Personal Legal Counselor and Interpreter of the Law via Machine Learning

Derek Yan, Tianyi Wang, Patrick Chase
{zhyan, tianyiw, pchase}@stanford.edu

***Abstract*—The goal of this project was to predict the likelihood of winning a new legal dispute based on results of past cases. We collected over 5000 legal proceedings in the form of case briefs from the internet and used various language processing techniques to parse the raw text into feature vectors. We then used this feature vectors to train several binary classification algorithms, including Naive Bayes, random forests, logistic regression and an SVM. The SVM model achieved the highest test set accuracy of 62%, which was an improvement over the random 50% baseline. In this paper, we explain the details of how we transformed the raw text of the case briefs into feature vectors, and how we used them to build several models. We then discuss the results obtained by each of the models and suggest future work that could be done in the area.**

***Index Terms*—law, machine learning, case briefs, court cases.**

I. Introduction

"THe first thing we do, let?s kill all the lawyers" - William Shakespeare, 2 Henry VI, 4.2.59.

Any major transaction, legal procedure, or patent dispute always requires an attorney-at-law in the due process. However, paying an attorney, even for a consultation, can be too expensive and out of reach for much of the population. Due to the exorbitant cost of legal action, many cases are unresolved or dropped. Our goal was to create a tool that would provide legal counsel to people who would otherwise not have access to it. In particular, our model would tell someone the probability they have of winning a given case, which would allow them to make the a better decision of whether or not to pursue further legal action.

II. Dataset

Our dataset was obtained from url-www.casebriefs.com. It consisted of 5,836 legal case briefs, where each case brief was split up into four segments of text, the *Parties in Dispute*, the *Summary of Facts*, the *Issue of Law*, and the *Verdict*. Fig. 1 is an example of a very short case brief.

Parties in dispute	Smith v. Doe
Summary of facts (features)	Smith was fired from her job as a cashier at Doe's store because she refused to work on Saturday because of Sabbath.
Issue of law (features)	Is the Free Exercise Clause denied when a claimant is discharged from work because of her religious practices?
Verdict (label)	Yes, Doe violated Smith's right to the free exercise of her religion

Fig. 1. Example Case Brief

III. Features and Preprocessing

First, we parsed the *Verdict* section to obtain the binary labels for the cases. When the *Verdict* was held, meaning the answer to the *Issue of Law* was "yes," we gave a positive label to the example, but when the answer to the *Issue of Law* was "no" we gave a negative label to the example.

After extracting the labels, we found that there were 2099 positive cases and 3737 negative cases.

A. Training and Testing Data

We then split up the data into the training and testing datasets described below.

	Training Dataset	Testing Dataset
Total Examples	3800	400
Positive Examples	1900	200
Negative Examples	1900	200

B. Feature Generation

To create the feature vectors, we used the *Summary of Facts* section and *Issue of Law* section. We did not use any of the text from the *Verdict* section because that section was used to determine the positive or negative label.

First, we processed the raw text of the case briefs by transforming each word to its stem using the Lancaster Stemmer from the NLTK, Natural Language ToolKit [1]. We then formed a dictionary by scanning through all the

words in our dataset. After forming the dictionary, we used the dictionary to represent the case briefs using the bag-of-words representation. In our representation, the ith element of the feature vector for an example corresponded to the number of times the ith stem occurred in the given case brief.

We also experimented with adding bigrams and trigrams to the feature vectors. For the algorithms that took dense matrices as input, this made the input too large and cause the algorithms to take too long or run out of memory. However, the SVM implementation took in a sparse matrix input and this was able to run with bigrams and trigrams.

(Add part of example feature vector?)

IV. Models

For all the models we used the SciKit-Learn package [2] for python.

A. Naive Bayes

Details.

B. Random Forests

Details.

C. Logistic Regression

Details on regularization.

D. SVM

Details on SVM.

V. Results

Large table of results and confusion matrices.
Important words/phrases

VI. Discussion

VII. Conclusions

VIII. Future Work

References

[1] E. Loper and S. Bird. Nltk: The natural language toolkit. In *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics - Volume 1*, ETMTNLP '02, pages 63–70, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

[2] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.