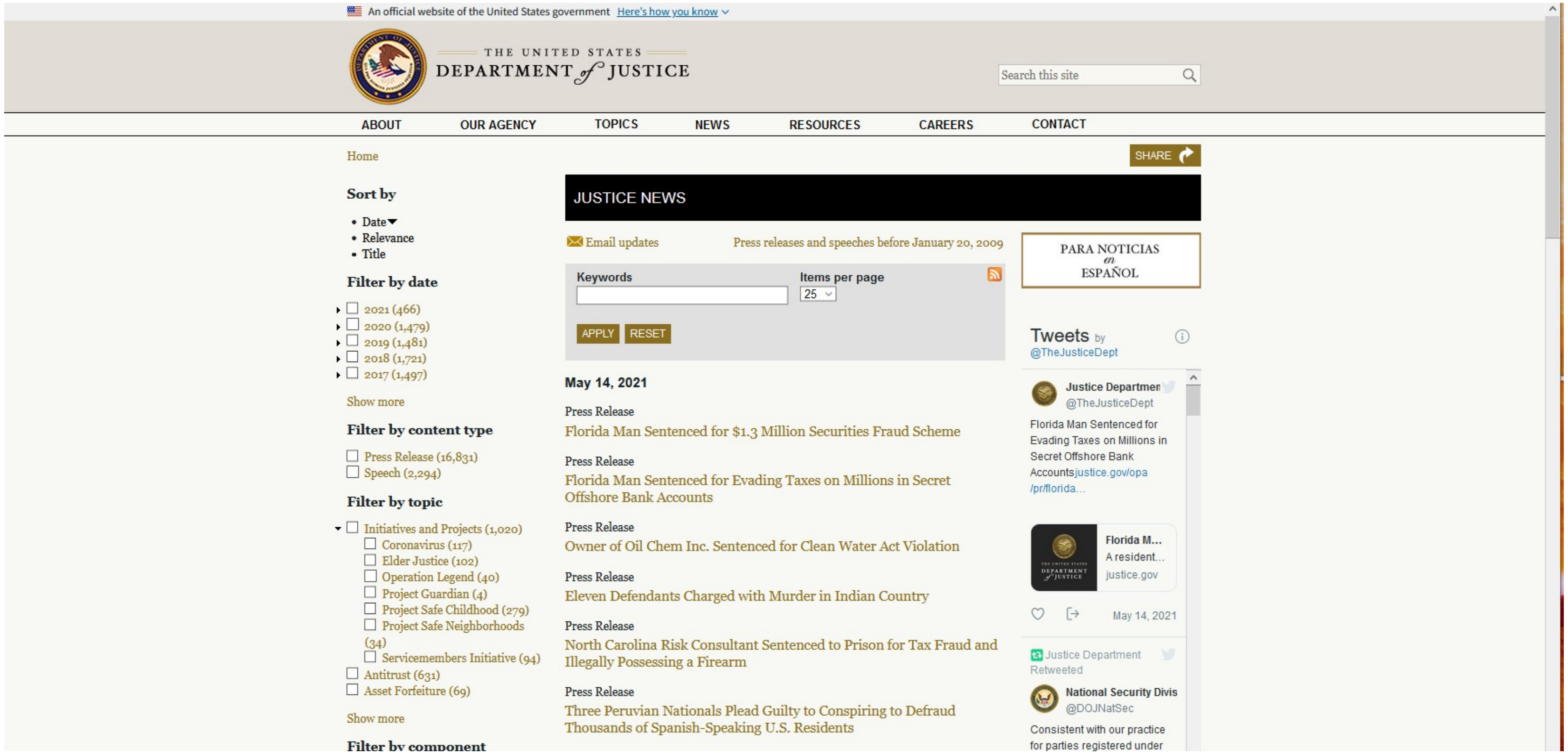
Natural Language Processing of DOJ Press Releases

by Matthew Quander

Introduction

- US Department of Justice regularly publishes press releases to inform the general public about active criminal prosecutions, civil enforcements, initiatives and projects, and any other non-sensitive information
- From 2009 to 2018, the DOJ was under 2 different administrations and was led by 3 different Attorneys General (2 from 1 administration)
- Since AGs are appointed by the president, political influence over the Department is often suspected
- *What has been the trend in cases brought by the Department from 2009 to 2018?*

justice.gov/news



- Kaggle dataset: Department of Justice 2009-2018 Press Releases
- As a former contractor to the Financial Fraud Section of the DOJ, I contributed to some of these cases
- By using Python's Natural Language Processing tools, and its Machine Learning and Clustering algorithms, I hope to uncover trends in the cases brought by the Justice Department within this timeframe

Dataset Description

- Kaggle dataset consists of 13,087 press releases from justice.gov/news, ranging from 2009-2018
- Data set provider scapped the data using a Python script and organized it into JSON format:
    - **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

- All fields populated with string data, common for NLP to operate on
- “id” field arbitrary number, “title” and “contents” fields most text heavy
- “date” field provided useful context in sequencing when cases brought and to investigate trends
- “topics” and “components” could provide insight into the type of case prior to processing the “contents” of the record

Exploratory Data Analysis

- used panda's `read_json` function to create a dataframe
- used several functions to check for null and missing values:

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

- ~64% of the records have missing values in “topics” attribute
- ~0.015% missing “contents”; ~0.138% missing “components”

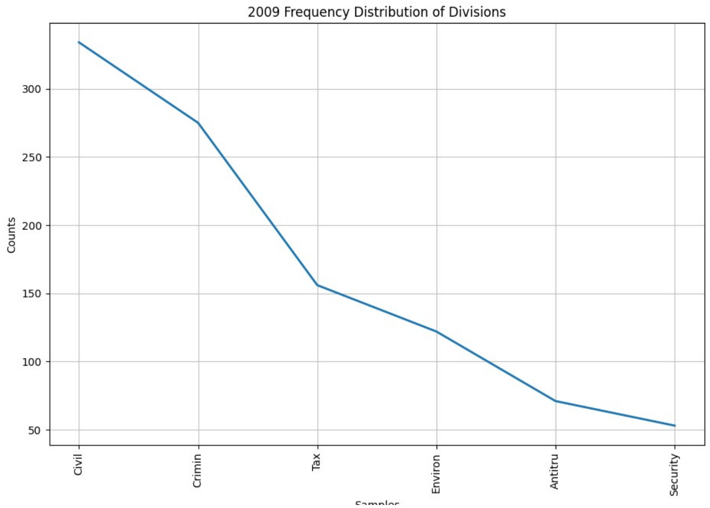
- Importing `nltk` library, used `stem`, `sub`, and `lower` functions to process the string of the “contents” field and remove stop words

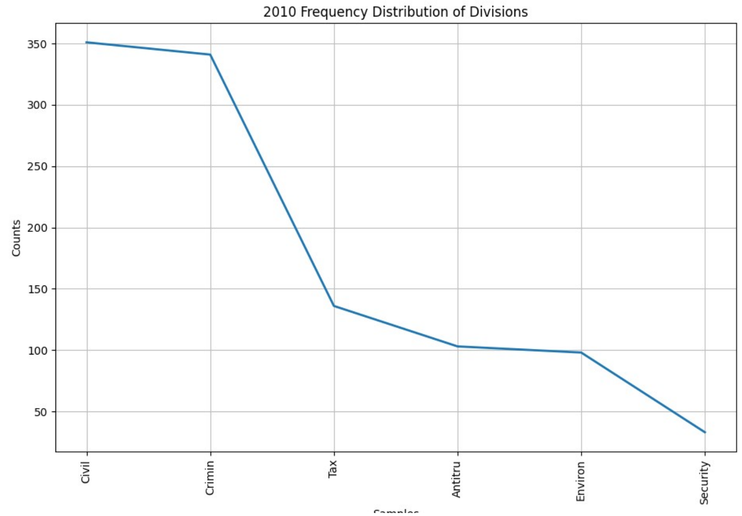
```
articles_df['cleaned content'] = articles_df['contents'].apply(lambda x: " ".join([stemmer.stem(i)
                                          for i in re.sub("[^a-zA-Z]", " ", x).split()
                                              if i not in stop_words]).lower())
```

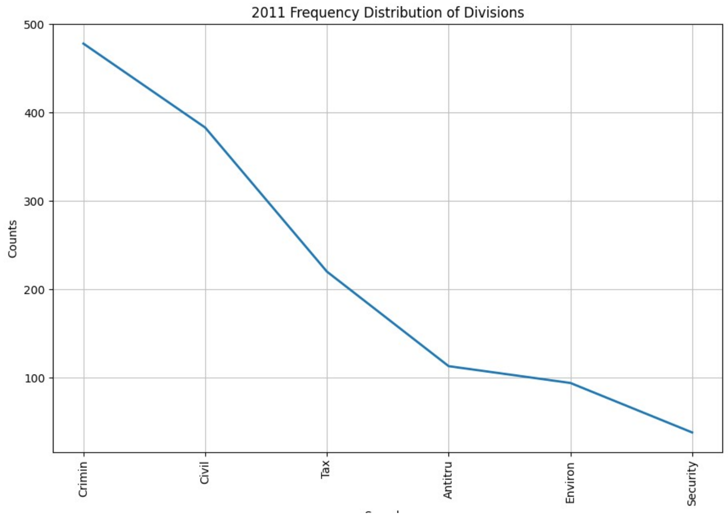
- Created 10 separate dataframes by year, 2009-2018
- Declared set of litigating divisions: {Antitrust, Civil, Criminal, Environmental and Natural Resources, National Security, and Tax}
    - ignored administrative divisions in the press releases (e.g. JMD)

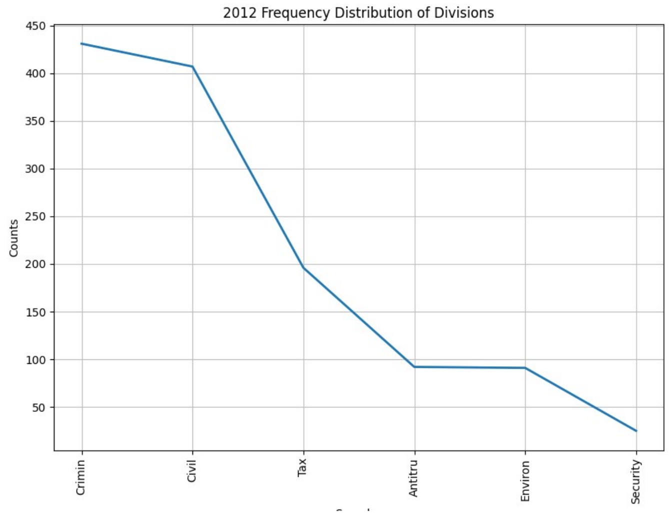
Frequency Distribution

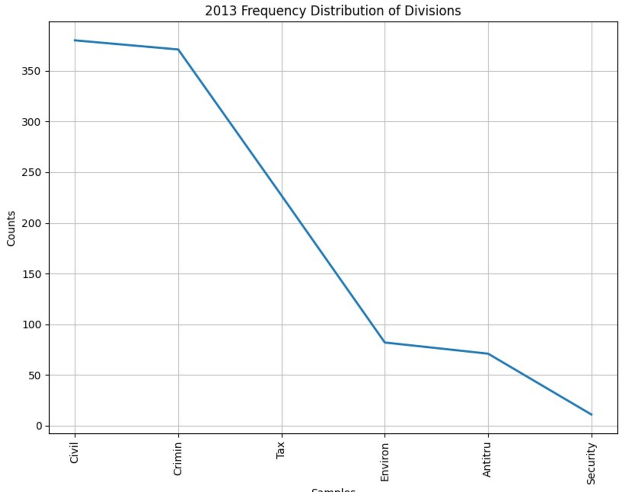
```
2009 freqDist:
{'Antitru': 71, 'Civil': 334, 'Crimin': 275, 'Environ': 122, 'Security': 53, 'Tax': 156}
2010 freqDist:
{'Antitru': 103, 'Civil': 351, 'Crimin': 341, 'Environ': 98, 'Security': 33, 'Tax': 136}
2011 freqDist:
{'Antitru': 113, 'Civil': 383, 'Crimin': 478, 'Environ': 94, 'Security': 38, '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}
```

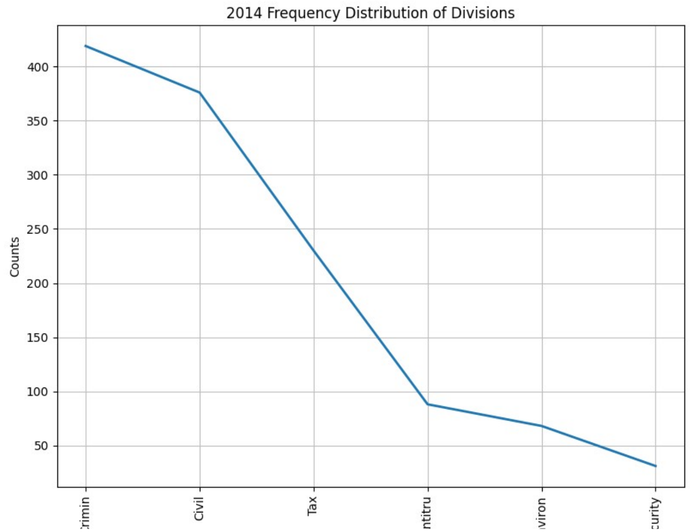

2009 Frequency Distribution of Divisions
Counts
Civil
Crimin
Tax
Environ
Antitru
Security

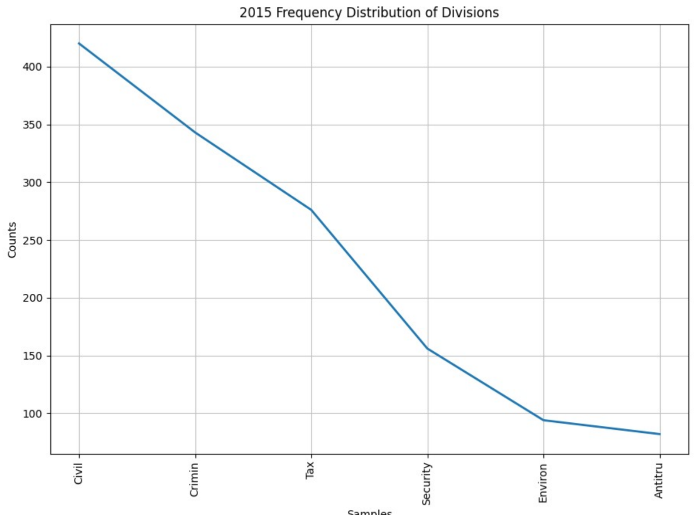

2010 Frequency Distribution of Divisions
Counts
Civil
Crimin
Tax
Antitru
Environ
Security

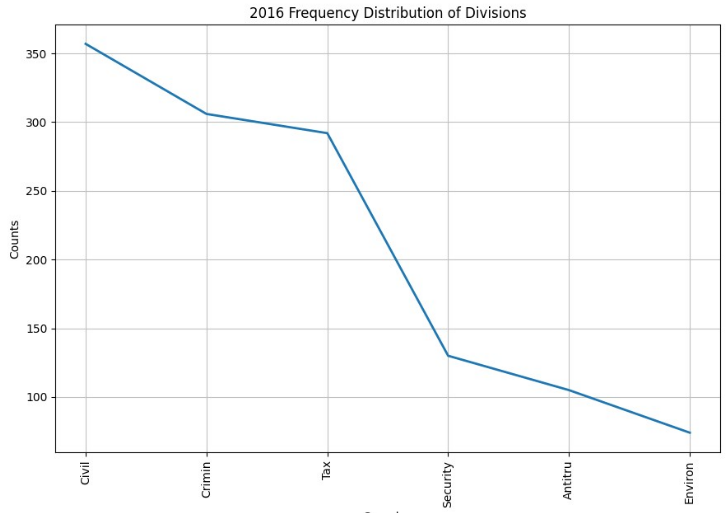

2011 Frequency Distribution of Divisions
Counts
Crimin
Civil
Tax
Antitru
Environ
Security

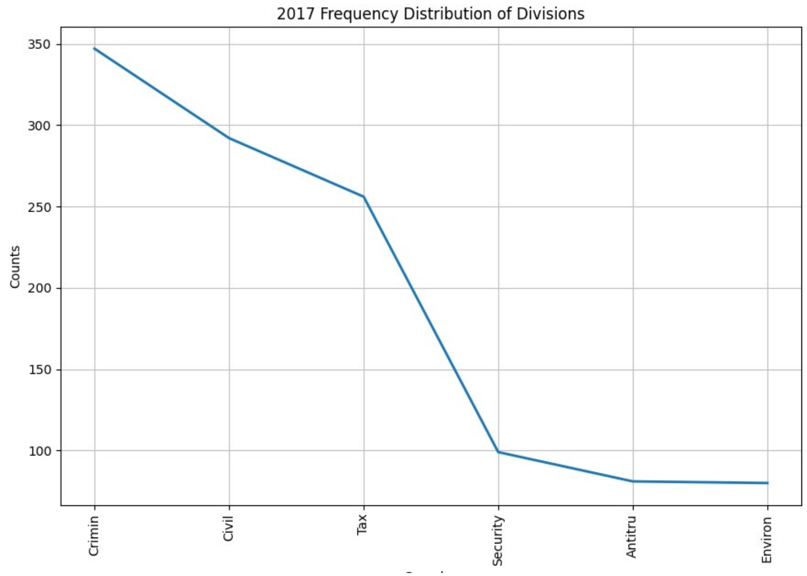

2012 Frequency Distribution of Divisions
Counts
Crimin
Civil
Tax
Antitru
Environ
Security

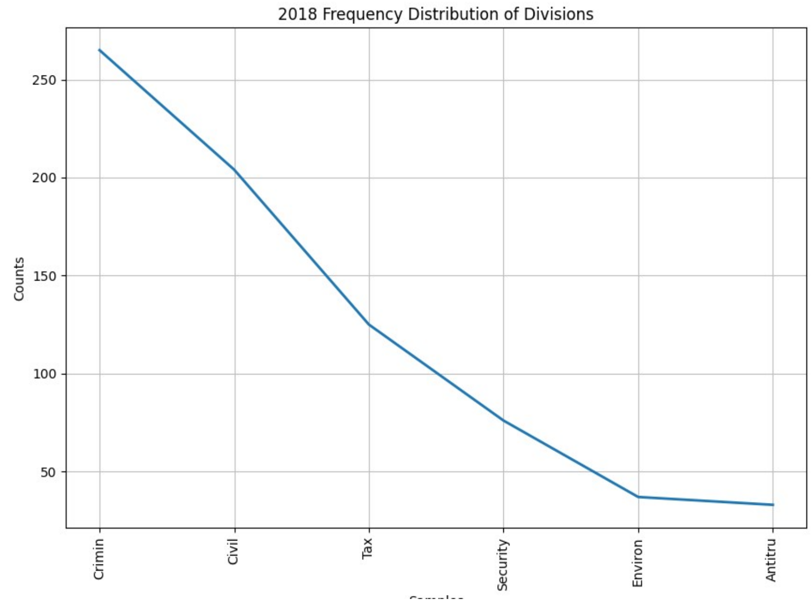

2013 Frequency Distribution of Divisions
Counts
350
300
250
200
150
100
50
0
Civil
Crimin
Tax
Environ
Antitru
Security


2014 Frequency Distribution of Divisions
Counts
400
350
300
250
200
150
100
50
Civil
Tax

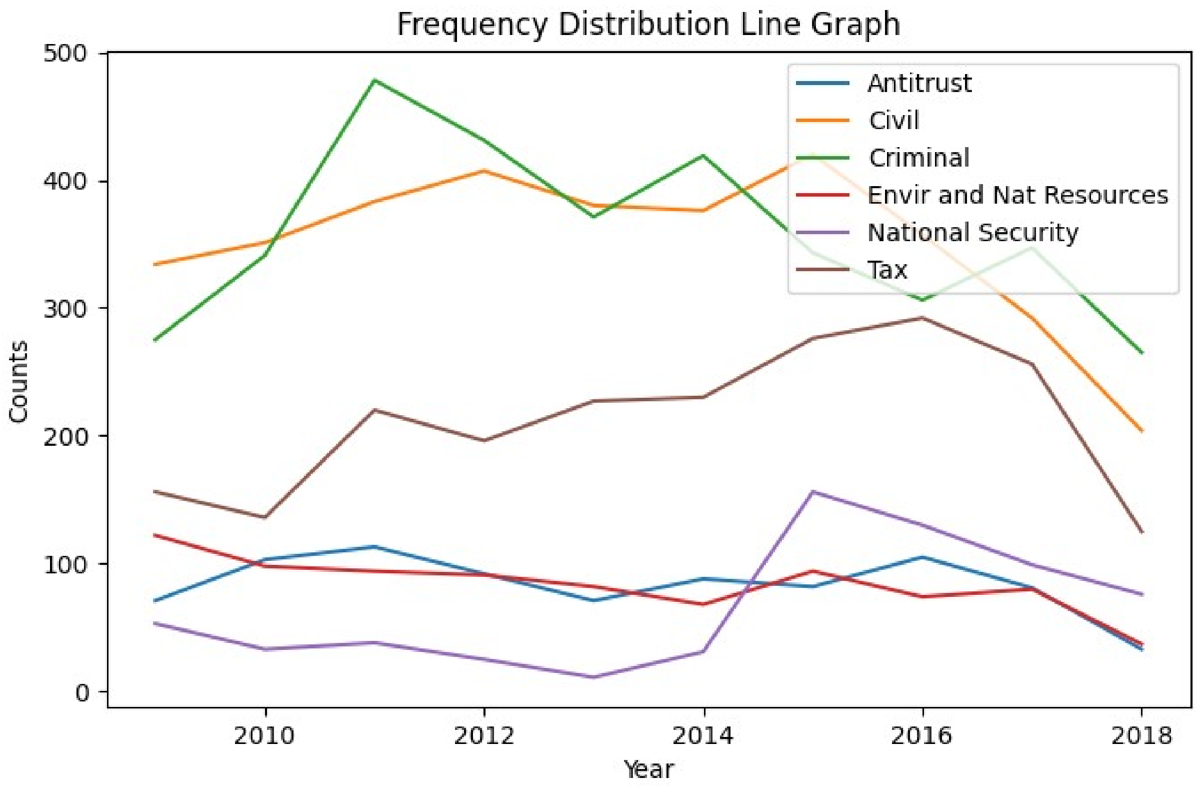

2015 Frequency Distribution of Divisions
Counts
400
350
300
250
200
150
100
Civil
Crimin
Tax
Security
Environ
Antitru


2016 Frequency Distribution of Divisions
Counts
350
300
250
200
150
100
Civil
Crimin
Tax
Security
Antitru
Environ


2017 Frequency Distribution of Divisions
Counts
350
300
250
200
150
100
Crimin
Civil
Tax
Security
Antitru
Environ


2018 Frequency Distribution of Divisions
Counts
250
200
150
100
50
Crimin
Civil
Tax
Security
Environ
Antitru

Aggregate Frequency Distribution



Word Cloud

- Visual display of commonly used words
  - joined strings of each row using `cat` function, passed entired string to `word_tokenize` function
  - removed words of 3 or less characters and removed common words (e.g. “attorney”, “department”, “justice”, etc.)
  - ex: 2011 and 2017

Experiment

- Term Frequency – Inverse Document Frequency (TF-IDF)
  - Term Frequency - the number of times a word appears in a document, divided by the total number of words in that document
  - Inverse Document Frequency - the log of, the total number of documents divided by the number of documents that contain a certain word w

$TF = \Sigma w/d$

w=word, d=total words in document

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

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

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

TF-IDF

- TF-IDF score for a word will initially increase if frequency is high within a document, but then decrease if the word is frequent throughout the data set
  - due to the IDF factor of the formula
- Words with higher TF-IDF score will be more relevant for specific topics of interest
- From Python's `sklearn` library, used `CountVectorizer`, `TfidfVectorizer`, and `TfidfTransformer` packages

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

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

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

- 2010 - “vehicle” has highest score of ~0.49, followed by “emissions,” “test,” and “air.” Suggests case related to vehicle emissions scandal was unique for the year 2010
- 2016 - “visas” has highest score of ~0.53, followed by “harbor,” “alien,” “conspiracy,” and “student.” Correlation of terms suggests a unique case involving fraudulent student visas
  - “fraud” has lower score likely because it's used frequently in the data set
- 2018 - “site” has highest score of ~0.48, followed by “epa”, “cleanup” and “river.” Suggests unique case(s) with EPA and ENRD (DOJ) related to hazardous waste spill

Support Vector Machine

- Attempt to predict the “components” field based on the text of the “contents” field
  - Created sub-data frame of only the litigating DOJ entities
- Used the `train_test_split` function with the “cleaned contents” and “cleaned components” fields, with a 70-30 training-test split
  - SVM requires int values for classification labels, so converted them with the `fit_transform` function
- From the `svm` module, used the `SVC` to build hyperplane and fit the training set attributes and the training set labels
- Then called the `predict` and `accuracy_score` functions, resulted in a ~99.3% accuracy.

SVM code

test.py > ...

```python


train_X, test_X, train_Y, test_Y = model_selection.train_test_split(sub_df['cleaned content'], sub_df['cleaned components'], test_siz
encoder = LabelEncoder()
#print(train_X[0:12])

train_Y = encoder.fit_transform(train_Y)
test_Y = encoder.fit_transform(test_Y)
print("train_Y type: ", type(train_Y))
print("train_Y head: ", train_Y[0:22])

tfIdfVectorizer=TfidfVectorizer() #use_idf=True
tfIdfVectorizer.fit(sub_df['cleaned content'])

train_X_tfidf = tfIdfVectorizer.transform(train_X)
test_X_tfidf = tfIdfVectorizer.transform(test_X)


print("train_X_tfidf type: ", type(train_X_tfidf))

SVM = svm.SVC(C=1.0, kernel='linear', degree=3, gamma='auto')
SVM.fit(train_X_tfidf, train_Y)
predictions_SVM = SVM.predict(test_X_tfidf) # predict the labels on validation dataset
print("SVM Accuracy Score -> ", accuracy_score(predictions_SVM, test_Y)*100) # Use accuracy_score function to get the accuracy
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 2: Python

```
3     9   15-089  ...  defend admit to target parent of hospit childr...     Criminal Division
4    10   14-660  ...  dure past week fbi local state feder law enfor...     Criminal Division

[5 rows x 8 columns]
train_Y type:  <class 'numpy.ndarray'>
train_Y head:  [2 1 4 1 4 3 3 3 3 1 0 1 2 2 3 1 2 3 3 0 1 2]
train_X_tfidf type:  <class 'scipy.sparse.csr.csr_matrix'>
SVM Accuracy Score ->  99.3
```

K-Means Clustering

- Used to group the articles' text into similar groups
  - used the `fit_transform` function on “cleaned contents” field
- Created an object from the `MiniBatchKMeans` class and obtained the top keywords
- “cleaned contents” text then grouped by the cluster object declared
- predetermined value of k = 10 clusters

```
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
```

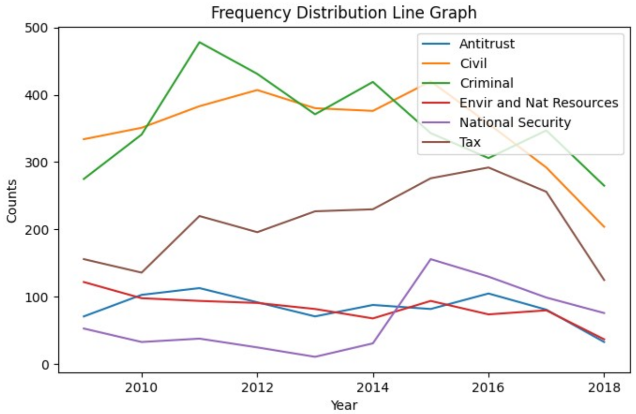
- Some words better clustered than others, e.g. Cluster 2 has terms “trafficking,” “inmate,” “civil,” and “right” - wide variety of cases
  - Similarly, Cluster 8 has general terms “general,” “nation,” “justice,” and “attorney”
- More accurate clusters are Cluster 3 which deals with child exploitation and Cluster 7 which deals with healthcare fraud

Results Analysis

- From the experiments, much information can be derived:
  - TF-IDF revealed the press releases of cases that were reported on less
    - done so by giving a higher score to words that were frequent in a document but infrequent throughout the data set
  - SVM successfully predicted the “components” field given the corresponding “contents” field
    - done so by splitting the data into a training and test set and learning from the training set
  - Clustering was less informative, case type of some clusters was easily discernable while others were inconclusive
    - likely due to the general words captured which could have manipulated the algorithm

Conclusion

- TF-IDF, SVM, and K-Means Clustering all proved effective in gaining information from the data set
- The NLP tools built into them gave an advantage in analyzing 13,087 records
- However, the tool which could answer the posed question was the frequency distribution



- Aggregate decline in press releases from 2016 to 2018, implying a decline in cases brought by the Department
- Actual cause remains unknown
  - could be funding, cooperation agreements, lack of evidence, or other reasons
  - requires further study

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).