = '%.2f' % (random.randint(20, 100) + random.randint(1, 100)/float(100))
headers = {
   'User-Agent': 'Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/56.0.2924.87 Safari/537.36',
   'User-Agent': 'Mozilla/5.0 (compatible; MSIE 8\.0; Windows NT 5\.1; SV1)
Chrome/%s.2924.87 Safari/537.36' % agent_version
}


def get_free_proxies(url, anonymity):
   # scrapes the site with the free proxies

   # perform the page request
   page = requests.get(url)
   # convert the page to a BeautifulSoup object
   soup = BeautifulSoup(page.content, 'html.parser')

   # find the html tags with the free proxies
   table = soup.find('table', id='proxylisttable')
   table_body = table.find("tbody")
   trs = table_body.find_all("tr")

   # create a list to store the proxies
   proxies = []
   # extract the details of each proxy from the corresponding html tags
   for tr in trs[0:]:
      tds = tr.find_all("td")
```

```
            if tds[4].text.strip() in anonymity:
                ip = tds[0].text.strip()
                port = tds[1].text.strip()
                proxy = ip+':'+port
                # append the proxy to the list
                proxies.append(proxy)

    # return the list of proxies
    return proxies

def update_proxy_pool(site):
    # select one of the two sites and scrape the available free proxies
    print('Updating proxy_pool...')
    if site==1:
        print('Site No. 1')
        url = 'https://free-proxy-list.net/'
    else:
        print('Site No. 2')
        url = 'https://www.sslproxies.org/'

    # call the function to scrape the selected site
    proxies = get_free_proxies(url, 'elite proxy')

    # convert the list to cycle
    proxy_pool = cycle(proxies)

    # return the proxies
    return proxy_pool


def initializeCsv(filename=''):
    # creates a .csv filename with the needed column names

    # create a dictionary with the needed .csv columns
    dataFormat = OrderedDict()
    dataFormat['Category']= ''
    dataFormat['SubCategory']= ''
    dataFormat['url']= ''
    dataFormat['N']= ''
    dataFormat['Title']= ''
    dataFormat['Price']= ''
    dataFormat['Description']= ''
    dataFormat['Page']= ''
    dataFormat['brand_name']= ''
    dataFormat['brand_url']= ''
```

```
    # create and save the .csv file
    keys = dataFormat.keys()
    with open(filename, 'w', newline='', encoding='utf-8') as output_file:
        dict_writer = csv.writer(output_file)
        dict_writer.writerow(keys)

def appendDictToCsv(filename='', data={}):
    # writes a python dictionary to a .csv file

    keys = data.keys()
    with open(filename, 'a', newline='', encoding='utf-8') as output_file:
        dict_writer = csv.DictWriter(output_file, keys)
        dict_writer.writerow(data)
```