Career Timelines Extracted from Wikipedia

Firas Dib
ada10fdi@student.lu.se

Simon Lindberg
ada10sli@student.lu.se

Abstract

We have created a system that will, given a persons Wikipedia article page, create a timeline of their work experience. We do this by connecting persons to professions through analysis of part-of-speech and dependency relations. We did not achieve any worthwhile results parsing the dates which is why we have not taken dates into account when calculating recall, precision or F-score. The process has a recall of 74% and a precision of 95%, resulting in a F-score of 83%.

1 Introduction

We have created a system that will take a given persons Wikipedia page as an input and output a career timeline for the person this article is referencing. This is done by utilizing several tools to parse and extract information from natural language and then analysing the data. A practical use of this software could be to expand the Wikidata profiles.

Although not attempted, a more generalized software could built based on ours that would not only look for peoples relations to occupations, but rather on any category and link them through another set of verbs.

We chose to use Wikipedia as the source because it contains a large quantity of quality resources with great tools to build upon. As of now we are strictly dependent on Wikipedia, although we only ever use the text itself so the source could easily be replaced.

2 How we did it

A number of tools have been used to make this project possible. We are going to walk through each one used, what role it played for our project and how it has been utilized.

2.1 Wikidata

Wikidata is a free knowledge database from the wikimedia foundation. We have used it to create a baseline of professions to work from. The baseline list of professions is obtained through first creating a graph where the nodes are all entries in Wikidata and the edges are the properties **Instance of** and **Subclass of**. We then loop through all the nodes, checking if any node leads to the `Occupation`-node. If it does, it is determined to be a profession and is added to the list. We chose `Occupation` as the root node since `Profession`, `Job` and `Labour` are all an **Instance of** `Occupation` according to Wikidata.

The list is created in a pre-processing stage, independent of the actual article parsing.

2.2 Wikiforia

This framework has provided us with a solid foundation of which we have built our project on top which has provided us with a pipeline structure where we can seamlessly pipe the data from one tool to another and at chopping up the archive in sizeable chunks allowing for a multithreaded execution in order to speed up the entire process. The complete pipeline built Wikiforia looks like this:

$$\xrightarrow{Wikipedia\ article} Sweble$$
$$\xrightarrow{plaintext} Stagger$$
$$\xrightarrow{POS-tagged\ article} Maltparser$$
$$\xrightarrow{Dep.\ parsed\ article} CareerParser$$
$$\xrightarrow{Career\ Profile}$$

2.3 Sweble

This is the first step of our pipeline. Here a full Wikipedia-article is submitted in order to have the Wiki-syntax parsed and the data we are interested in extracted.

2.4 Stagger

We chose Stagger as our part-of-speech tagger since we wanted to have the ability to parse both Swedish and English (although we never did any actual parsing in English). In addition to POS-tagging, Stagger also includes `Named Entity Recognition` (NER), which we use further down in the pipeline to extract persons from the sentence.

2.5 Maltparser

The third step is to run the part-of-speech tagged text through a dependency parser. Maltparser uses the `Nivre arc-eager` algorithm by default and we found no reason to change it. We have decided to use `Maltparser` for its easy integration and multithreaded support.

3 Career Parser

The fourth and final step of our pipeline is where our own work comes into play. This is where we parse the text to find out which person works with what, during what time frame and try to stitch all of these things together. Below you will find a detailed outline of how this process is performed.

All parsing is done sentence by sentence.

3.1 Finding persons

The first step in our Career Parser is to first find out who is referenced in the sentence. A person is identified if one of following holds true for a word:

- It is tagged as a person by Stagger, which includes a Named Entity Recognizer.
- It matches a regex based on the name of the person of interest.
- It is a pronoun in singular form (Han, Hon, Hans, Hennes)

All the persons matched are stored in a list which is later used to match each profession with a person and see if it belongs to the person of interest (the person the article is about). Which is done in the path finding stage.

3.2 Finding jobs

The next step is to find all the jobs referenced in the sentence. This is done using the set of jobs extracted from `Wikidata`. We started out by check each word in the sentence, comparing it the list previously collected in order to determine if this is a job or not. However, there is a drawback to this solution. Since the list is incomplete it results in us missing out on a large number of professions. We have in response to this attempted to extend the matching in order to capture as many professions as possible. The method used for this follows this simple list:

1. Check if this word, without any modification, is a job. If not, continue.
2. Check if this word is a concatenation of two words, where the last word is a profession. This check is done by dropping words from the beginning of the word until what is left is a profession. This however does not ensure it is a job, so we continue.
3. Validate that the dropped characters themselves make up a word. This is to eliminate false-positives such as `kre`**`tsar`**. If the dropped characters do make a word, the full word is a job, else continue do not make up a word, continue.
4. In Swedish (and other nordic languages as well) it is very common to either add a textbfs or a exchange the last vowel of the first word, which is why we have devised the following rules:
    - If the last letter of the word ends in an **s**, remove it and check if the new word is valid. This allows for matching: `utbildning`**`s`**`minister`
    - If the last letter of the word ends in a vowel, exchange it for another vowel and see if that makes up a word. This allows for matching: `förskol`**`e`**`lärare`

If all of this fails, we deem that this word is not a profession and continue.

3.3 Finding verbs

This step is one of the most crucial steps as this is what actually links a person to a job. Without this step, we would tag things like `Simon gillar poliser` incorrectly and output that `Simon` has worked as a `polis`. Clearly, this is incorrect. What we have done to solve this is to only allow the verb connecting the two words to be one in the following set:

- vara

- bli
- arbeta
- jobba
- praktisera

3.4 Finding a path

So far we have just found all the persons, jobs, and verbs and stored them in their own respective lists. We have not yet actually linked any persons to any jobs, all we know is that there are some persons, jobs and verbs in the sentence.

In this step we try to find a path between one of the jobs we have found with one of the persons. The path is created by traversing the dependency relation between each word until we find a common ancestor. There is however one crucial criteria: the path from job to person must walk through a key verb.

Even though we might have found path between a person and a job it does not necessarily mean this path is the shortest one, which is how we verify if this job belongs to this person or not. In order for a job to belong to someone they must have the shortest path to that job.

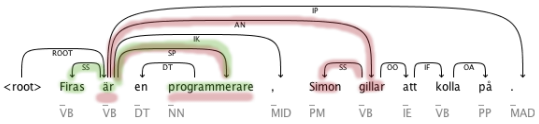

Figure 1: From the image above you can see how the path between `Firas` and `programmerare` is shorter than the one of `Simon` and `programmerare`

We therefore find all paths between all persons and all jobs, and only pick the shortest paths for each respective profession.

3.5 Finding Dates

After we have found a job and linked it to a person we start parsing for dates in the sentence. This has proven to be a very difficult task which is why our implementation has remained quite naive.

We take the the words preceding and succeeding the word representing the job in the sentence and look for something that resembles a date. If either word is a date we just use that; we only proceed further back or forward if the word is a preposition or belongs to a conjunction. Clearly, this analysis is not very well conducted and rather tailored to the patterns we have seen in the texts. A proper analysis would however be very difficult and a project in itself. This would miss out on date references located elsewhere in the sentence, not in relation to the job, such as: `Under 1996 arbetade Simon som programmerare.`

Our first attempt was to create a path finding algorithm, similar to the one we use to link persons to jobs, but instead link jobs to dates. The problem that arose here was that we do not know what word this date "belonged" to. The elimination process in our path finding process has always been to pick the shortest path, in our previous case, between several people and one job. The problem with doing this for dates is that we do not know which paths to compare against as the only path we will have found will have been from the profession to the date. The date could quite clearly have been pointing to another verb, such as in the sentence below, but we would have still thought it belonged to the profession.

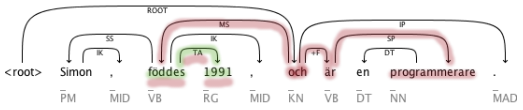

Figure 2: The figure shows the path between the profession and the date, but quite clearly they do not belong together.

4 Results

When measuring the accuracy of the system we took 10 random Wikipedia articles about people that was about one or two paragraphs long.

Recall	Precision	F-Score
74,1%	95,2%	83,3%

Disclaimer: since the articles were so short they were often to the point and did not contain any complicated language. This made the recall easier than if we would have tested against larger and more complex articles.

5 Shortcomings

Although we are happy with the results that we have achieved we are aware of several shortcomings in our system, the most notable being the difficulties regarding dates as previously mentioned. There are still some other issues that need to be tackled, which we have discussed below.

5.1 Negations

We do not pay any respect to any kind of negation in the sentences, such as `inte` or `aldrig`. While this has not yielded any false positive results so far, it is still an aspect we are lacking and need to be improved.

5.2 Occupational Activities and Workplaces

Since we only look for actual occupations we cannot associate work related activities or references to a workplace with a profession. Meaning that neither `writes articles` nor `works at New York Times` would register as a journalist or writer.

5.3 Wikidata Limitations

The search performed to find all occupations is quite wide, since what we are essentially doing is collecting anything remotely related to the **Occupation**-node. Due to this quite wide search we receive some unexpected professions in the baseline, for example:

- **Rare** - a video game studio, which is a software developer, which in turn is a profession
- **Slavery in the United States** - since it is slavery, which is unfree labour, which in turn is labour
- **Ant Man** - who is a superhero, which is a form of occupation

A more robust analysis could have been conducted to filter out these erroneous professions, for example, by controlling that they did not have a path to `business` or `superheroes`. But the number of possible edge cases are endless which would make this a tedious process, seeing as how very many things can eventually be linked to an occupation. This is however, none the less, a limitation in our design and implementation.

5.4 Naive Name Regex

The regex we run on the entities tagged as persons is very naive and matches more than we would like to. For example the regex for President Obama would be `barack|hussein|obama` which matches any reference to `Barack`, `Hussein` or `Obama`. While most of these references are accurate, this is not always the case. We have stumbled upon times where the text was referencing, say, the father instead which we could not deduce.

The mere fact that we use regex for this at all is naive, considering how there are frameworks specialized in accomplishing this very task, so called `Named Entity Disambiguation` (NED) or `Entity Linking`. If we had more time on our hands we would have liked to integrate some kind of NED into the project.

5.5 Pronoun References

While looking for persons in the sentence we also check for pronouns. The problem introduced here is that we do not evaluate what the pronouns are referring to, and instead assume that it is a reference to the person of interest. The expression `he is a lawyer` would result in the person of interest being given the occupation `lawyer`, regardless of context. Once again, if we had more time on our hands we would have liked to dig deeper into pronoun reference evaluation and anaphora resolution.

5.6 Swedish Only

For now the software only supports the Swedish language, which in international circuits is a big shortcoming. It is however fairly simple to adjust the software to work for English as well, since all the tools under the hood already have native support for it.

Most of our Career Parser would still work for English, with modifications needed for the language specific verbs and profession checker.

6 Summary

In conclusion, we are very happy with how this project went. The result is on par with what we were expecting and the journey was eventful, worthwhile and fun. As we have previously mentioned, the precision and recall was good and in general our system performed well enough to produce accurate career profiles. Complicated cases did reveal the shortcomings in our program, which has been thoroughly discussed. We believe that with more time (and effort), most of these the flaws could be ironed out (perhaps with the help of other existing frameworks) and produce even greater results.

In a future revision one might attempt to link work experiences to locations, or something of that nature.

Acknowledgements

We would like to thank Pierre Nugues for his motivation and guidance and Marcus Klang for his work on Wikiforia which was (and is) a great foundation to work from. Furthermore we would also like to give credit to the following projects, which without the project would not be possible:

Sweble: a wiki markup parser (Dohrn and Riehle, 2013);

Stagger: a POS-tagger for Swedish and English (Östling, 2013);

Maltparser: a dependency parser (Nivre et al., 2006).

References

Dohrn, H. and Riehle, D. (2013). Design and implementation of wiki content transformations and refactorings. In *Proceedings of the 9th International Symposium on Open Collaboration*, WikiSym '13, pages 2:1–2:10.

Nivre, J., Hall, J., and Nilsson, J. (2006). Maltparser: A data-driven parser-generator for dependency parsing. In *Proceedings of the fifth international conference on Language Resources and Evaluation (LREC2006)*.

Östling, R. (2013). Stagger: an open-source part of speech tagger for swedish. *Northern European Journal of Language Technology*, 3.