Data Mining Project 2019/2020

Alberto Bellumat
alberto.bellumat@studenti.unitn.it
ID : 213898
Master's degree : First Year
Computer Science : ICT Innovation
Don't know yet

1 INTRODUCTION

For the project of the Data Mining course, we are given the following situation: in a typical supermarket, customers buy different kinds of ingredients (food) for the recipes they are planning to prepare at home.

We have two datasets: a market basket dataset and a dataset that provides for each recipe its ingredients, called recipe dataset. The main task is to identify types of customers based on the food they eat.

Note that the customers may buy only some of the necessary ingredients for making a recipe; also, the same ingredients may be used in different recipes, or for the same recipe different ingredients may be used.

For each "kind" of customer, a basic textual description should be provided that gives a human understanding of what this kind of customer is about.

There are many ways to define the "types" of customers: the customers who plan to make certain recipes, the customers who plan to make recipes of certain cuisines, and so on.

In the Problem Statement section, we will explain the various challenges hidden in the main task.

2 RELATED WORK

In this section, we will introduce the algorithms and articles that are related to the problem at hand.

We start by explaining the algorithms. Despite at first sight they seem to be not related to the main task at all, the reader will understand better in the next section.

Woodstock '18, June 03–05, 2018, Woodstock, NY
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

Almost all algorithms and notions we used for the project are present in the book Mining Of Massive Datasets, available at the following link http://www.mmds.org/#book.

TF.IDF

TF.IDF is a weighting scheme that assigns each word in a document a weight based on its term frequency (TF) and inverse document frequency (IDF).

It is introduced in the book Mining of Massive Datasets, subsection 1.3.1.

We denote W as a word in the d document, which belongs to a corpus C. To compute the term frequency, denoted as TF(W, d), and the inverse document frequency, denoted as IDF(W, C), there are many ways, but we stick with the ones used in the coursebook; with a small addition regarding the IDF.

The TF(W, d) is computed as the number of occurrences of the term W in d divided by the maximum number of occurrences of any term in d.

$$TF(W,d) = \frac{(No\ of\ occurrences\ W\ in\ d)}{(Max\ no\ of\ occurrences\ of\ any\ term\ in\ d)} \tag{1}$$

We compute the inverse document frequency, denoted by IDF(W, C) , as the logarithm of the division between the total number of documents in C and the number of documents, in C, containing the term W.

$$IDF(W,C) = \log(\frac{(No\ of\ documents\ in\ C)}{(No\ of\ documents\ in\ C\ containing\ W)}) \tag{2}$$

Then, TF.IDF(W, d ,C) is the product of TF(W, d) and IDF(W, C).

$$TF.IDF(W,d,C) = TF(W,d) \cdot IDF(W,C) \tag{3}$$

Stemming

Stemming is the process of reducing an inflected word W to its root form.

Even though we did not study any particular algorithm, we clarify that the Porter Stemming Algorithm was used. You can check more at the following link https://tartarus.org/martin/PorterStemmer.

A-Priori

The A-Priori is an algorithm for frequent item set mining and association rule learning over relational databases.

It is deeply explained in the book Mining of Massive Datasets, subsection 6.2.5.

The frequent itemsets determined by the algorithm can be used to discover association rules.

Levenshtein distance

The Levenshtein distance between two words W and Q is the minimum number of single-character edits (insertions, deletions, or substitutions) required to change one word into the other.

It is presented in the book Introduction to Information Retrieval, subsection 3.3.3.

Cosine Similarity

The cosine similarity computes the cosine of the angle between two vectors A and B. It is calculated as the dot product of A and B, divided by the product of their lengths.

$$\cos(\theta) = \frac{A.B}{\|A\|\|B\|} = \frac{\sum_{n=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \tag{4}$$

One advantage of cosine similarity is its low-complexity, especially for sparse vectors: only the non-zero dimensions need to be considered. This advantage will be very useful for our solution.

Cosine similarity is introduced, not directly, in the book Mining Of Massive Datasets, subsection 3.5.4.

Articles And Books Related

Our solution was greatly inspired by the following article https://towardsdatascience.com/tf-idf-for-document-ranking-from-scratch-in-python-on-real-world-dataset-796d339a4089.

The author shows various steps for computing the similarity between a query document and a collection of documents; he uses in the process text normalization, stemming, TF.IDF, and cosine similarity.

We also implemented a very naive spelling corrector in the solution, based on the one defined by Peter Norvig at the following link https://norvig.com/spell-correct.html.

The task of computing text similarity is discussed in the subsection 9.2.2 of the book Mining of Massive Datasets; it is also deeper explained in the section 6.3 of the book Introduction to Information Retrieval by Christopher D. Manning. We used the information in both books for scoring documents, given a query document.

3 PROBLEM STATEMENT

In this section, we will explain the faced problems, and how we will apply the algorithms explained in the previous section. The next section provides a detailed description of our solution.

Before we go any deeper, we need to clarify the notions and assumptions we are making regarding the two input datasets, based on the real ones provided for the project. Most of the notions are taken from the coursebook Mining of Massive Datasets.

We start by defining a document as a sequence of words and corpus a collection of documents. The notion of corpus will be mostly used in the third step of our solution.

An example of corpus is {"test summary", "Rome every day"}, which contains the two documents "test summary" and "Rome every day".

We define a market-basket model a many-to-many mapping between two kinds of objects: the baskets and the items.

A basket is a set of items, and we assume that the number of items in a basket is much smaller than the total number of items.

The market-basket model can be used for many kinds of data, but in our case, we will use it for the analysis of true market baskets in step two. By true market baskets, we mean physical shopping carts brought to the register of a supermarket for checkout.

Thus, items and baskets acquire deeper meanings: the items are the products that a supermarket sells, and the baskets are sets of products bought by undefined customers. Also, for our solution, we will consider an item as a sequence of words.

So, for example, the basket { "whole milk", "cooking chocolate" } has the following products "whole milk" and "cooking chocolate", where the product "whole milk" is represented by the words "whole" and "milk" and the product "cooking chocolate" is represented by the words "cooking" and "chocolate". The products in the basket were brought to the register of a supermarket for checkout by an undefined customer.

Then, we define a recipe as a set of ingredients, which are used for making a particular dish. A recipe **must** belong to a cuisine, which is a collection of recipes.

A clear example of a recipe is { "eggs", "tomato", "salt", "vegetables" } belonging to the cuisine "Italian" and is defined by the following ingredients "eggs", "tomato", "salt" and "vegetables". The ingredients in the recipe are used for making a particular dish of Italian cuisine.

As you may guess, recipes and baskets are, conceptually speaking, very similar. Both are sets of items, which in turn defines them, and such similarity will be very crucial in the third step of the solution.

Now, let us provide a formal description of the market basket dataset and recipe dataset.

A market basket dataset is a collection of baskets recorded by a local supermarket. It is crucial to note that, in the market basket dataset, there is no data that can give us any information regarding the customers, such as identification codes or names.

So, for example, there could be the customer Jhon Doe who purchased Italian ingredients, but, in the dataset, Jhon Doe is represented as a basket of the items he bought.

Also, we have no data regarding the supermarket itself, such as its name and location.

A recipe dataset is a collection of recipes and their cuisines.

Assuming the existence of a market basket dataset and a recipe dataset, the main task implies the solving of various problems, which we divided in four steps:

1. Data Cleaning
2. Market Basket Analysis
3. Comparison between baskets and recipes
4. Customer Profiling

First Step : Data Cleaning

Unfortunately, we cannot simply assume that the data in the two datasets is immediately ready to be processed.

Since we are working with text data, it would be very convenient to normalize it, which is to transform it into a canonical form that it might previously not have had. Thus, we make sure that each word in the two datasets has no numbers, specials characters, and is in lowercase.

It is also reasonable to expect misspelling errors between the two datasets. For example, we could find in the market basket dataset the word "applu", which might reasonably refer to the word "apple" in the recipe dataset; thus, we will perform spelling correction by using the **Levenshtein distance**.

After the correction process, we will delete those words that are very commonly used and can be deleted because they do not provide any information at all; such words are also defined as **stopwords**. Examples of stopwords are "the" and "a".

Then, we will deal with cases of words like plural nouns, such for example "apple" and "apples", and verbs, such as "shredding" and "shredded"; these cases can create ambiguity when we are in the third step of our solution. Therefore, we will replace each word with their root form, obtained by applying the **stemming** technique.

After the cleaning process is done, we will have a more clear view of the data; it will significantly help us when we compare each basket to each recipe.

In summary, we are using in the first step stopwords, Levenshtein distance, and stemming.

In the next section, we denote the Levenshtein distance as Levenshtein(W, Q) and Stemming as stem(W), where W and Q are words.

Second Step : Market Basket Analysis

Now, before we compare the baskets and recipes, there might be insightful frequent patterns in the baskets of the market basket dataset. We may find interesting information which could help us in the third step.

We start by finding frequent itemsets with the application of the **A-Priori** algorithm with **support threshold** S; a typical S would be the 1% of the baskets.

After the frequent itemsets are determined, we find within them the association rules with confidence equal or greater than C; the **confidence threshold** C of such rules should be reasonably high, say 50%, or else the rules might be useless.

However, we are more interested in the association rules that represent true relationships, where the items in the left affect somehow the item in the right; therefore, of the association rules gathered so far, we will collect only those that have interest equal or greater than I; a reasonably **interest threshold** I is 0,5.

Note that we are not interested in rules with negative interest, meaning that the presence of the items in the left discourages the item in the right.

After the filtering process, the discovered association rules have support equal or greater than S, confidence equal or greater than C, and interest equal or greater than I.

Then, we will use such rules to "characterize" the baskets. By characterizing them, we influence the predictions towards certain recipes rather than others.

For example, given the dataset of a supermarket of Trento, we discover that the customers who buy "whole milk" are very likely to buy "cooking chocolate" as well. Maybe the reason of such strong relationship is that the customers plan to make a dessert, which requires “whole milk” and “cooking chocolate”. If, in the dataset, we find a basket containing "whole milk" but not "cooking chocolate", then there is a good chance that the customer who bought "whole milk" might already have at home the product "cooking chocolate". Thus, we "characterize" the basket of the customer by storing the item "cooking chocolate" to the said basket, even though the customer did not actually buy it.

In summary, what we are doing is the following: "Given the following strong association rule {"whole milk"}$\rightarrow$"cooking chocolate" , if we find a basket that contains the product "whole milk" but not the product "cooking chocolate", then we can say that there is a good chance that the customer who bought "whole milk" might have already the product "cooking chocolate" at home. Thus, we apply such prediction by storing the item "cooking chocolate" to the basket".

When we compare the basket to the recipes, we will keep in count also the predicted product "cooking chocolate", stored in the basket.

In the next section, the A-Priori algorithm is denoted as A-Priori(S), where S is the support threshold.

Third Step : Comparison between baskets and recipes

We cleaned the data and tried to discover frequent patterns in the market basket dataset, now we need to see which recipes one customer could make with the items he bought.

It is logical to think that certain ingredients have particular importance in a cuisine rather than other cuisines.

For example: let us assume that, in the recipes dataset, there are 490 out of the 500 recipes in Italian cuisine that use the ingredient "olive" and only 10 out of 400 recipes in the Greek cuisine use "olive". If we compare the following basket {"olive"} to the Italian and Greek recipes, we are more prone to say that the customer is making Greek recipes than Italian recipes, because the ingredient "olive" is essential for those 10 Greek recipes than for those 490 Italian recipes.

So, the reasoning is the following: "Given the basket {"olive"}, the product "olive" gives us very little information to which Italian recipes a customer might plan to cook, because "olive" is used in almost every Italian recipe, but it gives much more information if we compare the basket to the Greek recipes". Thus we need an algorithm that allows us to add a weight to each ingredient in order to represent their importance relative to the cuisine they are used in; the algorithm we are talking about is **TF.IDF**.

However, if we plan to apply TF.IDF, we need to consider the baskets and the recipes as documents, and not as sets of items anymore.

The main reason why, in the cleaning part, our focus was entirely directed towards only the single words was that it greatly help us to use the TF.IDF algorithm.

Then, we will compare a document basket and the document recipes using the **cosine similarity**.

For each basket, we will return the top N recipes with the highest similarity scores in a decreasing order, and the result will be saved on a file; we decided N to be 4.

Note that the variable N is used because otherwise we would return, given a basket, thousands of matched recipes; thus, it is wiser to filter the matches.

Then, from the generated data, we can group the users based on many criteria we may want.

In the next section, we denote TF.IDF as TF.IDF(W, d, C) and cosine similarity as cosineSimilarity(A, B), where W is a word, d is a document in the corpus C, and A and B are vectors.

Fourth Step : Customer Profiling

From the third step, we know, given a basket, the top four recipes one (undefined) customer could make with the products he/she bought, based on the cosine similarity score.

In this final step, we will create **customer profiles**. A customer profile, in our solution, is a vector whose components are the M most common recipes we discovered in a market basket dataset, taking into consideration also the position of the recipes in the rankings. The M indicates first, second, and so on.

An example is the following: given four baskets, we discover that the products in the baskets are used to make respectively {Rec1, Rec4, Rec8, Rec5}, {Rec2, Rec10, Rec8, Rec1} , {Rec1, Rec10, Rec16, Rec12} and {Rec1, Rec5, Rec8, Rec12}. Remember that the ranked recipes are sorted in decreasing order.

Then, a customer profile would be {Rec1, Rec10, Rec8, Rec12}. That is, in the first position, Rec1 is the most common recipe; in the second position, Rec10 is the most common recipe; in the third position, Rec8 is the most common recipe; and in the fourth position, Rec 12 is the most common recipe.

When we are looking for the first most common recipes, we are also taking into consideration in which order they appear compared to the other recipes of the same ranking set.

Now, we could repeat the same process for creating another profile, looking for the second most common recipes, and so on.

4 SOLUTION

In this section, we describe the detailed solution for each challenge, given the algorithms and notions explained in the previous section.

We use the notations of the previous section for explaining the solution in a pseudo-code style.

Note that during the process we will left the cuisines untouched.

First Step : Data Cleaning

We start by replacing every single word in the two datasets by their normalized form.

The process is also called "Text Normalization" in the Natural Language Process field. There are no standard rules for the normalization process; in our case, given a word W, the normalization, denoted by the function normalize(W), consists of the following steps:

(1) Converting all letters in W to lower case.
(2) Removing all numbers in W.
(3) Removing all punctuations, accent marks and other diacritics in W.
(4) Return the normalized word W.

After each single word is normalized, now what we would like is that each word in the basket dataset to, if possible, exactly match to at least one of the words in the recipes dataset. There could be misspelling errors and variations of the words, so we will also perform spelling correction by using the Levenshtein distance.

We collect and save each word in the recipes datasets to a variable called *temporary*; the variable is initialized empty.

Then, each single word W in groceries dataset will be replaced by the result of the function called correction(W), which compute the following steps:

(1) Return W if W is found in *temporary*.
(2) If W is not present in *temporary*, then find the set L1 of words for which the Levenshtein(W, P) equal to 1, where P belongs to L1. Of all the words in L1, find the word Y which is the most common in *temporary*.
(3) If L1 is empty, then find the set L2 of words for which Levenshtein(W, P) equal to 2, where P belongs to L2. Of all the words in L2, find the word Y which is the most common in the *temporary*.
(4) If L2 is empty, return an empty string.
(5) Return Y.

Let us explain clearly what the function correction(W) does. If W is present in *temporary*, then there is no need to correct it. If the word W cannot be found in *temporary*, then in the second and third steps, we try to correct it, using *temporary* as a dictionary. The fourth step means that the word W is not present in *temporary*, and we cannot find a word P in temporary such that Levenshtein(W, P) is equal or lesser than 2; thus, we will return an empty string. The fifth step means that we return the most common word either in the set L1 or L2. Note that we could find all the words such that Levenshtein distance is equal or greater than 3, but there is a high chance that it will create noisy data; thus, the tradeoff is not worthy, and we will stick with a maximum distance of 2.

Then, we create the variable *StopWords*, which is the set of all the predefined stopwords.

We replace each word in both datasets with their root form through stemming; that is, each word W in both the datasets will be replaced by the result of the function stemmingWords(W), which compute the following steps:

(1) If W is in *StopWords*, return an empty string.
(2) If the length of W is 1, return an empty string.
(3) Return the result of stem(W).

If W is a stopword, return an empty string; if W is a single character, return an empty string; if W is not a stopword or a single character, then return its root base.

Second Step : Market Basket Analysis

In this step, we apply the A-Priori algorithm with support threshold S, meaning we apply the A-Priori(S), and store the frequent itemsets in a variable, called *freqItemsets*. In *freqItemsets*, we find the association rules with confidence equal or greater than C, and interest equal or greater than I.

Now, we use the association rules to characterize the baskets; we replace each basket B with the result of the function characterization(B), which computes the following steps:

(1) Check if one or more items of the basket B are in the left side of one or more association rules; if the right items of the matched association rules are not in the basket, store them.
(2) Return B.

Third Step : Comparison between baskets and recipes

In this part, we now consider the baskets and recipes not as a set of items anymore, but instead as documents. Such conversion can be easily achieved: a document is a sequence of words, and the items of a basket or recipe are represented as sequences of words; thus, a converted document is obtained by creating an empty string and adding it each word present in a basket/recipe in order of appearance.

For example, the basket {"grape tomato", "light salt"} becomes the document "grape tomato light salt".

So, we denote the conversion process as conversionToDocument(L), where L is a set of items, which executes the following steps :

(1) Create an empty variable, called *temp*.
(2) For each word in L, store it in *temp*.
(3) Return *temp*.

And we replace each basket and recipe with the result of the conversionToDocument function.

Consequently, the definition of a cuisine changes as well; now, it is a collection of (recipes) documents, meaning that it is a corpus.

At this point, we can apply the algorithm TF.IDF. Note that we calculate the IDF(W, C) relative to the cuisine C the word W is used. This means that a word W will have an IDF value in a certain cuisine and a totally different IDF value in others.

By doing so, we create for each cuisine C a word-document matrix associated with C, denoted as T(C). In a word-document matrix, each row corresponds to a word used in the cuisine C, and each column corresponds to a document in C. The value in each entry $T[W_i, d_j]$ of the matrix is the value of TF.IDF(W_i, d_j, C), where C is the cuisine, W_i is the i-nth word in d_j , which is the j-nth document in C.

For each document basket B, we will pass it to the function comparison(B), which computes the following operations:

(1) Create an empty variable, called *tempStore*.

(2) For each cuisine C, perform steps 3, 4 and 5.
(3) Convert the basket B to a vector V, whose components are calculated, for each word W in B, as IDF(W, C) x TF(W, B), that is the IDF value of W in the corpus C multiplied by the TF value of W in the document basket B.
(4) Calculate the cosine similarity between the vector V and each column D of the term-document matrix T(C), such that D is the representation of the recipe R that shares at least one word with B.
(5) Store the columns D, their represented recipes and similarity scores in *tempStore*.
(6) After you iterate over all the cuisines, return the top four highest similar documents in *tempStore*.

Let us explain what the function does.

In the first step, we create an empty variable, in which we will store the result in each cuisine.

In the second step, we iterate over all the cuisines.

In the third step, we need to represent the document basket B as a vector, in order to compare it to the recipes in C. If there are any words in B that do not appear in any recipes in C, it means the IDF values of such words do not exist in C. If a word W is present in the recipes, but not in B, then its value in the vector V is equal to 0, because TF(W, B) = 0. Note that we are not adding the document B to the corpus C, and we are using the IDF values already encoded in C. By doing so, we can compute the word-document matrix T(C) only once, and reuse it any time we want.

In the fourth step, we compare the vector V to the columns of the word-document matrix T(C); as said previously, the columns of T(C) are the recipes represented as vectors. Now, a naive approach would be to compare to ALL the columns in T(C), but it would be computationally expensive. Instead, a trick is this: if basket B and a recipe R have no words in common, their cosine similarity is 0. Thus, we will compare the vector V, which is a representation of the basket B, to only those columns D that have for at least one word W in B the value TF.IDF(W, d, C) greater than 0, where d is the recipe represented by the column D. Such columns represent recipes that share at least one word with B.

In the fifth step, we store the recipes and their similarity score in a variable.

In the last step, the iteration is finished, and we return the top four recipes, whose similarity scores are the highest in *tempStore*.

We save the results of comparison(B) in a file, which we call *similarRecipes*.

Fourth Step : Customer Profiling

We look in the file *similarRecipes*, and we create M customer profiles, where M indicate the first most common recipes to the M most common recipes. Then, we save the customer profiles in a file, called *finalReport*.

5 EXPERIMENT EVALUATION

We now apply our solution to the real datasets in Python language, and see how the data will react/change. You can see the python code in the file *solution.ipynb*.

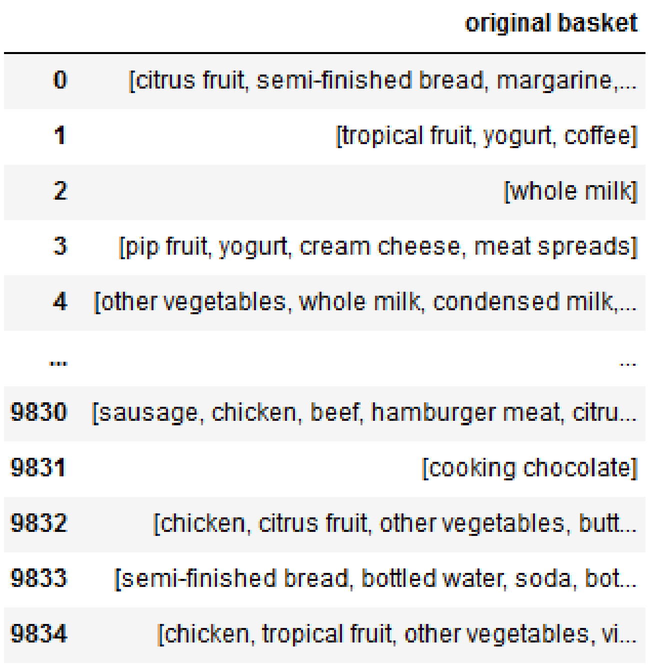
To be clear, we will not apply our algorithm directly to the files, but instead, we will work on their representation.

In Python, we can represent the two datasets as data frames. A data frame is a 2-D labeled data structure. One can also consider a data frame as an in-memory representation of an excel sheet via Python programming language.

By doing so, we keep the original files intact, while still working on the data contained in said files.

One note: the real market basket dataset has one column for each item in a basket. Thus, for simplicity, in the data frame, which represents the market basket dataset, we collect the items in one list inside on a single column.

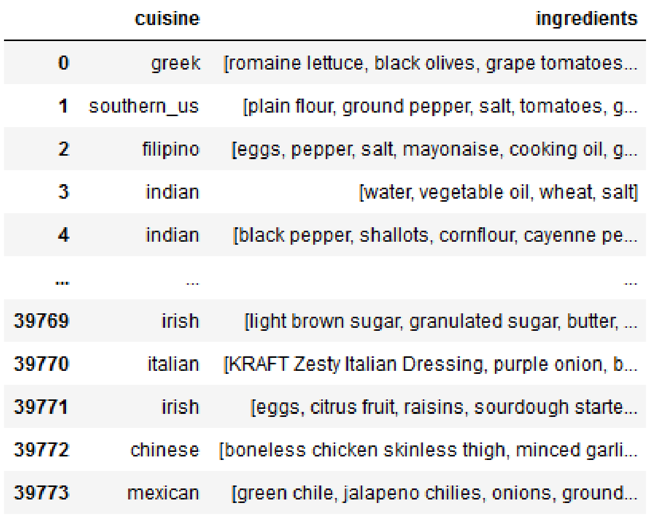
The datasets, represented now as data frames, have the structures in figure 1 and figure 2.



	original basket
0	[citrus fruit, semi-finished bread, margarine,...
1	[tropical fruit, yogurt, coffee]
2	[whole milk]
3	[pip fruit, yogurt, cream cheese, meat spreads]
4	[other vegetables, whole milk, condensed milk,...
...	...
9830	[sausage, chicken, beef, hamburger meat, citru...
9831	[cooking chocolate]
9832	[chicken, citrus fruit, other vegetables, butt...
9833	[semi-finished bread, bottled water, soda, bot...
9834	[chicken, tropical fruit, other vegetables, vi...

Figure 1: Market Basket Dataset represented as a data frame. Each row is a basket.

We stated that the baskets and recipes are sets of items. In Python terminology, the baskets and recipes are represented as lists of items. A list in python is similar to an array, with the only difference that it can contain different types of elements inside.



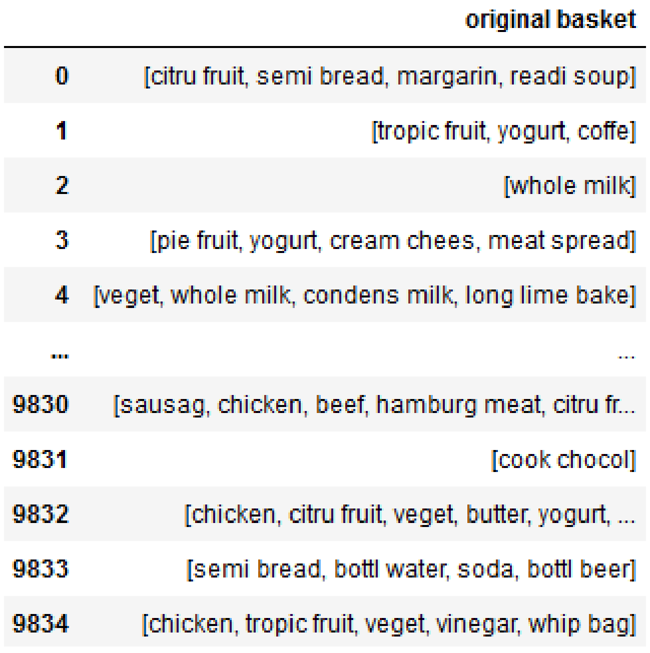
	cuisine	ingredients
0	greek	[romaine lettuce, black olives, grape tomatoes...
1	southern_us	[plain flour, ground pepper, salt, tomatoes, g...
2	filipino	[eggs, pepper, salt, mayonaise, cooking oil, g...
3	indian	[water, vegetable oil, wheat, salt]
4	indian	[black pepper, shallots, cornflour, cayenne pe...
...	...	...
39769	irish	[light brown sugar, granulated sugar, butter, ...
39770	italian	[KRAFT Zesty Italian Dressing, purple onion, b...
39771	irish	[eggs, citrus fruit, raisins, sourdough starte...
39772	chinese	[boneless chicken skinless thigh, minced garli...
39773	mexican	[green chile, jalapeno chilies, onions, ground...

Figure 2: Recipe dataset represented as a data frame. Each row is a recipe and its cuisine.

Data Cleaning

We normalize, correct and stem each word in both datasets.

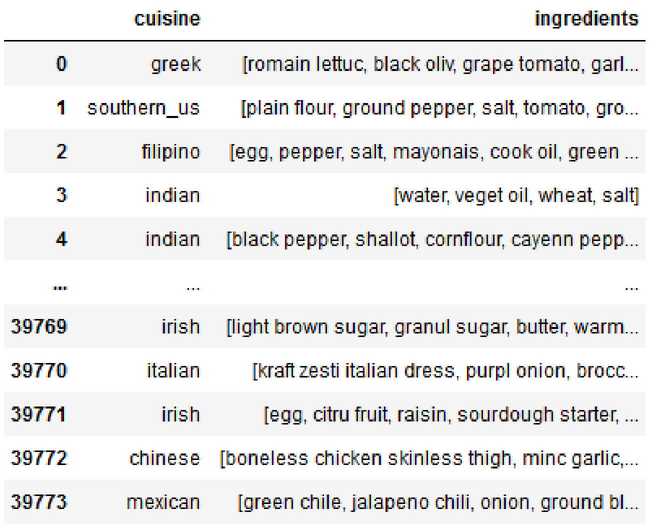
After this phase, the data has the structures in figure 3 and figure 4.



	original basket
0	[citru fruit, semi bread, margarin, readi soup]
1	[tropic fruit, yogurt, coffe]
2	[whole milk]
3	[pie fruit, yogurt, cream chees, meat spread]
4	[veget, whole milk, condens milk, long lime bake]
...	...
9830	[sausag, chicken, beef, hamburg meat, citru fr...
9831	[cook chocol]
9832	[chicken, citru fruit, veget, butter, yogurt, ...
9833	[semi bread, bottl water, soda, bottl beer]
9834	[chicken, tropic fruit, veget, vinegar, whip bag]

Figure 3: Market Basket data frame after Data Cleaning process

As you can see, each word is in lower-case, with no presence of any numbers or special characters; keep in mind that we left untouched the cuisines.



	cuisine	ingredients
0	greek	[romain lettuc, black oliv, grape tomato, garl...
1	southern_us	[plain flour, ground pepper, salt, tomato, gro...
2	filipino	[egg, pepper, salt, mayonais, cook oil, green ...
3	indian	[water, veget oil, wheat, salt]
4	indian	[black pepper, shallot, cornflour, cayenn pepp...
...	...	...
39769	irish	[light brown sugar, granul sugar, butter, warm...
39770	italian	[kraft zesti italian dress, purpl onion, brocc...
39771	irish	[egg, citru fruit, raisin, sourdough starter, ...
39772	chinese	[boneless chicken skinless thigh, minc garlic,...
39773	mexican	[green chile, jalapeno chili, onion, ground bl...

Figure 4: Recipe data frame after Data Cleaning process

Note that there are not the stem forms of the words "finished" and "other" in the market basket data frame. This means that, during the correction, the word "finished" was not present in the variable *temporary*, and there was no word W in *temporary* such that the Levenshtein(W, "finished") was equal or lesser than 2; thus, we replaced the word "finished" with an empty string.

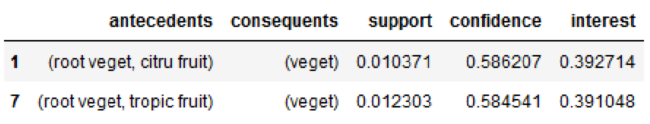
For "other", we defined it as a stopword; thus, we replaced it with an empty string.

From this phase, we get that all the words in the market basket data frame are also present in one or more recipes in the recipe data frame.

Market Basket Analysis

After the data is cleaned, we would like to discover strong association rules in the grocery dataset.

We apply the A-priori algorithm, with support threshold to be 1% of the baskets, and then we find the association rules with confidence equal or greater than 50% , and interest equal or greater than 0,35. We find the association rules, represented in figure 5.



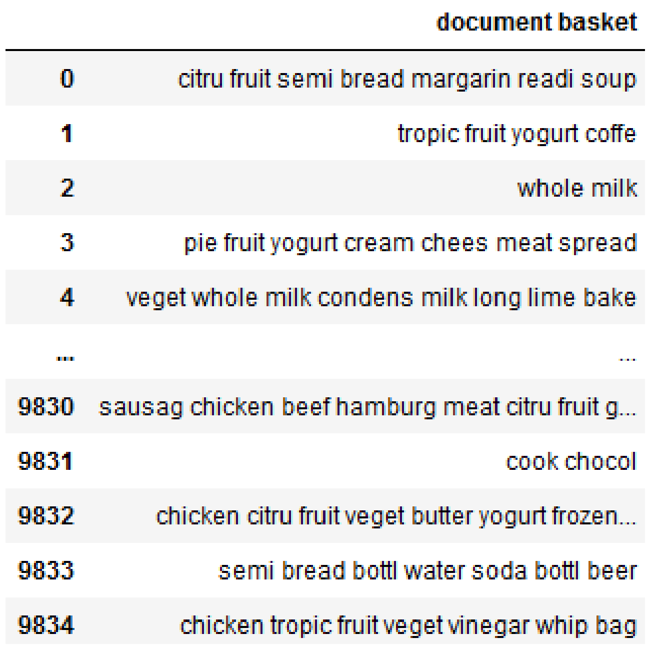
	antecedents	consequents	support	confidence	interest
1	(root veget, citru fruit)	(veget)	0.010371	0.586207	0.392714
7	(root veget, tropic fruit)	(veget)	0.012303	0.584541	0.391048

Figure 5: Association rules represented as a data frame. Each row is an association rule.

What we do now is to search in each basket if there are the items in the left side of one or more of the association rules; if the left items are in the basket and the right items are not inside it, we store the right items in the basket. For example, if we have a basket {"root veget", "tropic fruit", "chicken"}, then we add the item "veget" to the basket, acting as the customer bought it.

Comparison between baskets and recipes

We convert the baskets and recipes to documents. In the Python environment, it means we convert the lists into strings.



	document basket
0	citru fruit semi bread margarin readi soup
1	tropic fruit yogurt coffe
2	whole milk
3	pie fruit yogurt cream chees meat spread
4	veget whole milk condens milk long lime bake
...	...
9830	sausag chicken beef hamburg meat citru fruit g...
9831	cook chocol
9832	chicken citru fruit veget butter yogurt frozen...
9833	semi bread bottl water soda bottl beer
9834	chicken tropic fruit veget vinegar whip bag

Figure 6: Market basket data frame converted

In figure 6 and 7, you can see the converted data frame.

For the comparison of each basket to each recipe, we will show what happens in the "greek" cuisine, having as document basket the following "ziti activ semi bread".

We create the word-document matrix associated to the "greek" cuisine, represented in the figure 8.

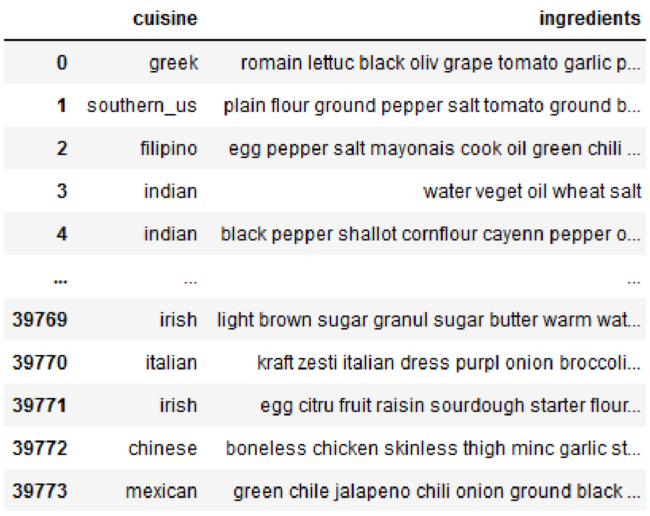
Then we need to represent the document basket "ziti activ semi bread" as a vector V, which the reader can clearly see in figure 9.

Now, we compare V to those columns which have the TF.IDF values of at least one of the words in B greater than 0.

Then we calculate the cosine similarity between V and the matched columns and store the encoded recipes and their similarity scores in a variable.

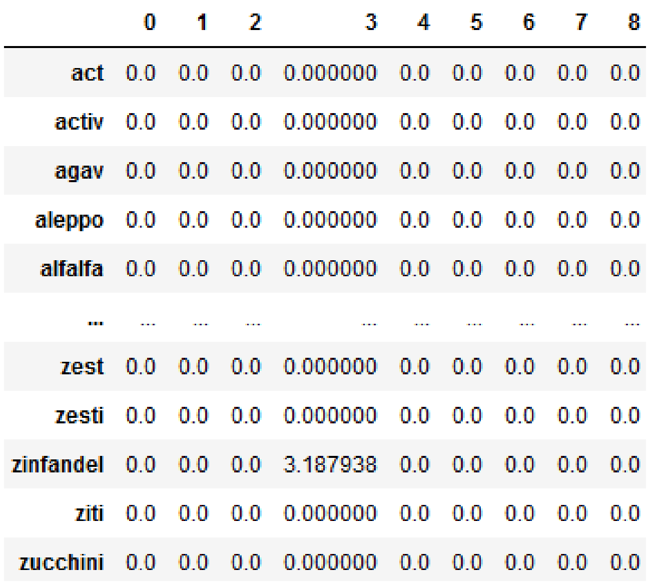
This process is done for every cuisine in the recipe data frame.

Then for the document basket B we return the four recipes, whose similarity scores are the highest in the variable.



	cuisine	ingredients
0	greek	romain lettuc black oliv grape tomato garlic p...
1	southern_us	plain flour ground pepper salt tomato ground b...
2	filipino	egg pepper salt mayonais cook oil green chili ...
3	indian	water veget oil wheat salt
4	indian	black pepper shallot cornflour cayenn pepper o...
...	...	...
39769	irish	light brown sugar granul sugar butter warm wat...
39770	italian	kraft zesti italian dress purpl onion broccoli...
39771	irish	egg citru fruit raisin sourdough starter flour...
39772	chinese	boneless chicken skinless thigh minc garlic st...
39773	mexican	green chile jalapeno chili onion ground black ...

Figure 7: Recipe data frame converted



	0	1	2	3	4	5	6	7	8
act	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0
activ	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0
agav	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0
aleppo	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0
alfalfa	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...
zest	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0
zesti	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0
zinfandel	0.0	0.0	0.0	3.187938	0.0	0.0	0.0	0.0	0.0
ziti	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0
zucchini	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0

Figure 8: Word-document matrix. For visualization purpose, we show only the first 9 recipes in the Greek cuisine.

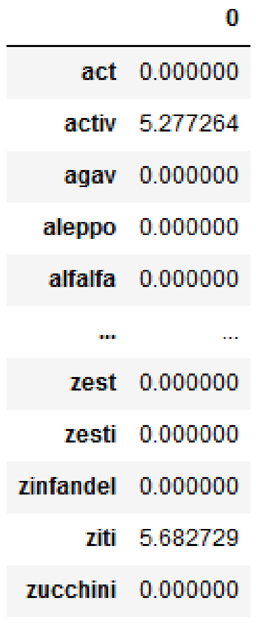
We repeat for every document basket in the market basket data frame and store the results in a file.

Customer Profiling

From the file we obtained previously, we generate two customer profiles, and we store said profiles in *finalReport.txt*.

The first customer profile has the following recipes :

(1) butter, yeast, purpos flour, salt, sugar, beer (Avg.Score: 0.61).



	0
act	0.000000
activ	5.277264
agav	0.000000
aleppo	0.000000
alfalfa	0.000000
...	...
zest	0.000000
zesti	0.000000
zinfandel	0.000000
ziti	5.682729
zucchini	0.000000

Figure 9: Vector V represented as a data frame

(2) milk, purpos flour, cook oil, salt, larg egg, beer (Avg.Score: 0.61).
(3) ground black pepper, garlic, dri thyme, butter, carrot, can low sodium chicken broth, bake potato, salt, veget, heavi cream, onion (Avg.Score: 0.55).
(4) soy sauc, green onion, long grain rice, chuck roast, veget oil, water, sesam oil, toast sesam seed, lettuc leav, beer (Avg.Score: 0.52).

What this profile tells us is the following : "Given all the baskets in the real market basket dataset, and the top four recipes associated to each basket, the most common recipe in the first position of all rankings is the recipe "butter, yeast, purpos flour, salt, sugar, beer", and its average cosine similarity is 0.61. The most common recipe in the second position of all rankings is "milk, purpos flour, cook oil, salt, larg egg, beer" and its average cosine is 0.61; and so on."

The second customer profile has the following recipes :

(1) water, coconut, sugar, salt, fruit, rice flour (Avg.Score: 0.62).
(2) vanilla, oil, whole milk, cream chees, roll, sour cream, whip cream, condens milk (Avg.Score: 0.51).
(3) milk, purpos flour, cook oil, salt, larg egg, beer (Avg.Score: 0.52).
(4) marmit, onion, chees, bun, toast (Avg.Score: 0.49).

What this profile tells us is the following : "Given all the baskets in the real market basket dataset, and the top four recipes associated to each basket, the second most common recipe in the first position of all rankings is the recipe "water, coconut, sugar, salt, fruit, rice flour", and its average cosine similarity is 0.62. The second most common recipe in the second position of all rankings is " vanilla, oil, whole milk, cream chees, roll, sour cream, whip cream, condens milk" and its average cosine is 0.51; and so on."

The detailed report is in the file *finalReport.txt*.

Input Basket Dataset	*Input Recipe Dataset*	*Execution Time*
Real	Real	7,3 Minutes
Real	50MB	30,5 Minutes
Real	100MB	59,8 Minutes
5MB	Real	16,5 Minutes

Table 1: Execution times. Each row represent the inputs we pass to the algorithm and the execution time. Real indicates one of the two real dataset provided for the project, 5MB indicates marketData5MB.json, 50MB indicates recipeData50MB.json, and 100MB indicates recipeData100MB.json.

Synthetic Data

We worked on relatively small datasets so far. In the real market basket dataset we have 9835 baskets, and in the real recipe dataset we have 39774 recipes.

Now we would like to see how well our algorithm scales when applied on larger data; how much time our implementation of the algorithm takes to be processed.

We created recipeData50MB.json, and recipeData100MB.json. These two files imitate the schema of the real recipe dataset, and the recipes and their cuisine in the datasets were randomly generated.

In recipeData50MB.json, we have 250000 recipes.

In recipeData100MB.json, we have 500000 recipes.

Then, we created the file dataMarket5MB.json, which imitates the schema of the real market basket dataset. The baskets in the datasets were randomly generated.

In dataMarket5MB.json, we have 25000 baskets.

As you may guess, the names of the files indicate the types of dataset and their approximated sizes.

The test will be done as the following: we will apply the algorithm with as inputs the real market basket dataset and real recipe dataset, and we will time the execution of our solution, from the first to the fourth step.

Then, we will time the execution of our implementation with as inputs the real market basket dataset and the dataset defined in recipeData50MB.json. And so on.

The results are in table 1.

Input Basket Dataset	*Input Recipe Dataset*	*Execution Time*
Real	Real	4,2 Minutes
Real	50MB	14,0 Minutes
Real	100MB	25,2 Minutes
5MB	Real	6 Minutes

Table 2: Execution times using the second approach. Each row represent the inputs we pass to the algorithm and the execution time. Real indicates one of the two real dataset provided for the project, 5MB indicates marketData5MB.json, 50MB indicates recipeData50MB.json, and 100MB indicates recipeData100MB.json.

6 COMPARISON WITH A BASELINE ALGORITHM OR SOLUTION OF YOUR CHOICE

Now, a different approach to the main task is the following: we go through the process of data cleaning and market basket analysis as always, but, in the third step, we find documents that are similar using Jaccard Similarity, Shingling, MinHash, and Locality-Sensitive Hashing; this approach is described in the subsection 3.4.3 of the coursebook Mining of Massive Datasets. The fourth step remain untouched.

The python code is in the file *LSHSolution.ipynb*.

During the execution of this approach, we do not take into consideration the cuisines anymore; the entire recipe dataset is one single giant corpus. Also, note that, because of the data cleaning process, we decided, in the shingling part, to convert the documents in unigram shingles sets. We will not go any more in-depth in the description of this approach, as it is not part of the task of this section.

By using the real market basket dataset and the real recipe dataset as inputs, let us see how is the first customer profile:

(1) whole milk, lime (Avg.Score: 0.45).
(2) lemon, salt, whole milk (Avg.Score: 0.39).
(3) guin beer, ice cream, irish whiskey (Avg.Score: 0.13).
(4) mexican beer, gin, fresh lemon juic (Avg.Score: 0.12).

The detailed report is in the file *finalReportMinHashLSH.txt*.

As expected, the profiles generated by the two approaches are very different.

The main reason is that the notion of "similarity" in the second approach is lexical; two or more documents are similar if they share one or more words, and we are not taking into consideration also the context the words are used in. By doing so, we are wasting valuable data regarding the cuisines and not considering the importance of certain ingredients.

But on another hand, if we time the executions, represented in table 2, as we previously did, we note that he execution times are lower than the ones of the previous approach.

This is mainly thanks to Locality-Sensitive Hashing, which allows us to avoid computing the similarity of every pair of minhash signatures; meaning that we will not take into consideration most of the pairs that do not meet our threshold of similarity. Thus, LSH improves the performance of our solution significantly.

In summary, we have presented two ways of computing document similarity: on the one hand, we use TF.IDF and cosine similarity; on the other hand, we use Jaccard Similarity, Shingling, MinHashing, and Locality-Sensitive Hashing.

We would go with the first way if we are dealing with relatively small datasets; it tries to understand the importance of the ingredients used in different cuisines. However, the second way is more suitable for massive large-scale datasets.

7 CONCLUSION

For the project, we applied various notions and algorithms studied during the course.

We normalized, corrected, and stemmed the data before applying any technique.

Then, we analyzed the market basket dataset in order to find typical behaviors of the customers; in such patterns, we tried to discover useful information.

After that, we created a very basic search engine; given the products in a basket, we returned the top four recipes in the recipe dataset, using the cosine similarity. We used the results for creating customer profiles.

We observed how our algorithm performs with different size of datasets, and we then confronted the results with a new approach, using a different notion of "similarity".

We finish the report by stating that the algorithm can be improved in many aspects, from the first step to the fourth step. However, such improvements take a significant amount of time that we could not afford.