Personal Legal Counsellor and Interpreter of the Law via Machine Learning
CS229 Milestone Report

Patrick Chase, Tianyi Wang, Derek Yan

Introduction:

The goal of this project is to predict the outcome of a legal proceedings or class actions based on previous court proceedings. Legal advice is often very expensive, which would make a tool that provides free counseling by examining previous cases very useful to anyone with a legal dispute. In addition, this application would give user an impartial assessment of the chances they have of winning their case.

Dataset:

For this project, data was collected from the legal case briefs from www.casebriefs.com. There are 5,836 cases from multiple categories ranging from administrative law to torts. 2099 of the cases had an affirmed holding (where the judge ruled in favor of the plaintiff), and 3737 were denied. A very brief example is shown below:
INPUT-Case Title: Eisenstadt v. Baird
Category: Family Law
Facts: Appellee (Baird) was convicted for exhibiting and distributing contraceptive articles under a law that forbid single as opposed to married people from obtaining contraceptives.
Issue: Is there a rational ground for the different treatment of married and unmarried persons under the Massachusetts State law?

OUTPUT-Holding: No.

Features:

Since the input is text, we have decided to use the bag-of-words model to represent the input. We first removed all punctuations and stop-words from the cases. Then after building a dictionary of all the words in the dataset, we transformed each case brief into a feature vector where the *i*-th element corresponds to the number of times the *i*-th word appeared in the text.

Algorithms:

The legal counsellor is a classic binary classification problem. Since the problem is a basic text based learning problem, we consider using 3 different learning algorithms: SVM, logistic

regression, and GDA. These algorithms can provide both the predicted label and the probability of the label, which we think will be interesting because we could provide a new case with the probability of winning, rather than only predicting whether they will win or not.

So far the SVM and logistic regression have been implemented. Currently, we have built SVM for our application using libsvm, and we have built logistic regression using SciKit-Learn.

After the implementation of these algorithms, we will perform several rounds of error analysis to fine tune the features and clear the noise in the training data. And then we will use cross-validation to select the best model with the highest testing accuracy.

As a metric to evaluate the probability of winning, we will draw calibration plot. We divide the test data into 10 bins. The i-th bin contains the instances with probability of winning in such range: $(i-1)/10 \leq P < i/10$. And we calculate the estimated probability of each bin as the ratio of the number of instances with label 1 to the total number of instances in this bin. With a good probability prediction, the estimated probability of i-th bin should be around $i/10 - 0.5$.

Example Flow:

Using the example case "*Eisenstadt v. Baird*", we show an example flow on how SVM and logistic regression are put in action:

SVM:
The feature vector would be a dictionary of 1-3 word grams with punctuations and stop words removed. We give an inflated rating for phrases (2 word grams and 3 word grams) since these would be more useful in matching for content. For the section of "Issues", it would look like {"there": 1; "there rational": 2; "there rational ground": 3; "rational": 1; "rational ground": 2 … "law": 1}. The output is extracted from the holding section, where we used a regular expression to find out whether words such as "yes", "affirm", "held", or "positive" appear. If they appear, then it means y is 1, otherwise, it's 0. The feature vector and label are stored in a tuple, where the feature vector is a dictionary (from words/phrases to their weights). This is then passed to the algorithm to perform learning and predicting.

The algorithms have training and predicting part. In training part, The features and the labels (holding) are passed in to the algorithm to learn a weight vector. In predicting part, only the features are passed into the algorithm and the algorithm outputs a label. If the label is 1, then the holding is predicted as "Yes", and if the label is 0, the holding is predicted as "No". From this holding, the user can get the answer to his issue. Together with the label, the algorithm will also evaluate a probability. For example, using SVM algorithm, the probability evaluation can be generated by using "-b 1" option.

Logistic Regression:

The feature vector is a (number of examples) X (number of features) matrix. For the features of logistic regression, we do not use 2 word grams and 3 word grams like we do in SVM because the sparse matrix used in the program would exceed most every-day computer's memory limit. Therefore, the number of features is simply the number of all the unique words from all the cases. Matrix(i, j) would be how many times *j*-th word appeared in the *i*-th example. The output is extracted from the holding section as described in the SVM part. Then we feed such matrices to the algorithm for training and predicting similar to SVM.

Preliminary Results:

SVM:

Testing set accuracy: 0.6404

Logistic Regression:

Size of Training Set	Training Accuracy	Testing Accuracy
1000	0.99900	0.640
2000	0.99700	0.635
4000	0.99725	0.625

Plan for Improvement:

We believe there is a lot of room for improvement to our model as the current accuracy is not ideal in both SVM and logistic regression. First, since there is a problem of overfitting, we will experiment with different regularization methods. In addition, the model would likely benefit from more sophisticated feature selection. Currently, no words other than stop words are being removed from the dictionary, but it is likely that many of the words don't have much predictive power and the model could be improved by removing them. We would also like to convert words to their root forms (apples => apple, happily => happy, etc) We are currently looking into consulting some legal experts on what other features would be appropriate to add to the feature vector. And using some NLP package to extract some features such as dependency path might be helpful. Lastly, we could improve the text features either with something standard like tf-idf weighting of the words or we could take advantage of the fact that the case briefs have structured text and experiment with weighting words for different sections differently or excluding some sections altogether.