# Legal Verbiage and Topic Modeling

Anthony Chi & Chris Han

**May 21st, 2018**

## I. Definition

### Project Overview

Docket texts are public court records that detail legal case events in a chronological order. New texts/records are available when the case status has been updated by the court. These new texts are sent to the responsible attorneys, often via email. It may be necessary to respond or react to these new records, meaning for law practices, it is a business as usual task. To help medium to small practices attorneys efficiently handle these records, ideally, we want to automate the docket text handling processes for law practices.

The As-Is process for many law practices, especially smaller practices: Lawyers spend time reading through docket text emails, and determine whether or not it’s necessary to respond and react to the docket texts, as well as how to respond and react.

The To-Be process: an algorithm scans the newly received docket texts and determine whether or not to respond and react. If response is necessary, the algorithm would also make the appropriate response such as draft emails, schedule appointments with the clients, etc.

### Problem Statement

Ideally, this should be a supervised classification exercise, as in for every docket text, we would have corresponding targets telling us how an attorney would respond. However, unless we have historical data recording responses, or ask attorneys to go through our dataset and provide responses, we can’t proceed with the supervised classification exercise.

Instead of a supervised classification exercise, we can first try an unsupervised classification exercise. More specifically, we can try topic modeling. If we determine how many topics the docket texts can be classified into, the algorithm will identify the most likely topics each text belongs to. Therefore, instead of using brute force to classify each docket text, attorneys can identify the responses more efficiently based on the topics assigned.

There are many difficulties with this exercise:

1. Difficult to acquire a large database: even though the docket texts are publicly available, they are only available with a fee. The process of acquiring data means someone with access to the government court website would have to log in, search for cases that may or may not be relevant, and download the docket texts.
2. This is an unsupervised modeling exercise: there is no target for this dataset, meaning the dataset does not come with answers on whether a lawyer should respond and react, or how to respond or react. Therefore, unless someone is willing to go through each texts to determine the targets, this is an unsupervised classification (topic modeling) exercise.
3. SME is required: to perform NLP or topic modeling for this exercise, we need to avoid GIGO (garbage in, garbage out). This meant that at each step, we needed experts to interpret our input and output results:
   1. During NLP, we need to clean enough to remove all the unwanted texts, but not too much so the meanings or relevant keywords are lost.
   2. After NLP, is there an easy way to filter the features to provide the correct targets?
   3. During Topic Modeling, we need to determine the optimal number of topics, as well as interpret each topics correctly.
   4. Testing: we require SMEs to tell us whether our model is assigning topics correctly for the test dataset.
   5. After Topic Modeling, we need SMEs to tell us if any action needs to be taken, and what type of actions are relevant.
4. If topic modeling does not work, how should we proceed?
   1. We can collect more data. But this comes with a price.
   2. Make this exercise a supervised classification exercise by asking the SMEs to provide targets for each feature.

### Metrics

Since this is an unsupervised exercise, we rely on our SMEs to determine the correctness of our classifications. Accuracy will be the metric that we measure/compare our modeling results.

## II. Methodology

### NLP (Natural Language Processing) – Cleaning

Organization and Name identification

Oftentimes names and organizations are irrelevant in topic modeling, as names and organizations are too specific on the case bases. Therefore, I’m using the tagger (Stanford NER) that Stanford developed to identify and remove names and organizations out of the docket texts.

However, we’ll later find that Stanford NER didn’t do a thorough job in this task. This means we’ll iterate through the NLP and modeling process to find more irrelevant object in our docket texts for removal.

Lemmatization and POS Identification

Lemmatization is a process to return words back to its original form. For example, plural back to singular, and past tense into present. This will help us simplify our texts without changing the meaning of the texts.

Parts of speech identifiers help us identify structure of sentences and words that should be removed. Numbers, dates, and pronouns are examples of things to be removed without changing the meaning of the texts. Our goal is to make our cleaned texts as simplified as possible without loosing meaning or keywords.

Stop Words Identification

Stop words are words that we often use and do not have particular meaning. These are also removed from the docket texts.

Punctuation Identification

Many ready-to-use NLP libraries have difficulties identifying punctuation and symbols that are used for specific context in a particular industry. We have identified punctuation and symbols that needed to be removed because the NLTK library didn’t.

Misc Identification

There are many legal and clerical jargon that adds no meaning to the docket texts. For example, contents in brackets “()” are always initials or dates. These occurrences should be removed during the NLP process.

Phrase Modeling

A phrase is combination of words that have meaning that is different from each individual constituents. It is important to identify phrases going into topic modeling, as the model will examine phrase frequencies instead of word frequencies.

**Keyword/Topic Pairs**

It may be difficult for topic modeling to produce results that are relevant. Therefore, another thought is that we would identify relevant topics in the docket texts if they contain specific keywords. This may be a good way to filter the docket texts for low hanging fruits, then allow topic modeling to handle texts with miscellaneous topics.

**Topic Modeling**

We are using the gensim library to perform LDA (Latent Dirichlet Allocation). LDA assumes that documents are probability distribution over latent topics. Topics are probability distribution over words. LDA looks at a number of documents and assumes that the words in each document are related. It then tries to figure out the 'recipe' for how each document could have been created. We just need to tell the model how many topics to construct and it uses that 'recipe' to generate topic and word distributions over a corpus. Based on that output, we can identify similar documents within the corpus.

### In order to understand the LDA process, we have to know how LDA assumes topics are generated:

1. determine the number of words in the document
2. choose a topic mixture for the document over a fixed set of topics (ie. topic A 20%, topic B 50%, etc)
3. generate words in the document by:
   * pick a topic based on the document's multinomial distribution
   * pick a word based on the topic's multinomial distribution

### Working backwards

Suppose you have a corpus of documents, and you want LDA to learn the topic representation of K[¶](http://localhost:8888/notebooks/Docket Text Topic Modeling - Building Model (Spacy and LDA) v4.ipynb" \l "2-Topics:)

topics in each document and the word distribution of each topic. LDA would backtrack from the document level to identify topics that are likely to have generated the corpus.

### LDA's Magic

1. randomly assign each word in each document to one of the K topics
2. for each document
   * assume that all topic assignments except for the current one are correct
   * claculate two proportions:
     1. proportion of words in document d that are currently assigned to topic t = p(topic t | document d)
     2. proportion of assignments to topic t over all documents that come from this word w = p(word w | topic t)
   * multiply those two proportions and assign w a new topic based on that probability. p(topic t | document d) \* p(word w | topic t)
3. eventually we'll reach a steady state where assignments make sense

**Topic Modeling Visualization**

The goal here was to examine the available data, and achieve the following objectives:

Compare Data Sources: many data sources are becoming unavailable, turned into paid services, or only available on proprietary platforms. Many APIs were also inactivated, and webpages were inactivated or unable to be scrapped. I was only able to access three data sources: Quandl API, NASDAQ, and finance.yahoo.com.

After I compared the three data sources, there were apparent differences in Adjusted Closing Price and Adjusted Volume. There were also occasional missing data from Quandl API; the NASDAQ maximum historical data only spanned 10 years. With these difficulties, I relied soley on finance.yahoo.com’s data.

<insert the difference in data chart>

Using the available features, including open, close, high, low, volume and adj. close, I generated additional features:

* High-Low Range, as a percentage of previous day’s close: this described how volatile the day’s movements were
* Open-Close Range, as percentage of same day’s open: this described the general movement of the day’s activities
* Moving average (adjusted close & volume) for a specific time period.

The new and old features needed to answer the following questions or explore ideas:

* What am I trying to predict: specific price, change in price, simple direction, or general trend/direction? This will determine if this is a classification (bi or multi) or a regression exercise.
* Were there any relationship between the independent and dependent features?
* What did the feature distributions look like? Did I need to make any transformation?
* Were there any missing or outliers that I needed to look into?
* Did I needed to use scalers, or other feature engineering techniques such as PCA?
* Was there a model that can be applied to all equities? Or did I need to train for each equity?

### Exploratory Visualization

**Outliers**

<insert distribution plots for features>

**Missing Data**

<insert chart for missing feature>

**New Features created**

**Target**

<insert target distribution and transformation chart>

**Features Transformed**

**PCA**

### Benchmark

1. I used two benchmark models, namely XGBoost Classifier and Lasso Regression. The ultimate goal was to use LSTM to beat these benchmark models. Supposedly, LSTM should be able to incorporate long and short term memories to generate additional insights, and further improve the predictive power of resulting models.
2. During the LSTM optimization exercise, I attempted to explore how the hyperparameters such as window, epoch, batch size, and LSTM constructs affected the model performances. This surely helped me tune the model for better performances.
3. Also note this was a regression, as well as, a classification exercise.
   * 1. As the exercise became computationally expensive, I moved computational and storage needs onto AWS.

**III. Methodology**

**Data Preprocessing**

scalers

**Target**

**Outliers**

**Missing Data**

**Delete Feature**

**Feature Types Transformed**

**One-hot Features**

**New Features created**

**Features Transformed**

**PCA**

### Implementation

### Refinement

### **Lasso Regression** **(benchmark)**

### **XGBoost (benchmark)**

### **LSTM**

## IV. Results

### Model Evaluation and Validation

### Justification

## V. Conclusion

### Free-Form Visualization

### Reflection

# Findings and questions

# Questions:

# Predicting next day price may not be the best thing to do. We may want to predict price in 5 days, or even much more ahead? What’s a reasonable window to predict price before randomness takes over?

# Should I predict volume?

# Should I include other features to predict price?

# Does dropout improve performance?

# Does multiple layers improve performance?

# Should we actually try to predict price? Or should we predict a range? For example, -1~1% is neutral, 1~5% is good, 5%+ is excellent, etc.

# Findings:

# Best results is with window 15 (about 3 weeks), epoch 2000, and batch-size 50-250.

# There may be upward bias, meaning all that we have seen is upward trend (most of the time).

# When selecting hyperparameters, training predict time, testing predict time, training error eval time, and testing error eval time are negligible.

# The larger the window, the longer the training time. The higher the epoch, and the smaller the batch size, the longer the training time.

# Best training error achieved is with window 15, batch size 10, epoch 500: 2.5616e-06

# Best testing error achieved is with window 10, batch size 500, epoch 2000: 7.58083e-05

### Citation and Sources

**Relevant Files and Folders**