1. Introduction

"Word Association Game" - BigData Project

For this project, we want to analyse the WARC web crawl by looking at the game, so-called *"word association game"*. In this game, one comes up with a starting word and keeps naming words which come to mind with the last said word. For example:

$$\text{BigData} \to \text{Database} \to \text{DuckDB} \to \text{Duck} \to \text{Duckling} \to \text{Yellow}$$

There, generally, is not really a point to the game or some way of winning. There is no sense of competition really, but it is still interesting to analyse the game and see if we can learn some things about it.

1.1 Word Association

Of course, any word can in theory, be related to any other word, but for this project, we try to restrict ourselves to words which can be related in some meaningful way. We will say words are *associated* if they are discussed on the same webpage. So, if webpage 1 discusses topics A and B and webpage 2 discusses B and C, we have:

$$\text{A} \to \text{B} \to \text{C}$$

Staying with the previous example, we could imagine some webpage discussing different *databases* and has a short section on *DuckDB*, and some other webpage which discusses *DuckDB* and has a short introduction about what DuckDB has to do with *Ducks*.

1.2 The General Goal

With this method of associating topics, we will aim to build a graph where words/topics are vertices and

vertices are connected if they are mutually discussed on some webpage.

With this graph, we can ask ourselves some interesting questions: *Is the graph connected?*, *What is the longest minimal path?*, *What are some big clusters?*, or if there is an abundance of computing power *Is there a Hamiltonian Path/Cycle?*

2. The Problem in Detail

We have to figure out a number of things. First, we have to determine how to find the topics of a page. We propose to do a word count on the `WET` files (we don't want all words to be related to "the" or "`<script>`"). We then pick the 4 or 5 most common words and take those to be the topics of the page (the exact number should be fine-tuned by hand). The problem here, of course, is that we will only see the topics 'is' and 'be'. Moreover, we will also get that "pencil" and "pencils" as distinct topics.

A possible solution would be to tokenise and lemmatise the `WET` files. The downside of that approach would be that "university" and "universe" become the same lemma, which is probably not what we want.

However, considering the scope of this project, we will still use basic tokenisation and lemmatisation. Followed by removing everything which is not a noun. This is a little "harsh", however, implementing a better custom NLP tokeniser and lemmatiser solution seems a bit out of scope. Moreover, in any case, we can simply say that any two words which have their "lemma-version" in common can probably be linked in the word association game (with as argument word similarity).

With the methods outlined above, we will convert all web crawl files to a short list of topics. We then convert this to a graph. Since this will probably be quite a sparse graph, we will store the graph as a list of tuples, not as a matrix. When we have this graph, we can answer the fun and interesting questions.

3. Reducing WET to Topics

As described above, this consists of two parts. First, we "tokenise" the WET file, with a custom tokenisation method. Second, we have to do a word count and get the n most common words.

We first work in a simple Zeppelin notebook on a WET file I plucked from the internet[1]. It is around ≈128MB zipped and should be more than sufficient for this stage of the project.

3.1 Spark NLP

To tokenise and lemmatise all the text we make use of Spark NLP. Since we are working with a WET file, we can easily filter on `warcType==conversion` to get all text from web pages.

Then we setup a basic NLP pipeline:

$$\text{Tokeniser} \Rightarrow \text{Normaliser} \Rightarrow \text{posTagger} \Rightarrow \text{Lemmatiser}$$

So we split up the page in tokens, then we "normalise": we remove anything which is not in the alphabet and convert everything to lowercase. After that, we add a "pos-tag", we tag each word with the *part-of-speech*, so mark words with *noun* (NN), *verb* (VB), or *adjective* (JJ)[2], for example. Finally, the

[1] For the specific file name see end of report

[2] The POS tag list contains 36 different tags, including different forms of adjectives and nouns, but we do not consider those here. We only want simple singular nouns.

lemmatiser converts tokens to “lemmas”, which reduces each word to its *stem*, so “university” becomes “univers”.

When this is done we can simply filter on nouns, done by filtering lemmas on the `NN`-tag.

3.2 Word Count

Now we need to do a word count on the nouns per page. A simple test on one page — using a simple groupBy — showed that picking the five most frequently occurring nouns works quite well.

4. WebCrawl Topics to Graph

With the topics extracted from the `WET` files, we can now create a Word-Associated Graph. As described in the previous section.

In the graph, we want each topic to be a vertex, and we want an edge between two topics if there exists a page discussing both. To be able to remove noise later, we will not just make an edge but keep track of the number of times a topic pair occurs. This will allow us later to compute some *association intensity* measure and filter out the lowest edges.

We realise this graph in two simple steps.

1. For each page, add each pair of topics on that page to a DF (disregard (x, x))
2. `groupBy` the pairs and call `count` to get the intensity of each edge.

For step 1, we can either make a UDF which directly implements this, but we choose to make use `join` and filter out (a, a) and (b, a) where $b >_{lex} a$, this provides us with the same result. The downside is that we at some point handle about twice the number of data points, but we feel this would still outperform

a UDF.

4.1 Running at scale

With this code all working, we want to start analysing the graph. To make this interesting, we need a graph of some considerable size. Therefore, we now run the code for generating the graph on the `WET` file on the cluster and save the resulting graph. Since we only take 5 topics from each page we can be quite certain that storing the graph will not create memory problems.

We save the graph as a parquet file with columns:

```
+----------------------+------------+
| edge                 | occurences |
+----------------------+------------+
| (bigdata, databases) | 120        |
| (databases, duckDB)  | 25         |
| (duckDB, ducks)      | 18         |
| ...                  | ...        |
```

The resulting dataset has over 300K rows and is about 3.18MB.

We can illustrate the graph created from the first 1000 rows. (For the purposes of illustration this graph was already filtered of noise)

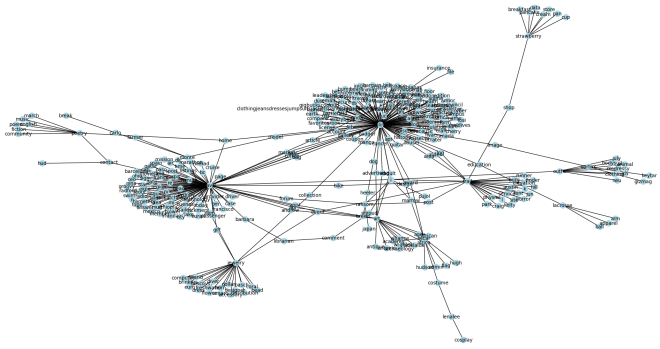

Figure 1: Graph constructed from the first 1000 rows

We can now move on to do some analyses of this graph, and answer some question about the *Word Association Game*.

5. Analysing the Graph

Now that we have this rather small file, we can create a Zeppelin to analyse the graph. We can then make this Zeppelin into a Scala program, combine the two programs and run an analysis on larger data.

5.1 Ranking Edges

We currently store the number of occurrences. This is not really a good representation. We need a way to properly handle different imbalances and strong or weak connections between topics which have diverging "popularity". To do this, we use the Pointwise Mutual Information method:

$$\text{PMI}(A,B) = \frac{\mathbb{P}((A,B))}{\mathbb{P}(A)\mathbb{P}(B)} = \frac{\#(A,B)\cdot N}{\#A\cdot \#B}$$

The following table allows us to see that this performs quite well. First, we consider a set of four topics, each with some count of occurences in the crawled pages.

	A	B	C	D
#	6000	7000	89	10^5

Here we give the mutual occurences between some of the topics and their PMI score.

	(A,B)	(A,C)	(B,C)	(B,D)	(C,D)
#	5000	87	5	320	89
PMI	1.129	1.265	-0.042	-1.220	0.053

So, for pairs (A,B) and (A,C) that have good scores, which is wanted, (A,B) clearly has a strong correlation. For (A,C) we see that a page about C almost always covers A and compared to the total number

of pages, the difference in number of occurrences between A and C should not be too big of a problem. In the case for (C, D) we see the same type of relation, but here we have a very low score, which is something we want since almost all pages cover D so we cant really say that there is a link between C and D. It is as if saying there is strong link between couches and Wikipedia when you are analysing Wikipedia data. (B, C) is clearly a weak relation and this is represented in the PMI score. Finally, (B, D) similarly shows a weak association for the number of occurrences of B and D.

The analyses of the graph will be done using GraphX. The results we will describe are performed on the graph derived from the `WET` file. With this PMI score we could filter out edges with a PMI score less than $\leq \mu - 1.96 \cdot \sigma$, which nicely cleans up the graph. The resulting graph has 40663 vertices and 670782 edges[3].

5.2 Connected Components

Determining whether the graph is connected is a bit more subtle than simply checking if it is connected. In figure 1, we only show the largest component, but we leave out many smaller components. To figure out if the graph is connected we compute the size of each of the connected components. First of all, we filter out any component of size ≤ 5, since these are one-page clusters and are not important. When we look at the component sizes we get the following results:

[3] These are actually directed edges, so in theory there are only $\frac{670782}{2} = 335391$ edges, however, for data processing that does not matter.

name	#vertices
art	39962
keskuu	17
slanina	13
lieky	9
…	…

The “name” of a connected component is simply the first word in the component in lexicographical ordering. This does not signify some topic, but it does give us some insight. As far as I am aware, the topics in the other clusters don’t really make sense. These results lead me to believe that if we scale up the amount of data, all other clusters either get swallowed in the big cluster or get filtered out due to a low PMI.

5.3 Shortest Paths

We start by a simple case, what are the words hardest to associate with “online”. If we look at the words with the longest shortest paths to “online” we get the following list.

topic	distance
hodnotou	6
nnn	6
islice	6
printf	6
warianty	5
…	…

Some of these words definitely do not make sense. The appearance of “printf” makes me believe that there definitely is not enough data here. Of course, “printf” and “online” might not be directly related, but it probably should not have a shared *last* place with the topics *least* associated with “online”.

Now, what two words are the words to associate

with each other? To answer this, we need to find the two words with the longest shortest path. We can not simply compute all shortest paths between each vertex, since this would take too much time. However, since the graph is (mainly) one big connected component, we can iteratively search for the longest shortest path and each iteration take the furthest point as your starting point to find the longest shortest path when the distance does not increase anymore. We can do this with a simple while-loop (we do add a max iterations condition to ensure this won't take 20 days before it is finished). The only requirement to make this work is that we need a starting word which is in the connected component for sure. Here, an obvious pick would be something like "online".[4]

The two words which are hardest to associate are:

$$\text{xac} \longleftrightarrow \text{hodnotou}$$

With the shortest distance of 9. I have never heard of either of these words. When doing these kinds of measurements it is a lot easier to get these strange words. We could hope that increasing the amount of data would help filter these words are we could have a higher threshold for PMI score.

But what is the association between these two words, then? GraphX does not natively support a method for explicitly giving this path. We did implement a custom BFS for this purpose, however, due to time constraints, we decided not to run this custom BFS

[4] Even more obvious would have been "art" but I did not know that art would be included before running the program. Betting that "online" would be a major topic on the internet was quite a safe bet.

function. It is included in Zeppelin, and there you may test it on the provided graph.

5.4 Clusters

In the figure 1 we can see some "hot spots" in the graph. These are the topics with many associated topics. We would like to see what topics create these "clusters". In the word association game, this might be very useful to know. You might not find the shortest path, but hopping between these main topics might be an easy way to go from topic A to B. In figure 1 we see, for example, that going via *online* or *travel* is a good option.

Finding these "hot spots" is easily done by finding the nodes with the highest number of edges. This can be done with GraphX, and this gives the following list of vertices.

topic	degrees
post	7138
news	6048
page	4784
photo	4592
comment	4548
review	4522
video	4498
home	4486
email	4480
view	4154

This actually make sense! I think that all these words are very prominent topics on the internet and it is not strange to believe that these are the hottest spots. This actually works very well because this is not so dependent on the specific pages you include, these are general trends online, while a word as "hodnotou" will probably not be included if you pick a dif-

ferent set of pages.

What I find interesting here is that “post” is the vertex with the highest degree. Since this is a `WET` file and since the degree is only about 7000 and not 70.000 I believe this is not a consequence of some `HTML POST` requests. Of course, a post on social media is a prominant topic online, however, it might also be the case that the NLP pipeline labels post in “I post a picture” as a noun, which could also contribute to the popularity very much.

I also find it interesting that “news” is so high and that besides the first two all have about the same number of degrees, but I will (for the sake of brevity) stop interpreting the results for now.

6. Scaling Up

For now, all analysis has been done on one single `WET` file of about 128MB zipped, which gave a graph of about 40K vertices and 670K edges. We did not have time to actually deploy the scaled-up version with a larger amount of data, since the NLP pipeline is very slow and processing the zipped 128MB `WET` file already took 9+ hours on silver level. Since the cluster does not contain `WET` file we had to adopt the code to handle the `WARC` files. We have included that in the Zeppelin and the code provided. We will here describe the challenges and solutions.

First, the `WARC` files provided were not all in English. There were two approaches to solving this: adding a language filter in the NLP pipeline, or filtering on the HTML language tag. We choose to do the latter. The downside of this is that this HTML tag is a rather unreliable method of detecting the language of a page, however in this particular application, I believe that

we will filter out all non-English topics, as long as the input is large enough and the HTML language tag is reasonably accurate. The advantage of the HTML tag method is that it is considerably faster then the NLP solution. Therefore we choose the HTML tag option. A compromise would be to combine the two solutions, since most English pages have an English tag and non-English pages sometimes also have an English tag, to get a high ranking on Google. However, I think this would not really affect the results, since most of the input will be English and thus topics from other languages will probably be filtered out.

Secondly, we had to filter out the body from the `WARC` file. This was easily done using JSOUP. I combined this with the HTML language tag filtering to create a fast and easy method to get a column of plain text.

Lastly, `WET` files already filter out so-called stop-words. To handle this I expanded the NLP pipeline to include this filter. This was not too big of a problem, since I already needed to include a stopword filter to filter out all lemmas of size ≤ 2.

7. Final Remarks

So what can we conclude about the *Word Association Game*? Well, for starters, we are pretty sure that there exists some path of association among all words. We also see that this path is rather short — the longest shortest path is 9 on a set of > 40K words. From the list of most popular topics we might also conclude that looking at online pages might be biased to certain topics; “post” probably is not *the* most popular topic in everyday non-online conversation.

The analysis was run on the `WET` file:

```
CC-MAIN-20130516134005-00099-ip-10-60-113-184.ec2.internal.warc.wet.gz
```

The code ran on the cluster for analysis on the `WET` file is included by the name: `REDBAD_WET_ANALYSIS_s1098110.scala`. The adapted code for analysing `WARC` files is included by the name: `REDBAD_WARC_ANALYSIS_s1098110.scala`. Finally, a Zeppelin file is included which walks through all parts of the code.

DEPENDENCIES Running the Zeppelin file requires adding:

```
com.johnsnowlabs.nlp:spark-nlp_2.12:5.5.3,
org.apache.spark:spark-graphx_2.12:3.5.0
```

To the comma-separated list: `spark.jars.packages`.

For assembling the code, an updated `build.sbt` file is required which is also included in the submission, named: `build.sbt`.