Business Insider: Extracting {CEOs, Companies and Percentages}

IEMS 308 - Data Science and Analytics, Homework 3: Text Analytics

Saif Bhatti

March 5, 2020

BUSINESS INSIDER
UK

1 Executive Summary

Business Insider provides important second-hand news stories regarding the latest in innovation and progress in the business world. An important task for BI might be identifying trends across news by understanding where relevant information like CEO names, company names and percentages are used. This might be the first step in forming a Question-Answer system, which would be able to parse queries and respond by collecting results across the corpus of text.

- In order to find CEOs, companies and percentages within the corpus of text, it is key to find matches by first splitting the text into small chunks and running text analysis on these sections. This includes cleaning techniques and preliminary analysis for named-entity-detection.
- First, initial data exploration was performed on each article (of 730 total), and then the open-source named-entity-recognition (NER) library spaCy was used to examine used to find entities in the text, and allow quick parsing.
- The results were impressive. Percentage extraction was automated using NER tools within spaCy, and the Logistic Regression model (with TF-IDF vectorization) was able to achieve over 95% recall (proportion of actual positives was identified correctly), and 95% precision (proportion of positive identifications was actually correct) for both Companies and CEOs. The results can be seen in the respective .csv files at the Github (See Appendix).

2 Problem Statement

The Business Insider articles were .txt files representing a full article, which represented data retrieved from their API in 2013 and 2014. There was a little less than one article per day, for a total of 730 articles (this will be henceforth referred to as the 'corpus' and the required tasks were as follows:

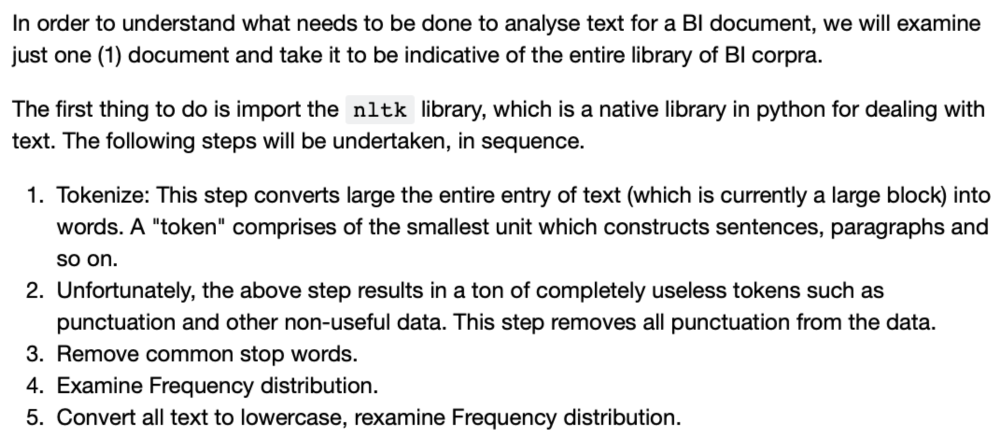
- Extract all company names from the files.
- Extract all numbers involving percentages. Note that sometimes the corpus has "0.5%" and other times "point five percent." (There are enumerate ways to refer to percentages.)
- Extract all names of CEO's.

3 Data Methodology

The steps taken are itemised as below.

3.1 Exploratory Data Analysis

- Run Exploratory Data Analysis on each article within the corpus. All EDA is conducted within **text_eda.ipynb**.



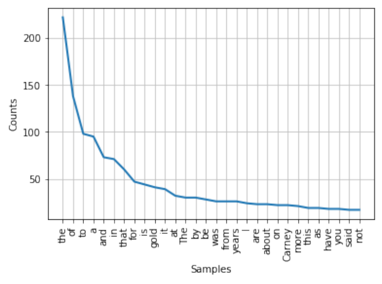
In order to understand what needs to be done to analyse text for a BI document, we will examine just one (1) document and take it to be indicative of the entire library of BI corpra.

The first thing to do is import the `nltk` library, which is a native library in python for dealing with text. The following steps will be undertaken, in sequence.

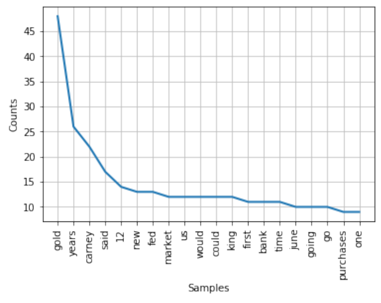
1. Tokenize: This step converts large the entire entry of text (which is currently a large block) into words. A "token" comprises of the smallest unit which constructs sentences, paragraphs and so on.
2. Unfortunately, the above step results in a ton of completely useless tokens such as punctuation and other non-useful data. This step removes all punctuation from the data.
3. Remove common stop words.
4. Examine Frequency distribution.
5. Convert all text to lowercase, rexamine Frequency distribution.

Figure 1: Pre-processing pipeline

- Based on these improvements from the Pre-processing pipeline, the following changes can be seen in the Frequency Distributions.



(a) Pre Pre-processing



(b) Post Pre-processing

Figure 2: Frequency Distributions pre and post Pre-processing

- However magnificent this may look, there is no way to properly scale this solution given the length of time at hand. There are many edge cases available in the corpus of text which foil even the most devious and length regex query. Rather than engage this rabbit hole, we can divert our attention to more sophisticated and efficient tools. It is with this view that the rest of this project deploys the named-entity-recognition library spaCy, which is available to use in python via pip.

3.2 spaCy: an NER tool

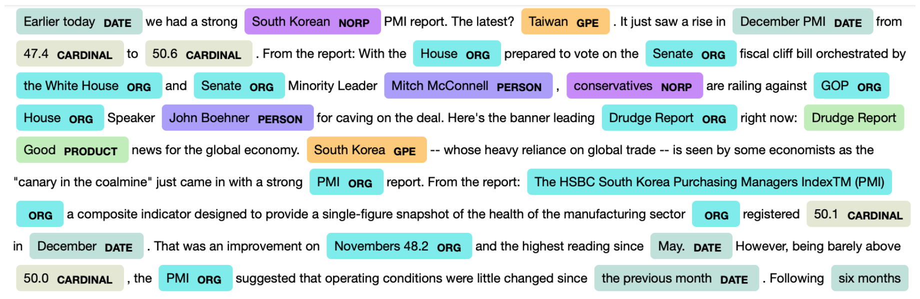

Earlier today DATE we had a strong South Korean NORP PMI report. The latest? Taiwan GPE . It just saw a rise in December PMI DATE from
47.4 CARDINAL to 50.6 CARDINAL . From the report: With the House ORG prepared to vote on the Senate ORG fiscal cliff bill orchestrated by
the White House ORG and Senate ORG Minority Leader Mitch McConnell PERSON , conservatives NORP are railing against GOP ORG
House ORG Speaker John Boehner PERSON for caving on the deal. Here's the banner leading Drudge Report ORG right now: Drudge Report
Good PRODUCT news for the global economy. South Korea GPE -- whose heavy reliance on global trade -- is seen by some economists as the
"canary in the coalmine" just came in with a strong PMI ORG report. From the report: The HSBC South Korea Purchasing Managers IndexTM (PMI)
ORG a composite indicator designed to provide a single-figure snapshot of the health of the manufacturing sector ORG registered 50.1 CARDINAL
in December DATE . That was an improvement on Novembers 48.2 ORG and the highest reading since May. DATE However, being barely above
50.0 CARDINAL , the PMI ORG suggested that operating conditions were little changed since the previous month DATE . Following six months

Figure 3: An example of spaCy's NER capabilities, run on an article from corpus

- In the above html rendering, NER from spaCy has been applied on a single article. Within the document, spaCy has clearly picked out relevant entities, including DATE, ORG, PERCENT, etc. This will form the backbone of entity recognition for all 3 chosen fields. To make this concrete, the entities will searched through as follows:
  - PERSON will be searched for CEOs.
  - ORG will be searched for companies.
  - PERCENT will be returned as the list of percentages used.
- The entire corpus has been extracted from the directory, and run through a sentencizer pipeline within spaCy. The below pipeline converts the data from just a list of strings (each article) into a flat list of sentences of the entire corpus. As a result, the remainder of the data processing is conducted at the scope of sentences.

```
processed = [nlp(article) for article in tqdm(content[0:len(content)])]
```

  - Warning: this takes 20 minutes minimum to run.
  - Processed holds the entire corpus of data (which are now spaCy.doc.docs, allowing spaCy methods like label_ and entity_).

3.3 Feature Engineering (v1.0)

Given that feature engineering for both CEOs and companies was essentially the same pipeline, they will be explained together.

- First, process (the list of spaCy.doc.docs) was filtered to only return sentences if they contained the right type of entity. (Refer above for clarification)
- These were converted into pandas DataFrames.
- **Feature Labelling**: a matching function was applied to the dataframes, which checked whether any of the training labels were substrings within each sentence in the corpus. In general, this is used to label the data as positive or negative samples.
- **Number of Capitals**:
- **Number of Words**: As written on the tin, this
- **Company Word**: If you made it this far and were anticipating a regex string, this function finds if there are any uses of the following common company attributes are present in the sample.

- Features above were calculated for both companies and CEOs, and then the entire dataset was put through a standard ML pipeline which features train/test splits, and classifier accuracy comparisons between a variety of increasingly confusing ML models. (See homew3ipnyb skeleton for more evidence of this).

```
if (bool(re.search(r'(Advisors|Partner|LP|Associate|Co|Group|LTD|AirLL|Management|
    Capital)',sentence))) return 1
```

3.4 Feature Engineering (v1.1)

3.4.1 TD-IDF

- Tf stands for 'term-frequency' while tf-idf means term-frequency times inverse document-frequency. This is a common term weighting scheme in information retrieval, that has also found good use in document classification.
- In attempt to improve TDidf explores and rank orders features by constructing weights corresponding to that features' utility in providing relevant information.
- In order to start using TfidfTransformer, first a CountVectorizer is run to count the number of words (term frequency), apply stop words, and then let it compute across the features. The lower the IDF value of a word, the less unique it is to any particular document.

4 Results

4.1 Percentages, using NER

- Conveniently, there is a built-in function for percentage recognition in spaCy. This makes it a relatively easy task to create a matching function that takes the values to search in, and the type of entity, and return a list of percentages found across the entire corpus.
- This is then extracted into a pandas DataFrame object, and saved as csv. It can be viewed as 'take-home_percent.csv' in the Github directory. Find the information on accessing Github repo in the Appendix.

4.2 Results (Machine Learning)

4.2.1 Performance Analysis Metrics

- Before engaging in figuring out models that are apt to be used, it is useful to first preempt to understand what classification success looks like. As it turns out, there are three main metrics used in Machine Learning performance analysis: accuracy, recall and precision.
- As it turns out, in this use case, accuracy can be a misleading metric when analysing highly imbalanced data. Instead, recall and precision form more informative metrics. The definitions for both are given in the graphic below.

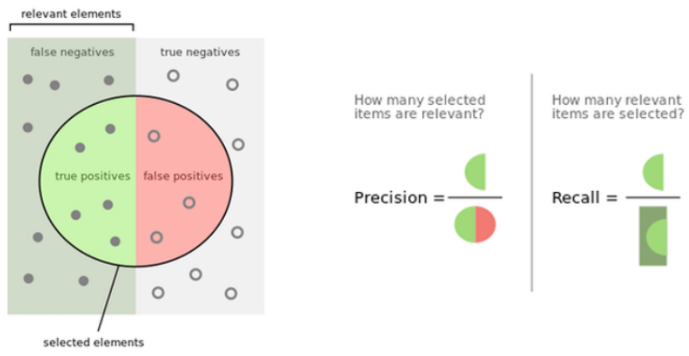


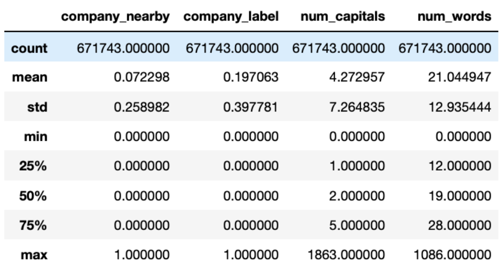
Figure 4: Definitions of Precision vs Recall

4.2.2 General Machine Learning Framework

- The general purpose machine learning used in this classification task is laid out below. The application can be found in text_ml.ipynb.
  1. Load the data, use scikit-learn's train-test split to make training and test sets.
  2. Use Feature engineering algorithm (vertorising with TD-idf)
  3. Run a logisitic regression, extract the classification report, and tune the model
  4. Extract correctly classified results
  5. Run NLP on these results, and extract ORG (companies) and PERSON (CEOs) and save to file.

4.2.3 Using Feature Engineering v1.0

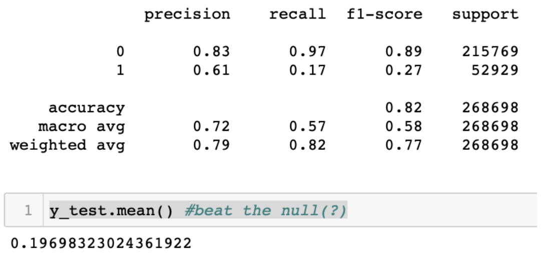
- The following table is the final list of CEO features, described.



	company_nearby	company_label	num_capitals	num_words
count	671743.000000	671743.000000	671743.000000	671743.000000
mean	0.072298	0.197063	4.272957	21.044947
std	0.258982	0.397781	7.264835	12.935444
min	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	1.000000	12.000000
50%	0.000000	0.000000	2.000000	19.000000
75%	0.000000	0.000000	5.000000	28.000000
max	1.000000	1.000000	1863.000000	1086.000000

Figure 5: Final List of CEO features

- The results from this indicate that roughly 7% of the sentences contain matches. Given that there were almost 350k sentences containing persons, 7% would be 24,500 sentences containing matched ceos from the training labels. This seems reasonable (sanity check).
- The classification rates for this Logistic Regression are pretty poor. The recall and precision are both really low for positive samples, and as a result this is barely better than just guessing.



```
              precision    recall  f1-score   support

           0       0.83      0.97      0.89    215769
           1       0.61      0.17      0.27     52929

    accuracy                           0.82    268698
   macro avg       0.72      0.57      0.58    268698
weighted avg       0.79      0.82      0.77    268698
```

```
y_test.mean() #beat the null(?)
```

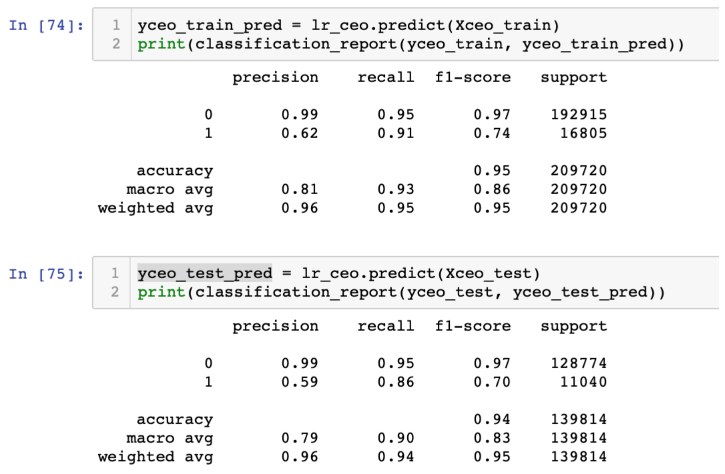
```
0.19698323024361922
```

Figure 6

4.2.4 Using Feature Engineering v1.1

- Following the dismal performance, I turned to TD-idf in order to improve the feature selection aspect of the problem. It seems that this approach is somewhat contentious because it means the actual features of the problem are less interpretable, but improve the performance of the overall classification. Given that the point of the problem is to correctly classify as many relevant samples as possible, I decided to deploy this method.

4.3 CEOs



```
In [74]: yceo_train_pred = lr_ceo.predict(Xceo_train)
         print(classification_report(yceo_train, yceo_train_pred))
```

```
              precision    recall  f1-score   support

           0       0.99      0.95      0.97    192915
           1       0.62      0.91      0.74     16805

    accuracy                           0.95    209720
   macro avg       0.81      0.93      0.86    209720
weighted avg       0.96      0.95      0.95    209720
```

```
In [75]: yceo_test_pred = lr_ceo.predict(Xceo_test)
         print(classification_report(yceo_test, yceo_test_pred))
```

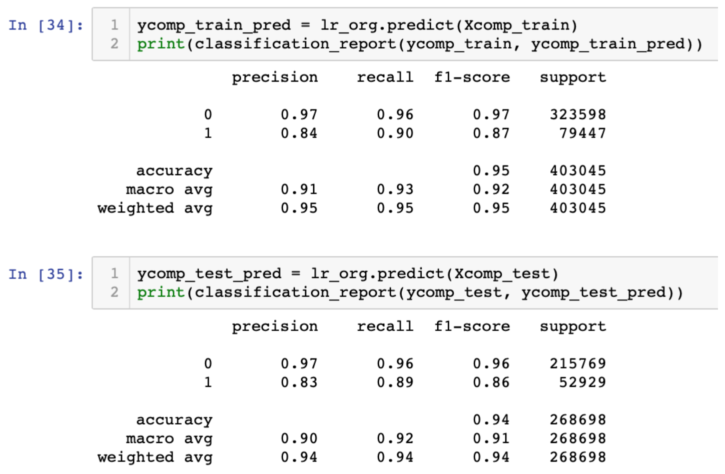
```
              precision    recall  f1-score   support

           0       0.99      0.95      0.97    128774
           1       0.59      0.86      0.70     11040

    accuracy                           0.94    139814
   macro avg       0.79      0.90      0.83    139814
weighted avg       0.96      0.94      0.95    139814
```

Figure 7: Classification Results for CEO

4.4 Companies



```
In [34]: ycomp_train_pred = lr_org.predict(Xcomp_train)
         print(classification_report(ycomp_train, ycomp_train_pred))
```

```
              precision    recall  f1-score   support

           0       0.97      0.96      0.97    323598
           1       0.84      0.90      0.87     79447

    accuracy                           0.95    403045
   macro avg       0.91      0.93      0.92    403045
weighted avg       0.95      0.95      0.95    403045
```

```
In [35]: ycomp_test_pred = lr_org.predict(Xcomp_test)
         print(classification_report(ycomp_test, ycomp_test_pred))
```

```
              precision    recall  f1-score   support

           0       0.97      0.96      0.96    215769
           1       0.83      0.89      0.86     52929

    accuracy                           0.94    268698
   macro avg       0.90      0.92      0.91    268698
weighted avg       0.94      0.94      0.94    268698
```

Figure 8: Classification Results for Company

5 Other Considerations

- **pickling**: Pickling is an incredibly useful and tactful technique to avoid re-running swathes of code that takes an inordinate amount of time. In doing so, it also prevents data loss in the case of a Kernel crash, which is common when dealing with large (but not big) data such as 'processed'. When pickled using pickle_me_hearties(), processed comes out to being almost 10GB stored on disk, and can be loaded again using return_me_hearties() To illustrate the point, some useful runtimes include:
  - Computation of processed: $>$ 20 minutes.
  - Storing processed to disk using pickle: $\sim$ 2 minutes.
  - Loading processed from disk using pickle: $\sim$ 1 minutes.
- Clearly, there is a major runtime improvement from storing data to file after it is computed once. In the cases where the recurrent neural nets were being trained and the jupyter kernel keeps dying, this removes a major hindrance of re-computation.
- **Parallel Computing**: With the advancement of computing, it is possible to split processing effort across multiple cores on a machine, and as a result this speeds up runtime. Using ipyparallel is a notable improvement to be made to this model, to cut notebook runtime from almost 45 minutes down to a more managable number.

6 Next Steps

- At the bottom of the homew3ipnyb, there is evidence of attempts to set up a recurrent neural net. This was attempted, but did not result in a successful train.
  - Mostly, this was a function of the jupyter kernel running out of memory and crashing. It seems as though adding in an entity within spaCy's framework, and training it on examples, would certainly result in a major boost to classification rates.
- It is also important to consider the case of multiple entities within a single sentence, which results in missed data. But, if these sentences are sanitised and replicated each one entity left in each time, this reduces the meaningfulness of feature extraction with TD-inf.

7 Appendix

1. Data can be found at these hyperlinks: Business Insider, 2013, Business Insider, 2014, Business Insider, Trained Test.
2. Source code can be found at this hyperlink: saif1457-github