A fuzzy approach to Food Security through Microblogs

Alexander Buesser

Computer Science

École Polytechnique Fédérale de Lausanne

I hereby declare that this paper is all my own work, except as indicated in the text.

Signature ______________________________

Date ______/______/______

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Acknowledgements

Contents

Chapter 1

Introduction

Chapter 2

Data

For this report we downloaded 2 TB of Tweets from the archive over a span of October 2011 - September 2014.

2.1 Lexicon

In this section we describe the filtering process of the tweets and the creation of the two lexicons. One lexicon contains tweets with food related terms (e.g. rice, wheat, milk) where as the other contains tweets with factors influencing the price and supply of the goods (e.g. oil, unemployment, flood). The filtering process resulted with 1047698 food relevant tweets and 523549 tweets of influencing factors.

2.1.1 Keyword Selection

The filtering of the dataset was initially performed with a simple list of food related keywords. To avoid ambiguities we will refer to the initial keyword list as K_i. As a first source for our set K_i we used the most common traded food commodities as it would easily allow us to verify our results using the price dataset made available by IMF [1]. We further decided to include the ten most important staple foods that feed the world as defined by Allianz[2]. Tweets were retrieved through exact term matching i.e. a tweet containing foods would not match on the keyword food where the reverse is also true. We mimic the term matching twitter performs. In the initial round we aimed for maximum coverage and hence avoided further filtering steps. The result was a collection of 1047698 tweets posted by 949085 user.

[1]http://www.imf.org/external/np/res/commod/index.aspx

[2]http://knowledge.allianz.com/demography/health/?767/the-worlds-staple-foods

Looking at the distribution of the food related tweets we realised that we would have to categorise our lexicon in order to have sufficient data for further analysis. Where global keywords such as food are highly represented, more specific keywords such as beef only occur very infrequently. Other then the sparsity of the data we also have the problem of ambiguous keywords. Soy is such a keyword that refers in english to the "bean" and in spanish to the verb "to be". To create categories we chose to mimic the categorisation of the FAO [3]. The FAO tries to measure the overall food fluctuation by five different food categories namely *meat, dairy products, cereals, vegetable oil* and *sugar*. The weighted average of those five categories defines the international food price index. We additionally created a further category named *Other Food of Interest*. This category contains general keywords (e.g. food, dinner or lunch) and food keywords that can not be assigned to one of the five categories but frequently occur (e.g. coffee, tea). To be considered frequently the set of tweets containing the keyword needs to be $> 1\%$ of the total sample. Those six subsets s are $\in K_e$. C is a fictional set that contains the five categories *meat, dairy products, cereals, vegetable oil, sugar* each building a subset containing all possible food items belonging to a specific category (e.g. the subset dairy would contain all possible dairy products). If the following relationship holds $k \in C$ for any keyword $k \in K_i$, we consdier $k \in K_e$. For all keyword $k \notin C$ the condition of it being frequent is evaluated and if true added to K_e. Food commodities that could not be assigned to one of the six categories were discarded. Upon manual examination of the dataset we realised that people are much more likely to talk about a specific food product rather then the raw material. Cereals are not a public interest, however products such as bread or flower occur much more frequently. The set K_e was further enriched by using food products that have been identified by [**?**] in set K_f only $\forall\ k \in K_f$ that are also $\in C$. To further improve our coverage of the six food categories we filtered for synonyms and contextual similar words using HAL. What we mean by contextual similar and how HAL retrieves those keywords is further described in Section 3.3.1.

Our approach. We took several steps in order to improve our detection of the desired food commodities. K_e was created as follows:

1.) We add all keywords $k \in K_i$ to K_e only if $k \in C$ or k is frequent

2.) Further add all keywords $k \in K_f$ to K_e only if $k \in C$

3.) From a subsample of 10% we create a HAL space with all keywords that occur > 100. $\forall c \in C$ we pick the keyword $k \in K_e$ that most frequently occurs and

[3]http://www.fao.org/worldfoodsituation/foodpricesindex/en/

retrieve the top 500 similar terms. We hand select those that are $\in C$.

The keyword set K_e was used to perform exact term matching on the tweets collected from the archive. The resulting set of keywords in K_e forms our Food Lexicon.

Lexicon / Subset s	Keywords (i: from initial set, e: from K_f , h: from HAL space)
K_i Food	meal (i), meals (i) ,food (i), foods (i), wheat (i), rice v, maize (i), carley (i), soybean (i), soy (i), meat (i) , beef (i), cattle (i), chicken (i), poultry (i), lamb (i), swine (i), pork (i), fish (i), seafood (i), shrimp (i), salmon (i), sugar (i), bananas (i), oranges (i), coffee (i), cocoa (i), tea (i), milk (i), yams (i), cassava (i), potatoes (i), sorghum (i), plantain (i), nuts (i), onion (i), salt (i), egg (i), dairy (i), cereals (i)
K_e Meat	meat (i), lamb (i), pork (i), swine (i), chicken (i), poultry (i), beef (i), sausage (e), rib (e), pastrami (e), kidney (e), liver (e), ham (e), bacon (e), chorizo (e), salami (e), sheep (e), boeuf (e), oxen (e), kine (e), steak (e), cow (e), brisket (e), veal (e), tenderloin (e), sirloin (e), poulet (e), volaille (e), hot dog (h), hamburgers (h), meatballs (h), burgers (h), goat (h), cattle v, turkey (h), pig (h)
K_e Cereals	wheat (i), atta (i), starch (i), farina (i), bran (i), ethanol (i), biofuel (i), rice (i), corn (i), maize (i), ravioli (e), barley (e), scotch (e), whisky (h), oat (h), bread (h), flour (h), gluten (h), pasta (h), noodles (h), beer (h)
K_e Oil	coconut oil (i), corn oil (i), olive oil (i), palm oil (i),peanut oil (i), sunflower oil (i), rapeseed oil (i), safflower oil (i),soybean oi (i), sunflower oil (i), soybeans (i), soya (i), soy sauce (i), soja (i)
K_e Sugar	sugar (i), sugarcane (i), syrup (e), energy drink (e), cola (e), chocolate (e), nestle (e), cookies (h), cupcakes (h)
K_e Dairy	dairy (i), egg (i), milk (i), kefir (e) , butter (e), yogurt (e), quark (e), mozzarella (e), cheddar (e), parmesan (e), buttermilk (e), ricotta (e), feta (e), romano (e), provolone (e), colby (e), edam (e), eggnog (e), pimento (e), cheshire (e), roquefort (e), icecream (h), milkshake (h), cheese (h), cream (h)
K_e Other	meal (i), meals (i), food (i), foods (i), fish (i) , prawn (i), seafood (i), salmon (i), tea (i), coffee (i), dinner (h), lunch (h), breakfast (h), dish (h), cuisine (h)

Table 2.1: A summary of the evolution of our Food Lexicon

2.1.2 Factors Lexicion

Text describing factors lexicon goes here........

Chapter 3

Filtering

3.1 Feature Definition

From our basic food lexicon we proceeded to extract features that we could use to predict the price and the global food security index. The FAO measures food security based on four dimensions namely *Access, Availability, Stability* and *Utilisation.* Where *Access* mostly captures the supply of food, *Availability* is concerned with the affordability of the basic goods. *Utilisation* captures the nutritional value of the food and lastly *Stability* is a measure of the other three dimensions over time. For food security objectives to be realised, all four dimensions must be fulfilled simultaneously [?].

To model food security we focus our work on those four dimension namely *Access, Availability, Utilisation* and *Stability.* Together those predictor categories build the set C_p. Attempts have been made to capture Availability by the UN [?]. Bellow we will describe an algorithm that builds on the UN's idea namely selecting tweets that match a certain pattern of keyword categories.

We define the predictor category *Access* by looking for tweets containing price as a keyword as in [?] but improve the recall by including synonyms of price that appear in the same context. *Availability* was defined in similar fashion by matching keywords that appear in the context of food availability. Unlike [?] we don't measure food Utilisation by observing the exact diet but capture the people's food needs. Lastly as a measure of *Stability* we focused our attention on economic stability. Keywords in the context of poverty were selected to match this predictor category. Our framework for capturing keywords that describe C_p is noted bellow.

3.1.1 Food in Context

In this section we try to comprehend which words are associated with the above mentioned categories in C_p. More specifically what words are represented in the context of food supply, food price, food needs and food poverty. To achieve this we need a plausible methodology for representing the meaning of a word. The reason why we analyse the context of a word is to identify new words that have a similar meaning or given the same context express the same thing. The later is concerned with identifying synonyms where as the former looks at contextual similarity. For example let's look at the word *"mold"* and *"available"*. Those two words seem unrelate but given the context of food the express the same thing. Namely an abundance of food. Through the role of the context they posses elements of items similarity but by themselves they would never be considered words with similar meaning. It's important to stress that they are not similar because they occur frequently locally but because the occur frequently in similar sentential context. Burgess et al. [**?**] argues that a simple local co-occurence analysis misses to capture a lot of relationships. For example the word street and road are basically synonyms however the seldomly locally co-occur. They do, however occur in the same contexts. This observation motivated us to deviate from the commonly used co-occurence analysis an take a step further to improve the precision of our filtering framework. We now proceed to describe the algorithm Hyperspace analogues language (HAL) [**?**] used to identify contextual similar words for our four categories.

The algorithm

For each word in some corpus HAL records the frequencies that other words in the corpus tend to occur before and after the word within a given window size. This is achieved by storing a vector for each word with the number of co-occurences of every other word in the corpus. Hence if our corpus contains N different words the resulting HAL space would be an N x N square matrix of co-occurences. For each appearance of a word in the corpus, these co-occurnence vectors are updated. It is important to note that only the context before the word is recorded because the context after the word will appear in the column in the matrix that corresponds to that word. For each co-occurence HAL applies a scoring function. Words that appear closer receive an inversely proportionated score to its distance. To illustrate the idea Table 3.1 gives an example of a simple sentence *"The horse raced past the barn fell."* with a sliding window of five.

Following the creation of the matrix we take both the column and row vector of a word, thus including both the preceding and following contexts of the words, and concatenate them. The result is a meaning vector that now can be compared. This can be simply done by using some distance measure. In this paper we use the cosine similarity.

	Barn	Horse	Past	Raced	The
Barn		2	4	3	6
Fell	5	1	3	2	4
Horse					5
Past		4		5	3
Raced		5			4
The		3	5	4	2

Table 3.1: Toy example of HAL

Let's consider the first row. *"The"* precedes *"Barn"* twice. Once within a distance of five and the other time it directly precedes the word *"Barn"*. Hence that cell receives a score of five for the proximate one and a score of one for the word further away resulting in a final score of six.

Our approach

We use a large text corpus of around 23860931 words. As a source we used a random sample of 10 % from our food related tweets. Our corpus of food related tweets has a number of appealing properties as it covers a large vocabulary centered around food. Unlike most corpras that represent formal business reports or specialised dictionaries our food corpus represents everyday speech. This gives us a closer proximation on how people would talk in the context of our predictor categories.

The vocabulary of the HAL model contains 14084 words. The initial set of words in our corpus was filtered only to contain those words that appear at leat 100 times. Words occurring infrequent were discarded as well as stop words and punctuations. We shall refer to this set of words as F_c. Using the words $w \in F_c$ we produced a 14084 by 14084 matrix wit the co-occurences within a window size of five. Since vector similarity measures are sensitive to the magnitude of the vectors we normalized all the vectors to a constant length. Once the HAL space was created we performed the following steps to retrieve the desired keywords for our four categories.

1.) $\forall k \in K_e$ choose the keyword k with the highest occurrence. Let's call it k_{max}

2.) $\forall w \in F_c$ perfomre a similarity meassure with k_{max}

3.) Retrieve the 500 most similar words and handselect one word for each element of C_p (e.g. since we have four categories the result should be four words)

4.) For each of those handselected words apply HAL and compare it $\forall\ w \in F_c$

5.) For each predictor category retrieve the 500 most similar words and manually select relevant keywords.

The high-level intuition of this procedure is as follows. The first step will give us the most prominent food term. This is most likely going to be something general such as the keyword *"Food"*. Step 2 and 3 will allow us the identify the most contextual similar keywords for each category. So the keyword is retrieved that is most likely used to describe supply in the context of food. in step 4 and 5 we aim to retrieve similar words that could describe supply but maybe appear more frequently in different contexts. In other words we aim to find synonyms here.

Result

Show retrieved keywords

3.1.2 Filtering Implemenation

Following the creation of the categories we used polarities to model the price variation. For example, the category *"price"* has two polarities: *"high"* and *"low"*. The following word list belongs to *"high"* polarity:

'high', 'expensive', 'costly', 'pricey', 'overpriced',
and these are words from *"low"* list:

'low', 'low-cost', 'cheap', 'low-budget', 'dirt-cheap', 'bargain'.

Likewise, a category "supply" has "high" polarity:

'available', 'full', 'enough', 'sustain', 'access', 'convenient',
and "low":

'run-out', 'empty', 'depleted', 'rotting'.

The dictionary with a total of 4 categories (each having at least two polarity word lists) was built (*"price", "poverty", "needs", "supply"*), let's call it D.

Then, for each tweet a feature vector is built, representing the amount of words from each category and polarity. Several cases have to be taken into account. First of all, a word may be not in its base form ("price" -¿ "prices", "increase" -¿ "increasing"), which will prevent an incoming word from matching one from D. Therefore, we use stemming technique to reduce each word to its stem (or root) form. Another problem is misspelled words ("increase" -¿ "incrase", "incraese"), and for tweets it happens more than usual due to widespread use of mobile devices with tiny keyboards. Our solution to this problem is covered in the next section.

Here is the overview of predictor category extraction algorithms we implemented:

Preprocessing: For each relevant word in D a stem is computed using the Lancaster stemming method, and the stem is added to reverse index RI, which maps a stem to a tuple: (category, polarity).

function get_category(w):

```
Compute a stem $s$ from $w$
Check if $s$ is present in $RI$.
if yes then
    return the corresponding tuple.
else
    ask spell checker for a suggestion
    is suggestion stem returned?
    if yes then
        return the corresponding tuple from $RI$
    else
        return None;
    end
end
```

On a high level, every tweet is split into words, and then each word (token) is passed through 'get_category' function. But here's another problem we face using this approach: each relevant word we encounter may have a negation word (particle) before it, which subverts the meaning: "increases" -¿ "doesn't increase", "have food" -¿ "have no food", etc. To deal with this problem, we employed the following method: we added a special 'negation' category with a list of negation words ("not", "haven't", "won't", etc.), and if there is a word with "negative" category before some relative word (to be more precise, within some constant distance from it, say 2), then we change the polarity of relative word's category. For example, if a word is from category "poverty" and has "high" polarity (like "starving"), then negative category word right before it (such as "aren't") will turn the polarity to "low".

Tweets spell checking

People often do not pay much attention about the proper word spelling while communicating over the Internet and using social networks, but misspelled words may introduce mistakes in processing pipeline and significantly reduce the amount of filtered tweets, since the relevant, but incorrectly written word might not be recognized by the algorithm.

Several spell checking libraries were checked (Aspell and Enchant to name a few), but their 'suggest' method lacked both performance (several seconds to generate a suggestion for thousands words, which is very slow) and flexibility (it's not possible to specify the number of generated suggestions, as well as a threshold, such as maximal edit distance between words). Therefore, we decided to use a simple approach which involved computing edit distances between a given word and words from predictor categories dictionary (D).
For each given word w we compute its stem s and then edit distance (also known as Levenshtein distance) to each word (stem) from D. It can be done really fast thanks to C extensions of *python-levenshtein* module.
After that, we choose the stem with minimal edit distance (using heap to store the correspondence between distances and words and to speed up selection of the minimal one), and check if the resulting number of "errors" (which is equal to distance) is excusable for the length of word w. For example, we don't allow errors for words of length 5 or less, only one error is allowed for lengths from 6 to 8, etc. If everything is alright, then the suggestion is returned, otherwise the word is discarded.

The approach proved to be fast and tweakable, and was successfully used for tweets processing.

Chapter 4

Analysis

4.1 General Stats

We see in the bellow Figure **4.1** that the distribution of the number of tweets per user follows a power law where a lot of individuals have sent only a few tweets about the subject and only a small number of users have sent a large amount of tweets.

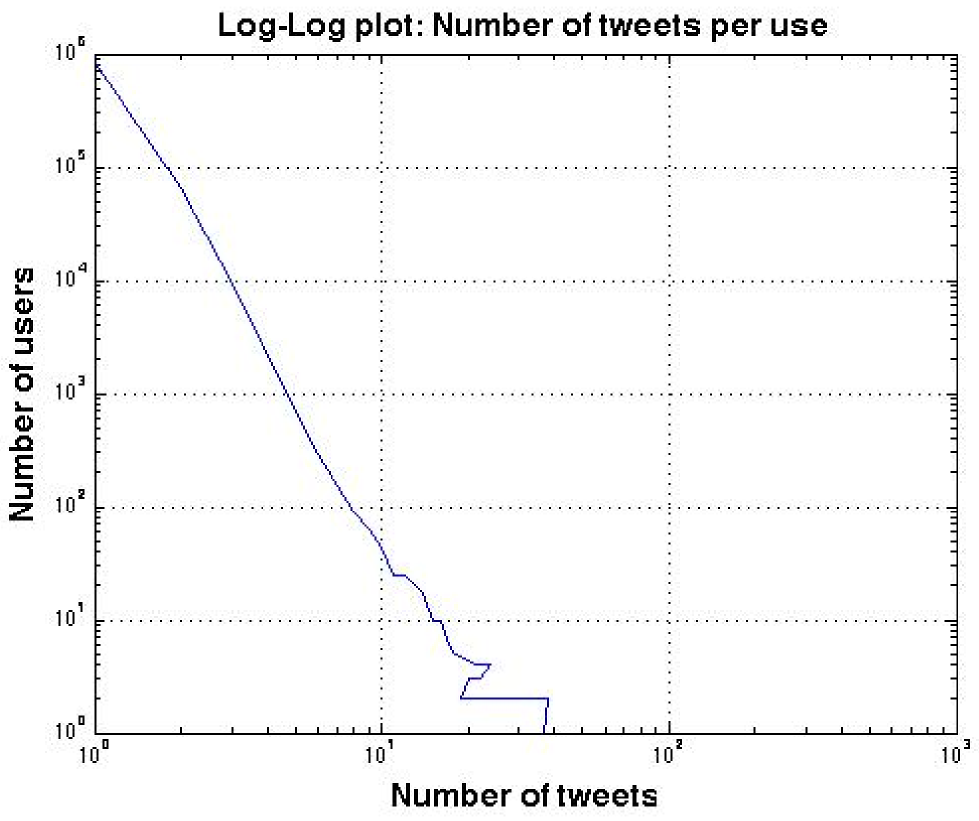


Figure 4.1: Distribution of Tweets per User

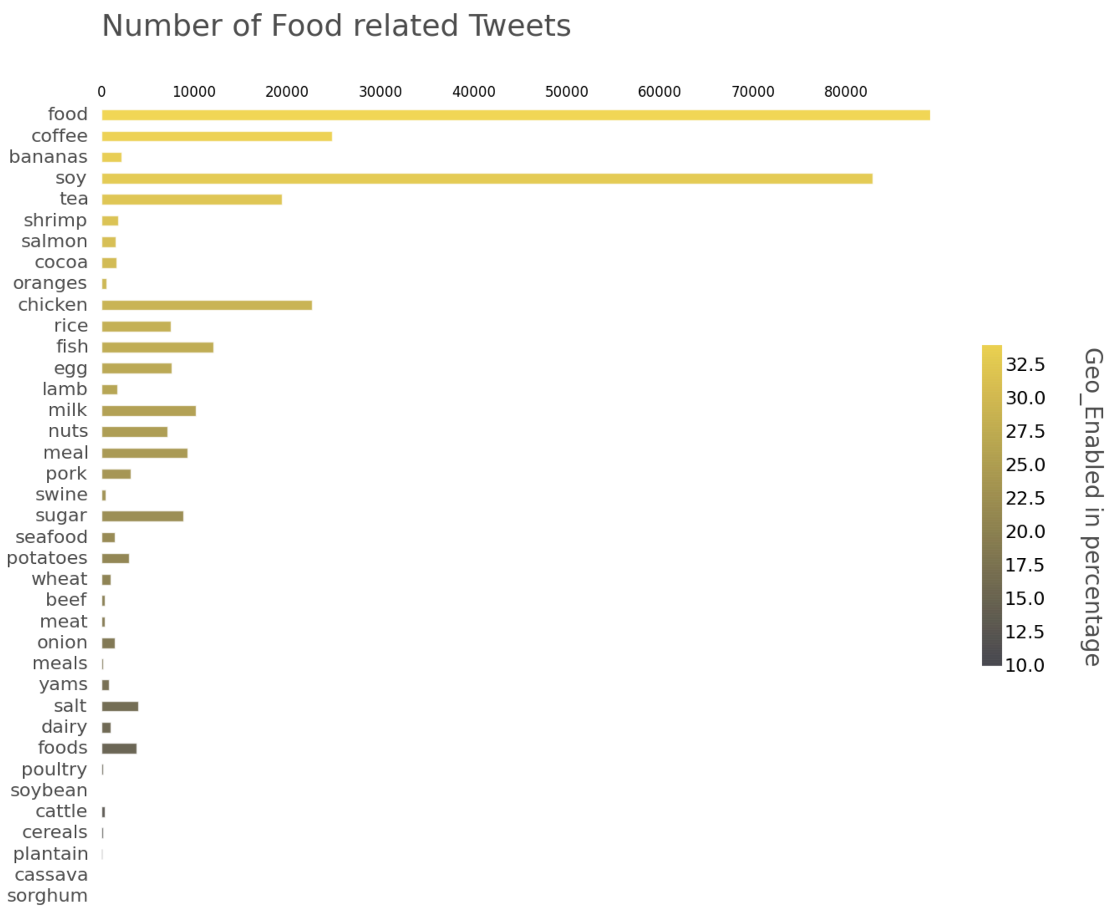


Figure 4.2: Keyword Distribution

Appendix

4.2 Processing and Storage

To facilitate the storage and processing of this large amount of data we used an AMD supercomputer with 64 cores. Inspired by the map reduce paradigm we split the dataset into 64 parts and assigned each to a single core. To efficiently use the hardware resources we manually controlled for the memory assignment using numactl. As illustrated in **4.3** eight cores directly access one out of eight memory blocks. Each dataset was filtered in parallel reducing the 64 dataset to two lexicons.

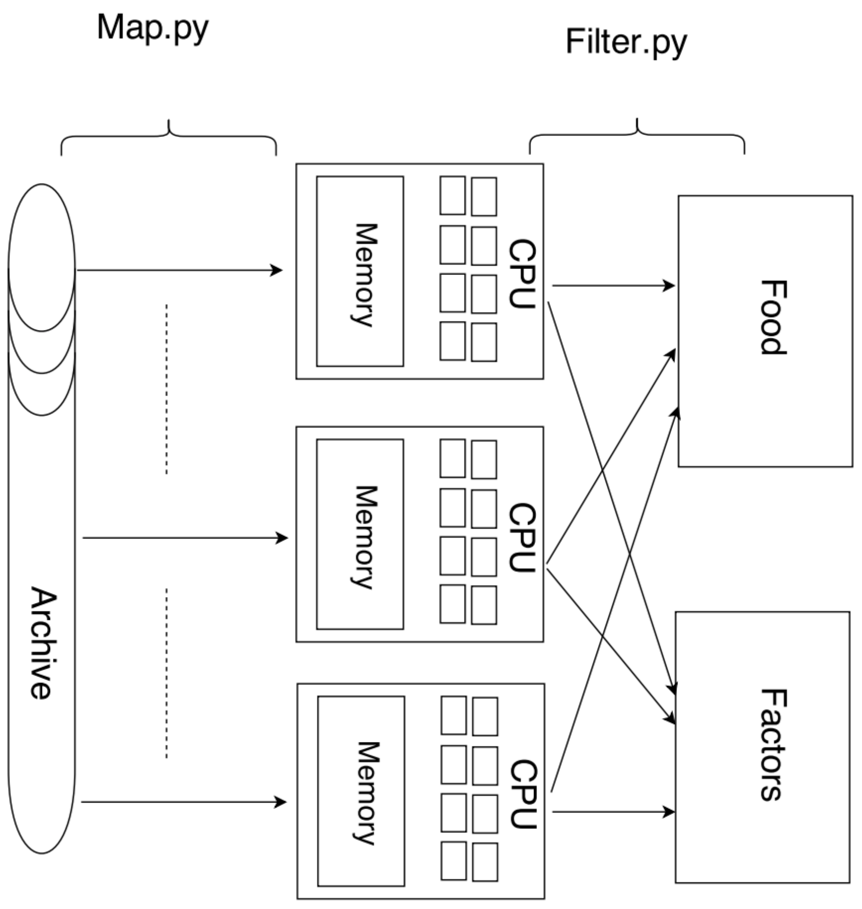


Figure 4.3: Dada Processing

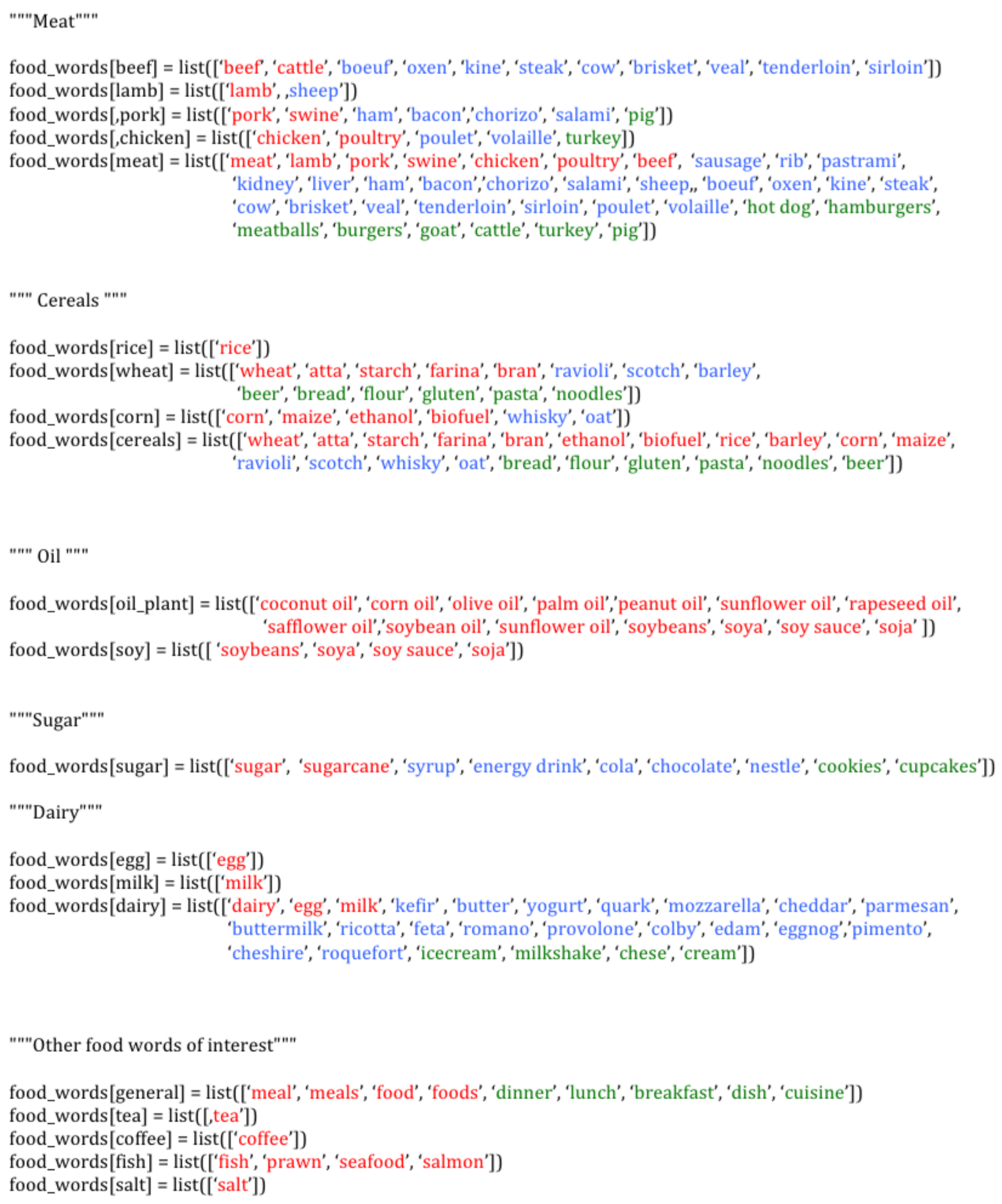
```
"""Initial List"""

food_words[food]  = list([‘meal’,’ meals ‘, ‘food’,’foods’, ‘wheat’, ‘rice’, ‘maize’ ‘carley’, ‘soybean’, ‘soy’, ‘meat’
                    ‘beef’,’cattle’, ‘chicken’, ‘poultry’, ‘lamb’, ‘swine’, ‘pork’, ‘fish’, ‘seafood’, ‘shrimp’, ‘salmon’,
                    ‘sugar’, ‘bananas’, ‘oranges’, ‘coffee’, ‘cocoa’, ‘tea’, ‘milk’, ‘yams’, ‘cassava’, ‘potatoes’, ‘sorghum’,
                    ‘plantain’, ‘nuts’, ‘onion’, ‘salt’, ‘egg’, ‘dairy’, ‘cereals’])
```

Figure 4.4: Keywords: Food Lexicion

Initial Keywords:

Extended Keywords:

Hal Keywords:



```
"""Meat"""

food_words[beef] = list([‘beef’, ‘cattle’, ‘boeuf’, ‘oxen’, ‘kine’, ‘steak’, ‘cow’, ‘brisket’, ‘veal’, ‘tenderloin’, ‘sirloin’])
food_words[lamb] = list([‘lamb’, ,sheep’])
food_words[,pork] = list([‘pork’, ‘swine’, ‘ham’, ‘bacon’,’chorizo’, ‘salami’, ‘pig’])
food_words[,chicken] = list([‘chicken’, ‘poultry’, ‘poulet’, ‘volaille’, turkey])
food_words[meat] = list([‘meat’, ‘lamb’, ‘pork’, ‘swine’, ‘chicken’, ‘poultry’, ‘beef’,  ‘sausage’, ‘rib’, ‘pastrami’,
                         ‘kidney’, ‘liver’, ‘ham’, ‘bacon’,’chorizo’, ‘salami’, ‘sheep,, ‘boeuf’, ‘oxen’, ‘kine’, ‘steak’,
                         ‘cow’, ‘brisket’, ‘veal’, ‘tenderloin’, ‘sirloin’, ‘poulet’, ‘volaille’, ‘hot dog’, ‘hamburgers’,
                         ‘meatballs’, ‘burgers’, ‘goat’, ‘cattle’, ‘turkey’, ‘pig’])


""" Cereals """

food_words[rice] = list([‘rice’])
food_words[wheat] = list([‘wheat’, ‘atta’, ‘starch’, ‘farina’, ‘bran’, ‘ravioli’, ‘scotch’, ‘barley’,
                          ‘beer’, ‘bread’, ‘flour’, ‘gluten’, ‘pasta’, ‘noodles’])
food_words[corn] = list([‘corn’, ‘maize’, ‘ethanol’, ‘biofuel’, ‘whisky’, ‘oat’])
food_words[cereals] = list([‘wheat’, ‘atta’, ‘starch’, ‘farina’, ‘bran’, ‘ethanol’, ‘biofuel’, ‘rice’, ‘barley’, ‘corn’, ‘maize’,
                         ‘ravioli’, ‘scotch’, ‘whisky’, ‘oat’, ‘bread’, ‘flour’, ‘gluten’, ‘pasta’, ‘noodles’, ‘beer’])


""" Oil """

food_words[oil_plant] = list([‘coconut oil’, ‘corn oil’, ‘olive oil’, ‘palm oil’,’peanut oil’, ‘sunflower oil’, ‘rapeseed oil’,
                              ‘safflower oil’,’soybean oil’, ‘sunflower oil’, ‘soybeans’, ‘soya’, ‘soy sauce’, ‘soja’ ])
food_words[soy] = list([ ‘soybeans’, ‘soya’, ‘soy sauce’, ‘soja’])


"""Sugar"""

food_words[sugar] = list([‘sugar’,  ‘sugarcane’, ‘syrup’, ‘energy drink’, ‘cola’, ‘chocolate’, ‘nestle’, ‘cookies’, ‘cupcakes’])

"""Dairy"""

food_words[egg] = list([‘egg’])
food_words[milk] = list([‘milk’])
food_words[dairy] = list([‘dairy’, ‘egg’, ‘milk’, ‘kefir’ , ‘butter’, ‘yogurt’, ‘quark’, ‘mozzarella’, ‘cheddar’, ‘parmesan’,
                        ‘buttermilk’, ‘ricotta’, ‘feta’, ‘romano’, ‘provolone’, ‘colby’, ‘edam’, ‘eggnog’,’pimento’,
                        ‘cheshire’, ‘roquefort’, ‘icecream’, ‘milkshake’, ‘chese’, ‘cream’])


"""Other food words of interest"""

food_words[general] = list([‘meal’, ‘meals’, ‘food’, ‘foods’, ‘dinner’, ‘lunch’, ‘breakfast’, ‘dish’, ‘cuisine’])
food_words[tea] = list([,tea’])
food_words[coffee] = list([‘coffee’])
food_words[fish] = list([‘fish’, ‘prawn’, ‘seafood’, ‘salmon’])
food_words[salt] = list([‘salt’])
```

Figure 4.5: Keywords: Food Lexicion