Natural Language Processing of DOJ Press Releases – Final Paper

Matthew Quander
Dept. of Computer and Information Sciences
Towson University
Towson, MD, USA
mquand1@students.towson.edu

Abstract— **The United States Department of Justice prosecutes criminal cases and imposes penalties for civil violations of the law. In this paper, I analyze a data set from Kaggle consisting of the press releases from the Justice Department's justice.gov/news website. As a former contractor to the Department, I contributed to some of these cases. By implementing Natural Language Processing tools, and Machine Learning and Clustering algorithms, I hope to uncover trends in the cases brought by the Justice Department.**

Keywords–- **natural language processing, machine learning, clustering, text analysis, news, article, government**

I. INTRODUCTION

On a weekly basis the U.S. Department of Justice provides press releases to the public on their justice.gov/news website. These press releases include information about active cases (previously made public), new indictments, new civil complaints filed, and any other case related information that is ready to be made available to the general public [2]. From 2009 to 2018, the Justice Department was under two different administrations and was led by three different Attorneys General (two from one Administration). Given that these officials are appointed by the president, a perception of political influence over the Department is often suspected. Nothing could be more evident of this than the cases brought in court. By using many of the tools in Python's libraries, I analyzed the Kaggle data set Department of Justice 2009-2018 Press Releases [1]. The purpose of this analysis was to uncover any patterns or correlations to the types of cases brought by the Department during this time period. I, therefore, pose the question:

What has been the trend in cases prosecuted by the Justice Department, from 2009 to 2018?

II. DATA SET DESCRIPTION

The Kaggle data set consists of 13,087 press releases from the Department of Justice justice.gov/news website, ranging from 2009 to 2018 [1]. The Justice Department regularly publishes press releases to inform the general public of active criminal prosecutions, civil enforcements, initiatives and projects, and any other non-sensitive information [2]. Using a Python script, the data set provider scrapped certain aspects of the data from the website and organized them into JavaScript Object Notation (JSON) format [1]. The 52MB preprocessed JSON file contains the following fields and descriptions:

- Id: press release number
- Title: title of the release
- Contents: the full text of the press release
- Date: date the press release was posted on the website
- Topics: a set of topics covered by the release, if provided
- Components: a set of the agencies and departments involved in the particular release, if provided

[1]

In reviewing the JSON file in Visual Studio Code, I observed that all of the fields contain string formatted data. This should not lead to increased difficultly in analyzing the data, as Natural Language Processing is implemented on such data types. Further, the "id" field was not significant in the analysis since it consists of an arbitrary, identifying number.

The most text heavy sections are "title" and "content." The "title" field could be useful is scanning for preliminary information, prior to digesting the "content" field which is the typical length of an article. Given the length of the "content" section, processing this section took up the majority of the code's run time. The "date" field provided a useful context in sequencing the cases brought by the Department. If any trends are detected, they could be investigated further with respect to the year.

Lastly, the "topics" and "components" sections, when populated, provided insight prior to scanning the more lengthy "contents" section. For example, some press releases are tagged with a category and/or related agencies. Occurrences of certain topics and related agencies could be counted to measure any changes in trend over time.

III. Exploratory Data Analysis

Upon reading in the JSON file to a data frame, I used the `head` function in Python's pandas package to view the first few records with the respective fields. The pandas `read_json` function populates an index field from 0 to the last record. By using the `isnull` and `sum` functions on the data frame object, I easily calculated the amount of null (NaN) values in the data set, a total of 227 all of which were in the "id" field. This, however, did not account for any of the blank values. In order to count the empty string values, I ran the `sum` function on the result of the Boolean expression which tested each column with an empty string. This found 2 rows in "contents" that were blank. The "topics" and "components" sections were formatted as lists of strings, so to check for missing data for each column I used the `str` object and the `len` and `eq` functions. I was able to find those lists that were of length 0, a total of 8399 for "topics" and 18 for "components."

```
Blank Field Values:
id:  0 (227 null)
title:  0
contents:  2
date:  0
topics:  8399
components:  18
```

TABLE I. Blank Field Values

Since ~64% (8399/13087) of the data has missing values for the "topics" field, this field will be dropped and not considered in the analysis. Conversely, since the "contents" and "components" fields only have a few missing values (~0.015% and ~0.138%, respectively), the corresponding records can be discarded from the data set without having a significant impact.

Next I processed the text of the "contents" field by applying a set of Python functions. After importing `SnowballStemmer` and `stopwords` packages from the `nltk` library, I applied the `stem` function to remove the stop words and get the stem for each word. I also removed punctuation and extra spaces using the `sub` method in the `re` Regular Expression library. Lastly, I converted all of the text to lower case using the `lower` function. Since the "contents" field consists of one long string for each record, processing it with these functions will allow for better text analysis.

Since the data set consists of many records, each with a lengthy "contents" field, I decided to split the data frame by year to better organize it. I did this by scanning the "date" field and applying the `contains` method, passing a string of each year from 2009 to 2018. Each of these sub-data frames were stored in separate variables with the indexes reset. This will also allow for more efficient analysis to uncover any trends that occurred over the years.

Since I will be focusing primarily on the cases brought in court, I declared a set of stem words to represent the Justice Department's litigating entities. These are meant to capture the following divisions: Antitrust, Civil, Criminal, Environmental and Natural Resources, National Security, and Tax. By scanning the "contents" and "components" text, this set of words can be used to identify the type of case brought. Other press releases that are administrative announcements from the Department won't be considered. By declaring these Justice Department entities, I built the below frequency distribution of the press releases for each year:

```
2009 freqDist:
{'Antitru':  71,  'Civil':  334,  'Crimin':  275,
'Environ': 122, 'Security': 53, 'Tax': 156}
2010 freqDist:
{'Antitru':  103,  'Civil':  351,  'Crimin':  341,
'Environ': 98, 'Security': 33, 'Tax': 136}
2011 freqDist:
{'Antitru':  113,  'Civil':  383,  'Crimin':  478,
'Environ': 94, 'Security': 38, 'Tax': 220}
2012 freqDist:
{'Antitru':  92,  'Civil':  407,  'Crimin':  431,
'Environ': 91, 'Security': 25, 'Tax': 196}
2013 freqDist:
{'Antitru':  71,  'Civil':  380,  'Crimin':  371,
'Environ': 82, 'Security': 11, 'Tax': 227}
2014 freqDist:
{'Antitru':  88,  'Civil':  376,  'Crimin':  419,
'Environ': 68, 'Security': 31,  'Tax': 230}
2015 freqDist:
{'Antitru':  82,  'Civil':  420,  'Crimin':  343,
'Environ': 94, 'Security': 156, 'Tax': 276}
2016 freqDist:
{'Antitru':  105,  'Civil':  357,  'Crimin':  306,
'Environ': 74, 'Security': 130, 'Tax': 292}
2017 freqDist:
{'Antitru':  81,  'Civil':  292,  'Crimin':  347,
'Environ': 80, 'Security': 99, 'Tax': 256}
2018 freqDist:
{'Antitru':  33,  'Civil':  204,  'Crimin':  265,
'Environ': 37,  'Security': 76, 'Tax': 125}
```

TABLE II. Frequency Distribution

What is more revealing is the combined graph, overlaying the frequency distributions of these Justice Department entities. From the above results, I built a separate data frame to create an overlapping visual of the frequency distribution. The below graph suggests certain trends in the cases brought by the Department. Most notably, the Tax Division saw a decrease in the amount of press releases from 2016 to 2018, with a steeper decline beginning in 2017. This follows a sharp spike in Tax Division press releases from 2014 to 2015. Similarly, the Antitrust Division, which typically had a lower frequency during this time period, also saw a decrease from 2016 to 2018. The Civil Division and the National Security Division saw consistent decreases in the amount of press releases from 2015 to 2018. An interesting observation that I made was that the inflection point for the frequencies of these divisions occur during or near the election year of 2016. Another observation from graphing the data frame is that all 6 of the divisions have a decreasing frequency from 2017 to 2018. This is notable since a new administration was in control at this time.

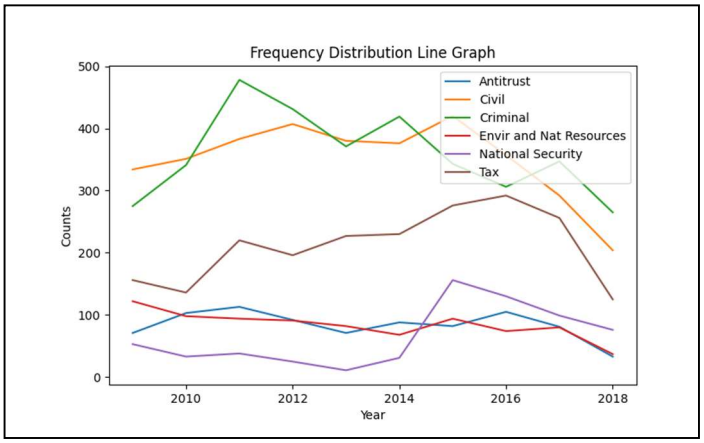

Fig. 1. Frequency Distribution Graph.

One of the many features that Python provides is that of creating word clouds, which give a visual display of the commonly used words in a text. To build this word cloud, I first had to process the text of the "contents" fields of each sub-data frame. I began by joining the strings of each row of the sub-data frame using the `cat` function and `str` object, separated by an empty string. I then passed this string to the `word_tokenize` function to tokenize it. To get a more accurate depiction I removed any words that had 3 or less characters, as well as common words related to the Department that were used in the press releases (e.g. "attorney", "department", "justice", etc.). After configuring the plot size, font size, and maximum amount of words to display, the `generate_from_text` function was used [4]. A few of the resulting word clouds are below:

Fig. 2. Word Cloud of 2011 Press Releases "contents"

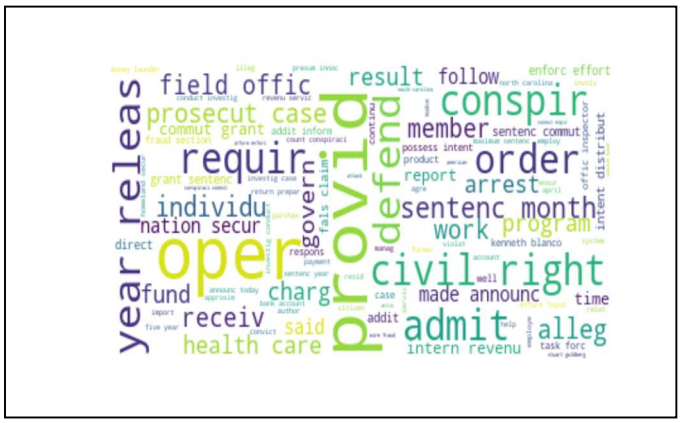

Fig. 3. Word Cloud of 2017 Press Releases "contents"

IV. Experiment

One common practice in analyzing a set of documents is to compute the Term Frequency – Inverse Document Frequency (TF-IDF). The Term Frequency is the number of times a word appears in a document, divided by the total number of words in that document [5]. The Inverse Document Frequency is the log of, the total number of documents divided by the number of documents that contain a certain word *w*, this determines the weight of rare words across the set of all documents [5]. The TF-IDF is then calculated by multiplying the Term Frequency by the Inverse Document Frequency [5].

$$TF = \sum w/d$$

w=word, d=total words in document

$$IDF = \log(N/D(w))$$

N=number of document, D(w)=number of documents containing word w

$$TF\text{-}IDF = TF * IDF$$

The TF-IDF score for a word will initially increase when that word has a high frequency within a document, however, as that word is used more across the set of documents its score will decrease. The decrease is due to the IDF factor of the formula, since a word that is frequent across the set of documents has less importance than a word that's only frequent within one document [3]. Therefore, a word with a higher TF-IDF score will be more relevant for particular topics of interest.

Using Python's `sklearn.feature_extraction.text` library, I imported the `CountVectorizer`, `TfidfVectorizer`, and `TfidfTransformer` packages to compute the TF-IDF. By first creating `TfidfTransformer` and `CountVectorizer` objects, I applied the `fit_transform` function of the `countVectorizer` object to the cleaned

"contents" field and stored it in a variable. I then passed that variable to the `TfidfTransformer` object's `fit_transform` function and stored that in a variable. A new data frame was then created from this variable. A display of the highest TF-IDFs and their respective words can be seen below:

TF-IDF 2010	
vehicl	0.494134
emiss	0.458472
test	0.290667
air	0.235928
nevada	0.204033
clean	0.156728
falsifi	0.143278
falsif	0.122377
analyz	0.116405
pass	0.113070

TABLE III. TF-IDF 2010

TF-IDF 2016	
visa	0.525740
harbor	0.325162
alien	0.321206
conspiraci	0.227706
student	0.218689
unnj	0.200166
profit	0.190308
new	0.165527
jersey	0.153425
fraud	0.145062

TABLE IV. TF-IDF 2016

TF-IDF 2018	
site	0.480124
epa	0.290745
cleanup	0.283443
centredal	0.236808
river	0.224689
manor	0.207207
superfund	0.206141
emhart	0.177606
settlement	0.159827
woonasquatucket	0.148005

TABLE V. TF-IDF 2018

For the year 2010, the word "vehicle" had the highest TF-IDF score of ~0.49, followed by "emission," "test," and "air." This suggests that the case related to the vehicle emissions scandal was unique with respect to the other articles in the sub-data frame of 2010. In analyzing the TF-IDF scores for 2016, "visa" has the highest of ~0.53. The next highest words are "harbor," "alien," "conspiracy," and "student". The correlation of these terms indicates that there was a unique case involving fraudulent student visas. The term "fraud" is also in the list, however, with a lower value. This is likely because "fraud" is used frequently in the data set, therefore having a lower IDF number. Finally the top TF-IDF terms in the 2018 sub-data frame, "site," "epa," and "cleanup" provide some insight into what unique case or cases occurred that year. The Environmental Protection Agency (EPA) works with the Justice Department's Environmental and Natural Resources Division to help prosecute violations of the nation's environmental laws. Furthermore, the term "cleanup" suggests that the case was related to a hazardous waste spill on natural land.

Next, I used a Support Vector Machine in an attempt to predict the "components" field based on the text of the "contents" section. Since the "components" contained a wide variety of Department entities, some of which are not litigating units, I created a sub-data frame with only the prosecuting divisions: Antitrust, Civil, Criminal, Environmental and Natural Resources, National Security, and Tax. After resetting the index and dropping the old columns which were cleaned into new columns, I called the `model_selection`'s `train_test_split` function with the "cleaned contents" and "cleaned components" fields passed as parameters and the test size set to 0.3. This function, a part of the `sklearn` library, randomly splits the data frame into a training and test set [6]. With a test size value of 0.3, there will be a 70-30 training-test split. In order for the SVM algorithm to run successfully on the data set, the target variable field has to be converted to unique integer values to represent the possible classification labels [6]. Therefore, an object of type `LabelEncoder` from the `sklearn.preprocessing` package was created to call the `fit_transform` function on the train and test labels. A TF-IDF is than calculated for the training and test attributes to help build a better model. From the `sklearn` library, I imported the `svm` module to use the `SVC` (support vector classification) to build a hyperplane and fit the TF-IDF training set attributes and the training set labels. This `svm` object was stored in a variable, which I used to predict the labels for the test set attributes by calling the `predict` function. Using the `sklearn.metrics`'s `accuracy_score` function and multiplying by 100, I obtained the accuracy percentage of the prediction which was ~99.3%.

Lastly, I performed clustering on the data set to group the articles' text into similar groups. This was done by first using a `TfidfVectorizer` object to call the `fit_transform` function on the "cleaned contents" field. Creating an object of type `MiniBatchKMeans`, from the `sklearn.cluster` library, I used a script to obtain the top keywords [1]. It takes the "cleaned contents" field data and groups it by the cluster object previously declared. It then loops through the rows of the sub-data frame and sorts. This was run with a predetermined value of k = 10 clusters. The output is below:

Cluster 0
price,consum,acquisit,settlement,workshop,depart,merger,propos,competit,antitrust
Cluster 1
indict,racket,isil,polic,charg,attorney,terrorist,murder,member,gang
Cluster 2
polic,indict,traffick,inmat,assault,attorney,offic,civil,victim,right
Cluster 3
imag,ceo,safe,children,project,childhood,sexual,exploit,pornographi,child
Cluster 4
attorney,injunct,file,fals,refund,incom,ir,prepar,return,tax
Cluster 5
access,depart,civil,vote,employ,ada,disabl,hous,right,discrimin
Cluster 6
investig,attorney,briberi,govern,offici,armi,compani,bribe,crimin,contract
Cluster 7
claim,bill,oig,medic,patient,hhs,fraud,care,health,medicar
Cluster 8
communiti,general,nation,offic,justic,law,drug,state,depart,attorney
Cluster 9
conspir,conspiraci,account,attorney,crimin,charg,tax,financi,bank,fraud

TABLE VI. DATA SET CLUSTERS

In reviewing the output, the success of Python's clustering algorithm can be observed. It should be noted, however, that some words appear to be better clustered than others. For example, Cluster 2 has terms that relate to "trafficking," "inmate," "civil," and "right," which covers a wide variety of types of cases. Similarly, Cluster 8 has many general terms, such as "general," "nation," "justice," and "attorney," which make it difficult to determine the case type. On the other hand, Cluster 3 clearly deals with cases of child exploitation and Cluster 7 deals with cases of healthcare fraud.

V. RESULTS ANALYSIS

From the experiments performed in this project, much information can be derived. The Term Frequency – Inverse Document Frequency algorithm revealed the press releases of cases which were reported on less throughout the data set. The algorithm achieved this by giving a higher score to the words that were frequent in a document, but were infrequent throughout the entire data set. It can then be inferred that these cases receive less attention by the Department as compared to others.

The Support Vector Machine algorithm proved successful in predicting the "components" field for the given "contents" of a press release article. By splitting the data set into a training and test set, the algorithm was able to learn from the text in the "contents" field to determine the appropriate "components" label. This resulted in a ~99.3% accuracy.

In analyzing the output of the clustering algorithm, this appeared to be the least informative of the methods used. Although the case type of some of the clusters was easily discernable, others were inconclusive. This could be due to certain words being used many times throughout the data set. Such general words invite the possibility of the algorithm being manipulated and clustering these terms inaccurately.

VI. CONCLUSION

In conclusion, the Term Frequency – Inverse Document Frequency, Support Vector Machine, and K-Means Clustering algorithms proved effective in gaining information about the data set. Although some were more effective than others, the Natural Language Processing tools built into them gave an advantage in analyzing the 13,087 records. Ironically, the most useful tool in answering the posed question was the frequency distribution as displayed in Fig. 1. By graphically presenting the yearly sums of press releases for each division, the trends became evident. From 2016 to 2018 there had been an aggregate decline in press releases, implying a decline in cases brought by the Department. Although this trend is has been detected, the actual cause remains unknown. As many could speculate about political influence, alternative causes could be related to funding, cooperation agreements, the lacking of evidence to bring a case, or other various reasons. In order to answer this additional question of causation, further study would be required.

REFERENCES

[1] *Department of Justice 2009-2018 Press Releases*, Kaggle, July 29, 2018. [Online]. Available: https://www.kaggle.com/jbencina/department-of-justice-20092018-press-releases

[2] Accessed: Apr. 8, 2021. [Online]. Available: https://www.justice.gov/news

[3] M. Borcan. "TF-IDF Explained and Python Sklearn Implementation." Towards Data Science. https://towardsdatascience.com/tf-idf-explained-and-python-sklearn-implementation-b020c5e83275 (accessed May 2, 2021).

[4] C. Silva. "Headlines Articles Analysis and NLP." Towards Data Science. https://towardsdatascience.com/headlines-articles-analysis-and-nlp-4013a66dbac (accessed Apr. 8, 2021).

[5] Qaiser, Shazad & Ali, Ramsha. "Text Mining: Use of TF-IDF to Examine the Relevance of Words to Documents," in *International Journal of Computer Applications*, July 2018. [Online]. Available: https://www.researchgate.net/publication/326425709_Text_Mining_Use_of_TF-IDF_to_Examine_the_Relevance_of_Words_to_Documents

[6] G. Bedi. "A guide to Text Classification(NLP) using SVM and Naïve Bayes with Python." Medium. <link> (accessed May 5, 2021).