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

Abstract

Acknowledgements

Contents

Chapter 1

Introduction

Appendix

We describe here the relevant information to setup the Planetlab environment and how to properly deploy Hive2Hive.

Deploy Hive2Hive

Chapter 2

Data

2.1 Processing and Storage

For this report we collected 2000 GB of Tweets from the archive over a span of October 2011 - September 2014. 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 **2** eight cores directly access one out of eight memory blocks. Each dataset was filtered in parallel reducing the 64 dataset to two lexicons. One lexicon contains food relevant keywords where as the other contains factors influencing the price and supply of the goods. The filtering process resulted with 60 GB of food relevant tweets and 30 GB tweets of influencing factors. Similar to the initial dataset we split the two lexicons into smaller json objects of 5 MB to allow for list operations.

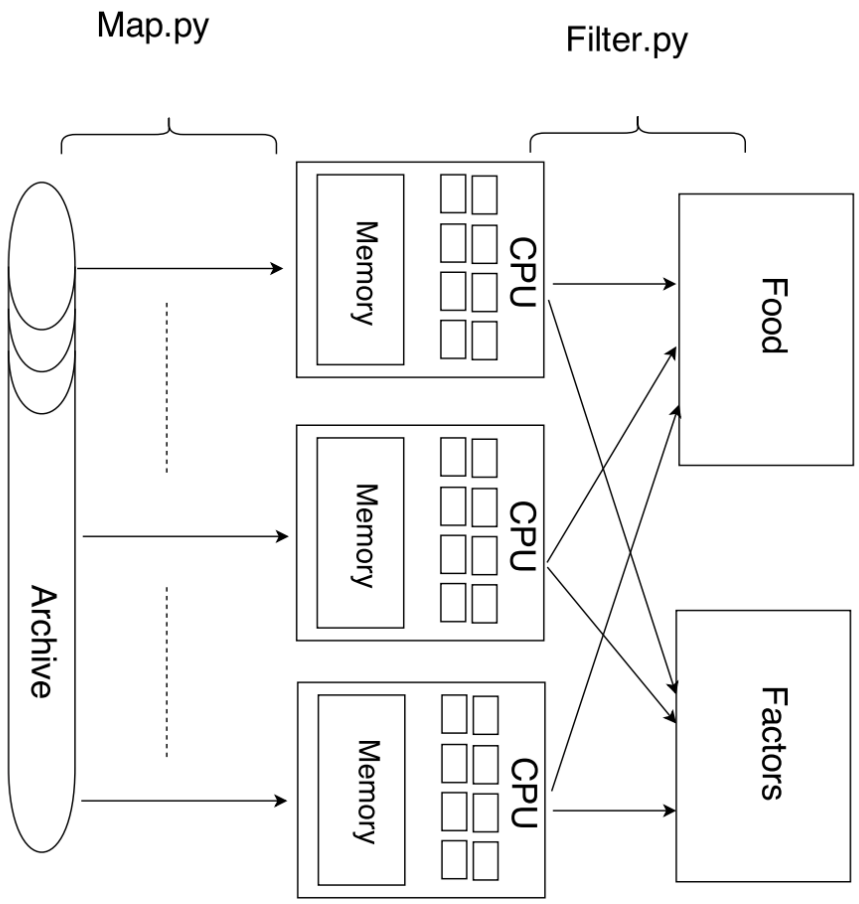

Figure 2.1: Dada Processing

.1 Basic Analysis

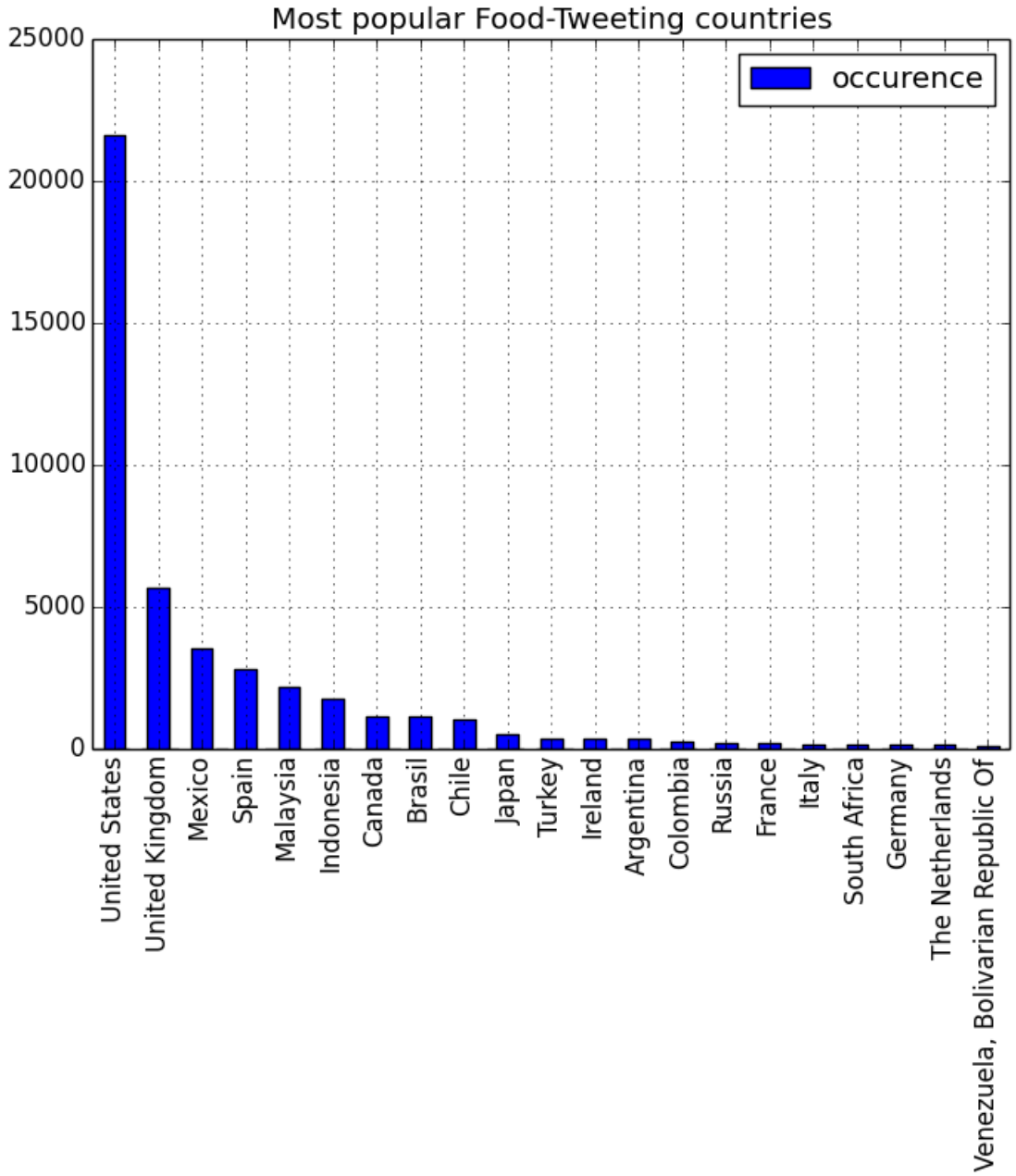

Figure 2: Country Distribution

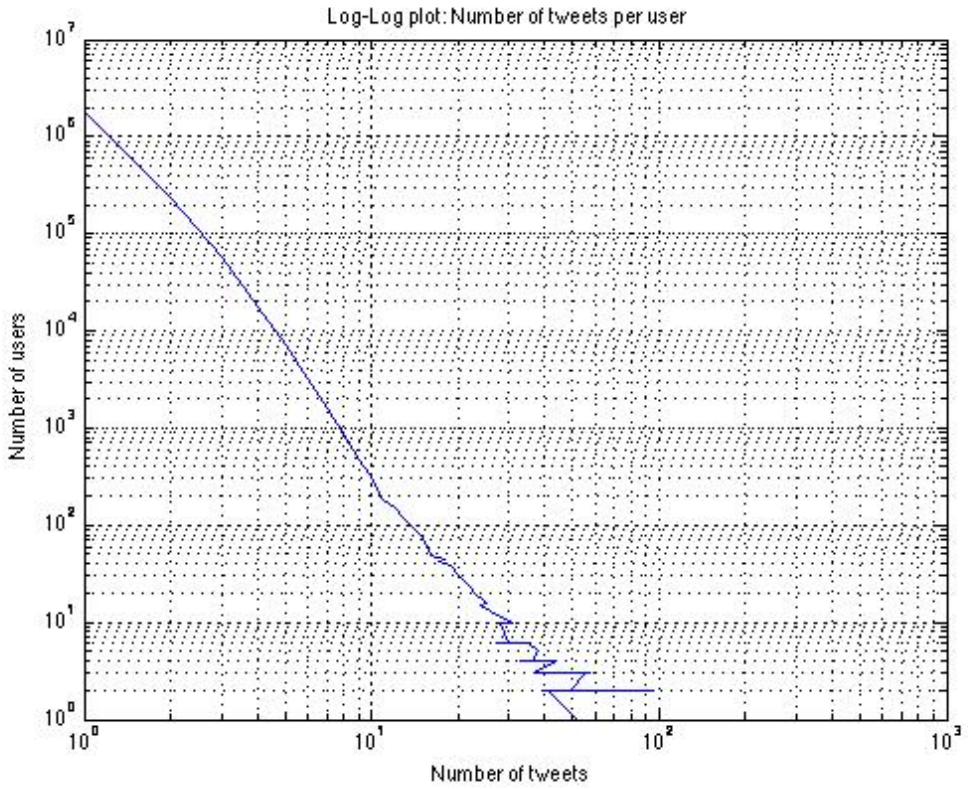

Figure 3: Country Distribution

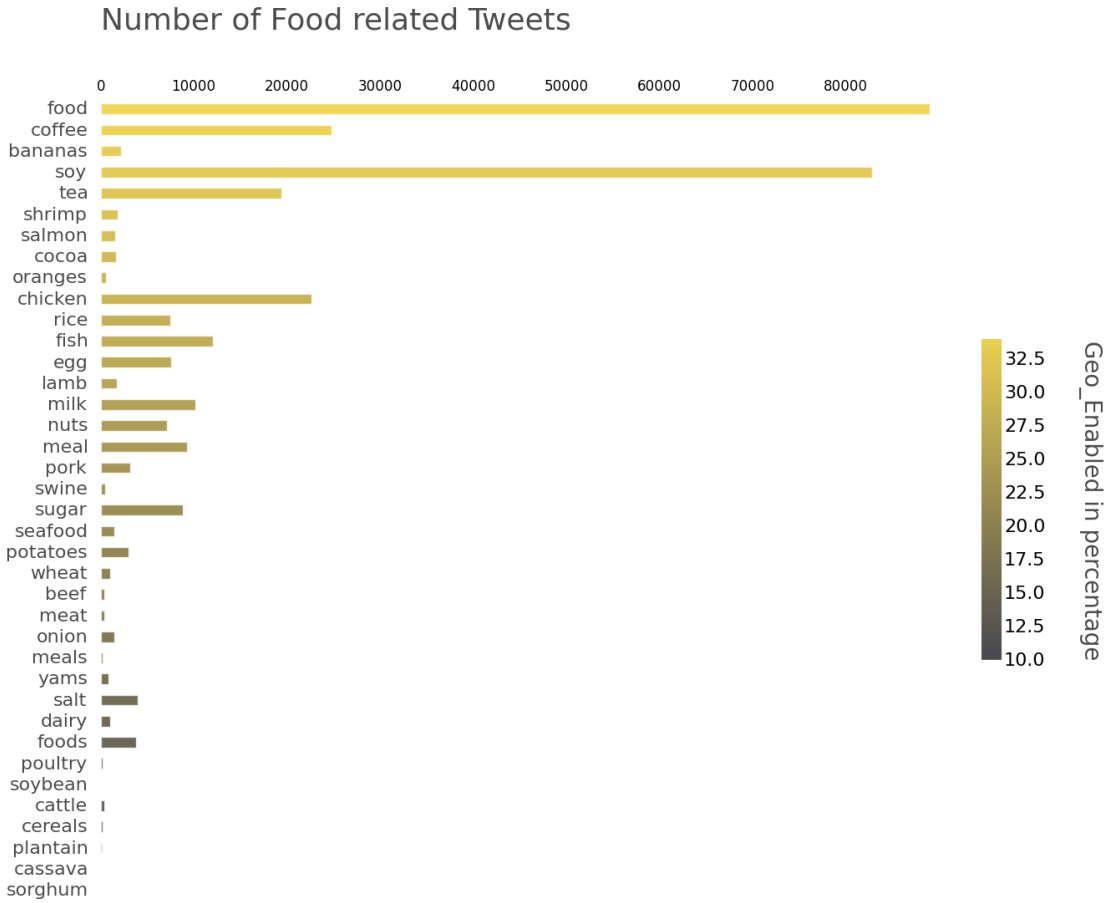

Figure 4: Country Distribution

Extracting predictor categories

First, several "predictor categories" were selected. These categories represent different aspects of price variation, and each category includes several sets of words of different polarities. 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 6 categories (each having at least two polarity word lists) was built (*"predict", "price", "sentiment", "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.