Reputation Analysis - Dathena competition

By Max Premi and Charles Gallay

Motivation

Perhaps the most critical, strategic, and an enduring asset that a corporation possesses is its reputation. Although corporate reputation is undoubtedly a significant and relevant corporate asset, formidable measurement challenges exist nowadays. We propose the creation of a "reputation index" based on public articles and all media resources, which are accessed on the internet and are relevant to the particular company.

Our approach is based on a Sentiment analysis strategy to evaluate all the events related to the company and its employees, client's opinion and calculate the general index, which can define the level of the reputation.Several studies have focused on the same topic of the sentiment changes over time in online media. Gilbert & Karahalios (2010) have estimated information about future stock market prices based on the analysis of the emotions expressed in blogs. Based on over 20 million posts made on the site LiveJournal, they found that increases in expressions of anxiety predict downward pressure on the S&P 500 index. O´Connor et al. (2010) have found that a relatively simple sentiment detector based on Twitter data replicates consumer confidence and presidential job approval polls.

Introduction

Our goal is to find the 10 best and 10 worst document about each company: Keppel (Keppel Corporation Limited) and Prudential (Prudential Assurance Company). To do so, **Name Entity Recognition** and **Machine Learning Classification** are going to be needed.

In this report, we present our pipeline for the whole process of extraction and sentiment analysis.

Our first approach was a naive one, and we build more features on top of this one to get to our final result. The final pipelines contain *Text extraction, Company Recognition, Text Summary, Training of a model for Sentiment Analysis, Features Addition, and finally Sentiment Analysis*. The exact pipeline is presented in details in **Section 2.3**.

In the following lines, we present the layout of this report. The first step is about understanding data, states assumptions, and to look at how they can be extracted. Then, the second part focuses on the cleaning of data. The third part deals with the naive implementation and the steps taken to end with our current pipeline. In the end, we address the future work in the **Optimizations Section** and a conclusion on our work.

1 Observations

1.1 Data

We looked through the data and noticed some interesting things, and points we should not forget. The data come from Web-scraping meaning some of them are totally irrelevant to the task. Indeed not all documents are talking about one of the 2 companies (Actually most of them are not talking about the interesting companies).

Example: `BEINGS: PHASES OF COMMITING THE CRIME.pdf`. *Robert Keppel* is the author of one document stated in the text, it has nothing to do with *Keppel Corporation*.

The documents are of 3 forms:*PDF, doc(x), xls(x)*. The metadata that we could use is *frequency* and *occurrence* of the company word, as well as *Title* of the document. We only extract text from documents, and noticed that some files were corrupted or could not be opened (such as `businessassociatelist--1-.xls`).

The library used to open the documents are : *docx, PyPDF2, PdfReader, xlrd* are the python library we used to open all document or extract Metadata, except the `.doc` extension. Indeed it seems no python library could open this so we used *Antiword* to convert them to `.txt`.

All needed library are stated in the **README** of the Github repository.

1.2 Assumptions

The company studied are:

- Keppel company basically was offshore specialized. Singapore based. The company consists of several affiliated businesses that specialises in offshore & marine, property, infrastructure and asset management businesses.
- Prudential plc is a British multinational life insurance and financial services company headquartered in London, United Kingdom. **Do not mix with Prudential Financial** which was not the company asked for!

Company detection is done in the following way:

We did a parsing of all phrases containing “Keppel”, “Prudential” and then looked “manually” (fill an array with all different occurrences) at the combination of the word of the company and the words that it follows. This way we were able to look at which document to drop and which to keep.

- For **Keppel** the keywords to look for are *'keppel corporation', 'keppel capital', 'keppel reit', 'keppel infrastructure','keppel o&m', 'keppel energy', 'keppel offshore', 'keppel dhcs', 'keppel fairway', 'keppel professorship', 'keppel reit', 'keppel group', 'keppel land' , 'keppel singmarine', 'keppel fels', 'keppel shipyard','keppel shipyard' , 'keppel gas','keppel t&t', 'keppel bay', 'keppel telecom','keppel corp','keppel seghers','keppel center','keppel cebu','keppel thai','keppel philippines','keppel tatle','keppel telecoms','keppel tower'*.
    - These are mostly the branches of keppel, we drop all other documents.
    - Other were talking about *institutions*:Keppel Highschool, *people*:Edwin Keppel, Keppel-Jones, or *places*:Keppel Free Trade Zone, Keppel Harbour , Keppel Street, ...
    - We drop around 75 documents out of the 193 documents.
- For **Prudential** the keywords to look for are *'prudential mutual','prudential plc'*
    - The dropouts are about *law*:Prudential laws, *word*:prudential is a word, but mostly *other Companies*: Prudential Finance, Prudential Real Estate Investors (belongs to the first one), Prudential Regulatory Authority, Prudential Retirement.
    - We drop 180 (which is a lot) documents out of 194.
    - **We noticed that a lot of document were about Prudential Financial**

With this approach, we guarantee that the documents we fetch are only documents talking about the companies, which should have given us a Precision of 100% in spite of a weaker Recall. Sadly, we realized the last day of submissions that some documents were parsed from the wrong company event with the filtering. We could easily fix this problem by removing a filtering word, but we would end up with a really few documents. We really think that mixing document from different companies are really bad for the purpose of the classifier. But because of the lack of time and few documents, we kept this.

Concerning accuracy on Kaggle: After seeing that Kaggle was doing the Mean Average Estimation (MAE) on the files and that our assumptions were potentially different than the State of the Art approach, we decided not to base our result on this, but to look ourselves at the document and decide whether we were satisfied with our result. (Indeed if you submit a csv with all 0 MAE is $2 * (10 * 2 + 9 * 2 + 8 * 2... + 1 * 2)/(193 + 194) = 0.56847$)

2 Implementation

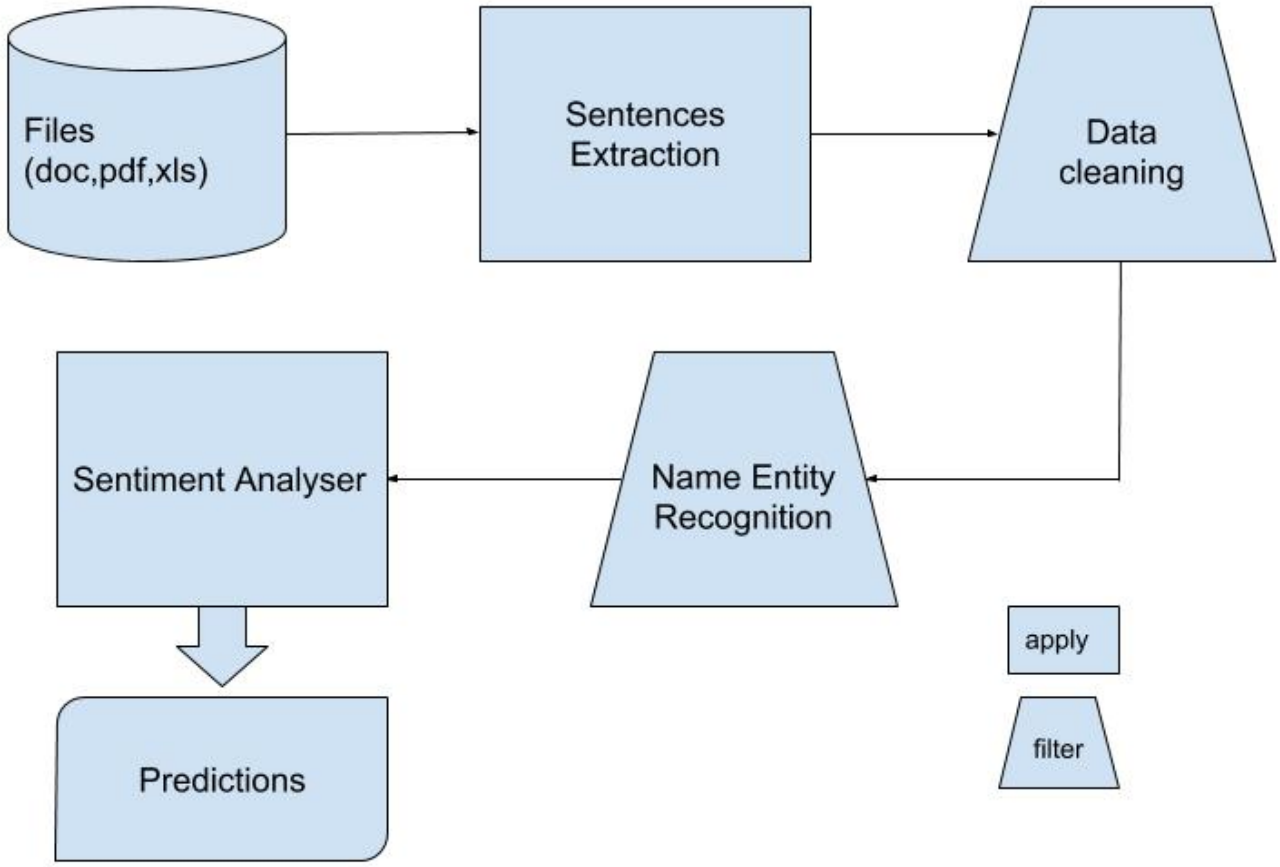
2.1 Naive Implementation

Our first implementation was a naive but quite quick one. We extracted from all document the phrases containing either **Keppel or Prudential**. To do so, we looked at all occurrence of the word and took the 10 surrounding words on the left and right. This way we had sentences about the "company" (was not true, but naive approach), for each document. Then the phrases were load in a model in Spacy library that process all sentences loaded. The pipeline used by the default models consists of a *Part-Of-Speech tagger* and a *Tokenizer*. Eventually, we just use NLTK provided pre-trained Sentiment Analyzer returning sentiment $\in [0, 1]$ on each of these sentences, and do the mean of all result for one document, and classify in function of these values, with *0 negative*, *0.5 neutral* and *1 positive*.

The result of this naive implementation was not that bad, but clearly not good enough. Some documents were classified correctly, but other not at all (Award winning company for Keppel was at -10). This was clearly due to 3 issues:

- We did not run any "**Name Entity Recognition**," i.e, we might classify document talking good or bad about a Keppel Entity that has no link to our topic.
- The sentences we get are not the only one **relevant to the whole document**, i.e, we need a solution to take the sentiment of the document into account, and not just the sentiment of the sentences talking about the company.
- The sentiment analyzer **is not performing well**, we fix this in the next session by showing how to train it for our specific task and dataset. We think that the "bottleneck" for performing well is the sentiment analyzer.

2.2 Improvement made to the Naive implementation



To solve the 3 problems stated before, the following changes were made to the Naive model. The pipeline is shown above (we do not detail exactly as it will not be used in the future, the final pipeline will be more detailed.)

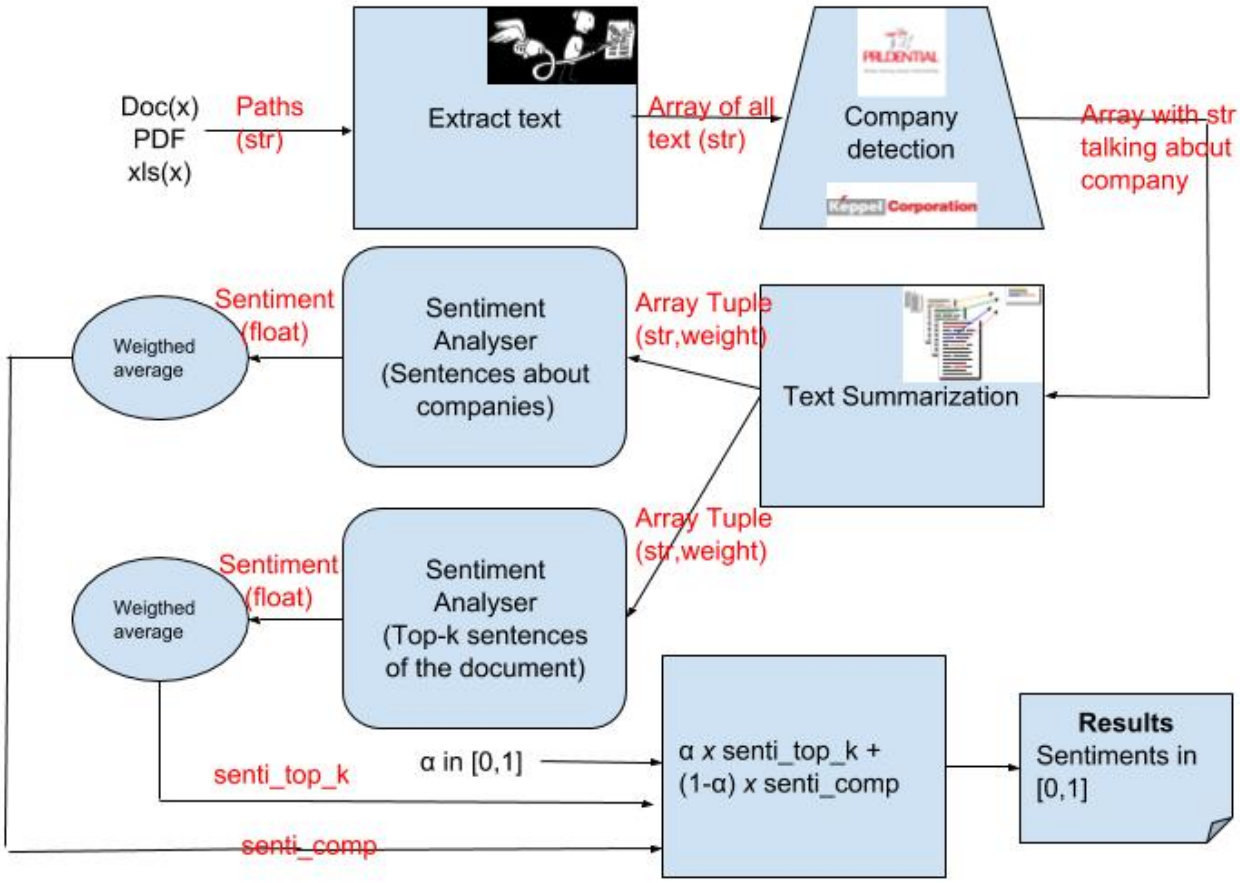
- First, we try different sentiment analyzer, using either already modeled one `NLTK` or even Stanford one* or *Convolutional Neural Network*.
- We also added an **NER** to filter out phrases that were not tagged as *ORG* using the tagger from `Spacy`, but it was not concluding.

These solutions were not satisfying enough to be kept. In the end, the **NER** will be done as the **Section 1.2** describes.

Also, the sentiment analyzer is now implemented with `Keras` library, by doing a Convolutional NN.

2.3 Final implementation and pipeline

We now develop the final pipeline and model we implemented and use to get our final submission.



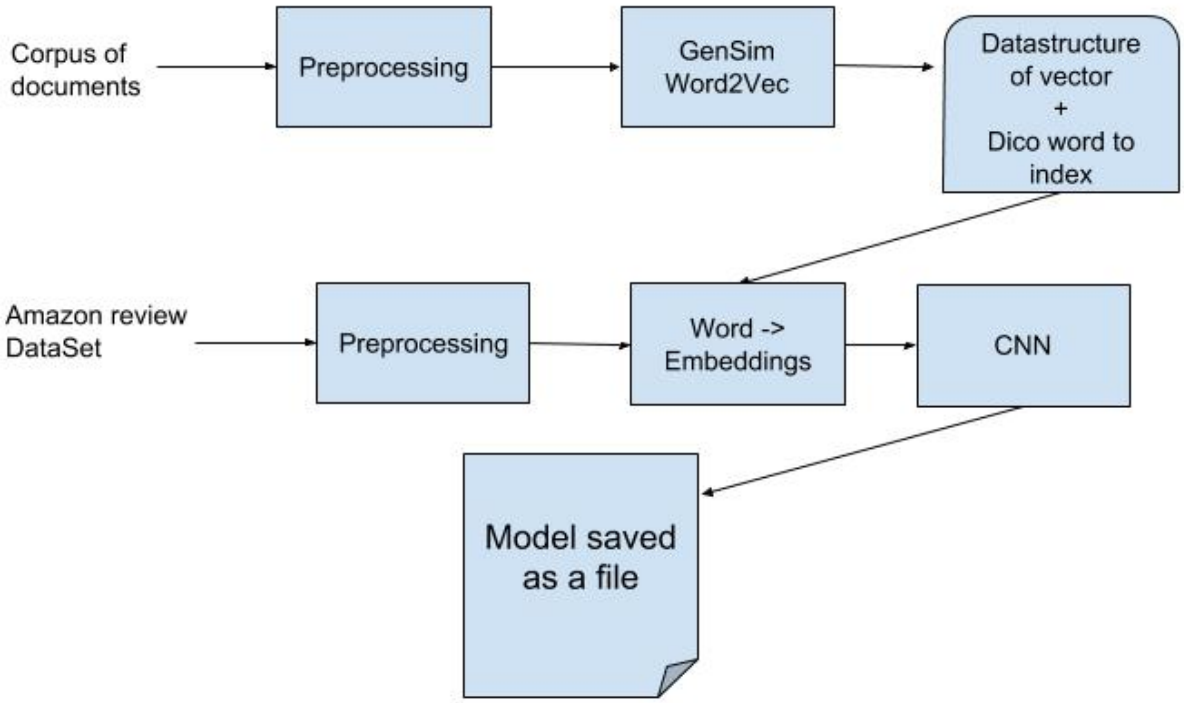
We extract the text from all document the same way as before. In a second step, we only keep the text that contains one of the words in relation to the company as describe in **Assumption** (Section 1.2) with a function called `doWeDrop`. This way we greatly decrease the number of documents to analyze. All dropped documents are labeled with a -500. Dropping documents as early as possible is a good strategy.

Then we extract 2 different things from each document. First, all the sentences that contain **Keppel or Prudential**, secondly with their weight (importance), using a library called `Sumy` (API), and also the *top-k* sentences, with their weight computed with TextRank.

The *hyperparameter* α is completely tunable. Its goal is to chose to which part the model should give importance when computing the final sentiment. With $\alpha > 0.5$ we consider more the whole document sentiment, while when $\alpha < 0.5$ we consider the sentences dealing with the companies more importantly. And if $\alpha = 0.5$, we consider both part equally.

This assumption is made with the following thinking: If the name of the company appears in a negative document, it is bad for the company. This is why we consider both.

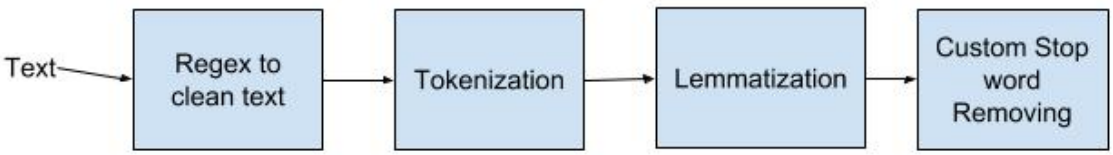
Now let's look more closely at what the *Sentiment Analyzer* is made of.



We use all the documents provided to create a pre-trained *Word Embedding*. First, the preprocessing is done on the text (see figure under). A regex is applied to remove all characters except the Roman alphabet, and some punctuations that contain sentimental information (..., !). This way there no weird characters, that does not belong to `utf-8` and numeric symbols. Then we use the `nlp` (pipeline) function of `spacy` to **Tokenize** and do a **Lemmatization** . At this point, each text contais words reduce to their root in function of the context, and the last step is applied. It is a removal of stop words, customized in our case, which are considered not useful for the sentiment analysis.

We use customize stop word as we do not want to remove the word **not**. This is due to the fact that sentiment analyzer struggle with phrase containing a negation. You can check this with an online Sentiment Analyzer that can not predict correctly the sentence: *you are not beautiful*. So we wanted to try to reverse the sentiment if a not was contained in the phrase with an adjective preceding it.

Example: "You are not beautiful" is rated positively. In classical sentiment analysis, after removing stop word this phrase is **beautiful**. With our idea, it should be **not beautiful**. This way we can predict only on *beautiful* and get the sentiment s_1. To reverse it we just apply $s_{not1} = 1_{not} - s_1$. You reverse the sentiment if and only if a *not* is present.



After *preprocessing* the corpus, we compile the `word2vec` from `gensim` to get vectors representing each word of our corpus. This is done so that words specific to the company or to this task get assigned a vector. The parameters to tune are the **length of each vector** fixed to 300 here, and **the maximum number of vector** fixed to 100 000. At the end of this step, the algorithm outputs a data structure containing the embeddings, and a dictionary with word index mapping

After getting the vectors, a data set coming from Kaggle about Amazon review is used to train a sentiment analyzer. The **same pipeline** of cleaning is applied to the reviews, and each word is mapped to the embeddings. The **CNN** is trained on the Amazon review, and the model is saved in a file.

Let's look at how the *CNN* is implemented:

The Convolutional Neural Network was implemented with `Keras` and `TensorFlow` as a backend. We based our model on a Medium post on which we added a regularizer on embeddings weights. We computed word embeddings on the documents about the companies provided on Kaggle. We wanted to make these embeddings modifiable as we trained the model on Amazon Reviews, because we wanted to use the information contained in our pre-trained embeddings with another text corpus. The idea is the following:

A word has a different embedding represented as Emb_{amazon} and $Emb_{companies}$. To take into account the context information of words from the Company corpus, we forced the cosine similarity distance between 2 words a, b in the Emb_{amazon} space to be roughly the same as the cosine similarity distance between a and b in $Emb_{companies}$ space.

This is done by a custom regularization function implemented in `Keras`.

Given 2 words a and b, and 2 embedding space Emb_{amazon} and $Emb_{companies}$. Each word has it's embedding in the space denoted as $v_a = E^a_{amazon}$, $v_b = E^b_{amazon}$, $u_a = E^a_{company}$ and $u_b = E^b_{company}$. To enforce the cosine similarity to be the same we compute , for each batch, random pairs of indexes embeddings $Sample = [(i_1, i_2), \dots]$ uniformally sampled and we minimize the following :

$$\textbf{square difference of cosine similarity} = cosSimDiff(v, u) = (\frac{v_{i_1} \cdot v_{i_2}}{|v_{i_1}| \cdot |v_{i_2}|} - \frac{u_{i_1} \cdot u_{i_2}}{|u_{i_1}| \cdot |u_{i_2}|})^2$$

$$\textbf{regularization term} = \frac{1}{1000} \sum_{a,b \in Sample} cosSimDiff(a, b)$$

We penalize big difference by adding the regularization term multiplied by a hyperparameter $\sigma = 0.1$ in our actual code.

3 Optimizations

We discuss here some optimizations that could be made to improve our model.

First, we could use the Brown Corpus along with our Corpus to get our embeddings from `word2vec`. This would allow having more precision and more words.

What could also be done is a correction of words. Indeed, the parsing is not perfect, and some documents are not either, so an algorithm to correct word could be useful. For example, correct a spelling mistake, or split 2 words that were mistakenly appended together.

Another interesting feature that could be implemented is language detection. Indeed, we noticed that some documents were containing some Chinese/Japanese characters, but also text in other languages (Polish or Hungarian not sure). If language detection was implemented, we could choose which model to load from `spacy` before doing the cleaning and thus, have more accuracy on foreign language.

Moreover, the summarization used is part of what is called *text extraction*. Indeed, we extract phrases that seem important. Another approach would be complete text generation with *abstractive text summarization*. It generates sentences about the text, to summarize it as well as possible, with deep learning methods.

This method seems complicated and is said not to be much better than the extraction so we did not focus on it. We believe that by using an attention-based RNN, we could even get an abstract summary of a particular word (here it would be the name of the companies). Still, we did not test it, but with the name in input, it should be able to activate the correct attention memory cells and then product a correct resume of the text given this word as an important feature.

On top of the CNN, we started the implementation of a SentiWord augmentation for our sentiment analyzer but did not have the time to finish it. This might have helped the analyzer by creating a new feature which is the mean of the sentiment of each word in a sentence.

4 Conclusion

The result we come up with is quite satisfying for us. We wanted to "think outside the box" and do different things or variations of what already exist. We had almost no notion of NLP before this project and we had a lot of fun playing with different libraries and learned a lot in only one week. We enjoyed implementing the Analyser and we are quite satisfied with the result. Of course, the implementation is still really clumsy, and a lot of improvements could be made as discussed previously. We thanks Dathena for this interesting challenge and opportunity to do a real-world application of what we are taught.

5 References

- Text Summarization
- SentiWord
- Spacy Training
- A really good online sentiment analyzer
- Keras