Data Mining Project 2019/2020

Final Report

David Ernesto Ayala Russi
Mat: 212622
david.ayalarussi@studenti.unitn.it
University of Trento
Data Science, 1st year
Industry Intention

ABSTRACT

This document contains the final report of the project of Data Mining for the academic year 2019/2020. This project is based on two unrelated datasets from Kaggle competences. The first dataset is a JSON file that contains a list of recipes with their corresponding ingredients and cuisine types[1]. On the other hand, the second dataset contains grocery market basket data in CSV format, this is essentially a set of baskets with their corresponding list of items[2]. The goal of this project is to group the market customers based on not only on what they are buying but what they are cooking based on the recipes dataset. To achieve this goal, recipes and baskets are treated as documents with a set of words. So, clusters of recipes have been created using k-means with previous processing using TF-IDF(Term Frequency-Inverse Document Frequency). On the other side, after removing all the items that do not correspond to any recipe, TF-IDF preprocessing was applied to the market basket dataset using the same vocabulary used for recipes. Finally, each basket was associated with the closest centroid of recipe clusters. In that way, the result is a classification of customers based on what they are buying related to the recipes dataset. At the end each cluster is labeled with a human-understandable text to interpret the results

KEYWORDS

Data Mining, Clustering, K-means, TF-IDF, Market Basket Dataset, Python, Sci-kit learn

[1] Recipe Ingredients Dataset: https://www.kaggle.com/kaggle/recipe-ingredients-dataset

[2] Groceries Dataset: https://www.kaggle.com/irfanasrullah/groceries

1 Introduction

This problem is built on top of two completely different and unrelated datasets. The first one is related to the recipes in which each element corresponds to a dish that contains a list of ingredients and one type of cuisine, such as indian, mexican, italian, etc. This dataset was originally constructed to create classifiers which target is type of cuisine. Therefore, it is composed by JSON formated training and testing sets. The training set contains 39774 recipes with 6714 unique ingredients and the test set comprises 9944 recipes and 4484 unique ingredients. On the other hand, the second dataset is a market basket dataset. in this case each basket represent a customer's purchase, that can be expressed as a list of items. Since both datasets are unrelated, items can be so diverse that could not be part of any recipe. This dataset is expressed in CSV format and contains a total of 9835 baskets with 169 unique items.

Concretely, the main problem of this project is to reveal customer clusters based not only on what they are buying but what are they preparing taking into account the recipes dataset. The challenge of this problem is that there is a many-to-many relationship between ingredients and recipes, which means that on one side, one ingredient can be used for a wide range of recipes and different types of cuisine, and a recipe contains many ingredients on the other side. In addition, a customer can buy not

necessarily all the ingredients of a recipe, but only a few ingredients of it, that makes more difficult the identification of which dish the user bought the ingredients for.

It is also important that each cluster makes sense to a human reading the results of the solution. That is why in the end, each customer group is labeled with a human-understandable text.

2 Related Work

The solution to this problem is mainly based on the idea of TF-IDF(Term Frequency-Inverse Document Frequency) and clustering algorithms. Both techniques will be described in this section.

2.1 TF-IDF

TF-IDF stands for **Term Frequency-Inverted Document Frequency** is not a data mining algorithm but it is an important concept that helps to measure the importance of a word in a collection of documents(corpus). In this project, recipes and market baskets will be handled as a document or a sequence of words. That is why this concept is applicable and useful. This measure increases when the number of appearances of a word also increases. So, that means that it is possible to find the topic of a document using this measure. However, in real documents the most common words will be prepositions or articles or connectors that in the end are just part of the language but do no define the topic of the document. These words are called stop words and are usually eliminated. On the other hand, there are also extremely infrequent words that do not tell anything about the topic of the document, therefore they will have a low TF-IDF score and are removed as well.

In practice, TF-IDF is computed in two parts. The left side is called *term frequency* and measures how often a word occurs in a single document. That is the frequency of a word in the document divided by the frequency of the most common word:

$$TF = \frac{frequency}{maximun\,frequency} \quad (1)$$

On the right side, inverse document frequency is the measure of how often a word appears in an entire set of documents. This is the total number of documents *N* divided by the number of documents in which the word appears *n*.

$$IDF = \log\left(\frac{N}{n}\right) \quad (2)$$

Finally, the TF-IDF score is computed multiplying the two quantities computed previously

$$TF - IDF = TF * IDF \quad (3)$$

These TF-IDF scores can be represented in a matrix which rows correspond to document and columns correspond to each word in the vocabulary.

2.2 Clustering

Clustering is a data mining technique that allows to group points according to a distance measure. In other words, similar data points are expected to have a shorter distance and therefore, be classified in the same cluster or group. This is a challenging problem because usually these data points belong to a high dimensional data space. There are two different strategies to address this problem *hierarchical approach* and *point assignment*.

2.2.1 Hierarchical algorithms. This type of clustering algorithms follow two approaches. Firstly, the *agglomerative clustering* or *botton up* approach, in which each point is a separate cluster and the goal is merging close clusters repetitively until a stop point or a define threshold. Conversely, the *divisive* or *top down* approach considers all points in the same cluster and the goal is recursively split it in separate clusters. These approaches have two main advantages, the number of clusters are discovered by the algorithm and are useful for every type of space and distance measure such as Euclidean, Jaccard or Cosine similarities, but the con is that it is expensive for large datasets because all the points need to be stored in memory.

2.2.2 Point assignment. The core idea of this type of clustering algorithms is that each cluster is

represented by a point(usually called centroid or clustroid). Each data point is iteratively assigned to the closest representative of the cluster. This leads in an update of the representative at the end of every iteration. A good example of this approach is the K-means algorithm.

2.2.2.1 K-means

This is a cluster algorithm based on the point assignment approach. It makes two important assumptions that make it different from the hierarchical algorithms. Firstly, It assumes an euclidean space, which means that the distance measure is euclidean distance. The second assumption is that the number of clusters has to be fixed as a parameter of the algorithm. However,, there are techniques to explore this number of clusters. For instance, one approach is performing the k-means algorithm many times and check a measure of cluster quality such as average radius or diameter. The optimal number of clusters will be this value of k where there is no much change in the measure of quality. There are also some considerations choosing the initial points. For example, points located as far as possible from each other are good candidates to be settled as initial centroids because most likely they belong to different they belong to different groups. The advantage of this algorithm is that it has variations that make it useful for large datasets that does not fit into memory. The k-means algorithm is described in the Fig. 1

```
Initially choose k points that are likely to be in
    different clusters;
Make these points the centroids of their clusters;
FOR each remaining point p DO
    find the centroid to which p is closest;
    Add p to the cluster of that centroid;
    Adjust the centroid of that cluster to account for p;
END;
```

Figure 1: K-means algorithm, taken from Mining of Massive Datasets, p 255

3 Problem Statement

In this section a formal definition of the problem will be introduced creating also a terminology useful to explain the solution. First, define the set of baskets to be $B = \{b_1, b_2, \dots, b_n\}$ being n the total number of baskets in the dataset. Simultaneously, each basket bj contains a list of items $\{e_{1j}, e_{2j}, \dots, e_{i,j}\}$ that will be denoted by $e_{i,j}$ which means the element i in the basket j. s_j will denote the size of the basket j The many-to-many relationship can noticed here, since each $e_{i,j}$ can be also present in a basket b_x where $x \neq j$.

On the other hand, there is the set of recipes that will be defined by $R = \{r_1, r_2, \dots, r_m\}$where m is the total number of recipes in the dataset. Likewise, the recipes are also a set of items, they will be called ingredients here, but the notation will be the same because in the end the goal is to find a match between ingredients from recipes and items from baskets. So, each $r_j = \{e_{1j}, e_{2j}, \dots, e_{i,j}\}$. It can be evidenced a many-to-many relationship in the recipes case as well because one ingredient can be used in more than one recipe.

The **input** of the algorithm will be these two sets $B = \{b_1, b_2, \dots, b_n\}$ and $R = \{r_1, r_2, \dots, r_m\}$ that will share elements $e_{i,j.}$ Since the goal of the project is finding groups of customers to characterize them based on what they are preparing, k will be defined as the number of groups in which the customers or baskets will be labeled. The **output** of the algorithm is a list denoted by L which size is n and contains the corresponding cluster label assigned to each basket. Formally it will be $L = \{l_1, l_2, \dots, l_n\}$

4 Solution

To solve the problem described above, this solution will handle baskets B and recipes R as sets of documents, which means that b_j and r_j are composed by a sequence of ingredients or items $e_{i,j}$ that will be interpreted as words. The next sections will describe step by step the processing of each set.

4.1 Recipes Processing

The goal of the recipes processing is to create **clusters of recipes**. The labels discovered here will serve later to classify the customers into different groups. One option to identify clusters of recipes could be using the *cuisine* field provided in the dataset. However, this solution uses this field just to validate that the discovered clusters make sense,

but it is not a parameter to group recipes mainly for two reasons 1) There are around 20 different cuisine types in the dataset. It will showed later that given the sparsity of the ingredients in the recipes, the baskets does not contain representative ingredients from all cuisine types, which means that somehow a cuisine aggregation is needed 2) Since there is a test set, the idea is to use it in order see how test solution performs with more data, this test set does not contain the *cuisine* field.

The function to create clusters of recipes is described in this pseudocode:

```
1. function recipes_clusters()
2.     data = load recipes dataset;
3.     drop columns 'id';
4.     unique_ingredients = list of unique
ingredients in data;
5.     replace ingredients by their indexes in
unique_ingredients;
6.     tf_idf, vectorizer = create TF-IDF matrix;
7.     k-means(data=tf_idf, k=4)
8.     return labels, centroids, vectorizer
```

This function loads in a tabular way the data which is originally in JSON format. The column *id* is removed because it is not used, so the ingredients column contains the list of ingredients for each recipe. Then it creates a list of the unique ingredients in all the recipes. It will be useful to save memory handling indexes instead of entire words. The next step is to replace the ingredients in text format by their corresponding index in the list. Next, the TF-IDF matrix is created, where columns are ingredients and rows are recipes. Finally, since there is a numerical matrix, it is possible to run a cluster algorithm such as k-means with $k = 3$, this is just because the cluster tuning showed that three is a good choice. The function returns *label* which is a list of size *n* that contains the corresponding group for each recipe, *centroids* of each cluster and *vectorizer* which is an object with the vocabulary and other useful information obtained from the TF-IDF matrix creation.

4.2 Baskets Processing

The main goal of the basket processing is to take the information returned by he *recipes_cluster* function and **assign each basket to a cluster.** This will lead with a characterization of customers based on what they are preparing according to the recipes dataset. The pseudocode of this processing is described below

```
1. function process_baskets(centroids, vectorizer)
2.     data = load baskets dataset;
3.     unique_items = list of unique items
4.     discarded = unique_items that are not in
recipes data
5.     filtered_baskets = remove discarded items
from baskets;
6.     corpus = replace items by their indexes;
7.     tf_idf = vectorizer.transform(corpus)
8.     for basket in tf_idf
9.           find closest centroid
10.          assign the basket to that cluster
11.    return labels
```

This function starts reading the dataset, in this case the data is presented in CSV format where each row corresponds to a basket and its items are separated by comma. The first goal in this function is to remove the items that do not belong to any recipe, since items like 'newspaper', 'soap', 'shopping bags', 'napkins', etc. are not useful to characterize customers based on what they prepare. To do this a list of unique items is created and then compared to the unique ingredients list created previously, so it is possible to identify the discarded items. The next step is to filter the baskets removing all the items present in the discarded list. After that, the basket corpus needs to be constructed replacing the items text by the index of *unique ingredients,* Then, the same vocabulary used to compute the TF-IDF matrix for the recipes is used to compute the matrix for baskets. As a result, the number of columns (words) is the same in both sets baskets and recipes. So it is possible to compare euclidean distances. This is exactly the next step, for each row the closest centroid is computed and the basket is assigned to that cluster. There is no update of centroids in this step because it is just an mapping.

4.3 Implementation

This solution has been implemented under the Anaconda environment. Using specifically **Python 3.7.4** and the library **scikit-learn 0.21.3** which is a very well known API for machine learning, data mining and predictive analysis. This library is built on top of other popular python libraries such as numpy and scipy. *Scikit-learn* has implemented

functions for the most important parts of this solution such as:

TfidVectorizer is the implementation for TF-IDF. It contains methods to compute the TF-IDF matrix and learn vocabulary from corpus. One important trick used here is that the *TfidVectorizer* allows cut off, which means that words under a certain score can be discarded because they do not contribute to characterize the corpus. Similarly, the cut off can be applied to the upper boundary, that means removing corpus stop words. This is very important because in terms of the project, it allows to reduce the number of ingredients/items and makes the clusters computation less expensive.

Kmeans is the implementation of K-means algorithm, using it is straightforward, it is enough passing the data, in this case TF-IDF matrix and the number of clusters. The output of this function is a list of labels corresponding to each row of the data.

Euclidean_distances is the implementation of the euclidean distance computation, it receives two arrays and returns the distance between them. It is useful to compute the distance between the processed TF-IDF matrix for baskets and the centroids computed from recipes clusters.

It also contains the *TruncatedSVD* function used just to reduce the dimensionality for visualization purposes.

4.3 Label Definition

Scikit-learn implementation of the K-means algorithm assigns arbitrary numeric labels. However a proper name for this labels is needed in order to understand the results. In this section the recipes clusters will be depicted and the label names will be defined. The number of clusters were designated to be three. In addition, the type of cuisine is used here to validate that the clusters make sense. The label *0* contains the cuisines: *chinese, filipino, japanese, korean, thai and vietnamese.* The name assigned to this category is *Asian cuisine.* The label 1 contains *british, irish and southern_us* that from now on will be labeled as *English cuisine*. Lastly, the label 2 contains the rest of cuisines, *brazilian, cajun creole, french, greek, indian, italian, jamaican, mexican, moroccan, russian and spanish.* This group will be called *Western cuisine.* The Figure 2 shows the distribution of the clusters in the space

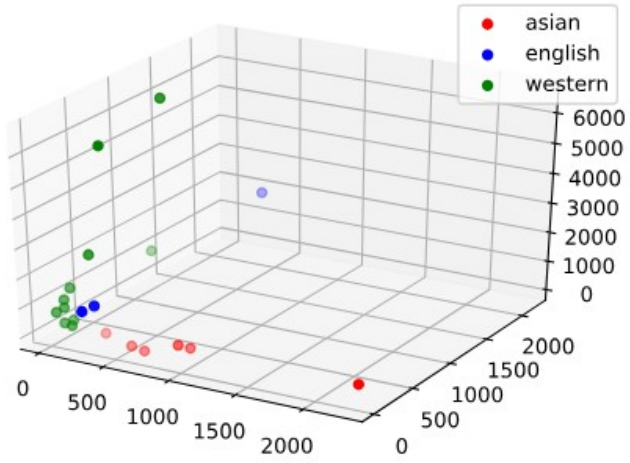

Figure 2. Recipes cluster by cuisine type

4.4 Customer Clustering

As mentioned above, the goal is customer clustering. The recipes cluster are already set and will be used to classify customers in the same categories. Figure 3 shows the customers clustering result. This is the result of assigning the processed basket to the closest centroid derived from the recipes clustering.

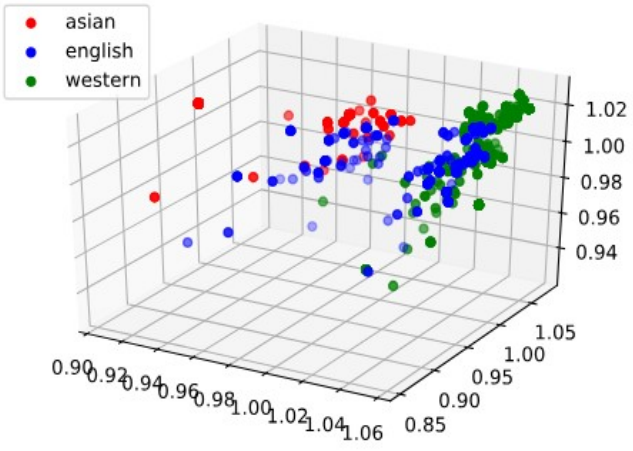

Figure 3. Shows the customers clustering

5 Empirical Evaluation

This section shows first the complexity of the solution and the performance with real data and synthetic data. Finally, it will expose a comparison between this solution and a baseline algorithm such as just clustering customers based on what they buy.

5.1 Complexity

This solution is divided in two parts. The recipes processing and the basket processing. Runtime complexity of TF-IDF is *O(nL log nL)* where n is the total number of documents and L is the average length of documents. On the other hand the runtime complexity of K-means algorithm is $O(n^2)$. However since tasks like removing discarded items and replacing ingredient names by their indexes also have a complexity of $O(n^2)$, the overall complexity can be approximated to $O(n^3)$. This means that the solution will not perform efficiently when the dataset becomes large. However it can be optimized running tasks like TF-IDF and K-means under a MapReduce approach. But as it is, the algorithm will not perform well when the datasets does not fit in memory.

5.2 Real Data

The original datasets are composed by 39774 recipes that contain 6714 unique ingredients. On average, the recipes contain 10.76 ingredients. Meanwhile, the total number of baskets is 9835 and the number of unique ingredients is 164. As can be seen, the difference in items proportion is large, which means that few items are part of the recipes. In fact, only 13 items could be found as ingredients in recipes. However, this deficiency can be improved by not only looking at textual comparison but defining a threshold of similarity between items and ingredients. That means finding a way such that items like 'whipped/sour cream' can be understood as the ingredient 'sour cream'. In this work textual comparison is performed. In terms of time, it takes around 30 seconds to process this amount of information.

As mentioned before, the *TfidVectorizer* allows to set up *cut off thresholds*, this is what takes more time, but it is very important because it helps to reduce the number of ingredients in the vocabulary. In other words it reduces the dimensionality of the matrix representation used in the clustering algorithm. For the real data, the minimum document score was set to 0.01 and the maximum was 0.6, which ends up with a final vocabulary of 194 unique ingredients. Although it is still a large number of dimensions compared to the original 6714 ingredients, it can be handled later by K-means.

5.3 Synthetic Data

In the code of this solution it is also included an utility to generate synthetic baskets. It is a simple script that picks items randomly and creates baskets of normally distributed sizes. The intention of this script Is increasing the number of baskets to test the performance of the algorithm. The Figure 4 shows the performance in time increasing the number of baskets

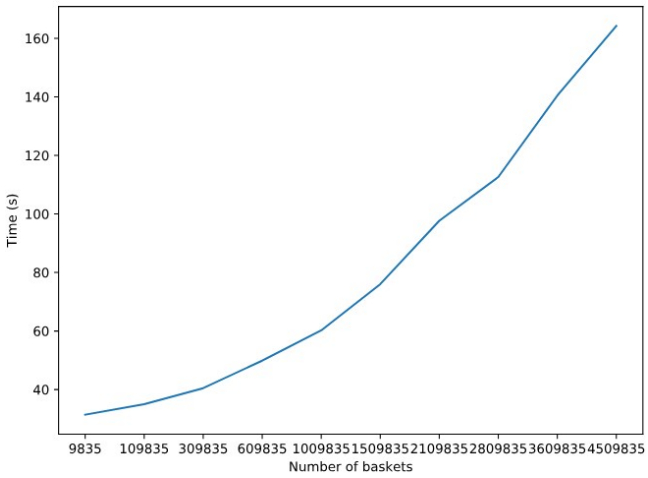

Figure 4. Performance in seconds vs number of baskets

As mentioned before, the test set of the recipes dataset will be used to see the performance of the algorithm with more recipes. The Figure 5 shows this performance increasing by around 1000 recipes per iteration.

In both graphs can be seen that this solution does not perform efficiently when the data. However is a solution that can be implemented in other environments that allows the MapReduce approach such as Spark. In terms of memory, It will fail when

the data is large enough to fit in it. To overcome this problem, other clustering algorithms should be considered, such as CURE or DBSCAN

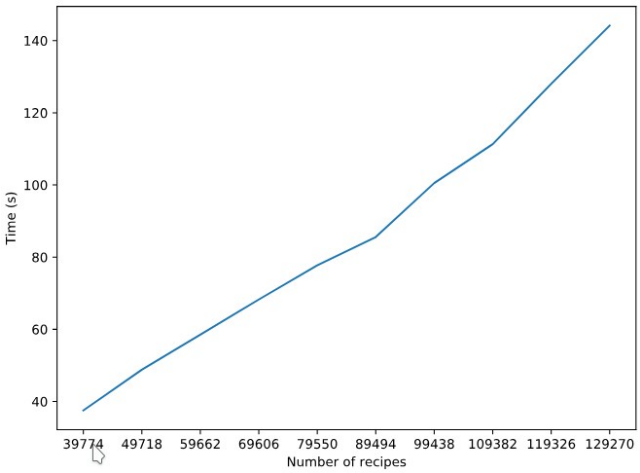

Figure 5. Performance of the algorithm increasing the number of recipes

5.4 Baseline Comparison

The most important part of this solution is the clustering of recipes. So the one provided in this solution will be compared against a cluster of recipes keeping all the ingredients. The original list of unique ingredients contains 6714 that means that building the TF-IDF matrix it will have 6714 rows or dimensions. On the other hand, the total number of recipes is 39774, which means that the final dimension of the matrix will be 39774 by 6714. The TF-iDF scores will be stored as float, so it means 39774 * 6714 * 64 bits of memory that corresponds to 2.1GB of main memory. The cut off applied in this solution ended up with 194 ingredients, which then leads in 0.06GB of memory. The memory consumption is significantly less to process the same data.

6 **Conclusion**

In this work was presented a solution useful to cluster transactions presented in a market basket dataset based on the food they cointans related to a dataset of recipes. This solution presents a way to cluster the recipes and the apply this groups to the baskets. This work handled recipes and baskets as documents or sequences of words, such that techniques like TF_IDF to find importance of words in a corpus can be applied. The main clustering algorithm used in this solution is K-means which is an algorithm that assumes euclidean space and a fixed number of clusters. The customers were classified in three groups: Asian Cuisine, English Cuisine and Western Cuisine.

This algorithm was implemented in scikit-learn a python library useful for data mining, machine learning and data analysis. However, it does not implement any kind of distributed processing. As future work this solution can be implemented in technologies like Spark or Hadoop to use all the power provided by the MapReduce approach

REFERENCES

[1] Jure Leskovec, Anand Rajaraman, JeffreyD. Ullman. 2014. *Minning of Massive Datasets* .

[2] Malay K Pakhira. 2014. *A linear* A Linear Time-Complexity k-Means Algorithm Using Cluster Shifting. IEEE Xplore

[3] Yingnan Cong, Yao-ban Chan, and Mark A. Ragana,. 2016. A novel alignment-free method for detection of lateral genetic transfer based on TF-IDF. PMC

[4] Scikit-learn official documentation