Classifying Court Documents by Violations

Justina Zou*
Department of Statistical Science
Duke University
Durham, NC 27708
justina.zou@duke.edu

Abstract

Mortgage enforcement actions can provide insight into the efficacy of policy, enforcement trends, and prevalence of law breaking. Traditional methods to extract information such as the specific violations of the law from texts can be costly. We explore the efficacy of machine learning models such as support vector machines and neural networks to aid in classification. We find that these methods can give high accuracy, but other metrics such as precision reveal they lack the ability to effectively find true positives.

1 Introduction

Retrieving information from court documents can be useful for many different disciplines. In the case of enforcement actions, learning the types of violations present in the documents can provide insight into the efficacy of policy, enforcement trends, and prevalence of law breaking.

Often, it is resource intensive to manually extract information from all the relevant documents. Approaches such as dictionary based text mining rely on constructing a dictionary, which requires domain knowledge that may not be comprehensive or available.

In this project, we explore multi-label classification using machine learning models such as support vector machines and neural networks. We found that document embeddings can be useful as word embeddings and neural networks can outperform support vector machines.

2 Data

The data consists of 153 hand labeled court documents from the North Carolina Commissioner of banks site:

https://www.nccob.gov/Online/NMLS/CommissionOrderListing.aspx.

The documents span from 2003-2009. Documents were converted from PDF to text using the Tesseract OCR engine. Each document can contain one or more violations, and 75% of documents contain one violation.

3 Related Work

Previous work for information retrieval from text documents include a variety of methods. A survey of text classification algorithms identifies several classes of algorithms: feature extraction methods

*Github: https://github.com/cmzou

(e.g. Word2Vec), dimensionality reduction methods, and classification algorithms (e.g. naive Bayes, support vector machines, and neural networks), metrics for evaluation, and current limitations. [1]

In some cases, both machine learning and traditional methods are used to extract information. The system History Assistant uses feature embeddings and support vector machines to augment traditional parsing methods and discourse analysis in accomplishing the task of extracting rulings and prior cases from court documents. [2]

There has also been work examining the effects of stemming, lemmatisation, and feature reduction on the performance of support vector machines on multi-label classification of financial textual data and legal text. [3]

4 Methods

Given a text document, we want to find a model that maps text input $\mathbf{X}$ to binary vectors $\mathbf{Y}$, which assigns a value of 0 or 1 for each label (in our case, a violation) in $\mathbf{Y}$.

We use the Continuous Bag of Words model to create document embeddings and neural networks and support vector machines to build a classification model.

4.1 Continuous Bag of Words

Embeddings are often used to transform non-numerical data into a numerical form, usually in the form of a vector. The goal of an embedding is to reduce the size of the space from $\mathbb{R}^{|V|}$, where $|V|$ is the number of unique words in a document, to something much smaller and encode semantic information. Therefore, embeddings are useful for transforming textual data into a form machine learning models can manipulate.

The key idea in the word embedding approaches under "Word2Vec" is that we consider a metric where words that show up together in the same context should embed close to each other in a vector space. The Continuous Bag of Words model predicts the current word based on the words around it. That is, given a sequence of training words $w_1, w_2, w_3, \ldots, w_T$, the word vector model seeks to maximize $\frac{1}{T}\sum_{t=k}^{t-k} \log p(w_t|w_{t-k}, \ldots, w_{t+k})$. [4] Once the word vectors are computed, words with similar meaning are mapped to similar positions.

The Paragraph Vector framework is similar. In addition to the words as input, the model is also fed a paragraph ID. Again, the model tries to predict the next word given the words ground it. The result of the model is then a vector that can be thought of as features for the paragraph. An advantage of the paragraph vectors is that they take into consideration the word order, which is lost when combining word vectors to create paragraph embeddings. In our model, we use the Document Vector framework, where a document ID is fed into the model to obtain a document embedding. We also use word vector framework and average all the word embeddings to form a single document vector for comparison purposes.

4.2 Support Vector Machines

The idea of linear support vector machines (SVMs) begins with a linear hyperplane that separates the data. Defining the margin as the distance from the hyperplane to the nearest example, the basic observation is that intuitively, we expect a hyperplane with a larger margin to generalize better than one with a smaller margin. We denote our hyperplane by $\mathbf{w}$ and we let $\mathbf{x}$ be a datapoint closest to $\mathbf{w}$, and we let $\mathbf{x}^{\mathbf{w}}$ be the unique point on $\mathbf{w}$ that is closest to x. Finding a maximum margin $\mathbf{w}$ is equivalent to maximizing $\|\mathbf{x} - \mathbf{x}^{\mathbf{w}}\|$.

For some $k > 0$ (for convenience), $\langle \mathbf{w}, \mathbf{x} - \mathbf{x}^{\mathbf{w}} \rangle = k$. Since $\mathbf{x} - \mathbf{x}^{\mathbf{w}}$ is parallel to the normal vector w, $\frac{k}{\|\mathbf{w}\|} = \|\mathbf{x} - \mathbf{x}^{\mathbf{w}}\|$. Without any loss of generality, we fix k to 1, and see that maximizing $\|\mathbf{x} - \mathbf{x}^{\mathbf{w}}\|$ is equivalent to maximizing $\frac{1}{\|w\|}$. This is equivalent to minimizing $\|w\|$. Thus, we can define the margin as the distance between the hyperplanes $\langle \mathbf{w}, \mathbf{x} \rangle = 0$ and $\langle \mathbf{w}, \mathbf{x} \rangle = 1$.

Since there are datasets that are not linearly separable, so we introduce slack variables ξ_i. We can still define the margin as the distance between the hyperplanes $\langle \mathbf{w}, \mathbf{x} \rangle = 0$ and $\langle \mathbf{w}, \mathbf{x} \rangle = 1$. We

can also intruduce an unregularized bias term b, leading to classification via a function of the form $f(x) = \text{sign}[\langle \mathbf{w}, \mathbf{x} \rangle + b]$.

The primal SVM problem becomes

$$\begin{aligned} \min_{\mathbf{w} \in \mathbb{R}^n, b \in \mathbb{R}} & C \sum_{i=1}^{n} \xi_i + \frac{1}{2} \|\mathbf{w}\|^2 \\ \text{subject to: } & y_i(\langle \mathbf{w}, \mathbf{x} \rangle + b) \geq 1 - \xi_i, i = 1, \ldots, n \\ & \xi_i \geq 0, i = 1, \ldots, n. \end{aligned}$$

MLTSVM was introduced to extend the functionality of twin support vector machines, a relaxation of the SVM requirement that hyperplanes be parallel, to multi-label problems. [5]

4.3 Neural Networks

The main idea behind deep neural networks is that the learning or inference problem is framed as learning a function or map $f : X \to Y$, where X are the inputs and Y are the outputs or our predictions.

The function $Y = f(X, \theta)$ can be thought of as a composition of many different functions, and can be described as a feedforward network, or directed acyclic graph. The depth of a neural network is the maximum number of layers. The intermediate layers are referred to as hidden layers.

An early version of a neural network is the perceptron algorithm. The input is a p dimensional vector $(x_1, \ldots, x_p)$ and the output is one of two classes $\{0, 1\}$. The perceptron performs three operations:

1. Each input element is multiplied by a weight: $u_i = w_i \times x_i$.
2. The products are summed: $h = \sum_{i=1}^{p} u_i$.
3. An activation function is applied to the sum. E.g.

$$y = \begin{cases} 0 & h \leq 0 \\ 1 & else. \end{cases}$$

Training is accomplished by updating the weights corresponding to the input-output pairs that correspond to an incorrectly classed input.

Multi-layer perceptron essentially combines several perceptrons. However, weights in each layer need to be learned, and backpropagation was developed to do so.

Let a multi-layer perceptron have depth d. Thus, the function that we are trying to learn $f : X \to Y$ is a composition of functions $f_d \circ f_{d-1} \circ \cdots \circ f_1$. Let τ_k be the number of hidden units in layer k. Let weights at a layer k be $w_k = \{w_{k,1}, \ldots, w_{k,\tau_k}\}$, where $w_{k,1}$ are the weights for the first hidden unit in layer k and w_{k,τ_k} are the weights for the last hidden unit in layer k. Then, the optimization problem that we are trying to solve is

$$\min_{w_1, \ldots, w_d} \left[L(w_1, \ldots, w_d) = \frac{1}{2} \sum_{i=1}^{n}) y_i - f(x_i)^2 \right].$$

At any layer j, the weights can be updated for a hidden unit i based on derivatives with respect to $w_{j,i}$

$$w_{j,i}^{(t+1)} = w_{j,i}^{(t)} - \alpha \frac{\partial L}{\partial w_{j,i}}.$$

The update is implemented using recursion and a differentiable activation function instead of the unit step function. Because we are also looking at a composition of functions, backpropogation also makes use of the chain rule to take the error at the output and pass it back through the network recursively to update the weights.

4.4 Metrics

In multi-class problems, the standard metric for evaluation is accuracy, how many labels were correctly predicted divided by the total number of predicted labels. Other metrics such as precision, recall, and the F-measure serve to evaluate the performance of the model on different classes. In the case of multi-label classification, each instance is associated with a set of labels, rather than a single label. Thus, there is the notion of fully correct, partially correct, or fully incorrect.

Let Y_i be the true labels for an observation i, and let Z_i be the predicted labels for an observation i.

In the context of this project, we have found the metrics *Accuracy* (Hamming Score), *Exact Match Ratio* (EMR), *Precision*, and *Recall* to be of relevance from a survey on multi-label learning. [6] [7]

4.4.1 Accuracy

Accuracy is defined as the Hamming Score, which is the number of the predicted correct labels to the total number of labels for that instance.

$$Accuracy = \frac{1}{n}\sum_{i=1}^{n}\frac{|Y_i \cap Z_i|}{|Y_i \cup Z_i|}$$

4.4.2 Exact Match Ratio

Exact Match Ratio is the number of fully correct labels to the total number of predicted labels. We use this metric because we are interested to know not only how well the algorithm can predict 0s and 1s, but also how well the predictions completely match with the real labels.

$$ExactMatchRatio = \frac{1}{n}\sum_{i=1}^{n} I(Y_i = Z_i)$$

4.4.3 Precision

Precision is the ratio of true positives to all labels predicted to be positive. It measures how well the algorithm can correctly predict 1. It is important that we look at this metric and *Recall* because of the sparsity in the labels–the model can achieve a very high accuracy simply by predicting 0 for each label.

$$Precision = \frac{TP}{TP + FP}$$

4.4.4 Recall

Recall is the ratio of true positives to true positives and false negatives. It measures how well the algorithm can find all the 1s.

$$Recall = \frac{TP}{TP + FN}$$

5 Results

We discarded labels with less than 5 occurrences as well as ambiguous violations such as "other." We were left with 18 unique labels. We reduced each document to a 100-dimensional document vector. We used both the Document Vector framework (Doc2Vec) and the Word Vector framework. We created document vectors with the Word Vector framework by averaging the word vectors.

The linear MLTSVM model with averaged word vectors was trained using empirical risk 0.0625, smoothing parameter 2, and regularization parameter 2. The linear MLTSVM model with docu-

ment vectors was trained using empirical risk 0.125, smoothing parameter 0.125, and regularization parameter 0.2. We used scikit-multilearn's implementation of MLTSVM. [8]

The neural network model was trained using two hidden layers. The first layer had 128 neurons, and the second had 64. We used the activation function ReLU for the hidden layers and sigmoid for the output layer. We trained the network with 25 epochs and 10 as our batch size.

We selected the parameters that gave the highest EMR for each model and used 5-fold cross validation to average our results.

Table 1: Model Results, best results bolded

Model	Accuracy	EMR	Precision	Recall
MLTSVM w/Averaged Word2Vec	0.8179	0.0133	0.2296	0.2977
MLTSVM w/Doc2Vec	0.8900	0.1172	0.4717	0.2905
Neural Network w/Averaged Word2Vec	0.8899	0.1305	0.4904	0.3134
Neural Network w/Doc2Vec	**0.9126**	**0.2226**	**0.6432**	**0.4110**

The neural network model using Doc2Vec as its document vector outperformed all four models in all four metrics.

6 Discussion

As expected, the models with Doc2Vec outperformed the models with averaged Word2Vec as input, although the MLTSVM model with averaged word vectors had a slightly higher recall than its counterpart. Both neural network models outperformed the MLTSVM, although the neural network model with averaged word vectors had a slightly lower accuracy. The MLTSVM with Doc2Vec had a similar accuracy to the neural network with Doc2Vec, but their difference in EMR, precision, and recall indicates the neural network can be better at finding true positives.

For further investigation, a comparison to dictionary based methods or other traditional methods can provide greater insight into the feasibility of using machine learning. For example, in some cases, the availability of labeled data may affect the ability to train the neural network, which tends to do better with large amounts of data. In addition, it can also be helpful to compare other machine learning models such as random forests.

7 Conclusion

We explored classifying enforcement actions by the violations they contain inside. We explored two different types of document embeddings and built two different models using them as inputs. We found that the Doc2Vec framework performed better than the Word2Vec framework and that the neural networks performed better than linear support vector machines.

The accuracy of all the models were higher than 0.75, displaying promise for this particular task. However, the exact match ratio is lacking (0.2226 at best). Looking at the precision and recall indicates that the model could have trouble finding true positive labels, likely due to the small data set, large number of unique labels, and sparsity of the labels. Whether it is "worth" using these models depends on the circumstances and tolerance for error.

References

[1] Kowsari, K., Jafari Meimandi, K., Heidarysafa, M., Mendu, S., Barnes, L., & Brown, D. (2019). Text Classification Algorithms: A Survey. *Information*, 10(4), 150. doi:10.3390/info10040150.

[2] Jackson, P., Al-Kofahi, K., Tyrrell, A., & Vachher, A. (2003). Information extraction from case law and retrieval of prior cases. doi:https://doi.org/10.1016/S0004-3702(03)00106-1.

[3] Gonçalves, T., & Quaresma, P. (2005). Evaluating preprocessing techniques in a Text Classification problem.

[4] Le, Q. & Mikolov, T. (2014) Distributed Representations of Sentences and Documents. *Proceedings of the 31st International Conference on Machine Learning*, Beijing, China.

[5] Chen, W., Shao, Y., Li, C., & Deng, N. (2016). MLTSVM: A novel twin support vector machine to multi-label learning. *Pattern Recognition, 52*, 61-74.

[6] Sorower, M.S. (2010) A Literature Survey on Algorithms for Multi-label Learning, Oregon State University, Corvallis, OR.

[7] 3.3. Metrics and scoring: quantifying the quality of predictions. (n.d.). Retrieved from

https://scikit-learn.org/stable/modules/model_evaluation.html#classification-metrics.

[8] Szymanski, P., & Kajdanowicz, T. (2017). A scikit-based python environment for performing multi-label classification. Corr, abs/1702.01460 Retrieved from http://arxiv.org/abs/1702.01460.