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 Forest, 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 explained 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 for prediction. We then discussed the results obtained by each of the models and suggested 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 very expensive and out of reach for much of the general 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

The processing flow is shown in the Fig. 1

Our dataset was obtained from 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. 2 is an example of a very short case brief.

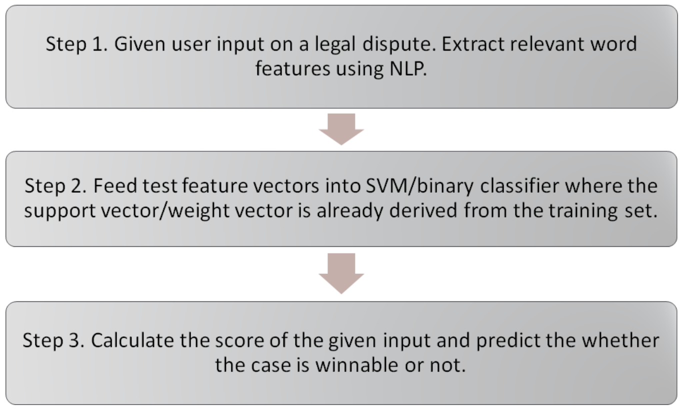
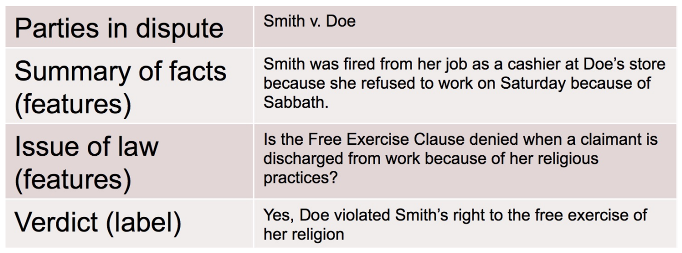

Step 1. Given user input on a legal dispute. Extract relevant word features using NLP.
Step 2. Feed test feature vectors into SVM/binary classifier where the support vector/weight vector is already derived from the training set.
Step 3. Calculate the score of the given input and predict the whether the case is winnable or not.

Fig. 1. Processing Flow



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. 2. Example Case Brief

III. Baseline and Oracle Results

Before tackling the actual problem. We first considered taking the baseline and oracle results of the legal case predictor. The baseline solution was using linear regression to serve as a predictor of future cases. We used 3-grams from the input text for feature extraction. When the new input was given, we extracted the features, and then used stochastic gradient descent to come up the learning-predictor to estimate the likelihood of the plaintiff winning the case. For the test data, we had an accuracy rate of around 51%.

The oracle was a manual interpretation of facts and issues given a test case by the group members. We read over the facts of a legal proceeding, the interpretation of law which was under question, and used our common sense to give a probable decision. For legal matters that were not familiar to us, we would research laws of such matter and make human predictions. This would serve as our oracle. For our test cases of 20, we had an error rate of 10%.

IV. 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 in our entire dataset.

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

Since we had a sufficient amount of positive and negative examples, we chose training and testing datasets with equal number of positive and negative samples to make it easier to analyze and compare the performance of our models.

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. In the {key, value} pair, key:string equals the stem (resp. n-gram) string, and value equals the count of that stem (resp. n-gram).

We also experimented with adding bigrams and trigrams to the feature vectors. For Naive Bayes, Random Forest, and logistic regression, the huge increase in the number of features caused the algorithms to take too long or run out of memory. However, the SVM implementation, which took in a sparse matrix input, was able to run with bigrams and trigrams.

For example, if the case description was "John was owed two weeks of pay for failing to submit his timecard". After removing stop words and transforming words to their stems, we get {"john", "owe", "two", "week", "pay", "fail", "submit", "timecard"}. Now we count how many times each word and word gram appeared, and form the dictionary/feature vector we need: {"john" : 1, "john owe" : 1, "john owe two" : 1, "owe" : 1, ...}

V. Models

For all the models we used the SciKit-Learn package [3] for python.

A. Naive Bayes

The implementation of Naive Bayes that we used assumed that the likelihood of the features was normally distributed:

$$p(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right) \tag{1}$$

There were no other parameters to set for Naive Bayes, so after choosing the distribution to represent the likelihood of the features, we trained our model.

B. Random Forest

When we first ran the out-of-the-box Random Forest classifier from SciKit-Learn, we got an accuracy on the training set of 97% and an accuracy on the testing set of around 53%, illustrating that we were overfitting the training data. To remedy this issue, we set the max depth of the tree to be 6, which greatly improved our overfitting problem.

C. Logistic Regression

When modeling the outcome of a legal case with logistic regression, we had the same issue with the out-of-the-box algorithm. We were drastically overfitting the training data, which led to a poor accuracy on the testing set. To fix this issue, we used a logistic regression model with L2 regularization. This decreased our training set accuracy from 99% to 67%, and improved the performance of the model on the testing dataset.

D. SVM

SVM model was implemented with the LIBSVM, which supports training, predicting, configuring regularization, cross validation and so on. The input feature vector used in LIBSVM was defaultly dictionary, but the keys of the {key, value} must be int type. So there was a transformer that compressed all the keys that appeared in the output dictionaries of the feature extractor into a vector, and then used the index of each element in the compressed vector as the key of the input dictionaries of SVM.

At first SVM was heavily favoring negatively labeled examples. i.e. it was more likely to predict new examples as negatively labeled. We fixed this by setting the gamma variable of radial basis function kernel to 0.005.

Then SVM with features of word-counter gave around 99% accuracy on the training set but around 50% accuracy on the testing set, which indicated a overfitting problem. To solve the overfitting problem, we integrated NLTK and changed word to stem, which decreased the feature set. Another technique we used was to remove stop words, such as ”to”, ”and” and ”is”, from the feature set. We also tried removing the words that have too high or too low appearance frequency from the feature set, considering those words might be meaningless, but this technique did not give quality improvement.

Our work decreased the training accuracy to around 60% and increased the testing accuracy to around 60%. The fact that the training error was similar to the testing error indicated that the overfitting is fixed.

VI. Results

The table below shows the training and testing error for each algorithm.

Model	Training Set Accuracy	Testing Set Accuracy
Naive Bayes	0.9463	0.5879
Random Forest	0.6318	0.5639
L2 Reg. LogReg	0.6768	0.6080
SVM	0.6920	0.6210

Here we see that the SVM had the best performance on the testing dataset. It was able to achieve 62.1% accuracy.

In addition, Fig. 3, Fig. 4, Fig. 5, Fig. 6, show the confusion matrices for each classifier. In the discussion section, we compare and contrast these results.

VII. Discussion

A. Most Indicative Features

To determine the most indicative features, we looked at the top five coefficients of the logistic regression model with the largest positive magnitude and the top five features with the largest negative magnitude. After mapping these stems back to their original words, we created the table illustrated in Fig. 7. We do see some interesting domain specific words come up, such as ”evidence.” It makes sense that the frequency of a word like ”evidence” in the case brief would lead to a larger probability of winning the case because there is likely

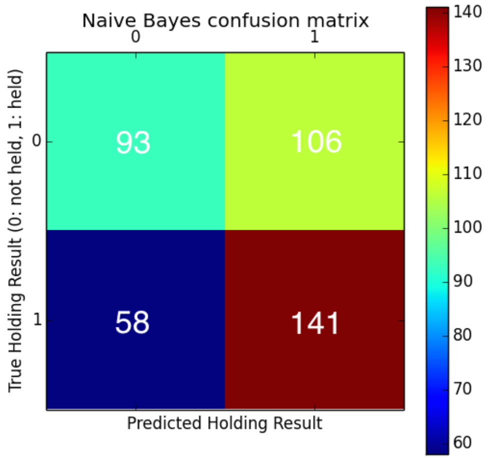


Fig. 3. Naive Bayes Confusion Matrix

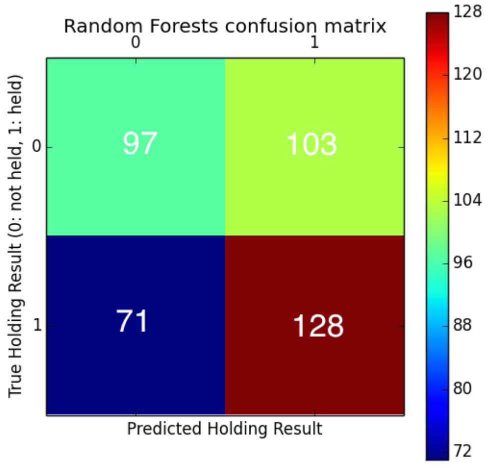


Fig. 4. Random Forest Confusion Matrix

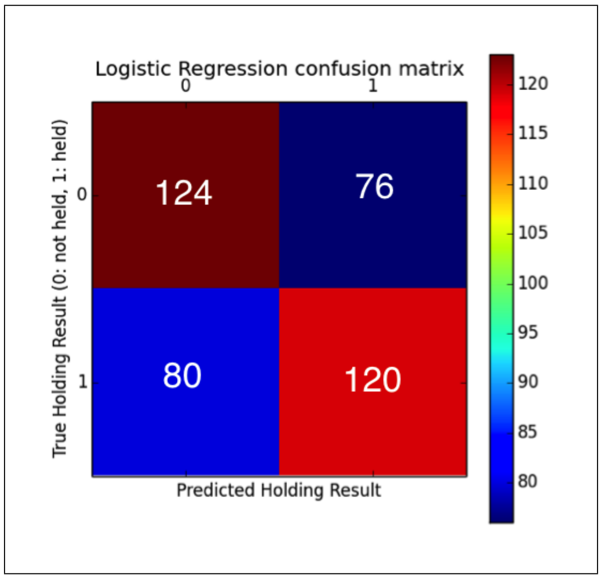


Fig. 5. L2 Logistic Regression Confusion Matrix

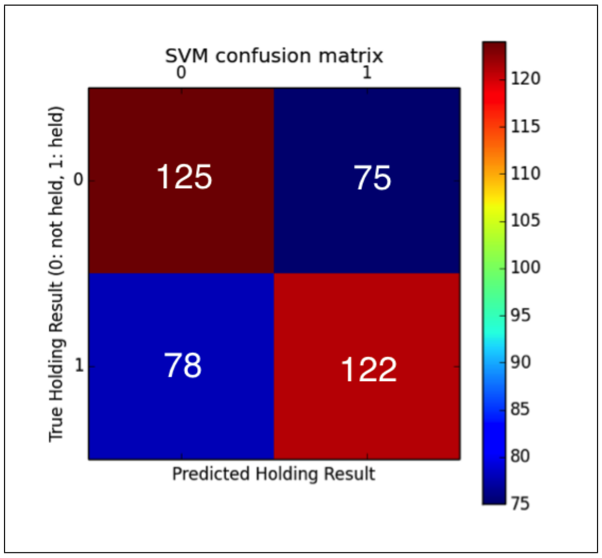
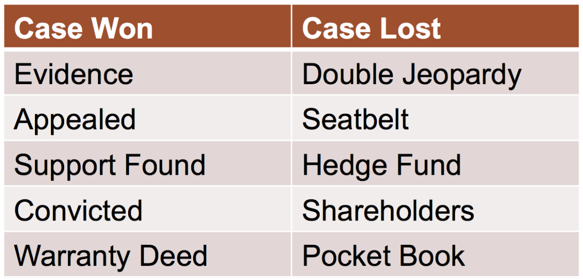


Fig. 6. SVM Confusion Matrix



Case Won	Case Lost
Evidence	Double Jeopardy
Appealed	Seatbelt
Support Found	Hedge Fund
Convicted	Shareholders
Warranty Deed	Pocket Book

Fig. 7. Most Indicative Features

more evidence for the plaintiff.

B. *Comparison of Models*

The confusion matrices give more information about how the binary classifier performed and make it easier to compare the four models. Both the Naive Bayes model and the Random Forest model have a very large number of false positives. A false positive in this case would be a bad result because you would advise someone to pursue legal action even when they have a large probability of losing the case. So, not only do Naive Bayes and Random Forest have a lower accuracy than the other two models, but they are worse because of the large number of false positives they have. In fact, the precision of each of these is less than 50%.

In contrast, the logistic regression model and the SVM have much more balanced confusion matrices. They have about the same false positive and false negative rates. The precision of both of these models is over 62%, which makes them much more desirable. Since the SVM has the best precision and the best accuracy, this is the best model out of the four. It's accuracy is 62.1% and its precision is 62.5%.

C. *Analysis of Accuracy*

We believe that predicting the outcome of a legal case just from the raw text of the case brief with an accuracy of 62% is a reasonable result for the first attempt at a very challenging problem.

Legal proceeding labels are subjective in the sense that ruling are influenced by the sentiment of the jury and may not be captured fully in case briefs [2]. An oracle of manually studying a dispute and researching online to give a well-informed prediction only results in a correctness of 90%.

The difficulty of this problem is that the legal information in the case brief is sophisticated, with many domain specific knowledge background. And our predicting output is not the normal true or false classification of the text like that in the spam email classification. What we are predicting is the relationship between the two entities in the case, the plaintiff and the defendant. The relationship is either the plaintiff defeating the defendant or the defendant defeating the plaintiff.

To address the first problem, feature set can be improved by collaborating with legal professionals to add nuances and hand-select features. With more domain specific knowledge, we can extract much more meaningful features besides the word counter.

We proposed using information extraction techniques to solve the second problem. The factor graph model is a good fit for extracting relationship between the plaintiff and the defendant. Important legal words and dependency paths between the two entities can be added as factors. And we also need entity linking to recognize mentions in the sentences. Since none of us have experience using these techniques, we did not implement these models in the current version given the time limit.

VIII. Conclusions

The outcome of a legal dispute was predicted with an accuracy of around 62% based on just the case brief using machine learning techniques. Past proceedings in the form of case briefs were used to extract features and labels. Feature extractor used techniques in NLP to remove word stems and map words of similar categories. The features that we used in the end were word grams with length 1, 2 and 3, and the labels were the verdicts of the proceeding. Naive Bayes, Random Forest, Logistic Regression and SVM were the four models used for prediction, and the SVM model achieved the best performance.

IX. Related Work

Machine Learning in law is relatively an untapped market. There currently is not a predictor based on past legal proceedings. With that being said, there are two existing websites that offer services in law through machine learning:

FindLaw.com: A website that allows users to search for relevant lawyers based on legal needs and provide legal counseling through a forum. A recommendation system is used for picking which lawyer is more relevant for the case at hand.

Judicata: Mapping unstructured legal data into a generative model to empower lawyers to find the most relevant and convincing past proceedings. This information then can be used to support an argument in court.

X. Future Work

As mentioned in the discussion section, integrating more domain specific knowledge and using relation extraction model together with more sophisticated NLP techniques may result in a more accurate model.

Besides, to help the users interact with our artificial counselor easily, an UI is needed. The UI should have at least two fields for users to input, case facts and issues. And the output is the predicted winning or losing.

Here is the link to the code for this project:
https://github.com/yan7109/LawPredictor

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] B. C. McKimmie B.M., Antrobus E. Objective and subjective comprehension of jury instructions in criminal trials. *New Criminal Law Review*, 17:163–183, 2014.

[3] 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.