Ariana Giorgi

2015

An Analysis of Methods for Information Retrieval

Advisor: Jonathan Stray

I. Introduction

In the field of data journalism, data comes in all forms - some data comes in the form of numbers on an Excel spreadsheet, while other data comes from the text fields of a PDF file. Different methods of data collection are used based on where the information is found and the amount of data there is to be sorted.

It becomes a challenging task when the information that a journalist wants is loosely organized. In 2013, Professor Sheila Coronel, as the Director of the Toni Stabile Center for Investigative Journalism at the Columbia University Journalism School, organized an assignment for a group of her investigative reporting students. Their assignment was to collect data from a database of court cases in order to identify larger trends. The cases document instances of international corruption and are provided by TRACE International - an organization with a mission to advance commercial transparency worldwide.

According to one of the students in the group, Caroline Chen (Columbia Journalism School Class of 2013), said that her group collected the data from the documents first and then decided on a story that could be written. They wrote a story on what they called the "bribe to benefit ratio," aiming to determine when it was worth it for a company official to accept the bribe. The story was never published.

Students viewed the cases by accessing them online through the TRACE database, and each case has a unique web address (see example here). Students were assigned different cases to read and fill out a form on Google for each case with information found from the case's site. When collecting the data, the students were always physically together. Every case was read three times; for each case, two different people entered the same information and a third person was settling the disagreements. The information was then stored in the form of an Excel spreadsheet. There are over 400 cases that were read and whose information the students entered manually into the Google form.

My goal for this research was to replicate the findings using methods like web scraping, natural language processing (NLP) techniques, and machine learning methods to see if the data could be collected faster with the same amount of accuracy. The challenge was to account for different types of information fields; some of these questions were classification problems, while others dealt more directly with information extraction within a certain space of the HTML document.

During the semester of this study, I was enrolled in both an NLP course and a machine learning course through Columbia University's Schools of Engineering and Applied Science. I used some of the concepts learned from these classes in my approach to answer questions. I had little prior experience to web scraping techniques en masse; much time was spent understanding different methods of information extraction.

In this study, I will detail the different types of information retrieval problems I dealt with and their unique challenges. I also hope to help identify and measure solutions for other journalists who come across the same challenges and questions in their own data.

II. Methodology

Each case is organized in the same simple HTML format with it's own unique URL address.

The HTML tags for any single case do not have unique IDs, so each section of the document must be addressed by the name within a certain h2 tag. Therefore, scraping the HTML document required more effort than simply extracting a field. I will go through the steps taken for each question I sought to answer. Once the desired data was extracted from each case, it was compared to the answer that the students concluded.

Data collection

File: grab_urls.sh

There were a total of 385 different cases that the students reviewed, and the IDs of the cases ranged from 1 to 465. Some numbers within that range either do not have a corresponding case or were not included in the Excel spreadsheet provided by the students. If a number does not have a corresponding case, the URL contains a simple document with the message "Case does not exist" - but the HTML can still be scraped. All URL addresses with IDs within the range of 1 to 465 were written to a .txt file.

Parsing the text files

File: parsefile.py

Python was used as the code of choice in this study. I am most comfortable with Python when using NLP techniques and familiar with data collection using the language. In order to more easily read in the HTML tags, I used BeautifulSoup software available online for download. Any text file that did *not* have a tag class corresponding to "msgBody" was considered. The only files that contained information in this tag were files corresponding to an ID without a case.

Perpetrator Company - Field Extraction

File: parsefile.py

In the text document, the only h1 tag corresponded to the perpetrator company for that case. Using BeautifulSoup, this was an easy extraction of just one line of code to retrieve the string within that tag.

Perpetrator Location of Origin - Field Extraction, Entity Interpretation

File: parsefile.py

The perpetrator location is under the h2 heading "Corporate Headquarters"; so using BeautifulSoup, I searched for the h2 section that matched that heading then read the string beneath it. The students were instructed to only name the country with the perpetrator company's headquarters is located, however, the document sometimes also includes the city or state.

Once I extracted the location, I then used a program called OpenCalais. OpenCalais is toolkit used to create metadata supported by Thomson Reuters and available for download online. The service is able to extract named entities, facts, and events from input text. I fed the location string into the OpenCalais function and then looped through the named entities that were found in the results. The answer that was paired with the entity "Country" was then assigned as the perpetrator location.

Perpetrator - Individual(s) - Field Extraction, Entity Interpretation

File: parsefile.py

Individual perpetrators were listed under the h2 heading "Entities/Individuals Involved" along with other companies or subsidiaries of the company that were involved in the case. Therefore, I had to find a way to differentiate between a person and a company. Each entity was listed on a new line, so I was able to interpret one line of the text below the heading at a line. Using the same technique as I did with the perpetrator location field, I entered the line of text as input into the OpenCalais function and extracted the answer that was returned if an entity type of 'Person' was found. This process was iterated through all of the lines beneath the h2 heading and if more than one perpetrator individual was found, a list was created.

Perpetrator Rank - Categorization, Entity Interpretation

File: parsefile.py

This question corresponded to the perpetrator individual's rank within the perpetrator company. The rank of an individual was listed in the same line and under the same h2 heading as the perpetrator individual from the previous category. When the OpenCalais function was run on the text from the line, I also checked to see if there was an entity identified as a "Position." If so, I extracted this text and then ran it through a separate test to identify what classification the position fit under.

On the Google form, the students were to choose the best fit for the perpetrator rank from the following seven categories: Founder, CEO/Chairman/President, Board of Directors, Vice President, Director of a Division, Company Official, or Other. Multiple ranks could be selected if two or more individuals fit under different categories. In order to identify the category that each position fit into, I searched for words within that string that seemed to classify that text correctly.

For example, if the words 'president', 'chairman', 'CEO', or 'Chief Executive Officer' were found in the text that was classified as a "Position," then the rank was determined to be CEO/Chairman/President.

<u>Result, Conviction & Investigation initiated by company - Yes/No Question, Bag-of-words Model and Mechanical Turk</u>

For these yes or no questions, I decided to try two different methods: the bag-of-words model, and Mechanical Turk. The goal was to compare the performance on the two techniques. I chose two different yes or no questions because I was looking for a sign of consistency.

Bag-of-words model - Result, Conviction

File: bagofwords.py

From the information provided on the case, the students also had to determine if the result of the corruption case was a conviction. They were to select from three possible answers: Yes, No, or Unclear from information given. Unlike the previous questions, this one presented a new challenge because the answer was not in a certain clear location. After reviewing some example cases, it seemed the answer could be most often determined by reading the paragraphs under the header “Enforcement Result”. There was no clear pattern on what text should be extracted, so I used the entire text.

I decided to go about finding the answer to this question in two different ways and then compare their accuracy. First, I decided to use the bag-of-words model that is frequently used as a simplification technique in NLP and information retrieval. Using this method, grammar and word order is ignored, but the number of times that a certain word occured in the in the “bag” or entire text is counted and makes up the feature vector that represents that document in a set of documents or, in this case, the paragraphs for each case in the set of all cases. A common way to ensure the most important words are considered more than common words is to additionally implement term frequency-inverse document frequency (TF-IDF) weighting. Stronger or rarer words will be given a higher weight in the text, while more common words like “the” or “and” are given a lower weight to decrease their influence upon the final outcome.

A helpful toolkit for this process is scikit-learn, which was created to aid machine learning processes in Python. Using the function 'TfidfVectorizer', scikit-learn easily converts a set of input texts to a weighted feature vector based of the words present in the whole set. In order to try and create a machine learning classifier that would predict future answers to the "Result, Conviction" question, I needed to extract the student answers as guideline labels for the vectors created in order to create training data for the classifier.

I decided to first consider the training cases to be any case with an ID between 1 and 100. Since this project is supposed to be a practical reflection on how a journalist can use code to read a large set of documents, I figured that a journalist doing a similar project may only want to dedicate time to reading 100 documents at most in order to find assign labels to the feature vectors for training. Reading any more than this amount would probably realistically defeat the purpose of a writing code to retrieve the information.

With the training data, I then trained two different classifiers - Linear Support Vector Classifier (SVC) and a Random Forest Classifier - in order to predict the labels from the untrained cases. The Linear SVC model uses lines to divide between the classifications for different labels, while the Random Forest Classifier is modeled on randomized decision trees that classify a vector by looking for the presence significant features.

After the classifiers are trained, I fed the classifiers the set of untrained feature vectors: the text from cases with an ID greater than 100. The prediction for each case was then translated as either “Yes”, “No”, or “Unclear”, as the categories on the Google form allowed.

I performed the bag-of-words model on training set sizes 200 and 300 as well.

The second way that I approached this problem was not using NLP or machine learning techniques. I simply extracted the text under the heading “Enforcement Result” (just as I did in the previous method) for each case and searched for the presence of the word ‘convict’ or ‘conviction’. If the word was included in the text, I predicted that the answer to “Result, Conviction” was “Yes”. Otherwise, I predicted “No”. The limitation of this method is that I did not account for the category “Unclear from information given.”

My intention was to compare the accuracy of both techniques.

Bag-of-words Model - Investigation initiated by company

File: bagofwords.py

I used the same method as described above. However, I did not compare against any extracted word like I did with the word “convict” for the previous category.

Mechanical Turk - Investigation initiated by company

File: create_csv.py

For this yes or no categorized question, I decided to try using Mechanical Turk. Mechanical Turk is a service provided by amazon where registered workers perform short tasks each for a small fee.

By reading through a few of the cases, it seemed most likely that the answer to this question would likely be under the h2 heading “How Conduct was Discovered”. However, some cases were missing text under this heading or the text was simply “Unknown.” In that case, I provided the text under the heading “Summary of Allegations” as a second attempt.

My goal was to just show a reader this portion of text from a case, along with the name of the perpetrator company, in hopes that they could determine whether or not the investigation was initiated by that perpetrator company. In addition to the “Yes” and “No” options, I also added an “Unclear” category to avoid the user from simply guessing the answer if it was unclear from the text provided.

In order to gauge the ability of an unrelated Mechanical Turk to identify the correct answer, I performed the Mechanical Turk task twice: once with one worker, and once with two workers. For one evaluation, I only looked at the answers from the first trial with one worker. For another, I compared answers from all three workers and selected the most common response for each question.

This method was designed to be a combination of code (in order to scrape the documents and set up the CSV file to submit to Mechanical Turk), human interpretation, and efficiency. There are instances where using an active audience or multiple workers to help refine and evaluate large amounts of data has been successful for a news an organization.

In 2013, ProPublica began a project called “Free the Files” in hopes of finding important stories in data from the FCC website concerning spending in the 2012 elections. The project was considered a success with over 1,000 people donating their time to looking through the files, which revealed patterns and important figures about political spending. Similarly, the New York Times created a photo mosaic in 2014 of the 43,000+ properties in Detroit that were nearing foreclosure in that year. In order to obtain a front-facing image of each property, the address was geo-located and a snapshot was generated and presented to a worker who would then confirm or fix the image so it would be a proper display.

Mechanical Turk - Result, Conviction

The same method was used as described above.

III. Results

Perpetrator Company - Field Extraction

File: test.py

Percent correct: 95.3% (the percentage of correct pairings between the student data and the information retrieval out of the total number of comparisons)

This instance of field extraction performed very well, likely because it was very clear where in the HTML document this information was located. It seems that for some instances, there was a mismatch between what was found by the code and what was recorded by the students because one overarching case was divided into multiple case IDs and required a bit more understanding beyond what the code could interpret from the h1 tag text.

Perpetrator Location of Origin - Field Extraction, Entity Interpretation

File: test.py

Percent correct: 98.0%

This category performed the best out of all of them, likely because for any location, OpenCalais can accurately identify the country.

Perpetrator - Individual(s) - Field Extraction, Entity Interpretation

File: test.py

Percent correct: 77.0%

Most of the ‘incorrect’ answers as determined by the code were really close to the actual answers, only missing one or two names from the list of individuals. This is likely because OpenCalais didn’t recognize the names. I would say that using this technique is still useable for a project like this, potentially as a way to quickly record at least a way to identify certain cases by a few names, or at least the most prominent ones.

Perpetrator Rank - Categorization, Entity Interpretation

File: test.py

Percent correct: 29.4%

Unlike perpetrator individuals, the collection of perpetrator rank answered performed much more poorly. This could be because OpenCalais wasn’t able to recognize certain descriptions as a “position,” and therefore, no information would have been extracted for that field. Possibly a way to improve the results would be to take *any* text after the perpetrator individual is identified and see if that matches a category for perpetrator rank. Also, the method of just matching the description with a word in the perpetrator rank categories could be flawed as well; there may be multiple ways to describe a position without using the text identified in the category title.

Result, Conviction - Yes/No Question, Bag-of-words Model, Mechanical Turk

File: bagofwords.py, create_csv.py

Bag-of-words, Training set size = 100

Linear Support Vectors Classifier percent correct: 60.0%

Random Forest Classifier percent correct, mean from sample: 56.0%

Bag-of-words, Training set size = 200

Linear Support Vectors Classifier percent correct: 54.3%

Random Forest Classifier percent correct, mean from sample: 56.1%

Bag-of-words, Training set size = 300

Linear Support Vectors Classifier percent correct: 66.7%

Random Forest Classifier percent correct, mean from sample: 68.7%

Identifying “convict” percent correct: 49.3%

Mechanical Turk

One worker percent correct: 47.8%

Three workers percent correct: 32.3%

Among the three different methods observed, the linear support vector model returned the highest percentage of matches with the answers provided by the students. This model likely performed poorly compared to its potential because the number of features (unique words among the set of training documents) is significantly higher than the number of samples. It's recommended for best performance to have the number of samples exceed the number of features. Therefore, the result might be improved with more samples or by limiting the number of features possibly by only using the features that have the highest weights in a certain case.

Because the random forest classification method is in fact random, the accuracy in predicting the correct answer can change every time you call it. After ten tests, the random forest classifier produced accuracies within the range of 0.439 to 0.658 and with a mean of 0.560.

The training set size made little difference, as the training set size of 300 gave the best results.

It's not surprising that simply by trying to identify the word "convict" or "conviction" was successful less than half of the time because there can be many ways to say that there was or was not a conviction.

The Mechanical Turk method was not as successful as the bag-of-words model or the extraction of the word "convict", and taking the most common answer of all three workers that responded produced a result that was worse. It’s possible that not enough information was given for the context of the question, or perhaps the workers answering the question did not have enough knowledge of the different ways to say that the result of a case was conviction - since the poor result of the extraction of “convict” showed that just looking for the word is not a reliable method in itself.

<u>Investigation initiated by company - Yes/No Question, Bag-of-words Model, Mechanical Turk</u>

File: bagofwords.py, create_csv.py

Bag-of-words, Training set size = 100

Linear Support Vectors Classifier percent correct: 72.2%

Random Forest Classifier percent correct, mean from sample: 68.5%

Bag-of-words, Training set size = 200

Linear Support Vectors Classifier percent correct: 68.6%

Random Forest Classifier percent correct, mean from sample: 65.7%

Bag-of-words, Training set size = 300

Linear Support Vectors Classifier percent correct: 71.0%

Random Forest Classifier percent correct, mean from sample: 68.7%

Mechanical Turk

One worker percent correct: 72.0%

Three workers percent correct: 76.8%

Over all training set sizes, the bag-of-words model performed consistently, and SVC proved to be a better model.

Mechanical Turk performed similarly to the bag-of-words model.

One reason why there might have been mismatch between the answers provided by students and those submitted on Mechanical Turk is because it's possible the context provided for the Mechanical Turk users was not enough to understand what happened in the case or the portion of the document that they were given did not contain the answer. A solution to this might have been to provide the whole document as background for the Mechanical Turk reviewer. However, if given more information to process, it's possible that this could jeopardize their focus or they could become unclear where to look for the answer.

Another way to potentially improve the accuracy of the Mechanical Turk results (assuming the students were correct and understood their material thoroughly) could be to require more than one Mechanical Turk worker to answer each question; in this experiment, I only required one worker per question. The journalist could decide to use the average answer received or to interpret discrepancies found among answers as an indication that the context provided for the case was unclear or confusing.

It should also be noted that Mechanical Turk was able to provide all answers within 50 minutes, which is a typically a much shorter time that it would take for a journalist to write code to interpret the text with the same level of accuracy.

Comparison of Yes/No Question Methods

	Bag-of-words						Mechanical Turk		Extraction
	Train =100		*200*		*300*		*1 worker*	*3*	**-**
	SVC	*RF*	*SVC*	*RF*	*SVC*	*RF*	-	-	**-**
Result, Conviction	60.0	56.0	54.3	56.1	66.7	**68.7**	47.8	32.3	49.3
Investigation initiated by company	72.2	68.5	68.6	65.7	71.0	68.7	72.0	**76.8**	-

(Values in terms of percent of answers that matched with those provided by students)

Time Consumption

I made sure to keep track of my time spent on this project. In total, writing the code and researching the best methods and implementation took me 30 hours.

Emmanuel Felton (Columbia Journalism School Class of 2013) said, “We had 6 people reading about 30 or so cases a week, my guess is it took each of us about 4 hours a week. We did it for 8 weeks.”

Felton estimates that it took a total of 28 hours for each student to read all the cases and collect the information manually. That is 168 total hours.

IV. Conclusion

The method with the best performance was the field extraction of the perpetrator company location. This is because this information was the easiest to identify in the document and is clearly communicated in the cases. If all of the desired information were in this format, it would be a clear advantage in terms of saving time to write a script that will pull the text in the same manner that I have done here.

The SVM and random forest classification models were, in general, unreliable. It seemed the SVM model result slightly improved with a greater number of training samples, though this method requires more time and focus from the journalist to label the training documents. I would only recommend relying on this method if the context of where the answer to the question lies within the document is substantially small in terms of number of words and clear in terms of location.

Using Mechanical Turk did not perform as well as I would have expected. For the category "Investigation initiated by company," the three-worker Mechanical Turk method performed best, but for the "Result, Conviction" category, it performed the worst. It seems that unless the desired information is presented clearly to workers, then the Mechanical Turk model will need a human checker to verify consistency.

It would also be interesting to see how well Mechanical Turk performs for a type of question besides a categorization.

For a data journalist who is already experienced in writing methods of web-scraping and NLP techniques, they would likely spend less time than I did writing the code for this project, which is still less than what Felton estimates the time that he and the other students spent reading the cases. In the case of extracting information where the answer is easy for a computer to locate, writing code to solve the problem is highly recommended.

V. CD

Files available on included CD:

PDF version of this document

bagofwords.py

calais.py

calaisREADME.txt

create_csv.py

files Folder (contains txt files of all cases)

grab_urls.sh

parsefiles.py

trace.xlsx

test.py

turk_results_convict1.csv

turk_results_convict2.csv

turk_results_initiated1.csv

turk_results_initiated2.csv

VI. Sources

Caroline Chen, cchen501@gmail.com, 650-868-1213

Emmanuel Felton, emmanuel.c.felton@gmail.com