Political Promise Evaluation (PPE)

Bhanu Jain, Rohit Begani

October 26, 2017

Abstract

PPE provides a unified view of promises made by Political Candidates while campaigning and the promises fulfilled during their tenure. PPE aims to scour and summarize Political Promises. We will compile a repository of newspaper articles about the political campaign and promises and then use the content of these articles to evaluate the current status of the promises.

1 Introduction

Elections play a huge role in deciding the growth path of a country for a long time to come. Selecting the right candidate can help a country take exponential leaps in their growth while inversely it can cause the country to lose out on years of progress.

We aim to provide unbiased natural language processing based evaluation for Political Candidates' Campaign Promises. This evaluation can be invaluable in helping people decide who to vote for specially during re-election. Moreover this evaluation tool provides an easy way for people to keep track of Candidate's promises and keep him accountable for the same.

In this age of Internet and due to the presence of multiple digital news sources, we can access real-time information about the world. Due to the vast influx of information it's usually very easy to miss out on important information. We aim to extract articles relevant to political candidates and their promises and then analyze the articles to understand the current status of the promises.

As a general newspaper reader it's usually very difficult to dig-in enough to read up on the less scandalous promises. Rather than showcasing the more click-bait articles we aim to provide a more objective evaluation about all the promises made by a candidate in a concise and unambiguous form.

2 Project Updates/Changes

2.1 First Attempt

1. For creating promises we started with PolitiFact API [4] over the Candidate's Campaign Manifesto since it would give us more accurate and better results.

2. We restricted our collected data to the Promise title so we would know what to match the articles against.

3. We attempted to extract the data from the Twitter API but realized upon trying that it wasn't possible to extract sufficient Tweets to give valid results for our project because of the restrictions established by Twitter[1].

2.2 Second Attempt

1. In this attempt we decided to collect Promise Result information from newspaper articles rather than Twitter so that we would be able to access a huge repository of archived data.
2. We pulled specific articles from Digital News sources to get a repository of relevant data.
3. After data mining we used bag of words for to match articles to the Promise. We were unsuccessful in getting usable similarities because the titles were too small and not descriptive enough.
4. We decided to extend the promise information to include description for the promises but we were again unsuccessful as a lot promises didn't have any associated descriptions.
5. The final way was to include tags for promises which didn't have an associated description.
6. Even after all these ways we weren't successful in getting a good matching because we were looking for exact keywords.

2.3 Third and Latest Attempt

We used TF-idf to perform document similarity. The idea is to treat both articles and promises as documents, generate TF-idf scores and compute cosine-similarity between articles and promises. In order to achieve this, we did,

1. Tokenized all the words using the NLTK Tokenize package[2].
2. Removed all the stop words to get relevant words.
3. We decided to use Stemming to include the root words for all the words to improve matching. To do this we used Porter Stemming[4] which is a part of NLTK. Stemming didn't work out well as we realized that it just takes the root form of a word rather than grouping together the inflected forms of the word. As an alternative we used Lemmatisation[6].
4. Finally we used the Tf-idf algorithm from the Scikit-Learn[5] to get the TF-idf scores which was used to find the Cosine similarity between the promises and the articles.

3 Data Preprocessing

3.1 Promise Preprocessing

Promise data was extracted using the PolitiFact API[4]

3.1.1 Data Selection

- The data provided by PolitiFact is for a multitude of candidates.
- During selection for data extraction we restricted the data to Donald Trump.
- Promises were restricted to the promises made by Donald Trump during his campaign speech.

3.1.2 Data Cleaning

- The data was extracted from the PolitiFact API[2]
- The data provided by the PolitiFact had a lot of extraneous elements which might be used to create a complete web based application for Trump's promises.
- We restricted the provided dataset to include only the Promise title and the Promise description.
- A lot of Promises didn't have a description attached to it in that case we included the Promise Tags to help us understand the context of the promise.

3.1.3 Data Representation

1. The data extracted was converted and stored as a JSON of promise arrays to allow fast retrieval and processing.

3.2 Article Preprocessing

3.2.1 Data Selection

For Promise status we're parsing newspaper articles. Due to the pace at which news is released everyday and because of the division of news in to multiple articles for more clicks we had to be smart about our article extraction to prevent excessive useless data.

1. The data was extracted using the multiple news sources available digitally.
2. Newspapers were selected because they are the source which are expected to cover all information regarding a Political Candidate.
3. A lot of research was done to identify neutral, left-leaning and right-leaning newspapers.
4. Extremely left or right leaning newspapers like BrietBart and OccupyDemocrats were ignored to ensure data integrity.
5. Restrict our search to the name of the candidate and the variations of his name in general news media.
6. Restrict newspaper articles to a specific duration to prevent collection of articles on the relevant candidate out of scope of the candidate's tenure as that is for sure going to be unrelated data.
7. We restricted the articles to those which were talking only about the Candidate's promises by adding a removal for all articles which seemed to be talking solely about the candidates win. This was done by reading the corpus of tags associated with all articles for all SEO purposes.

3.2.2 Data Cleaning

The data collected from News agencies contained a lot of information which are relevant to a newspaper article but were unnecessary for our purpose.

The data was restricted to article_url, article_text, article_summary, article_date and article_keywords.

3.2.3 Data Representation

We used the data to generate a JSON file to allow easy storing and parsing of the data.

4 Methodology

4.1 Methods/Models for Data Mapping

After collecting the whole corpus of articles and promises our next step was to assign all articles to their relevant Promises so they could be used for sentiment analysis later on.

To do the above we tried to follow multiple ways

4.1.1 Bag Of Words - Failed Method

1. We attempted to create a corpus of keywords for every article according to the frequency of every word in that article.
2. Then we tried to create a similar corpus of keywords for the collected promises based on the title and description of the promises. A lack of description for many promises made it extremely difficult to get a proper bag of words for them.
3. Many of the promises had zero associated articles even though we knew that there should have been more. We realized that this was because we were trying to match exact keywords in the promises which wasn't a good idea because we didn't have a big enough bag of words to get a proper result.

4.1.2 Tf-idf

Based on our failings in the earlier method we made a few changes:

1. We treated both the articles and the promises as documents.
2. We removed all the punctuation, the stop words and converted all the strings to lowercase.
3. We Tokenized all the words using the Tokenize method of the NLTK library[2].
4. After this we used the Tf-idf algorithm to get the Tf-idf numbers.
5. We used Porter Stemming[3] to improve word Tokenization.
6. The stemming actually removed all the plural versions of the words and converted it into the root form but it didn't make a difference.
7. We decided to Lemmaentize the words instead to get all the inflections for the words.
8. Finally we used Cosine similarity to match all the articles to their relevant Promises.

5 Model Improvement

Currently our model has a complexity of n-square as it computes cosine-similarity for all tokens in the vocabulary. We need to improve the algorithm to fit our memory and compute constraints.

6 Results

1. Since currently we have a complexity of n-square and our total corpus is of 2912016 documents so the operation runs for 2912016 * 2912016 times.

2. Consequently we're unable to get the cosine similarity on our systems because the process is slow and memory intensive.

7 Next Steps

Now that we have a method to match Articles to their relevant Promises our next steps would be:

1. Improve our document matching algorithm to give results for big corpus.

2. To extend our data repository to more Newspaper sources to get more accuracy.

3. We need to figure out a model to do sentiment analysis on the Articles related to every promise. Our first attempt would be to use a Naive-Bayes classifier but we might have try other Sentiment Analysis if we are unable to get a good enough result using Naive-Bayes.

8 Related Work

A lot of work is being done recently to generate awareness about lack of political promise fulfillment. Most of these are manual trackers which are hosted by individual media organizations or crowd-sourced.

1. Politifact [4]

2. TrumpTracker [7]

3. New York Times [8]

Additionally, there is some work done in the field of political sentiment analysis, as listed below:

1. Sentiment Analysis of Political Tweets: Towards an Accurate Classifier [9]

2. Analyzing Political Sentiment on Twitter [10]

3. Predicting Elections with Twitter [11]

9 References

1. **Twitter API** `https://developer.twitter.com/en/docs/tweets/batch-historical/overview`

2. **NLTK** `http://www.nltk.org/api/nltk.tokenize.html`

3. **M.F.Porter**, An algorithm for suffix stripping *Originally published in Program,14 no. 3, pp 130-137, July 1980.* `https://tartarus.org/martin/PorterStemmer/def.txt`

4. **PolitiFact** API `http://static.politifact.com/api/doc.html`

5. **Scikit-learn** `http://scikit-learn.org/stable/modules/feature_extraction.html`

6. **Stemming vs Lemmatisation** `https://goo.gl/89zD62`

7. **Trump Tracker** `https://trumptracker.github.io/`

8. **New York Times** `https://goo.gl/5pxb9R`

9. **Sentiment Analysis of Political Tweets: Towards an Accurate Classifier** `https://goo.gl/9x4xt9`

10. **Analyzing Political Sentiment on Twitter** `https://goo.gl/dN4JHb`

11. **Predicting Elections with Twitter** `https://goo.gl/kCj1Kv`