Delft University of Technology

Bachelor Graduation Project

Initial Research Report

UrbanSearch

Authors:
Tom Brunner
Marko Mališ
Gijs Reichert
Piet van Agtmaal

Supervisor:
Claudia Hauff
Clients:
Evert Meijers
Antoine Peris

May 1, 2017

Abstract

It is yet to be discovered how the importance of cities in the global network can be elucidated. In this document, the requirements for a solution to this problem are identified, as well as possible issues that might arise. We explore various methods and select the most appropriate ones that, if combined, satisfy these requirements and tackle the identified issues, in preparation for an attempt at solving the problem.

Keywords: urban, network, data mining, document analysis, filtering

Contents

1 Introduction

Common belief is that agglomeration benefits are key to economic growth [17]. However, it remains uncertain whether it is these benefits that primarily cause economic growth, or an increase in the position of the agglomeration in the global urban network.

The huge amount of textual data generated online and the numerous historic archives are great sources of information on social and economic behaviours. They constitute the bulk of information flowing among each other. Advanced text mining on newspapers and web pages containing city names would allow for a better understanding of the role of information in shaping urban systems. Similar to research efforts in other domains, such as financial trade [18] and sales forecasting [22], the idea is to develop search queries that capture urban-urban interactions. These interactions are retrieved from information corpora through the co-occurrence of geographical names in textual data. An example of such a query on the Google search engine[1] is `"Rotterdam + Amsterdam " OR "Amsterdam + Rotterdam"`, which searches for the co-occurrence of Amsterdam and Rotterdam. We thus answer the following question FIXME: correct?: what approach is best suited for extracting and visualising the global urban network from open data? FIXME: define subquestions?

First, we discuss related work in section 2. Second, we identify the requirements for a solution to the problem and discuss issues that might arise in section 3. Third, we develop a methodology for a framework that satisfies the requirements and tackles the issues in section 4. Last, we conclude in section 5 with the results of our research.

2 Related Work

3 Requirement Analysis

In this section, we first define the design goals. Then, we list the requirements which the FIXME thingy meet. We use the MoSCoW method [2] as a prioritisation technique. Four levels of priority are defined: must have, should have, could have and won't have (also known as would like).

3.1 Design Goals

The high-level design goals for this project have been provided by the client. These serve as a guideline to determine the priority label of the specific requirements, as defined in section 3.2 and are listed here, ordered by priority.

credible The client wants to impugn a widely spread belief. Therefore, the basis on which he does that must be sound.

understandable The results of the FIXME system? should be visually understandable, so it is easy for the client to deduce conclusions from them. Additionally, retrievable numeric data enable the client to further investigate the results outside of the scope of the FIXME system?, should the need arise.

scalable The client has expressed his concerns that restricting the FIXME system? to a set of non-English domains might impair the probability that his research will be published in an acknowledged journal. Allowing for investigating other domains would greatly help the client in a later stadium.

[1] `https://www.google.com`

plugable The client conveyed it might be interesting to let the FIXME system? perform analysis on different data sets without the need of a developer.

3.2 Product Requirements

3.2.1 Must Have

Requirements labelled as "must have" are key to the minimal performance of the FIXME product/whatever. If they are not met, the FIXME: thingy can be considered a failure.

1. General
   (a) Adding city names
   (b) Grouping relations and "zooming" on these relations
2. Search Engine
   (a) Filter results
   (b) Data mining
3. Filtering
   (a) Logic Filters
   (b) Relations Filters
4. Machine Learning
   (a) Types of relations
5. Visualisation
   (a) Statistics of relations? Query relations
   (b) Strength of relations
   (c) Types: ML CBS defined

3.2.2 Should Have

"Should have" requirements are those that greatly improve system performance and/or usability but might not fit in the available development time.

1. General
   (a) Pluggable datasets
2. Machine Learning
   (a) Generalising relations, grouping relations

3.2.3 Could Have

Requirements labelled as "could have" are useful and should be included in the system if time and resources permit.

1. General
   (a) International city names
2. Visualisation
   (a) Front end for the app

3.2.4 Would Like

"Would like" requirements have been agreed upon as not important to deliver within the current time schedule. However, they can be included in future releases.

4 Framework and Tools

In this section, we evaluate what data sources are useful. Additionally, we discuss several methods and tools that can be helpful in storing and ingesting data. Furthermore, we describe numerous methods to filter and classify textual data. Then, we elaborate on different methods to perform queries with. We conclude with an overview of the available visualisation tools that can be used for displaying the results of the analysis.

4.1 Gathering the Data

4.1.1 Common Crawl

Common Crawl [3] is a freely accessible corpus of the pages across the web. Their data are updated and released on a monthly basis. Many researchers have used the data for varying purposes [20] [13] [19]. Since the project requires us to crawl the web (see section FIXME), the corpus is a very suitable candidate for us to work with.

The data of Common Crawl come in three formats[2]:

WARC This is the default and most verbose format. It stores the HTTP-response, information about the request and meta-data on the crawl process itself. The content is stored as HTML-content.

WAT Files of this type contain important meta-data, such as link addresses, about the WARC-records. This meta-data is computed for each of the three types of records (meta-data, request, and response). The textual content of the page is not present in this format.

WET This format only contains extracted plain text. No HTML tags are present in this text. For our purposes, this is the most useful format.

For extracting data from Common Crawl, many open-source libraries are available. Common Crawl's official website refers to `cdx-index-client`[3] as a command line interface to their data. It allows for, among others, specifying which dataset to use, supports multiple output formats (plain text, gzip or JSON) and can run in parallel.

A simple query on the latest index using the online interface[4] yields 1676 pages of 15000 entries each, which are roughly 25 million entries in total. However, there are over 5.5 million registered domain names with top level domain `.nl`[5]. One would expect many more pages to exist with that number of domains. There are several explanations for this, including:

- Common large websites, such as `www.google.nl` and `www.wikipedia.nl` have not been fully indexed by Common Crawl, because their "parents", `www.google.com` and `www.wikipedia.org` have already been indexed almost entirely.
- Not every website allows their pages to be crawled. According to Common Crawl's official website, their bots can be blocked via the common `robots.txt`. Additionally, they honour so-called `no-follow` attributes that prevents the crawler to follow embedded links. Sites that use these features are therefore partially or not at all included in the indices of Common Crawl.

[2] https://gist.github.com/Smerity/e750f0ef0ab9aa366558

[3] https://github.com/ikreymer/cdx-index-client

[4] http://index.commoncrawl.org/CC-MAIN-2017-13-index?url=*.nl&output=json&showNumPages=true

[5] https://www.sidn.nl/a/knowledge-and-development/statistics?language_id=2

4.1.2 Delpher

Delpher [1] is a library consisting of millions of Dutch digitalised newspapers, books and magazines of ages differing from the fifteenth century up until now. Their data are free to use for private purposes. All data can be accessed however, if the organisation doing research explicitly asks for permission to use the data and agrees to Delpher's terms. Delpher could be useful to perform historical analysis and see trends over longer periods of time. This cannot be done using web pages as source, as the Internet is still very young.

Should historical analysis be desired, Delpher is the data source to use. Since their data are texts from reliable sources, there is no noise to be filtered out. Additionally, some valuable information can be extracted, such as area of distribution, cross-location referencing and target audience. These aid in the process of measuring the strengths of relationships.

4.1.3 Other Data Sources

Besides Common Crawl and Delpher, there are a plethora of other sources that might contain valuable information. The most notable are the Statistics Netherlands[6], the NOW Corpus[7] and However, due to time and resource constraints, we have chosen to exclude these from the project. Of course, in future versions, other data sources could be included.

4.2 Data Storage and Ingestion

4.2.1 Elasticsearch

Elasticsearch is an open-source search engine which centrally stores your data [4]. It is a fast and scalable solution that was designed with big data search in mind. According to Kononenko et al. [10] Elasticsearch has some significant advantages in comparison with traditional relational databases. Two of these advantages are scalability and performance.

Scalability According to Elasticsearch [4] their product has no problem with scaling horizontally. It automatically manages indices and queries distributed across a cluster. This is an important feature as it is likely that the amount of data that our solution will use and process will increase and we do not want to keep upgrading the server that contains database, which would be vertical scalability. FIXME: Referentie naar dat we alleen NL doen nu? En daarom scalability nogal belangrijk is

Performance Because Elasticsearch was designed to handle documents and perform full-text search it not surprising it performs well doing this. As we're going to be using the same kind of input data we expect Elasticsearch as the most choice. Kononenko et al. found that while scalability and schema-free documents are common for NoSQL systems, the combination of all three (scalability, agility, and performance) in one system is what makes Elasticsearch stand out from other systems. Following this, we conclude Elasticsearch would be a good choice as a data storage and search platform for our FIXME: solution—product—project.. (We moeten even kiezen welke we aanhouden zodat we dat overal hetzelfde doen.

Downsides A downside of Elasticsearch is that it does not have any form of security out of the box. This means that everyone with the server address could access the data. This is not a problem for CommonCrawl data, as this was already available online anyway. However, when using for example Delpher or other sources you need a license this becomes a problem. Next to that, it would also be possible for anyone to meddle with the data in Elasticsearch making the data unreliable. Elasticsearch provides a Security package for which you unfortunately need a

[6] https://www.cbs.nl/en-gb

[7] http://corpus.byu.edu/now/

paid license. However, to secure Elasticsearch while making it available for users we could use a plugin such as Search Guard[8] or use a special proxy as proposed by Kononenko et al. [10].

Another downside of Elasticsearch is that transactions involving multiple documents[9] are not ACIDic. Where ACID stands for the four properties atomicity, consistency, isolation and durability regarding transactions in database systems [8]. This means that we need to keep concurrency problems in mind and will probably need to enable some locking to prevent these concurrency problems when performing transactions on multiple documents.

4.2.2 Hadoop

Because we are designing a { *application*} that will use and parse a lot of data, it will be useful to use distributed computing. Although we only use the .nl data { *Totale grootte van data noemen?*} from CommonCrawl for this TODO: project (see section 4.1.1) we could still benefit from distributed computing, especially with the future of the { *application*} in mind. Therefore we want to use Apache Hadoop, which is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models [7]. With this open source software we could distribute the computations from a single server across many more devices, thereby speeding up the process.

4.2.3 MrJob

4.2.4 Neo4j

Neo4j is a highly scalable native graph database that leverages data relationships as first-class entities [14], enabling enterprises of any size to connect their data and use the relationships to improve their businesses. It is the single highly scalable, fast and ACID compliant (see section 4.2.1 for a short explanation) graph database available. Additionally, it is free to use for non-commercial use. To illustrate how scalable Neo4j is, consider that very large companies such as ebay, Cisco, Walmart, HP and LinkedIn[10] use it in their mission-critical systems. Holzschuher and Peinl compared the performance of Neo4j to the more classic and commonly used NoSQL databases and found out that the more natural representation of relationships resulted in significant performance increase gains [9].

There are some specific aspects of Neo4j that make it a very suitable candidate for the TODO: project. These are:

properties Any entity in the Neo4j graph can be given properties (key-value pairs) containing information about the entity. Properties are primarily meant to provide additional information and are less suitable to be queried on. As an example, a city can have a number of inhabitants and districts attached to it as a property.

labels Nodes can be tagged with a label, describing their roles in the network. These annotations are especially useful to filter the data set on one or more categories. For example, a city can be labelled as ”capital” to be able to distinguish between regular and capital cities.

relations Nodes can be connected using relationships. These are always directed, typed and named and can have properties. Using these properties, one can control how the graph is traversed. For example, if a path (relationship) is to be avoided unless absolutely necessary, the relation can be given a high cost. To give importance to some relationship, one could also assign a strength score to it. Since relationships are handled efficiently by Neo4j, nodes can have any number of relationships linked to it without decreasing performance. For our purposes, a relation could comprise the strength of the relationship between two cities (nodes).

[8] http://floragunn.com/searchguard/

[9] https://www.elastic.co/guide/en/elasticsearch/guide/current/concurrency-solutions.html

[10] https://neo4j.com/customers/

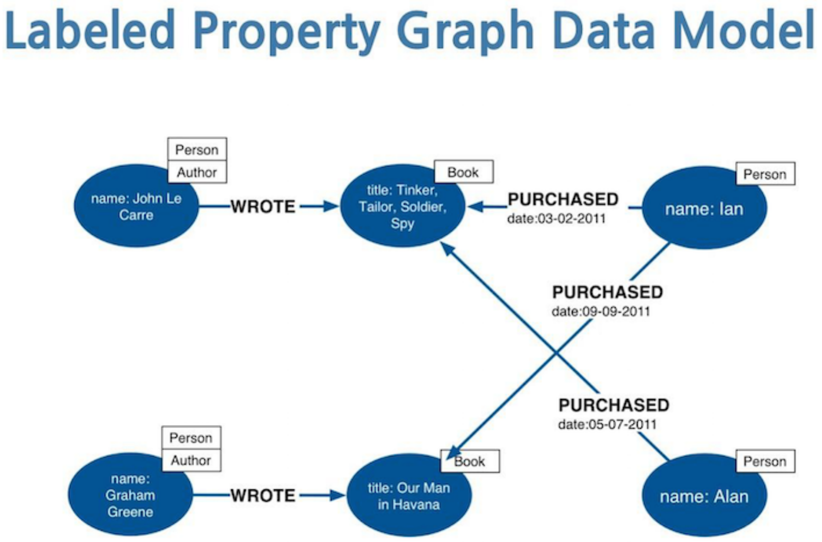


Figure 1: The Neo4j model

The Neo4j model can be depicted as shown in figure 1. It consists of nodes, relationships (edges), properties (within the nodes) and labels (rectangular blocks above the nodes).

Besides the aforementioned useful properties of Neo4j, we can put the graph to good use for visualising the global urban network. By adding a location property to a city, we can directly map nodes and relations to a geographical map. Most importantly, we can store indices of text files that mention the city as properties of nodes. That way, we are able to generate a subset of files that we can analyse for calculating the strength of the relationship between the nodes.

4.3 Selecting relevant data

4.4 Classifying

After the websites are stored in Neo4J we want to classify each website according to subject: for example economy, politics and migration. Classifying is done in two steps: choosing the classes and actual classifying.

Classes can be chosen manually, however, you can also apply certain techniques to atomise results. The advantage of this that you will get unbiased results, the disadvantage is that it takes more time to make these results and it may also cause classes to be chosen which the user can't use, for example when using Google Trends [6] results as "AFC Ajax", "Aeroflot" and "Eric Dane" may be found. The same problem occurs when searching for sub-classes.

For the classifying itself there are also several approaches. A standard approach is to use only the text from the websites and removing all other data. The big advantage is that it costs much less storage. Information in images may be lost however. Since we have to go over millions of pages, storage is a big issue for us therefor we choose this approach.

Text based classification itself can be split in classification on structured and unstructured text. Text is structured when sentences are used, meaning grammar is used. In the case of structured texts a word may say something about the next word in the sentence, therefor other techniques can be used (for example n-grams) then in case of unstructured texts.Because the text from websites can be structured as well as unstructured, we will most likely use unstructured approaches.

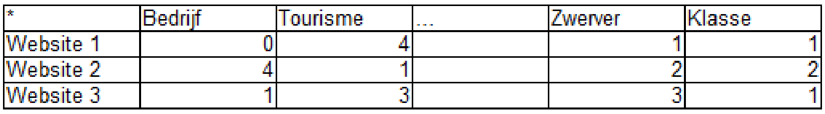
4.4.1 Machine Learning for classification of unstructured text

Text based machine learning for unstructured texts is done using the 'bag of words'. This model counts how often each word is used. There are 3 libraries available which contain most steps needed. There is scikit-learn [11] and TensorFlow [21] for Python and Weka [15] for Java. Since we write our program in python, and TensorFlow is only about neural networks, we choose to use scikit-learn. The machine learning works in 4 steps:

1. **Creating a feature extractor**
   Given text from a website, the "features" from this text are returned. Features are the words that occur in the text and the number of occurrences. Before this data is extracted stop words (the, is, at etc) are removed and the rest of the words are stemmed meaning all words will be changed to their root-forms (features - feature, controlled - control). The program Snowball [16] and NLKT [5] (which uses the snowball version) have a dutch implementation for this, although it might need to be improved a bit.

   Afterwards, to prepare the features for the machine learning algorithm. We need to give each feature a numeric id. Count each of these tokens. And we need to normalise the tokens. For this scikit-learn provides algorithms.

2. **Manually labelling**
   For each of the classes (e.g. business, tourism, art etc) we select a few websites we know fit to that class. From these websites all the words will be extracted and their occurrence will be counted. Possibly some normalisation functions are applied to get better values. We call these values the weight for each word for each class. From this we create a two dimensional array with in the rows each of the websites and in the columns all different words and one extra for the class. We fill the fields with the weights or a zero if the words don't occur.



*	Bedrijf	Tourisme	...	Zwerver	Klasse
Website 1	0	4		1	1
Website 2	4	1		2	2
Website 3	1	3		3	1

3. **Generating a classifier**
   The array is fed to a learning algorithms. This will generate a classifier. There are a multitude of classifiers importable from Scikit-learn and TensorFlow. For choosing the classifier we make use of the Microsoft Azure Machine Learning Test Sheet [12]. Several factors should be taken into account when choosing an algorithm. These are:

   (a) Accuracy - How well the algorithm separates the websites.

   (b) Training Time - How long it takes to train the algorithm.

   (c) Linearity - Linear regression assumes data trends follow a straight line. This is trade-off between accuracy and speed.

   (d) Number of Parameters - Adjustable parameters increase the flexibility of the algorithms. This is a trade-off between training time and accuracy.

   (e) Number of Features - A large number of features can make some algorithms unfeasibly long. Especially text data (what we are using!) has a large number features. Support Vector Machines are especially well suited in this case.

   (f) Special Cases - Some learning algorithms make particular assumptions about the data or the results.

   For textual data especially support vector machines are recommended, so it is most likely we will choose that machine learning algorithm. Depending on if we have time we might do some tests before making our decision however.

4. **Entering new examples**
   When a new (unlabelled) example (website) comes - extract the features and feed it to your classifier - it will tell you what it thinks it is (and usually - what is the probability the classifier is correct). Afterwards the classifier can be updated to include new features extracted from the example.

4.5 Domain specific search engine

4.6 Visualisation

TODO: Refer to Neo4j graph model & properties

4.6.1 Connection between cities

4.6.2 The Strength of these connections

5 Conclusion

References

[1] Koninklijke Bibliotheek. Delpher - boeken kranten tijdschriften. `http://delpher.nl/`. Accessed: 2017-04-26.

[2] Dai Clegg and Richard Barker. *Case method fast-track: a RAD approach*. Addison-Wesley Longman Publishing Co., Inc., 1994.

[3] Common Crawl. Common crawl. `https://commoncrawl.org/`, 2017. Accessed: 2017-04-25.

[4] Elasticsearch. Elasticsearch. `https://www.elastic.co/products/elasticsearch`. Accessed: 2017-04-26.

[5] Alex Rudnick et al. Nlkt. `http://www.nltk.org/api/nltk.stem.html`, 2017.

[6] Google. Google trends. `https://trends.google.com/trends/home/all/NL`, 2017. Accessed: 2017-05-01.

[7] Apache Hadoop. Hadoop, open-source software for reliable, scalable, distributed computing. `http://hadoop.apache.org/`. Accessed: 2017-04-28.

[8] Theo Haerder and Andreas Reuter. Principles of transaction-oriented database recovery. *ACM Computing Surveys (CSUR)*, 15(4):287–317, 1983.

[9] Florian Holzschuher and René Peinl. Performance of graph query languages: comparison of cypher, gremlin and native access in neo4j. In *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, pages 195–204. ACM, 2013.

[10] Oleksii Kononenko, Olga Baysal, Reid Holmes, and Michael W. Godfrey. Mining modern repositories with elasticsearch. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 328–331, New York, NY, USA, 2014. ACM.

[11] Scikit learn developers. Scikit-learn. `http://scikit-learn.org/stable/index.html`, 2017. Accessed: 2017-04-26].

[12] Microsoft. Microsoft azure machine learning algorithm cheat sheet. `https://docs.microsoft.com/en-us/azure/machine-learning/machine-learning-algorithm-choice`, 2017. Accessed: 2017-04-28.

[13] Hannes Mühleisen and Christian Bizer. Web data commons-extracting structured data from two large web corpora. *LDOW*, 937:133–145, 2012.

[14] Neo4j. Neo4j, the world's leading graph database. https://www.neo4j.com. Accessed: 2017-04-26.

[15] University of Waikato. Weka. http://www.cs.waikato.ac.nz/ml/weka/, 2017. Accessed: 2017-04-26.

[16] Richard Boulton Olly Betts. Dnowball. http://snowball.tartarus.org/algorithms/dutch/stemmer.html, 2017.

[17] Michael E Porter. Location, competition, and economic development: Local clusters in a global economy. *Economic development quarterly*, 14(1):15–34, 2000.

[18] Tobias Preis, Helen Susannah Moat, and H Eugene Stanley. Quantifying trading behavior in financial markets using google trends. *Nature: scientific reports*, 2013.

[19] Sameer Singh, Amarnag Subramanya, Fernando Pereira, and Andrew McCallum. Wikilinks: A large-scale cross-document coreference corpus labeled via links to wikipedia. *University of Massachusetts, Amherst, Tech. Rep. UM-CS-2012-015*, 2012.

[20] Jason R Smith, Herve Saint-Amand, Magdalena Plamada, Philipp Koehn, Chris Callison-Burch, and Adam Lopez. Dirt cheap web-scale parallel text from the common crawl. In *ACL (1)*, pages 1374–1383, 2013.

[21] Google Brain Team. Tensorflow. https://www.tensorflow.org/, 2017. Accessed: 2017-04-28].

[22] Lynn Wu and Erik Brynjolfsson. The future of prediction: How google searches foreshadow housing prices and sales. In *Economic analysis of the digital economy*, pages 89–118. University of Chicago Press, 2014.