# Data Mining project

PIOTR BRÓDKA, Erasmus student, student number: 210246, 2nd year, Industry intention
KAROL KIEREK, Erasmus student, student number: 213538, 2nd year, Industry intention

In a typical supermarket, customers buy different kinds of ingredients for the recipes they are planning to prepare at home. The aim of this project is to give some description of customers based on the food they eat. We work with two types of datasets. First one comprises purchases of products by clients. The other one has recipes, each recipe a has list of ingredients and a label about category of the recipe. Algorithms to handle this problem are proposed in this paper. Algorithm based on the Singular-Value Decomposition has been proved to work better than simple baseline method. In the paper, we describe use of both real and artificial (created by authors) datasets.

CCS Concepts: • **Information systems → Collaborative filtering**; **Data mining**.

Additional Key Words and Phrases: datasets, data mining, recommendation systems, singular-value decomposition, market basket analysis

**ACM Reference Format:**
Piotr Bródka and Karol Kierek. 2020. Data Mining project. *ACM Trans. Graph.* 1, 1 (February 2020), 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 INTRODUCTION

Currently, supermarkets collect data about purchases that their customers make. Analysis of this data is a big opportunity for targeted marketing. If we managed to find out customers culinary preferences, we could make quite a big advantage over competition. For example, if we found out, that some customer is particularly interested in Italian cuisine, we could make him/her special sales on certain products (of course with intention to earn some extra money on this move) or to offer them Italian cooking classes. In this paper, an attempt for such analysis for real and artificially created data will be shown.

## 2 RELATED WORKS AND TOPICS

### 2.1 Papers describing similar problems

This topic is relatively unexplored and as the best of our knowledge, there are no papers, that would address the problem accurately. In theory of Data Mining there are topics, that are connected to the topic of this paper and will be used in the proposed solution. Those are the **Singular-value decomposition** and **frequent itemsets** which will be described in the following subsection.

Authors' addresses: Piotr Bródka, piotrtomasz.brodka@studenti.unitn.it, Erasmus student, student number: 210246, 2nd year, Industry intention; Karol Kierek, karol.kierek@studenti.unitn.it, Erasmus student, student number: 213538, 2nd year, Industry intention.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2018 Association for Computing Machinery.
0730-0301/2020/2-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

### 2.2 Algorithms

*2.2.1 Singular-value decomposition.* From [Leskovec et al. 2014]. Let $M$ be an $m \times n$ matrix, and let the rank of $M$ be $r$. Recall that the rank of a matrix is the largest number of rows (or equivalently columns) we can choose for which no nonzero linear combination of the rows is the all-zero vector 0 (we say a set of such rows or columns is independent). Then we can find matrices $U$, $\Sigma$, and $V$ as with the following properties:

- $U$ is an $m \times r$ column-orthonormal matrix ; that is, each of its columns is a unit vector and the dot product of any two columns is 0.
- $V$ is an $n \times r$ column-orthonormal matrix. Note that we always use $V$ in its transposed form, so it is the rows of $V^T$ that are orthonormal.
- $\Sigma$ is a diagonal matrix; that is, all elements not on the main diagonal are 0. The elements of $\Sigma$ are called the singular values of $M$.

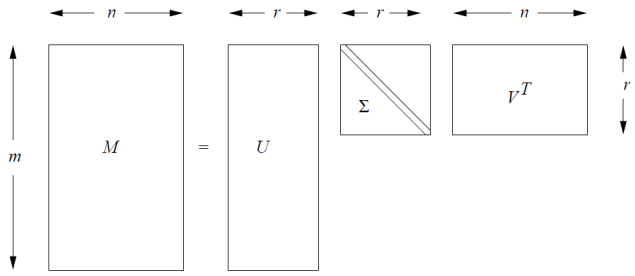


Fig. 1. The form of a singular-value decomposition

The key to understanding what SVD offers is in viewing the $r$ columns of $U$, $\Sigma$, and $V$ as representing concepts that are hidden in the original matrix $M$. Suppose we want to represent a very large matrix $M$ by its SVD components $U$, $\Sigma$, and $V$, but these matrices are also too large to store conveniently. The best way to reduce the dimensionality of the three matrices is to set the smallest of the singular values to zero. If we set the $s$ smallest singular values to 0, then we can also eliminate the corresponding $s$ columns of $U$ and $V$. How many singular values should we retain? A useful rule of thumb is to retain enough singular values to make up $80-90\%$ of the energy in $\Sigma$. That is, the sum of the squares of the retained singular values should be at least $80-90\%$ of the sum of the squares of all the singular values.

*2.2.2 Frequent Itemsets and the A-priori algorithm.* From [Leskovec et al. 2014]. Suppose we have a set of baskets and each basket is a set of items. We want to find **frequent itemsets** - i.e. such subsets of a set of items which appear in a set of baskets frequently. A threshold

value of a number of occurences in baskets for an **itemset** which it can be recognized as **frequent** is called **support**.

**A-priori** is an algorithm for finding these **frequent itemsets**. To understand it, some basic concept, called **monothonicity** is needed:

- if a set of items is frequent, then also all its subsets are frequent, and **contraposition**:
- if some itemset is not frequent, then for sure none of its superset will be.

Firstly, we are counting occurences of all items and we decide which of them are frequent. Because of **monothonicity**, a frequent pair cannot be composed from both frequent items. We are checking all candidates (composed of two frequent items) pairs for their frequency - there can be a pair that is not frequent even though it is composed of two frequent items. We can use this algorithm to create triples, quartets... To find the frequent itemsets of size $k$, we are creating them by restricting to their subsets of size $k-1$, that have been found frequent in the previous iteration. There are more efficient modifications of this algorithm. More information about them and examples can be found in [Leskovec et al. 2014].

## 3 PROBLEM STATEMENT

We have two datasets. One is composed of purchases of customers. For each customer we know set of products bought by them. We also have the other dataset with recipes. For each recipe we know a set of products needed for a recipe. All recipes have also a label which describes the type of cuisine that the dish belongs to. There are $c$ categories. For each customer we have $c$-element vector, where each element $c_i$ indicates the similarity to $i$-th label ($i$-th type of cuisine). Optionally (what will be done in this paper on the testing phase), for results interpretability, we will return:

- Category, that is the closest for an customer,
- Top 3 categories, that are the closest for an customer.

## 4 DATASETS

For the presented task we need two kinds of data:

(1) products bought by customers
(2) recipes - list of sets of products. A recipe needs to have some category eg. Italian / Indian / ... - this category will be used to describe people, that prepare this recipe (and recipes from this category)

### 4.1 Real datasets

We will use 2 datasets with products purchases and 1 dataset with recipes:

(1) Kaggle dataset: [Kag [n. d.]]. Example of an order:
    - citrus fruit,
    - semi-finished bread,
    - margarine,
    - ready soups
(2) Instacart is an American delivery company. They published part of their basket dataset for a non-commercial use: [Ins [n. d.]].
    They also organized a contest on Kaggle on the task to *"predict which previously purchased products will be in a user's next order"*. There are ca. 3400000 orders made by 206209 clients. Each order is a set of products. For example order with id: 1 consists of products of ids: 49302, 11109, 10246, 49683, 43633, 13176, 47209, 22035, so:
    - Bulgarian Yogurt,
    - Organic 4% Milk Fat Whole Milk Cottage Cheese,
    - Organic Celery Hearts,
    - Cucumber Kirby,
    - Lightly Smoked Sardines in Olive Oil,
    - Bag of Organic Bananas,
    - Organic Hass Avocado,
    - Organic Whole String Cheese
(3) Recipe-Ingredient database: https://www.kaggle.com/kaggle/recipe-ingredients-dataset. Sample recipe:

```
"cuisine": "indian",
"ingredients": [
  "tumeric",
  "vegetable stock",
  "tomatoes",
  "garam masala",
  "naan",
  "red lentils",
  "red chili peppers",
  "onions",
  "spinach",
  "sweet potatoes"
```

### 4.2 Artificial (generated) dataset

On the real data, we can evaluate the performance of an algorithm by manually looking at the things bought by a customer and evaluate, if the label given for customer is correct (so we need expert supervision).

For a quantitive evaluation, we created artificial datasets, which are composed of:

- for each customer, the products that he/she bought,
- set of recipes for cuisines, recipes are lists of products
- for each customer, his/her dominant (most preferred) cuisine. It is a ground truth, that we don't have its in real dataset and according to which our algorithm will be evaluated.

Assumptions, that were used for creating the data:

- There are **n_products** products, **n_cuisines** cuisines and **n_customers** customers,
- Each product has some overall frequency of buying, there are products globally infrequent and globally frequent. Some relative measure of frequency is used, e.g. frequent items are those with measure > 5,
- Each cuisine has from 5 to 15 frequent products used in recipes,
- Frequency of products in cuisines will be adjusted in such way, that frequent products from a cuisine will have 20% of overall frequency for all products in cuisine. Frequencies of

those frequent products are taken from the uniform distribution. Additionally, one product gets bonus of frequency, by multiplying by 2, and some other one - by multiplying by 1.5,
- Each cuisine will have some number of recipes, from 10 to 50, this number is sampled from uniform distribution,
- Each recipe will have from 5 to 15 ingredients, this number is sampled from the uniform distribution,
- Products for each recipe are chosen randomly, according to their frequency.

Datasets of different levels of complication were created. There are 7 datasets, that present increasing complexity of the data:

- **3 recipes from preferred cuisine**: Each customer will buy ingredients to prepare 3 recipes from his dominant cuisine,
- **3 recipes from preferred cuisine not full**: As above, but customers will not buy all products for recipes but only 70% of products for each recipe,
- **2 from preferred cuisine 1 from other**: Each customer will buy ingredients to prepare 2 recipes from his dominant cuisine and 1 from other cuisine,
- **3 from preferred cuisine 2 from other**: Each customer will buy ingredients to prepare 3 recipes from his dominant cuisine and 2 from other cuisine,
- **2 from preferred cuisine 1 from other not full recipes**: As in **2 from preferred cuisine 1 from other**, but customer buys only 70% of products for each recipe,
- **3 from preferred cuisine 2 from other not full recipes**: As in **3 from preferred cuisine 2 from other**, but customer buys only 70% of products for each recipe,
- **3 from preferred cuisine 4 from others not full recipes**: Each customer will buy ingredients to prepare 3 recipes from his dominant cuisine. Also, customer buys products to prepare recipes from 3 other cuisines and 2 from each cuisine. Customer does not buy all the ingredients. For preferred cuisine and one not-preferred cuisine customer buys 70% of products and for other 2 not-preferred cuisine customer buys 50% of products.

## 5 PROPOSED SOLUTION FOR THE PROBLEM

In this section, we will describe methods, that we used in the project.

### 5.1 Simplified Jaccard Similarity

When we have a set, representing the customer: $C$ and recipe: $R$, classical Jaccard Similarity is counted:

$$\frac{|C \cap R|}{|C \cup R|}$$

In our solutions we use a modification, that will be called the *Simplified Jaccard Similarity* in the following part of the report. It is counted:

$$\frac{|C \cap R|}{|R|}$$

### 5.2 Baseline algorithm

We created a baseline algorithm - a simple method, that we will be referring to, when evaluating results of the created algorithm. We present the method in form of pseudocode:

We have **n_customers** customers, we have **n_recipes** recipes, each with a label.

(1) For each customer i and recipe j in position we compute Simplified Jaccard Similarity between sets of words from products of customer i and sets of words from products that appear in recipe j
(2) For each customer we compute average Simplified Jaccard Similarity per cuisine. In that way, for each customer we get vector of similarities to cuisines.
(3) Optionally, we can return t most matching cuisines (with the highest Similarity).

### 5.3 Idea 1 - Frequent Itemsets (turned out, that it does not work well)

(1) For each cuisine we are searching for frequent ones, doubles, triples, quartets. Baskets are recipes and items in baskets are products needed in recipes. For each cuisine we filter frequent ones, doubles, triples, quartets in a such way, that we delete frequent itemsets, that appeared also in other cuisines - so we are characterizing cuisine by frequent itemsets, that are not frequent in any other cuisine. In our solution itemset is frequent if its support is equal or greater than 20%.
(2) For each customer, we are searching for similarity for cuisines. Among product, that he bought, for each cuisine we are searching for frequent itemsets and we are counting similarity of customer and cuisine as a sum of:
   - (number of ones) x 1
   - (number of doubles) x 2
   - (number of triples) x 4
   - (number of quarters) x 8
(3) We are searching for the cuisine with the biggest similarity. We are returning result if a maximal similarity is bigger than 15

### 5.4 Idea 2 - SVD (works well, it is our main algorithm)

Idea of the algorithm is based on the algorithms from the lectures about recommendation systems and dimensionality reduction (SVD).

We have **n_customers** customers, we have **n_recipes** recipes, each with a label.

(1) We create an utility matrix with dimensions (**n_customers**, **n_recipes**) For each customer i and recipe j in position i,j of matrix we count Simplified Jaccard Similarity between sets of words from products of customer i and sets of words from products that appear in recipe j
(2) We conduct the SVD decomposition. We take k strongest concepts. Customer is characterized by a vector of coefficients describing matching to concepts. A recipe is also characterized by a vector of coefficients describing matching to concepts. Thus each cuisine also can be characterized as such vector (mean over recipes).

(3) For each customer we can count a similarity for each cuisine (proximity of vector of customer to vector of cuisine)
(4) Optionally, we can return t most matching cuisines (with the shortest distance).

$k$ - number of singular values - is parameter of the algorithm

### 5.5 Cleaning of real data

Artificially created data is ready to use by the algorithm. Real data is more complex and needs preprocessing.

*5.5.1 Invalid categories of products.* In data about purchases we have information about the category that products belong to. To reduce the noise, we remove categories of products, that are not used in recipes, e.g.:

- cat food care
- mint gum
- vitamins supplements

*5.5.2 Complexity of real data.* Lets look at example names of products.

- Lightly Smoked Sardines in Olive Oil,
- Bag of Organic Bananas,
- Organic Hass Avocado,

This what really is interesting for us here, is that those are: sardines, bananas, avocado (or hass avocado - depending on the desired level of accuracy). For sure the fact that something is organic is a noise from our perspective (unless we want to find preferences towards organic food). That's why, data cleaning is needed. We describe, the steps that we used for cleaning:

(1) We take purchase data and we make list of all words in product names,
(2) We are filtering all those words:
    - We remove all words, that contain numbers,
    - We remove all words, that are composed of 1 or 2 characters,
    - We change form of words from plural to singular, so now from 2 words: "tomato" and "tomatoes" we get only "tomato" - and it is desired effect, because in recipes / names of products singular and plural form signify the same thing (in prevailing number of cases),
    - We change words to lowercase
    - We manually remove words, that (in expert opinion) will not have significance, e.g. "salt", "all", "purpose", "unsalted", "traditional", "natural", "whole"...
(3) We take recipe and we make list of all words in product names,
(4) We are filtering all those words as above.
(5) We take intersection of 2 sets of words.
(6) For each name of product (considering both shopping data and recipe data), we are filtering them as shown above and remove all words that are not appearing in intersection from point 5. If resulting name of string is an empty string (because all words, that this product was comprised of were deleted), we are deleting this product.

**Examples of cleaning of original names of products:**

- Green Chile Anytime Sauce -> green chile sauce
- Organic Turkey Burgers -> turkey burger
- White Pearl Onions -> white pearl onion
- Organic Spaghetti Style Pasta -> spaghetti pasta
- All-Seasons Salt -> *nothing*, because it got reduced to "salt" and after, this word was manually removed,

## 6 RESULTS

### 6.1 Quantitative results on artificial data

We conducted experiments on 7 created datasets. Algorithm returns proximity to all kitchens for every customer. As such output is not handy for interpretation, we are checking quality of algorithms by computing two metrics:

(1) **exact match**: Accuracy computed based of a number of examples, where predicted cuisine for customer is the same as in dataset.
(2) **approximate match**: Accuracy computed based on number of examples, where one of top 3 predicted cuisines for customer is the same as in dataset.

**Methodology of experiments:**

- **baseline**: we just run baseline algorithm on dataset, there are no variables to test.
- **A-priori method**: we just run baseline algorithm on dataset, there are no variables to test.
- **SVD method**: in this algorithm there is variable $k$ - number of concepts coefficients that are used to create profile of customer and cuisines. With change of this parameter, efficiency of the algorithm changes. In our solution we divided set of customers into 2 halves: *validation set* and *test set*. On *validation set* we are checking for which $k$ algorithm yields the best accuracy. And final accuracy, presented below is the accuracy for *test set*. Depending, on metric, that we want to maximize (**exact match**/**approximate match**), we are choosing $k$ maximizing this metric.

*6.1.1 Results of A-priori method.*

| dataset-> | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| baseline (accurate) | 0.954 | 0.883 | 0.641 | 0.558 | 0.591 | 0.492 | 0.317 |
| freq. it. (accurate) | 0.608 | 0.554 | 0.481 | 0.345 | 0.431 | 0.338 | 0.221 |
| baseline (approx.) | 0.998 | 0.986 | 0.985 | 0.995 | 0.942 | 0.968 | 0.711 |
| freq. it. (approx.) | 0.861 | 0.857 | 0.762 | 0.682 | 0.712 | 0.626 | 0.477 |

This method works much worse than the baseline at every time. It will be no longer fine-tuned and evaluated.

*6.1.2 Results of SVD method.*

| dataset-> | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| baseline (accurate) | 0.954 | 0.883 | 0.641 | 0.558 | 0.591 | 0.492 | 0.317 |
| SVD method (accurate) | 0.936 | 0.856 | 0.694 | 0.644 | 0.614 | 0.526 | 0.358 |
| baseline (approx.) | 0.998 | 0.986 | 0.985 | 0.995 | 0.942 | 0.968 | 0.711 |
| SVD method (approx.) | 0.98 | 0.96 | 0.95 | 0.986 | 0.894 | 0.938 | 0.692 |

In this comparison we showed, that our method yields better accuracies than naive approach. We also checked, how the distance metric can affect the results. In the following table we compare two metrics for computing distances between customer and cuisine:

(1) $L_1$ similarity:

$$sim(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} \mathbf{x}_i - \mathbf{y}_i$$

(2) cosine similarity:

$$\cos(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{xy}}{\|\mathbf{x}\|\|\mathbf{y}\|} = \frac{\sum_{i=1}^{n} \mathbf{x}_i \mathbf{y}_i}{\sqrt{\sum_{i=1}^{n} (\mathbf{x}_i)^2}\sqrt{\sum_{i=1}^{n} (\mathbf{y}_i)^2}}$$

| dataset-> | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| SVD method (accurate) - $L_1$ | 0.936 | 0.856 | 0.694 | 0.644 | 0.614 | 0.526 | 0.358 |
| SVD method (accurate) - cos. | 0.972 | 0.922 | 0.744 | 0.638 | 0.692 | 0.606 | 0.444 |
| SVD method (approx.) - $L_1$ | 0.98 | 0.96 | 0.95 | 0.986 | 0.894 | 0.938 | 0.692 |
| SVD method (approx.) - cos. | 0.998 | 0.984 | 0.972 | 0.978 | 0.908 | 0.96 | 0.734 |

Cosine similarity works better for this use case.

## 6.2 Subjective results on real data

Since there is no ground truth in real data (we don't have annotation about customers preferred cuisine, all that we can do is to look at the results returned by algorithm and try to subjectively assess them. We will play a role of experts.

We will use the algoritm from idea 2. - the one based on the SVD. We will test how the system works on both datasets:

- **Kaggle dataset**
- **Instacart dataset**: here, we will test the algorithm in two modalities:
  - Basket of customer is created from only one transaction of customer,
  - Basket of customer is created (concatenated) from all transactions of customer

In each case, we will look at classifications of the algorithm to given cuisine. We will look at customers baskets, that the algorithm classified to one of 3 cuisines:

(1) Italian,
(2) Mexican,
(3) Indian

We chose those cuisines, because we have some consciousness about what are typical things, that are prepared in those cuisines. For example, in Italian cuisine we would expect pasta, mozzarella, tomatoes... In listings of bought products, products, that would suggest given type of cuisine will be **bolded**.

We will return only those customers, for which their basket distance to closest is smaller enough than distance to second closest cuisine. *Small enough* means, that first distance should be less than x% than second distance. In our cases x is in range from 80 to 90. Such operation discards examples, when there is too little confidence to assign customer purchase to certain cuisine. $k$ - the number of concepts coefficients that are used to create profile of customer and cuisines was set to 50 (as an effect of experiments with subjective assesment).

*6.2.1 Test on Kaggle dataset.* Lets look at examples of customers, which were classified as having affiliation towards Italian cuisine:

- root **pasta** margarine food **coffee** bottled soda sparkling **wine** chocolate bar
- sausage hamburger, meat, tropical, fruit, milk, sliced, cheese, oil, food, soda
- hamburger meat curd hard cheese flour **pasta** margarine sugar food juice gum

The results are not good, they are not meaningful. We will not print out "Mexican" and "Indian" customers, because there, the results are also not meaningful. Lets try to figure out, what might have caused such poor performance. Lets look at example of customer purchases:

- citrus fruit,semi-finished bread,margarine,ready soups
- tropical fruit,yogurt,coffee
- whole milk
- pip fruit,yogurt,cream cheese,meat spreads
- other vegetables,whole milk,condensed milk,long life bakery product

Names of products are general. For example, tropical fruit can have many specific instances (like ananas or mango). As it was wrote in section about data cleaning, after some preprocessing only intersection of words from customer purchases and recipes is taken as a set of words, that can appear in products and recipes. Length of a vector of common words between customer purchases and recipes is:

(1) for Kaggle dataset: 123
(2) for Instacart dataset: 1839

Those are 5 recipes from original (not preprocessed) dataset of recipes:

- greek: romaine lettuce, black olives, grape tomatoes, garlic, pepper, purple onion, seasoning, garbanzo beans, feta cheese crumbles
- southern_us: plain flour, ground pepper, salt, tomatoes, ground black pepper, thyme, eggs, green tomatoes, yellow corn meal, milk,vegetable oil
- filipino: eggs, pepper, salt, mayonaise, cooking oil, green chilies, grilled chicken breasts, garlic powder, yellow onion, soy sauce, butter, chicken livers
- indian: water, vegetable oil, wheat, salt
- indian: black pepper, shallots, cornflour, cayenne pepper, onions, garlic paste, milk butter, salt, lemon juice water, chili powder, passata, oil, ground cumin boneless chicken skinless thigh, garam masala, double cream, natural yogurt, bay leaf

Folowing are recipes, that were cropped according to Kaggle dataset:

(1) greek: onion, cheese
(2) southern_us: flour, meal, milk, vegetable, oil
(3) filipino: cooking, oil, chicken, powder, onion, butter, chicken
(4) indian: vegetable, oil
(5) indian: milk, butter, juice, powder, oil, chicken, cream, yogurt

Folowing are recipes, that were cropped according to Instacart dataset:

(1) greek: romaine, lettuce, black, olives, grape, tomatoes, garlic, pepper, purple, onion, seasoning, garbanzo, beans, feta, cheese, crumbles

(2) southern_us: plain, flour, ground, pepper, salt, tomatoes, ground, black, pepper, thyme, eggs, green, tomatoes, yellow, corn, meal, milk, vegetable, oil
(3) filipino: eggs, pepper, salt, mayonaise, cooking, oil, green, chilies, grilled, chicken, breasts, garlic, powder, yellow, onion, soy, sauce, butter, chicken, livers
(4) indian: vegetable, oil, wheat, salt
(5) indian: black, pepper, shallots, cayenne, pepper, onions, garlic, paste, milk, butter, salt, lemon, juice, chili, powder, passata, oil, ground, cumin, boneless, chicken, skinless, thigh, garam, masala, double, cream, natural, yogurt, bay, leaf

This suggests, that the Instacart dataset might be more detailed and thus might serve better as a source for mining affiliations towards cuisines (because less general words may matter). Lets check that.

#### 6.2.2 *Test on Instacart dataset.*

- **Italian**:
  - almond, bag, banana, berry, blueberry, bottom, **cheese**, chocolate, dark, dat, **eggplant**, fruit, **fusilli**, globe, grape, grated, greek, **marinara**, medjool, mixed, **mozzarella**, nonfat, **parmesan**, pasta, raspberry, sauce, sea, strawberry, the, **tomato**, wheat, yogurt
  - **linguine**, **pasta**
  - burger, california, mushroom, **ravioli**, **ricotta**, veggie
- **Mexican**:
  - apple, **avocado**, blueberry, bran, butter, cereal, **cheese**, **corn**, cream, creamer, extra, french, grade, honey, jack, **jalapeno**, juice, kernel, lemonade, medium, **mexican**, mix, natural, nut, peanut, **pepper**, pure, raisin, **salsa**, shredded, slic, sliced, sour, taco, tomato, vanilla, vegetabl, vegetable, winter
  - **avocado**, blend, **cheese**, **corn**, has, **mexican**, plain, shredded, **tortilla**, yogurt
  - almond, **avocado**, broth, chia, chicken, **cilantro**, **corn**, free, gluten, halv, inch, oat, pie, piec, rolled, seed, shell, **tortilla**, walnut, white
- **Indian**:
  - almond, avocado, baby, bag, banana, brown, **butter**, **cilantro**, **coconut**, extra, firm, **ginger**, grain, green heart, hemp, lemon, mango, milk, millet, oat, oil, oliv, onion, pitted, ramen, raw, red, **rice**, rolled, root, seed, shelled, short, spinach, stock, thick, tofu, unsweetened, vanilla, vegetable, white
  - almond, butter, cacao, cashew, chia, chocolate, coconut, creamy, dat, **ginger**, ground, halv, heart, hemp, lemon, medjool, nib, oil, piec, raw, refined, root, seed, shelled, walnut
  - beverage, blueberry, chunk, **coconut**, fat, free, fusilli, **ginger**, **mango**, **milk**, natural, plain, premium, raspberry, root, unsweetened, yogurt

- **Italian**:
  - **cheese**, **mozzarella**, natural, pea, petite, string
  - apple, **arugula**, baby, **basil**, bell, blueberry, **broccoli**, **bucatini**, **cheese**, flour, grated, habanero, homemade, jalapeno, juice, medium, milk, **mozzarella**, **parmesan**, part, pepper, red, roasted, **salami**, salsa, sauce, skim, spinach, strawberry, **tomato**, tortilla, uncooked, vodka
  - apple, **basil**, butter, calorie, chip, chocolate, chunk, cookie, cooky, dough, extra, free, gluten, grated, greek, honey, **italian**, juice, medium, **mozzarella**, natural, nut, oil, olive, **parmesan**, salsa, sausage, shredded, soup, sparkling, spicy, strained, sweetener, **tomato**, yogurt, zero
- **Mexican**:
  - blueberry, **cheese**, mexican, shredded, **taco**, yoghurt
  - **avocado**, **has**
  - **avocado**, beef, blend, bun, **cheese**, classic, dog, **has**, hot, **mexican**, polish, sausage, shredded
- **Indian**:
  - milk, plain, **yogurt**
  - apple, bag, balsamic, banana, bartlett, berry, black, cereal, chipotle, currant, dressing, food, **ginger**, grape, greek, hibiscu, kale, lean, lite, lowfat, milk, mixed, pear, plain, raspberry, seedles, slic, spicy, strawberry, unsweetened, **yogurt**
  - apple, boneles, **chicken**, cilantro, **coconut**, crushed, cucumber, curry, enriched, extra, garlic, **ginger**, grain, ground, kirby, long, mint, onion, paste, powder, pulp, red, **rice**, skinles, thigh

In our opinion, system works. Situations that something completely unreasonable is returned are rare. What more, there are matches that are just in point. It is worth noting, that in majority of cases, for human expert it is hard (if possible) to tell definitely the connection of user with one of cuisines, in most of cases it is possible to find some evidence, sometimes stronger, sometimes weaker. Big amount of customers are buying sets of things, that are hard to match with certain recipe or cuisine. It may happen, when customers buy just everyday-need products like for example: milk, eggs, chicken - for human there is no possibility to tell exactly what this customer want to cook.

### 6.3 Finding for more sophisticated classifiers

At the classification phase, we tried to do something more complicated than just nearest neighbours methods. We tested simple neural networks - multilayer perceptrons with one/two hidden layers and softmax output layer. Unfortunately, results were worse than using nearest neighbours method, that's why we resigned from further development of neural networks.

### 6.4 Tests with Generalized Jaccard Similarity

Real data is imperfect. For example misspellings can happen. Also some words can be written in the different way, but in reality they signify the same, e.g. "yoghurt" and "yogurt" or "spaghetti" and "spagetti". There was need to find function for counting similarity between two sets of strings, that would work analogously to Jaccard Similarity, but also that would not discard close (but not perfert) similarities between elements of sets. Such solution is the Generalized Jaccard Similarity: [Gen [n. d.]]. If Jaro Similarity between two strings is bigger than given threshold, those strings are considered as the same.

We were testing for the best thresholds to set in algorithm and after experiments, it turned out that it is 0.8. Unfortunately, results

were not better (at least not significantly) while time of creating the utility matrix has grown significantly. That's why we resigned from using that solution.

# 7 CODE

## 7.1 Technologies

System was implemented using Python3 language with additional libraries. Algorithms that were not implemented by us, we used ready implementations:

- SVD: [SVD [n. d.]]
- A-Priori: [Apr [n. d.]]
- Generalized Jaccard Similarity: [Gen [n. d.]]

Code is located in Jupyter Notebook files (.ipynb).

## 7.2 Description of the repository

We describe how to navigate through the repository:

- **src**
  - **real_data**
    - *Products data cleaning - kaggle.ipynb* - notebook with process of cleaning Kaggle data; outputs:
      - *../../data/groceries/user_to_products_one_purchase_text.p* - dictionary with customer to cleaned list of products
      - *../../data/groceries/recipes_cropped.csv* - recipes cleaned in respect to Kaggle data
    - *Products data cleaning - instacart.ipynb* - notebook with process of cleaning Instacart data; outputs:
      - *../../data/instacart/products_cropped.csv* - products data with names cleaned
      - *../../data/instacart/recipes_cropped.csv* - recipes cleaned in respect to Instacart data
    - *UsersToProducts.ipynb* - notebook for creating data about purchases of customers needed for Instacart dataset (those are saved in pickle files: *.p in *../..data/instacart*)
    - *Recommender systems.ipynb* - notebook with main algorithm, it is possible to easily switch options there: Kaggle / Instacart, in Instacart we can choose whether we are creating customer profile from one purchase or from all accessible.
  - **artificial_data**
    - *Synthetic data creation.ipynb* - script for creating data synthetically
    - *Naive approach.ipynb* - notebook with implementation of baseline algorithm and test of performance on artificial data
    - *Frequent itemsets.ipynb* - notebook with implementation of algorithm basef on frequent itemsets and test of performance on artificial data
    - *Recommender systems.ipynb* - notebook with implementation of algorithm based on SVD decomposition and test of performance on artificial data
- **data**
  - **real_data**
    - **groceries** - folder with Kaggle customer shopping data,
    - **instacart** - folder with Instacart customer shopping data
    - **recipes** - folder with recipes
  - **artificial_data** - folders with created data (7 folders, for descriptions look in section 4.2.). Each folder contains 3 files:
    - *bought_products.p* - for each customer, the products that he bought are there,
    - *dominant_cusine_for_user.p* - for each customer, his dominant (most preferred) cuisine is written. It is a ground truth.
    - *recipes_of_cuisine.p* - set of recipes for cuisines.
    - **groceries** - folder with Kaggle customer shopping data,
    - **instacart** - folder with Instacart customer shopping data
    - **recipes** - folder with recipes

# 8 CONCLUSION

Algorithm based on the SVD not only has some useful intuition, that may suggest correctness, but also handles stated problem. On artificial dataset it works well. In practical case it also works, but here it is rather a fuzzy statement, because there is no way to assess it directly. In our opinion it works well enough, taking into consideration how complicated and unclear real data are. It is worth noting, that for big amount of examples from purchase dataset for human the task is also unclear.

# REFERENCES

[n. d.]. Apriori implementation), howpublished = https://pypi.org/project/efficient-apriori/, note = Accessed: 2020-01-10.
[n. d.]. Generalized Jaccard similarity measure), howpublished = https://anhaidgroup.github.io/py_stringmatching/v0.3.x/GeneralizedJaccard.html, note = Accessed: 2020-01-10.
[n. d.]. Groceries Market Basket Dataset), howpublished = https://www.kaggle.com/irfanasrullah/groceries, note = Accessed: 2020-01-10.
[n. d.]. The Instacart Online Grocery Shopping Dataset 2017), howpublished = https://www.instacart.com/datasets/grocery-shopping-2017, note = Accessed: 2020-01-10.
[n. d.]. Numpy SVD implementation), howpublished = https://docs.scipy.org/doc/numpy/reference/generated/numpy.linalg.svd.html, note = Accessed: 2020-01-10.
Jure Leskovec, Anand Rajaraman, and Jeffrey D. Ullman. 2014. *Mining of Massive Datasets.* Cambridge University Press.