Keywords Alternatives in lists of text

Rik van Outersterp
University of Twente
PO Box 217
Enschede, The Netherlands
r.m.vanoutersterp@student.utwente.nl

Han van der Veen
University of Twente
PO Box 217
Enschede, The Netherlands
h.vanderveen-1@student.utwente.nl

ABSTRACT

Using common text structure and Big Data, alternatives of subjects are extracted from many webpages. A common structure of lists in a text is used to determine these alternatives. These lists are combined into pairs and counted using MapReduce paradigm. The results are presented into a web application which can be searched on.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval; I.5.0 [**Pattern Recognition**]: General

Keywords

Big Data, text analysis, text structure

1. INTRODUCTION

"What if we can see immediately the alternatives for everything in the world?"

Nowadays advertisers have a great influence on people. Using advertisements in media such as the Internet, television, and radio they try to convince people to buying certain products. However, advertisements can make the market seem smaller than it actually is. Not every company has the ability to advertise their product on the same scale as other companies. That pushes people in certain directions, away from alternatives for which is advertised in a different way (e.g. another medium), in lesser extent, or not at all.

Our idea is that for every product in the world there is an alternative and that this alternative could be better than the original product. What we would like to do is to give people the option to find those alternatives and to compare them against each other and the original product. As advertisement is not always done for alternatives it can be difficult to find them. However, big data analysis can solve this problem.

In our research we choose to search only those alternatives in lists, because we think that will be sufficient to solve this problem. We use Big Data analysis on CommonCrawl data to retrieve alternatives for everything that is mentioned in the lists.

1.1 Related work

In a new moving science called Big Data, there are a lot of new techniques. Where Dean and Ghemawat [1] started with introducing MapReduce, new applications of MapReduce emerge. An overview of these techniques were given by Sakr et al. [3]. In our paper we use techniques to extract the keywords which are related. Another method is to look for keywords in sentences. Zhu [5] proposes a method for spotting keywords in large text, using a graph of the relations of words and the place of the words in the text to gather words into groups, and looking for keywords in those groups. This approach can also be applied when filtering the keywords. However, Big Data assumes that a lot of data will filter out the noise.

Text structure is important in our approach. Delgado [2] uses structure of a text for extracting information of a text. For example, an episode is often mentioned with a format consisting the number of the season and the number of the episode (e.g. s03e12). Delgado does not use lists in texts, however he uses common structures in texts.

In our approach we do not filter any of the results, but there may be a lot of the same which looks alike. We can filter them out by doing (as a post-filter) duplicate detection. Zhang [4] uses partials of a sentence of duplicate detecting. Duplicate detection can also be used to clean the resultset.

2. METHOD

Our method is to analyse the common structure of lists in text assuming that they are relate to each other. The analysis of the alternatives is done through looking for lists that are mentioned in the pages itself. A list has a certain structure. It can be a sentence such as *a, b or c* or *x, y and z*. We use this common structure to find alternatives, since we assume that the listed elements are alternatives to each other.

We pack the crossproduct of the sentence *x,y and z* into tuples. The tuples will be of the form *(x,y), (y,z) and (z,x)*. Our mapper outputs the tuples with a concatenation of x and y. For example, $x||y$ is the emitted key with count 1. Algorithm 1 is the mapper in pseudocode. We group on each pair. Our reducer does a sum on the occurences of the tuple. We assume that when a tuple has a high count, it is

Algorithm 1 Mapper of Linker

```
1: class Mapper
2:     method Map(docid $n$, doc $d$)
3:         $tuples \leftarrow$ pairs of $d$
4:         for all tuple $(a, b) \in tuples$ do
5:             Emit(string concat$(a, ||, b)$, count 1)
6:         end for
7:     end method
8: end class
```

most likely to be related.

2.1 Text processing

We extract our text with a simple regular expression, which can be found in the appendix B. The list is splitted on *comma*, *and* or *or*. Then the crossproduct of the pairs is emitted, taking in account that only one relation exists, such that *(a,b)* is only emitted and not *(b,a)*. This is achieved using a case-insensitive compare on a and b. Where a is lower than b, than a is before b and vise versa. These pairs are emitted into the MapReduce framework.

The MapReduce framework counts the results and the results are filtered with a small script to remove the results which have a count less than 25, which means that there the pair is reasonbly mentioned on the internet. These remaining results are loaded into a database through the parser. The database provides the basic interface for presenting the results.

2.2 Expectations

The expected result of the method is that on well known terms (e.g. Volkswagen) the results will be good. This is because that these terms are mentioned more often on the web. Furthermore, big terms are probably more related to other big terms. Also because these terms are mentioned more ofthen than other terms. The top results of each term will probably be related to other top results.

3. RESULTS

Our program has been tested on a sample of the dataset. The proposed test set for the Norvig Award was not there anymore so we tested it on a subset of the CommonCrawl output (about 5GB of data)[1].

3.1 Test set

We created 50 mappers and 1 reducer. The job was done in 31 minutes. Some of the jobs failed and that causes the long running time. If we use that for estimating what it takes using the full dataset, it takes about 100 days. However, on more data the cluster is faster, because every job can be runned directly after the other, so we think the job will be done in about 3 days. Our top results of our sample-subset is found in the appendix A.

3.2 Full set

We had some errors with trying our solution on the fullset, but we did manage to run it on a set of 2,7TB of webdata. The full job took about 7 hours minutes and our result was about 97 million relational tuples[2].

[1]with -maxfiles 50 switch
[2]report of this job can be found on github

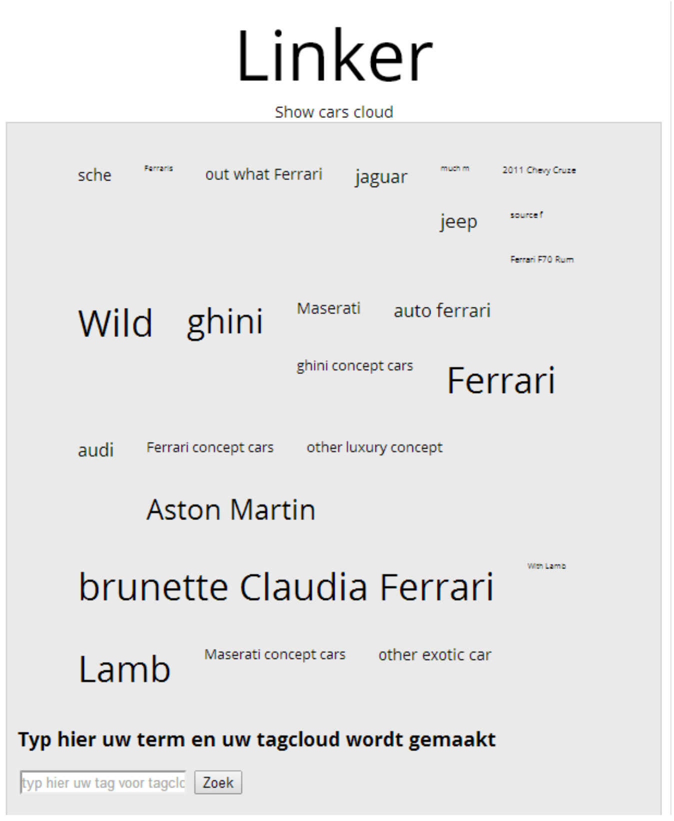

Figure 1: Search cloud for *Ferrari*

3.3 Final results

We created a tool for parsing the tuples and putting these in a database. The database can then be read through a simple web interface. In total almost 97 million combinations were found, which is equal to a database table of about 1.1 million rows. Note that we only have combinations with an amount of at least 25 times. The top three results in terms of combinations are: *hit enter - search* and *type - search*. All these combinations were found over 1.2 million times.

The alternatives are sometimes very good (i.e. the result is really an alternative for what we were searching for), but sometimes is really bad. The tool provides some basic relations, but not just alternatives like expected beforehand. For example, when searching on *Ferrari* some other cars are mentioned, but also the phrase *source f* which we could find no link with *Ferrari* for.

3.4 Killer result

The tuples are all collected in a database and thanks to the MapReduce framework the tuples are already sorted alphabetically on the first word of the tuple (although with a SQL-query you can always sort the results in the way you prefer). The *"killer result"* of our project is that of a simple webpage on which can be searched to find alternatives for the searched word. It should be clear that these alternatives are retrieved from the database.

The alternatives that are found for a search word are presented in a so called *word cloud* (see Figure 3.1). Alternatives with a higher count (i.e. high amount of combinations of the searched word and the alternative) are displayed with

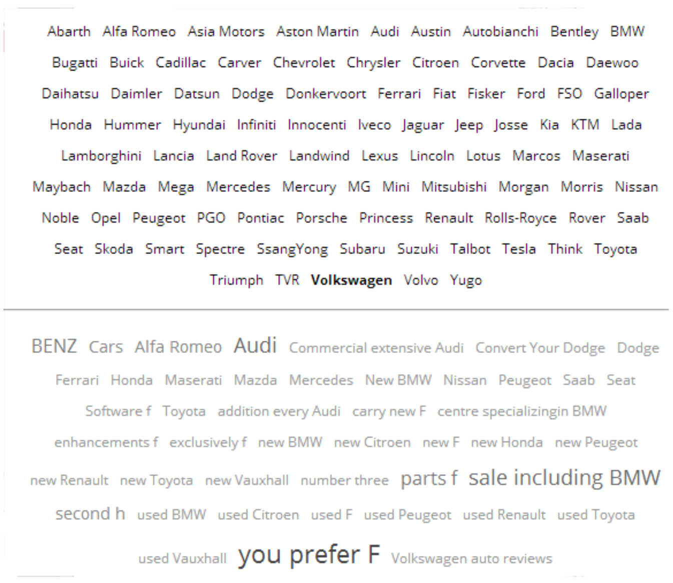
Abarth Alfa Romeo Asia Motors Aston Martin Audi Austin Autobianchi Bentley BMW
Bugatti Buick Cadillac Carver Chevrolet Chrysler Citroen Corvette Dacia Daewoo
Daihatsu Daimler Datsun Dodge Donkervoort Ferrari Fiat Fisker Ford FSO Galloper
Honda Hummer Hyundai Infiniti Innocenti Iveco Jaguar Jeep Josse Kia KTM Lada
Lamborghini Lancia Land Rover Landwind Lexus Lincoln Lotus Marcos Maserati
Maybach Mazda Mega Mercedes Mercury MG Mini Mitsubishi Morgan Morris Nissan
Noble Opel Peugeot PGO Pontiac Porsche Princess Renault Rolls-Royce Rover Saab
Seat Skoda Smart Spectre SsangYong Subaru Suzuki Talbot Tesla Think Toyota
Triumph TVR Volkswagen Volvo Yugo
BENZ Cars Alfa Romeo Audi Commercial extensive Audi Convert Your Dodge Dodge
Ferrari Honda Maserati Mazda Mercedes New BMW Nissan Peugeot Saab Seat
Software f Toyota addition every Audi carry new F centre specializingin BMW
enhancements f exclusively f new BMW new Citroen new F new Honda new Peugeot
new Renault new Toyota new Vauxhall number three parts f sale including BMW
second h used BMW used Citroen used F used Peugeot used Renault used Toyota
used Vauxhall you prefer F Volkswagen auto reviews

Figure 2: Word cloud for *Volkswagen*

a bigger font size and those with a lower count are displayed with a smaller font size.

Besides the search there is a special function available which only shows cars and their word clouds (see Figure 3.4). This is done since the prototype of our system only consisted of cars and it gives a nice impression of what our system is capable of. Selecting a car results in a word cloud of the alternatives for that car, similar to the search function.

At the top of the webpage the top three tuples with the most references are listed, together with the total amount of references. These results are mentioned in details in the Results section.

4. DISCUSSION

The script have parsed many alternatives. The idea was to give an alternative for everything. We sort of did that, using the alternatives given in lists by users in html data. In our database we have pairs of relations. So, a relation with a high count is found many times in the html pages, thus likely to be related.

There was a small bug that the matched sentence was splitted on *or*, *comma* or *and*, thus the sentence *parts for Audi* was splitted into *parts f* and *Audi*. Which is not correct. This was later fixed in the program. However, we did not manage to apply the fixed program on the full dataset, because we did not manage to run it again on the cluster.

A shortcoming of our solution is that there are many duplicates. The duplicates generates a lot noise and needs to filtered out. With proper duplicate detection the dataset can be cleaned. Our solution is limited to short lists. An improvement could be to focus on the whole sentence. And apply the context of the sentence.

An improvement could be focusing on larger lists. With larger lists there is more information, thus more relations to be found. Extracting the essence of larger lists could result in better alternatives.

5. REFERENCES

[1] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[2] M. Delgado, M. J. Martín-Bautista, D. Sánchez, and M. Vila. Mining text data: special features and patterns. In *Pattern Detection and Discovery*, pages 140–153. Springer, 2002.

[3] S. Sakr, A. Liu, and A. G. Fayoumi. The family of mapreduce and large scale data processing systems. *arXiv preprint arXiv:1302.2966*, 2013.

[4] Q. Zhang, Y. Zhang, H. Yu, and X. Huang. Efficient partial-duplicate detection based on sequence matching. In *SIGIR*, volume 10, pages 675–682, 2010.

[5] X. Zhu. Spotting keywords and sensing topic changes in speech. *2012 IEEE Symposium on Computational Intelligence for Security and Defence Applications, CISDA 2012*, 2012.

APPENDIX

A. SAMPLE SET RESULT

First column is word a, second column is word b and after the semicolin is the occurence count of the tuple (a,b).

```
broadcast||published;472
greeting cards||000 brochures;473
passw||Login with username;561
Bank Owned Homes||Columbus Ohio F;1630
Columbus Ohio F||eclosure;1630
eclosure||Columbus Ohio F;1630
```

B. REGEX

```
([a-z0-9\-\040]{3,25}[,])
([a-z0-9_\-\040]{3,25}[,]\s*){0,8}
([a-z0-9_\-\040]{3,25}(\s+(and|or)\s+)
[a-z0-9_\-\040]{3,25})
```

C. DATABASE STRUCTURE

```
CREATE TABLE `dataset` (
  `a` varchar(255),
  `b` varchar(255),
  `c` int(11),
  INDEX(a), INDEX(b), INDEX(c)
) ENGINE=InnoDB DEFAULT;
```

D. FILES AND DATA

Our project is public available on *https://github.com/haneev/linker*. On github project page can be found how to install the application and how to use it.