DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics: Games Engineering

# Enhancement and Evaluation of an Algorithm for Shopping List Generation based on Meal Plans

Marco Menzel

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics: Games Engineering

# Enhancement and Evaluation of an Algorithm for Shopping List Generation based on Meal Plans

# Erweiterung und Evaluation eines Algorithmus zur Einkaufslistenerstellung basierend auf Mahlzeitenplänen

| | |
|---|---|
| Author: | Marco Menzel |
| Supervisor: | Prof. Dr. Georg Groh |
| Advisor: | Monika Wintergerst, M.Sc. |
| Submission Date: | 15.12.2021 |

I confirm that this bachelor's thesis in informatics: games engineering is my own work and I have documented all sources and material used.

Munich, 15.12.2021 Marco Menzel

# Abstract

Unhealthy diets and malnutrition can lead to physical aswell as mental deseases. The usage of a meal plan can help with maintaining a health diet. To get the most nutrients out of any meal, we require fresh ingredients. To enable a more effective approach to the organisation of grocery shopping, automaticly generated shopping lists provide a time-effective alternative to traditional paper based lists. We took an existing program and tried to optimize it using natural language processing. We implemented data preprocessing into the grocery list generation aswell as categorisation.While our preprocessing algorithm performed well at making lists more compact, it had its problems with sorting items into food categories, as the relevent information between the tasks shifted significantly.

# Contents

# 1 Introduction

## 1.1 Section

Citation test [**latex**].

### 1.1.1 Subsection

See Table 1.1, Figure 1.1, Figure 1.2, Figure 1.3.

Table 1.1: An example for a simple table.

| A | B | C | D |
|---|---|---|---|
| 1 | 2 | 1 | 2 |
| 2 | 3 | 2 | 3 |

$$\begin{tikzcd}
R_1 \arrow[r, dash] \arrow[d, dash] & R_2 \arrow[r, dash] \arrow[d, dash] & R_5 \\
R_3 \arrow[r, dash] & R_4 &
\end{tikzcd}$$

Figure 1.1: An example for a simple drawing.

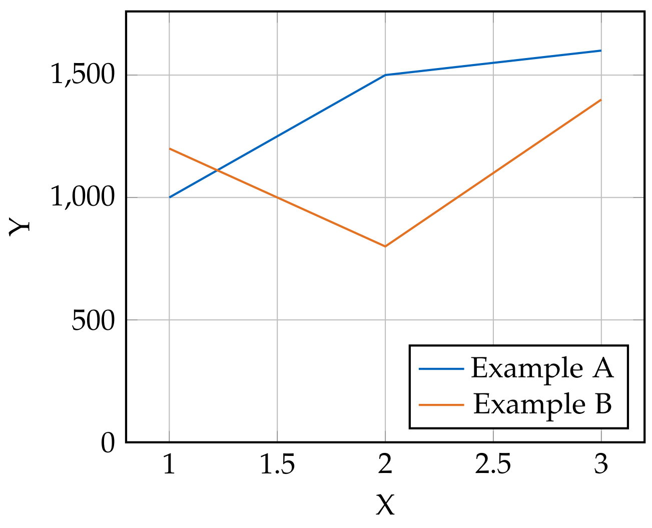

Figure 1.2: An example for a simple plot.

```sql
SELECT * FROM tbl WHERE tbl.str = "str"
```

Figure 1.3: An example for a source code listing.

# 2 Motivation

In this section we will lay down the main reasons to use a meal plan generated grocery shopping list. First we will show the efficiency of using a shopping list in general. Later we will take a look at physiological and psychological benefits of using a meal plan.

After housing, food is the second largest expense of an average household. Grocery shopping takes up over an hour per week. Shopping lists can help to organize this process and to reduce the amount of time aswell as money spend in a store. Women are more likely to use a shopping list: 69% of women are using lists compared to 52% of men. The majority of shopping lists are paper-based. Only 3% of shoppers use a digital list. Digital shopping lists are easier to maintain. In most of shopping list apps, items can simply be ticked off, or altered otherwise. For households with multiple people shopping groceries, synchronisation can be done quite easily. This can lead to less redundant purchases and less waste. A very simple feature, that helps organizing a shopping list is the categorisation of items on the list. This is another benefit over paper-based lists, on which you cant rearrange items. A good categorisation - that is similar to the organisation in the store - allows us to save a lot of time and searching while grocery shopping. We will show optimisation of categorisation in 5.3

Data from the AHA showed metabolic syondrome or syndrome X for 35.5% of adults in the USA in 2012, with numbers expected to have risen since then. There are links between metabolic syndrom and diet.Furthermore studies have shown, that a diet following a meal plan can help people with loosing weight. In conclusion weight reduction is a key element in the prevention and treatment of the metabolic syndrom. The metabolic syndrom is a metabolic disorder, causing an increased risk of coronary heart diseases, heart attacks aswell as type 2 diabetes. It is caused by high blood sugar, blood preasure and high levels of triglycerides aswell as low levels of HDL cholesterol. The data showed metabolic syndrome or syndrome X for 35.5% of adults in the USA in 2012, with numbers having risens since [**hirode2020trends**]. [todo: mealplan verknüpfung] There are links between metabolic syndrom and diet, and studies have shown that a diet following a meal plan can help against metabolic syndrom.

We have shown that a meal-plan based, digitaly generated shopping lists can have a positive impact on the efficiency of grocery shopping and provide a healthy diet.

# 3 Related Work

Digitalization has imposed great changes to the way grocery are purchased. But while there is an increasing interest in online grocery shopping platforms, there is also progress in digitalizing the shopping experience in local supermarkets.

In 2011 Heinrichs et. al. proposed "The Hybrid Shopping List [**heinrichs2011hybrid**]. It featured a combination of a digitalised and a paper-based list. In their work they found that only a very small part of used shopping lists were digital. The hybrid approach tried to make digital shopping lists more accessible, by still allowing the usage of traditional shopping lists. Using a digital pen, a paper based shopping list could be appended to the digital list. The digitalised list allready included some of the appealing features of digitalised shopping lists, like synchronisation between multiple devices within a household.

Focusing on the retailer side of technological advances, there are a lot of different proposals for the use of "Smart shopping Carts". One approach [**7932080**] uses an inbuild RFID-reader in the cart and RFID-tags in the items to automate billing. Also it can help with the logistics in the store, so if items run low, staff is informed automaticly. Other approaches use indoor navigation in combination with a shopping list to help customers path the store efficiently. Both approaches can improve the shopping experience for the customer. Attacking bottlenecks like searching items, waiting in line at the cash register, or the payment process itself enables customers to reduce their time in the store and the retailers to generate more revenue [**inproceedings**].

A very sophisticated way to reduce or extinguish the bottleneck, induced by the payment process, comes from Amazon. In the Amazon Go stores [**wankhede2018just**] customers are tracked using computervision. After a check-in with their phone, customers are tracked by optical sensors. The sensors detect if a customer picks up an item or put it back. With this information the items of every customer are tracked and the customer is automaticaly charged upon leaving the store.

# 4 Base Application

This thesis is focused on optimizing an already existing application. In order to show what optimisations were made, we must first go through the initial application. This section will focus on the underlying application, while the next will go into the optimisations in detail.

The application is API-based, consisting of three APIs: Grocery List API, Recipe API and MealPlan API. The recipe-API manages recipies, their reqired ingredients aswell as the respective quantities and units for them. In addition the individual ingredients are put into categories for the shopping list. For categorisation we use a database provided by the United States Department of Agracultures, containing around 8800 ingredient-category pairs. On this basis we can vectorise the ingredients, using a pre-trained languge model called BERT. When a new ingredient is added, we vectorise it with BERT. Afterwords we search our vectorspace for the closest vector, matching their category.

The MealPlan-API maintains the meal plans. Its main purpose is to map recipies to a stored meal plan. The API maps the recipies for a fixed timeslot aswell as a fixed number of meals per day.

Finally, in the GroceryList-API we create the shopping list. The GrocerList-API generates the list for a given meal plan and timeslot(in weeks). For the list creation

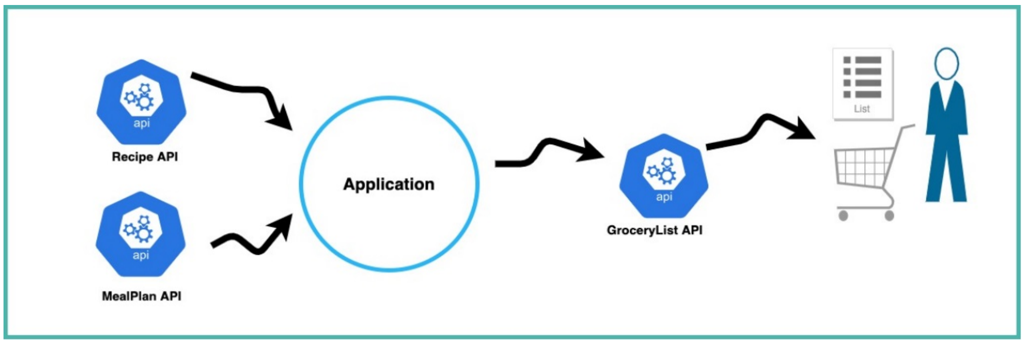

Figure 4.1: High level Architectire of the application

we iterate over each single meal per day for the entire timeframe. For every meal we iterate over the single ingredients, either increasing the quantity or adding it to the list, if not previously contained. Once the list is complete, we order the list according to a supermarket layout.

# 5 Optimization

In this chapter is the main contribution of this thesis. We will look at 3 different optimizations

## 5.1 Unit conversion

The application could previously only work on "gram", "ml" aswell as a item count. Recipies commonly feature units like tablespoons(tbsp), teaspoon(tsp), dashes, pinches, and in the US a lot of ingredients are measured in "cups". This system is based on volumes, unlike the usual metric approach where most measurements are made in mass. To provide a universal usability we have also included imperial aswell as us-customary units. While we can not convert between the us customary system and regular imperial system, we implemented a converter for both systems to the metric scale. The systems can be changed in the configuration of the application. Because we are using a meal plan provided by the austrailian government, the default system is set to imperial. All operable units are listed in table *insert table*.

| Name | Symbol | SI factor | Name | Symbol | SI factor |
|---|---|---|---|---|---|
| Imperial | | | US-Customary | | height |
| Volume | height | | | | |

Table 5.1: Units covered by the program

There are a couple of problems within the system of units that not covered by our algorithm. Some measurements, like the "pinch", are not even clearly defined within one system. Sticking with our example, the "pinch": a pinch is historically defined as "the amount that can be taken between the thumb and the forefinger" [**rowlett2000dictionary**]. Obviously this "measurement" varies significantly person to persons. To provide more consistent means to quantify a pinch in the US a pinch is defined as 1/16th of a taespoon, while in the UK it is defined as 0.355625g. This leaves us with a unit that can represent both mass and volume, without regard for density. This makes the "pinch" and similar inconsistent measurements unfitting for

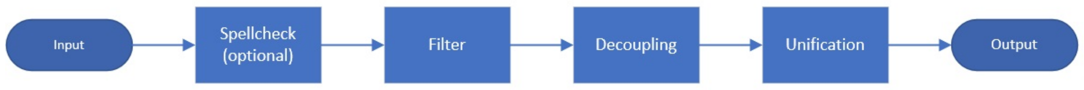

Figure 5.1: Preprocessing Pipeline

a scientific approach to a healthy diet. Our application features prestored recipies. These recipes contain the informaton for quantities aswell as units for each ingredient. This combination of user configurated unit interpretation and prestored, context-free, inconsistent units may lead to a deviation of the optimal nutrition levels defined by the mealplan. The combination of count, volume and mass units leads us to another problem not covered by our unit converter. The converter only converts within the context of mass or volume, but does not feature conversion between them. This may lead to the final shopping list containing entries for each type. Integrating a database with densities and average mass for food items should fix this problem.

## 5.2 Ingredient preprocessing

One problem of the initial application is that there are multiple occurences of the same ingredient, with a different suffix. For example the application might generate a list entry for "red onions" aswell as "red onions, thinly sliced". Simply dropping the suffix on the ingredients could help in some cases, like our onions, while for other cases droppign the suffix could be problematic. For "fried shallots" or "dried tomatoes" buying the product referenced in the recipe could save the user significant amounts of time, depending on the required procedure. In this section we will show, how we used data preprocessing to approach this problem. Our preprocesser is regular expression based. Given our context, the possible input strings are already quite well defined. As we do not need to fulfill complex functionality, state of the art maschine learning approaches seem a bit far fetched. We feel that regular expressions provide enough functionality to solve our problems.

Figure 5.1 shows how the individual ingredient names are preprocessed. First we apply spellchecking, then we filter unnecessary information from the string. Followed by decoupling for any String containing "and", finally we unify the format and spelling of the strings. In this section we will describe each step of our preprocessing pipeline in detail chronlogicaly.

### 5.2.1 Spellchecking

Before we can start processing the data, we need to make sure, our input is spelled correctly. Covering typos in regexes can be quite difficult and can also lowers readability of code. To get rid off any typos we use the pyspellchecker library [**spellchecker**]. The spellechecker uses levenshtein Distance to find permutations within a frequency list. The spellchecker introduces a large performance overhead, which lead us to implement a configuration option "use_spellchecker". This option allows the user to disable the spellchecker, if needed. Taking into account, that most people need to spend time reachign their local grocery market, the overhead should not be relevant in a "rolled out" application. Alternatively the spellcheck could be applied when adding new recipes to the database. This would reduce the impact to the performance of our programm significantly, since it only impacts our setup, but does not affect the program at runtime. Also we can assume that recipes are getting read multiple times, while we only need to write them once, further increasing the benefits of applying spellchecking while adding recipes.

### 5.2.2 Filter

Recipes usually do not specify the requierd ingredients in their natural form. Ingredients are often already processed, for example vegetables are usually cut into smaller pieces or peeled in order to provide lower cooking times, a more pleasent texture or a more appealing look. For the grocery list these specifications do not matter. Only in few cases the product, aimed to be bought at the store is affected by the additional information contained in the recipes desciption. One example for this could be "dried" fruits or vegetables, as the preparation time would exeed the time a regular person is willing to invest into their daily meals.
In the filtering step, our preprocesser aims to eliminate irrelevant information. There are three different types of information we try to eliminate:

- Descriptions of cooking procedures(e.g cleaned, sliced, crushed),
- adjectives that further specify the procedures(e.g. "thinly","roughly"),
- "fancy" product descriptions like "crisp salad" or "fresh tomatoes"

The first and the last point are handled with an identical approach. We use string matching for regular expressions and substitute any found pattern with an empty string. The pattern we are searching for are wordbound occurences of predefined strings. These strings are then combined in a disjunctive expression. The list of words contained in our filter is read from the config. The core reason to split up the list into

two parts was to enhance readability and maintainability of the lists.
To filter adjectives that further describe our input, we simply search for any word ending in "ly". To avoid falsely eliminating food items like "jelly" we also integrated a list exceptions in our config.

The final step in our filter is the elimination of redundant commata, spaces or "and"s. To do this we simply match any occurance in the beginning or end of our input, aswell as any surplus spaces or commas within.

### 5.2.3 Decoupling

Some ingredient descriptions feature a combination of multiple ingredients. To raise readability of our grocery list we try to decouple such items into a list of their components. For example, instead of having an item called "Salt and Pepper", we would prefer to split the item up into an entry for "salt" and another one for "pepper". For listings we have found that the usual format is having a leading generic term, followed by an instance. If lets say a recipe requires two different oils the item would be listed as: "oil, pumpkin seed and apple". To split the input in its components, we use this function:

```python
def decouple(input):
    pattern_and = re.compile(
        r"([\w\s]+,?␣)?([\w\s]+,?␣)?([\w\s]+,?␣)?([\w\s]+,?)?(and␣)([\w\s]+)"
    )
    match = pattern_and.match(input)
    components = []
    if match is None:
        return [input]
    if match.group(1) is not None:
        components.append(match.group(1))
    if match.group(2) is not None:
        components.append(match.group(2))
    if match.group(3) is not None:
        components.append(match.group(3))
    if match.group(4) is not None:
        components.append(match.group(4))
    if match.group(6) is not None:
        components.append(match.group(6))
    if components.__len__() > 2:
        head = components.pop(0)
        i = 0
        while i < len(components):
            components[i] = head + components[i]
            i = i + 1
    return components
```

The pattern seraches for any string containg "and " and groups text segments seperated by ",". If we can not match a "and"-pattern we return a list only containing our input. Otherwise we append every non empty group. Group 5 represents the "and " we are trying to eliminate and is conseqently not added. Furthermore if we have more than 2 components we pop the head of the list, and add it to every remaining component. Sticking to our earlier example, the function would transform "oil, pumpkin and apple" into ["oil, pumpkin", "oil, apple"]. The programm assumes even distribution over each component. So the quantity defined by the recipe is devided by the number of components.

Furthermore we differenciate between the word "and" and the ampersand(and symbol) "& ". The ampersand is used to describe flavours of ingredients. We must maintain any descriptions containing an "& " symbol as splitting up here would result in a false result.

### 5.2.4 Unification

At this point, our initial input has been spellchecked, most irrelevant information should be filterd, and the input has been decomposed into elemental items. As a last step, we need to make sure that any difference in spelling, wording and formating must be eliminated. To unify the remaining inputs we try to substitute any possible option into one predefined one. We split the unification process into four parts.

First we try to unify any terms, or generalize terms. There are a lot of possible wording options for terms like "fat-free" "gluten-free" and so on. Also there are product descriptions we can neglect. Brand loyalty is a the main factor in choosing products[**doi:10.1080/02642069600000006**]. Consequently we assume that a customer would buy his regular brand of olive oil regardless of our generated list specifying an extra-vergine olive oil or a regular one.

```
item = re.sub(r"fat[␣-]?free|non[␣-]?fat", "fat-free", item)
    item = re.sub(r"(extra␣vergine␣)?olive␣oil", "olive␣oil", item)
```

The two examples of suppstitutions show how we can easily unify different spellings, with the first attribute describing a pattern and the second one its replacement on the stringe defined by the third attribute. One generalisation not covered by our unification function, are requests for specific brands. While we can filter out upc-codes, brand names and their specific product descriptions require a much more complicated nlp system. For a single market and a fixed inventory list, this could be done by a filter similar to the one we described in (5.2.2), covering amount of global brands and number of their products with a word filter is infeasible.

As mentioned earlier we also need to unify spelling. While our spellchecker (5.2.1) should cover any typos, there are still some words with multiple correct spelling options. In the word "yoghurt" the "h" is common for most english speaking countries, excluding canada and the united states. Our unification function covers common food items with multiple spelling options and substitutes them in a identical fashion as the earlier examples.

At this point there are only two tasks left. The mapping of words in singular and plural and the unification of the format. To enhance readability and to unify the format we chose the to start with descriptions and end with the item name. To do that we invert the order of our string, at the first occurence of a comma.

```
pattern_unified_format = re.compile(r"([\w ]+), ([\w ]+)")
h = pattern_unified_format.match(item)
atch is not None:
item = match.group(2) + " " + match.group(1)
```

For handling singular and plural, we simply try to match strings with standard plural endings. We match while searching our grocery list for similar ingredients, and for any match of a singular and plural of the same word the plural word is written into the list.

## 5.3 Categorisaton

The optimization of categorization is done by processing the input with the in 5.2 mentioned preprocessor. The following chapter focuses on evaluating the categorisation aswell as grocery shopping list generation. We will compare our implementation with the inital implementation by asela

# 6 Evaluation

## 6.1 Evaluation of ingredient Categorization

As aforementioned categorisation is done with the BERT language-model provided by google and a corpus containing 8800 entries. For every new ingredient, BERT vectorizes the input and searches for a corpus-entry with the shortest distance. The category is then copied from this entry.
We applied the preprocessing algorithm 5.2 before the vectorization process. We tested the impact of our preprocesser using five meals. We chose different meals out of the north/south/latin american, eurpoean and asian cuisine. In addition to that, we chose each one cocktail recipies originated in these five regions. In spite of the regional influences, we take a separate look at the cocktails, because they have more shared or similar ingredients among each other than the other recipes of the same regions. We include Cocktails to cover a wider range of ingredient categories, compared to exclusively using regular dishes. We also cover most of the daily meals by including a breakfast and a dessert for each region. Our testset also features multiple vegetarian and pescetarian dishes aswell as dishes with meat. With these attributes we hope to cover a wide range of possible ingredients, to test our programm with. Our goal was to diversify our testset as much as possible, in order to see if there is any bias within our corpus. This information would enable us to optimize our corpus efficiently.

### 6.1.1 Testset

We will first take a detailed look at our testset. The 30 dishes include 425 different ingredients. These ingredients are divided into 25 categories. Processed by the program our testset covers: Dairy and Eggs, Spices, Fats/Oils, Poultry, Soups/Sauces, Sausages/Luncheon meats, Fruits and Fruitjuices, Pork, Vegetable, Nuts/Seeds, Beef, Beverages, Finfish/Shellfish, Legumes and Legume products, Lamb/Veal/game products, Baked Products, Sweets, Cereal Grain and pasta, Fast Food, Side Meals and Snacks. While we tried to cover a wide range of different ingredinets, we did not include ingredinets for the following categories: Breakfast cereal, Baby Food, Restaurant Foood and American Indian/alaska Native food. The following diagram shows

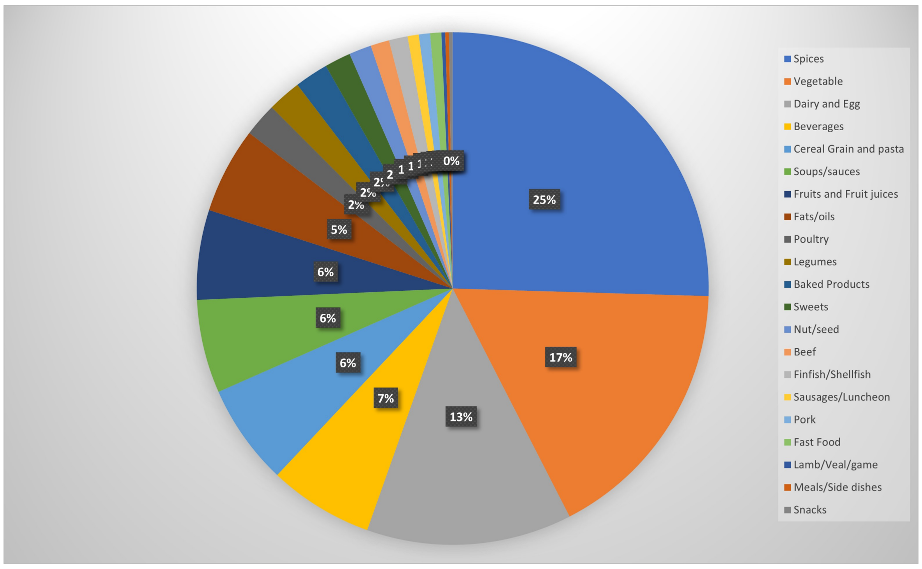

Figure 6.1: ingredient distribution over included categories

how many ingredients are contained by every category. The categtories Spices(108, 25.4%), Vegetables(72, 16.9%) and Dairy and Egg Product(55, 12.9%) contain the most ingredients. Figure 6.1 shows the distribution over all included categories. Figure 6.2 shows how the amount of ingredients is distributed over each region. When picking our ingredients, we searched for recipes with a larger amount of ingredients to get a larger testset. The List of recipes we used can be found in the recipe.xlsx file in the project git repository. The document also features more detailed information how many errors occured in each recipe.
We will now look at the result of our tests. First we will take a look at how each category performed. Followed by an analysis of each regions categorisation. We will differentiate between correct categorisation, wrong categoriation, and not assigned. The testset will be categorized by both our implementation aswell as the initial implementation by Asela, to gain better information about the impact of our preprocesser.

### 6.1.2 Analysis of Categories

### 6.1.3 Analysis of Regions

### 6.1.4 Problems with the preprocesser for categorization

## 6.2 Grocery List Generation

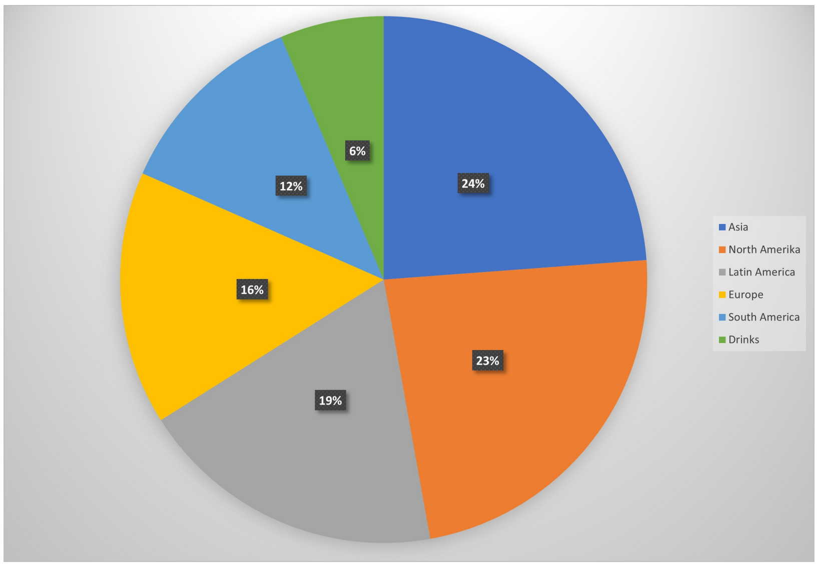

Figure 6.2: ingredient distribution over regions

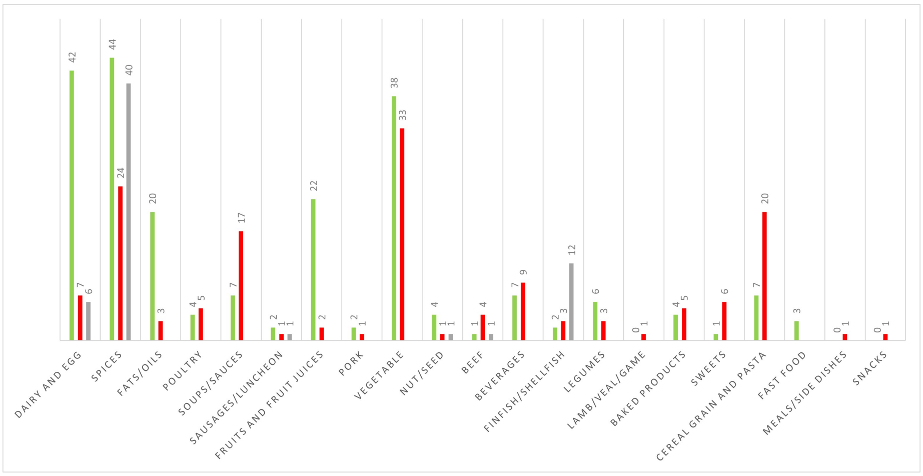

Figure 6.3: Categories without preprocesser

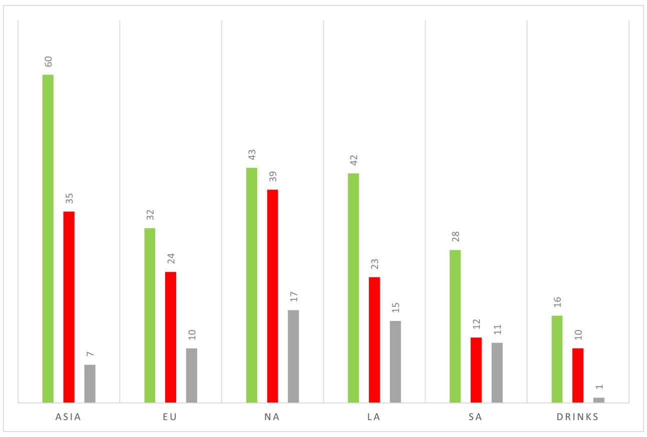

Figure 6.4: Categorization for ingredients from diffrent regions, initial Implementation

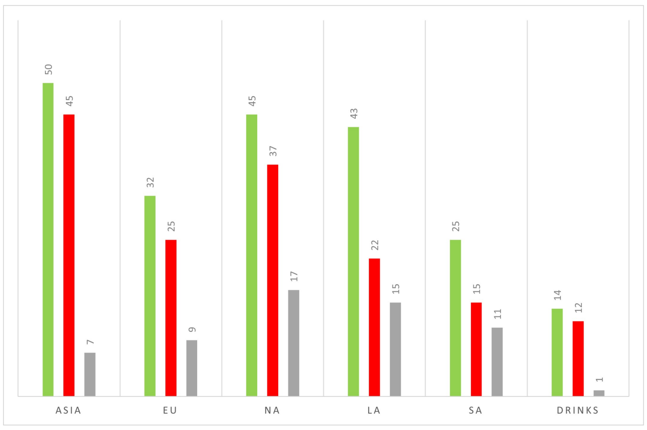

Figure 6.5: Categorization for ingredients from diffrent regions, our Implementation

# 7 Conclusion and future work

Our evaluation shows two things. The preprocessor can solve the problem it was initialy written for. This is cleaning up grocery lists and making them more compact. On the other hand we see that the cleaned up versions created by the preprocesser loose information for other systems that require context. The system would require multiple preprocessers for the seperate tasks at hand, as their purpose diverges. On one hand, we want to create an easily readable shopping list, without any irrelevant information. This allows us to purchase with higher efficency, saving us time and possibly money. On the other hand, knowledge on how a given ingredient could be processed yields relevant information about which food category an item belongs to. In further development of this project a preprocesser more tailored for the categorization-problem would be required. The properties described in 6.1.4 should be implementet into an additional preprocesser.

The unit converter as described in 5.1 still requires further development aswell. While we have added some additional systems, the system is still not universally usable. Right now only one of imperial or US-customary units can be processed. This disables us from combining recipies containing units of different systems into one mealplan. To solve this problem we could extend our ingredient model by an additional attribute for the system the unit is in. Furthermore another yet unsolved problem is the combination of different types of units. If our mealplan features recipes that contain volume and mass unit types aswell as counted items, the list will contain three instances of the same item. A quick fix for this would be an augmentation of the grocery list item. If we change the items to contain lists of quantites and units, similar to the json we are using to add recipes, we could reduce the instances to one. The best possible solution for this problem would be an inclusion of a database that features density aswell as the mass of an average item. This would allow us to combine the different unit types.

In 6.2 we have shown that the preprocesser can allready help cleaning up the grocery shopping list. The preprocesser is easily maintainable, by including a lot of configuration items. While adding our testset(6.1.1) we have seen that this maintainance is also required. An approach solely based on regular expressions can solve its task well, but will probably require much more maintainance to deliver similar results, compared to a maschine learning approach.

# List of Figures

# List of Tables