Tepes AI - Requirements Analysis

Table of Contents

1. Architecture Overview

For the application to be able to detect anomalies between a person's financial statements and earnings we propose the following architecture: The first step would be a manual one, in which we would have to search for information sources from which we can scrape data. After collecting enough sources and defining methods of retrieving data from them, we can automate this process by creating a **Crawler**. After retrieving the data using the crawler, we face a new problem. Since the data is retrieved from multiple sources and there isn't a standardized format for it (the data can come in the form of a pdf file / image with handwritten text), we need to create a **Normalizer** which is able to transform the scraped data into a usable format. With the information extracted, we now need to merge it in a standardized format which is useful for our anomaly detection purpose. For such a task, we will need to create an **Aggregator.** With the data now standardized, we can split it into two datasets (train / test) and use the train subset to train our **Anomaly detector.** After obtaining the desired accuracy, we can then use the **Anomaly detector** component to find anomalies in people's financial statements as stated in the problem's description.

2. Crawler

Before defining the crawler we will need to manually collect multiple data sources. After the collection of data sources, we will need to implement custom logic for scraping data from each source. For the application to be extensible to future potential data sources, we'll need to define a common interface from which we will build the logic for each source. At first, the crawler will be implemented in Python and it should contain two main components ->

1. CLI Crawler with the requests lib
    a. This crawler should be used for easily accessible targets which don't have many security features implemented (such as throttling, user-agent analysis, captchas).
2. GUI Crawler with the selenium lib / framework
    a. This crawler should be used for targets which use custom frameworks and for which the requests library isn't powerful enough. With it, we can mimic human behavior easier and better.

After collecting the data, the crawler should store it in a cloud storage, accessible for the whole team. We will have multiple data formats (text, pdfs, images).

3. Normalizer

As we mentioned, working with multiple data formats would require defining a template for our final data model. Since we are working with text formats (or images with text inside), Natural Language Processing is the best choice in order to prepare our data for the next steps.

For the text inside images we can use an OCR (Optical Character Recognition). We will proceed with the Keras-OCR, a tool which provides out-of-the-box OCR models and end-to-end training pipeline to build new ones.

After we gather all the data under a text format, we should proceed with expanding all the contractions, in order to get a clear visualization of the data. This can be done with a dictionary of contractions (with their corresponding expansions), further used to update the data with regular expressions.

Next, we tokenize the data, a process of cutting a text into pieces called tokens. We will use nltk's word tokenizer. Nltk also offers a set of functions which can be used in order to remove the punctuation.

After that we are stemming the data using Porter's algorithm, carefully, in order to eliminate the roots which are not actually English words.

Then, we are lemmatizing the data to obtain the final list of words. We pass this list to the POS tagger, in order to see which part of speech is suited for every word. With the tagged data, we can go further to the next steps, with the same template over each politician's information.

4. Aggregator

After the data collected by the crawler is transformed into an usable format by the normalizer, said data should be merged into a standardized format. For this purpose, we must implement an Aggregator. One could also define what we are trying to achieve as "summarizing" our collected data. This is done to make detecting anomalies much easier later on. As for what tool we will be using, MongoDB, a non-relational database, has a multitude of great tools suited for our purpose. A couple of effective approaches to aggregate our particular set of collected data are as follows:

- Filtering by incriminating news headlines and articles:

Since our data is centered around politicians, and the most corrupt ones are targeted to a lot of press coverage, gathering relevant data about the more well-known corrupt personalities in politics. For this, we will pass to our aggregator some relevant keywords so that we can differentiate useless data from actual useful data. This will help in gathering together data about potentially corrupt politicians who have recently been reported on by media outlets, thus giving us a good starting list of targets to investigate further.

- Filtering by ties to other known corrupt politicians:

Politicians who are known to have relations, either of friendship, or of membership of the same party to already highly suspected, known, or convicted corrupt politicians can be considered as more likely to be corrupt themselves. Filtering out these known relations helps us, once again, reduce the amount of junk data, revealing data that is more useful to our cause.

Obviously, there are many other ways by which we are able to filter our collected data. After aggregating said data in whatever manner, we can then proceed to the next step, anomaly detection.

5. Anomaly detector

Finally, we reach the final part of our proposed process. This step represents the actual assertion of whether a public person is 'corrupt' or not. After having our data processed, we can design a deep

neural network in order to decide if a given person is an **outlier**. We will tweak our model to take multiple factors into consideration, such as:

- Is the person underqualified for his/her position? - does this person have comparable experience / education to be able to be as qualified for the position? Is this level of competence comparable to other peoples' with the same job?
- Is there a significant discrepancy between a person's wealth declarations from consecutive years? Did that person get rich 'overnight'? Moreover, using **ontologies**, we may be able to decide whether the people connected to this individual also have fluctuations in wealth. That is because, if a group of people from, let's say, the same political party suddenly all declare more properties or assets in consecutive years at once, that may indicate that corruption has reached that particular group.
- We may also consult social media platforms / news outlets, to assert if there were 'scandals' about a public figure - in this context, a discrepancy would be, for example, if the wealth declaration of someone would account for no assets, and the article, on the other hand, would accuse said person of owning undeclared mansions or assets, inexplicable from a legal point of view.
- Finally, we may search for trials a person has gone through.

Based on the factors mentioned above, even though it is still to be decided, we may use this detector to assign each public person a 'corruption score'. Each factor would contribute with a certain weight in the final result. This would mean our model would be trained for regression. Using this computed score, we may filter / sort people based on different criteria.