State Health RegData Algorithm Documentation

State Health RegData aims to identify and quantify the amount of healthcare related regulations in each state regulatory code. To accomplish this, two steps had to take place. The first step was the actual gathering and documentation of state regulatory codes. This has been achieved via our State RegData project that can be found here. State RegData successfully collected the regulatory codes of 44 states plus the District of Columbia. The remaining states either had a regulatory code that was unpublished, not up to date, or behind a paywall.

The second step was the training of a machine learning algorithm that would be able to tell the difference between regulations that discuss healthcare related topics and regulations that discuss everything else. This is the majority of what will be covered during this section of the documentation.

Training Documents

The training documents were hand selected by researchers at the Mercatus Center and include positive and negative trainers from 44 U.S. state regulatory codes, the District of Columbia's regulatory code, and the U.S. *Code of Federal Regulations*. In total, 6,737 negative trainers and 17,786 positive trainers. The unit of analysis for a training document varies based on jurisdiction, but is always entirely inclusive of the text for a logical delineator of regulatory code. For the *Code of Federal Regulations* this is at the part level, for many states this is at the title or chapter level. As a rule of thumb, a median of 3,000 words and a mean of 12,000 is typical of the chosen unit of analysis for a given jurisdiction.

Preprocessing

A good bit of preprocessing is implemented for the training documents before they are used to form an algorithm. The first step that is taken is the removal of special characters and excess white spaces from text. This is completed through a series of regular expressions. Next, the contents of the documents are analyzed sentence by sentence, with each word in those sentences labeled with a part of speech tag. This is completed by using NLTK's package and pos_tag function that is described here. The tags are limited to adverbs, adjectives, verbs, and nouns, with the default tag being noun.

By attaching part of speech tags to words, the next step of the preprocessing is made easier. NLTK's WordNetLemmatizer takes the word and the pos_tag and outputs a lemmatized version of the word. Lemmatization is the grouping of similar words into one standardized word so that there is more efficient sentence analysis. An example of this is lemmatizing the words "change", "changing", and "changed" all to the word "change". After lemmatization, stop words are removed from the documents per NLTK's English stop words library.

One final stage is taken to prepare the training documents, tokenization. Tokenization is completed with Gensim, another open-source library for unsupervised learning and natural language processing. Gensim phrases takes a streamer of words (among other parameters) and outputs tokenized phrases if the phrase appears over a given threshold. The phrases that pass the threshold can then be stored as a custom genism package that can be used to preprocess other documents (prediction documents or documents that need to be classified). Tokenization was kept to phrases of length two in

maximum. Once all of these steps were complete, the training documents were ready to be used to train the algorithm

Per industry standards, these same preprocessing steps were implemented on every document before being classified by the final algorithm. This ensured that there is no information leakage between the training and classification steps.

Candidate Model Selection

After preprocessing the training documents, the QuantGov library was used to test and select an algorithm and specific hyperparameters. Three different models with various hyperparameters were tested: Logistic Regression models, Random Forest models, and Support Vector models. Outside of deep learning algorithms, these algorithms are typically the best algorithms for processing language.

All three models use TFIDF preprocessing while the logit models also implement a ridge penalty. The hyperparameters tested were as follows:

Logistic Regression Models: "C"

Random Forest Models: "n_estimators"

Support Vector Models: "C"

A range of these parameters were tested, each using a 5-fold cross validation method. After multiple days of running sample algorithms, evaluation scores were examined using the built in evaluation features in sklearn's library. The algorithm that performed the best was a logistic regression model with a "C" value of 100. This algorithm had a mean test score value of 0.960. The random forest models performed well, but generally underperformed the logistic models with scores around 0.940. Support vector models with "C" values greater than 1,000 also performed well, but never hit 0.960, topping out at 0.959.