Patent Cases Docket Classification

Ioannis Antonellis [*]   Panagiotis Papadimitriou [†]

Abstract

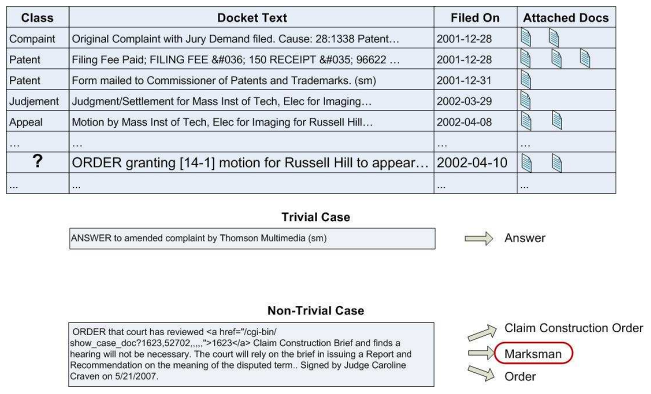
We contribute to the Intellectual Property Litigation Clearinghouse (IPLC) project by providing an extensive experimental evaluation of two classification techniques, namely classification trees and support vector machines. We focus on dockets belonging to the "Claim Construction order" class and we build classifiers that achieve up to 87% precision with 88% recall. This provides a 79% improvement on the best rules-based domain-specific classifier that IPLC possess.

1 Problem Description

Our work was a contribution to the Intellectual Property Litigation Clearinghouse (IPLC) [4]. IPLC aims to be a comprehensive online information source on IP lawsuits and will host general statistical information, as well as text-searchable dockets, complaints, select motions, judicial opinions, and related data.

The dockets that are going to be searchable comprise of 2-3 lines of human-written text and are accompanied by some pdf documents with 20-30 pages of images and text. These dockets are indexed by the Patent Case they refer to and can be classified to approximately 40 different classes such as Order, Motion, Patent, Judgement, etc.. In Fig. 1 we provide an example of the dockets that belong to a specific patent case. The information that the docket text conveys is unstructured, since there are no rules for writing a docket. People seem to follow general conventions for the docket text of particular classes, but this is not true for all the cases. For example, most of the docket texts of class ”Answer” start with the word ANSWER in uppercase letters. This convention results in an easy rule for the identification of dockets from a particular class. However, there are classes where simple rules cannot be derived; an example of such a class (”Marksman”) is shown in the same figure.

A human can classify any docket to a class, since he has access to the pdf documents that are attached to it. However, since the pdf documents are scanned, we need to apply some OCR method to get their content in machine-readable format and provide it as input to a learning algorithm. Given the cost of such a procedure we cannot consider it as feasible and any classification algorithm should constrain its input to the rest of the features that are available. The goal of our project was in the first place to investigate whether there is enough information in the rest of the features for the classification of a docket and then build an optimal classifier if that was possible.



Class	Docket Text	Filed On	Attached Docs
Compaint	Original Complaint with Jury Demand filed. Cause: 28:1338 Patent...	2001-12-28	📄 📄
Patent	Filing Fee Paid; FILING FEE &#036; 150 RECEIPT &#035; 96622 ...	2001-12-28	📄 📄 📄
Patent	Form mailed to Commissioner of Patents and Trademarks. (sm)	2001-12-31	📄
Judjement	Judgment/Settlement for Mass Inst of Tech, Elec for Imaging...	2002-03-29	📄
Appeal	Motion by Mass Inst of Tech, Elec for Imaging for Russell Hill...	2002-04-08	📄 📄
...	...	...	...
?	ORDER granting [14-1] motion for Russell Hill to appear...	2002-04-10	📄 📄
...	...	...	...

Trivial Case

ANSWER to amended complaint by Thomson Multimedia (sm)

Non-Trivial Case

ORDER that court has reviewed <a href="/cgi-bin/show_case_doc?1623,52702,,,,,">1623</a> Claim Construction Brief and finds a hearing will not be necessary. The court will rely on the brief in issuing a Report and Recommendation on the meaning of the disputed term.. Signed by Judge Caroline Craven on 5/21/2007.

Figure 1: Dockets of a Patent Case

2 Dataset

2.1 Description One of the non-trivial classes for classification is the ”Claim Construction Order” (CCO) class. Our Dataset consists of 3674 dockets that were classified by an editorial team. 899 of them are ”Claim Construction Order” (CCO) dockets and the rest 2775 are dockets of other classes. The dockets of the Dataset refer to 1369 different cases and in general case they are not related to each other, e.g. the non-CCO dockets are not responses to CCO dockets.

2.2 Preparation The dockets preprocessing steps performed are the following: i) Lexical analysis; ii) stopword elimination, that is the removal of very frequent words such as articles, prepositions, conjunctions, et. that carry little information about the contents of the processed dockets; iii) stemming, that is the replacement of all variants of a word with a single common

[*]Computer Science Dept, Stanford University. Email: antonell@cs.stanford.edu

[†]Electrical Engineering Dept, Stanford University. Email: ppapadim@stanford.edu

Table 1: Local and Global term weighting schemes

Symbol	Name	Formula
	Local term-weighting (l_{ij})	
`t`	Term frequency	f_{ij}
`b`	Binary	$b(f_{ij})$
`l`	Logarithmic	$\log_2(1 + f_{ij})$
`a`	Alternate log	$b(f_{ij})(1 + \log_2 f_{ij})$
`n`	Augmented normalized term frequency	$(b(f_{ij}) + (f_{ij}/\max_k f_{kj}))/2$
	Global term-weighting (g_i)	
`x`	None	1
`e`	Entropy	$1 + (\sum_j (p_{ij} \log_2(p_{ij}))/\log_2 n)$
`f`	Inverse document frequency (IDF)	$\log_2(n/\sum_j b(f_{ij}))$
`g`	GfIdf	$(\sum_j f_{ij})/(\sum_j b(f_{ij}))$
`n`	Normal	$1/\sqrt{\sum_j f_{ij}^2}$
`p`	probabilistic Inverse	$\log_2((n - \sum_j b(f_{ij}))/\sum_j b(f_{ij}))$

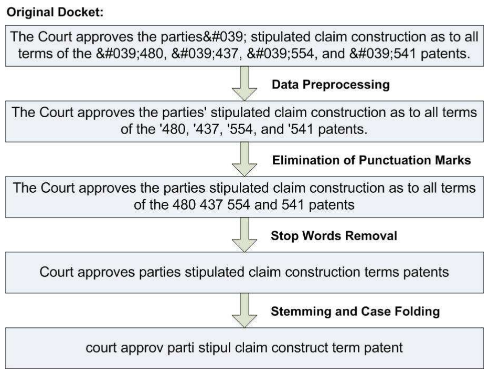

Original Docket:
The Court approves the parties&#039; stipulated claim construction as to all terms of the &#039;480, &#039;437, &#039;554, and &#039;541 patents.
Data Preprocessing
The Court approves the parties' stipulated claim construction as to all terms of the '480, '437, '554, and '541 patents.
Elimination of Punctuation Marks
The Court approves the parties stipulated claim construction as to all terms of the 480 437 554 and 541 patents
Stop Words Removal
Court approves parties stipulated claim construction terms patents
Stemming and Case Folding
court approv parti stipul claim construct term patent

Figure 2: Data Preparation

stem; iv) index-term selection, that is the selection of a subset of words encountered in the dockets to form the docket index;v) index construction. These steps are illustrated in Fig. 2. The output of this process is a term document matrix A. Each element α_{ij} of the term document matrix A measures the importance of term i in docket j and in the entire collection. There has been proposed various term weighting schemes using alternative functions for the local and global weighting for a term. Table 1 tabulates the various local and global weighting schemes we considered in our experiments. Another usual practise is the use of normalization for each docket vector. This normalization factor is used for the obliteration of bias towards longer documents. For the implementation of the preprocessing step we used the matlab toolbox Term Matrix Generator (TMG). The dimension of our feature vector is $n = 4947$.

After the extraction of the feature vector for each docket we divided our dataset into a training and a test set. The training set included 70% of the positive training examples and 70% of the negative examples of the original dataset, and the test set included the rest of the examples. We performed the partition of the dataset using *reservoir* sampling to guarantee the size of the resulting subsets.

3 Baseline Approach

The algorithm that is currently used for the classification of the dockets is rule-based. These rules derive from human heuristics and take advantage of domain-specific knowledge. To evaluate the performance of this algorithm, as well as the classification methods we propose, we use standard performance measures such as precision, recall and F-1 measure. We present the corresponding formulas for the classification of CCO documents in the following equations:

$$\text{Precision} = \frac{\text{classified in CCO} \cup \text{dockets in CCO}}{\text{classified in CCO}}$$

$$\text{Recall} = \frac{\text{classified in CCO} \cup \text{dockets in CCO}}{\text{dockets in CCO}}$$

$$F = 2\frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

The performance measures for the used classification method and the class CCO are shown in Eq. 3.1. We observe that precision is high and recall is small; this is so that the classification method can return a few false negative dockets. An editorial team reviewed the positive dockets returned by the algorithm and filtered out the true CCO dockets. We use the performance measures of this classification method as a baseline for comparison with our proposed methods.

$$\text{Precision}_{\text{base}} = 0.98$$
$$\text{Recall}_{\text{base}} = 0.32$$
$$\text{F}_{\text{base}} = 0.49 \tag{3.1}$$

4 Classification Trees

Tree-based methods for classification and regression partition the feature space into a set of rectangles and then fit a simple model in each one. The most common such model is the constant function. For our problem, we used CART [1], a popular method usually used in regression and classification problems.

The conceptual simplicity of the CART yields its limitations. For example, the partition of the feature-space into rectangles fails to capture non-rectangular distributions of the data. Trees can approximate other distributions only with a big number of small rectangles that increase the complexity and, consequently, the variance of the model. There are several proposed methods that handle CART's limitations such as bagging [2] and boosting [3], but they are out of the scope of this paper.

Despite their limitations, in our work we used only naive tree-based methods, because of the great interpretability of the classification algorithm decisions that they offer. As we discuss below, such trees reduce the classification problem into binary decisions on values of feature vector's dimensions. Our goal using classification trees was to gain an insight of the significant dimensions of the feature vectors that actually have an impact on classification. Since CART is conceptually simple, we could convey this insight to people with little or no knowledge of machine learning.

4.1 Description In our case we deal with a binary classification problem. The target variable has value 1 if a training example belongs to the class "Claim Construction Order" and value 0 if the example does not belong to this class. Our feature vector x_i shows the frequency of the words of our dictionary in docket i (corresponding to the weighting scheme `txx` as described in Section 2.2).

Let R_1 be the feature space of our training examples. In our case $R_1 = \mathbb{R}^n$ and it contains all $m_1 = m$ training examples. Let p_{10} be the ratio of examples in R_1 that do not belong to "Claim Construction Order" class. Similarly, we define the ration p_{11} for examples that do belong to the class. In general, for region R_k we define:

$$p_{k0} = \frac{1}{m_k} \sum_{x_i \in R_k} I\{y_i = 0\} \tag{4.2}$$

$$p_{k1} = \frac{1}{m_k} \sum_{x_i \in R_k} I\{y_i = 1\} \tag{4.3}$$

We start the training of our algorithm by classifying all the training examples of region R_1 to the majority value of the response variable y. Since most of our training examples do not belong to class "CCO", the most naive classifier would predict that a given docket does not belong to this class (since it tries to minimize the classification error). We define the impurity Q(1) of area R_1 using the 'Gini' index that we show in Eq. 4.4 for area R_k.

$$Q(k) = p_{k0}(1 - p_{k0}) + p_{k1}(1 - p_{k1}) \tag{4.4}$$

The 'Gini' index is an indicator of how "pure" an area is in terms of the different values that the target variable takes in that area. We get $Q(k) = 0$ if all training examples of one area belong to the same class.

The next step of the algorithm depends on the value of the *complexity parameter cp* and the gain $Q(1) - (Q(2) + Q(3))$ we obtain if we partition our training set into two distinct subsets that span regions R_2 and R_3 of our feature space, so that $R_2 \cap R_3 = \emptyset$ and $R_2 \cup R_3 = R_1$. First we explain how we determine the optimal partition of region R_1 and then we give more details in the role of *cp*. We partition the region R_1 into regions R_2 and R_3 in a way such that we can determine whether an element $x_i \in R_1$ belongs either to R_2 or R_3 with a binary decision based on the value of only one specific dimension of its feature vector. The binary decision is whether x_i's value for the specific dimension is le or $>$ than one splitting value. We select the dimension and the splitting value so that the gain $Q(1) - (Q(2) + Q(3))$ we have in the purity of the new regions R_2 and R_3 is maximized. In our case, the splitting of a region based on a value of a specific dimension of the feature vector yields partition of the dockets into two subsets based on the frequency of one specific word. For example, if word "constru" (derived from construct, construction, constructed, etc.) is the word that divides the dockets into two as pure as possible subsets, we will divide the dockets in these two subsets. In the best case, where "constru" appears in all dockets of our class and no document that does not belong to our class has the word "contru" in it, we will divide the dockets into two pure subsets and we get the maximum possible gain.

It is easy to see that without penalizing a region splitting we would stop splitting only in cases where we have pure areas, since it is easy to prove that we can always split an impure region into two, having a positive gain in the purity of the resulting regions. However, we limit the number of possible distinct regions by adding to the impurity of area R_k the term *cp*. Hence, in our case it would be worthwhile to split region R_1 into regions R_2 and R_3 only if the gain $Q(1) + cp - (Q(2) + cp + Q(3) + cp) = Q(1) - (Q(2) + Q(3)) - cp$ was positive.

After splitting a region into two, the next step of the algorithm requires to find first which region would yield the maximum gain and then find the optimal split for this region. In this optimization problem we should take into consideration that a region splitting that seems redundant may give the opportunity for subregions' splitting with great gain. To deal with the recursive nature of the optimization problem we construct the classification tree in a bottom-up fashion rather than a top-down that we described so far. We build initially a very large tree and then we prune it in such a way so that we maximize the purity of the nodes taking into consideration the complexity parameter that is associated with each terminal node.

A slight variation of the algorithm above uses a weighted version of the 'Gini' index of Eq. 4.4. This variation is used to penalize more the impurity of regions with respect to one class than the other. We show the updated formula in Eq. 4.5.

$$Q(k) = L_0 p_{k0}(1 - p_{k0}) + L_1 p_{k1}(1 - p_{k1}) \tag{4.5}$$

The new formula can result in different decisions for node splitting, since if $L_0 > L_1$, for example, we obtain a greater gain if we split areas where the majority of the examples belong to class '1' and, hence, we have misclassified examples of class '0'. [[probably give more detail]]

The algorithms finally returns a decision tree where each branch illustrates a decision on specific dimension of the features vectors. As a result, the dimensions of the feature vectors that do not appear in any branch are completely ignored during the classification of a new feature vector. In our context, that means that docket words that do not appear on the tree do not play any role in the classification of new docket that does not belong to our training set. Hence, the words that actually appear in the tree branches can be considered as the significant ones in our classification problem.

4.2 Tuning Classification Trees We fitted different classification trees to the data of our training set and test their performance measures in the classification of the test set. To construct different trees we varied the parameters L_0 and *cp*.

In particular, we considered the value of $L_1 = 1$ as fixed and we varied the parameter L_0 in the interval [0.2,4]. For each value of L_0 we varied the complexity parameter *cp* to all possible values that are greater than 0.0001. These values are finite if we consider only one value in every interval that results in a different pruning of the large tree we initially construct. Finally, for each L_0 we kept only the tree (an the corresponding value of *cp*) that had the smallest misclassification error on the test set.

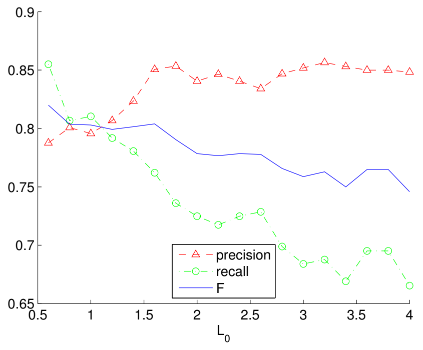


Figure 3: Performance Measures of Classification Tree for Varying L_0

4.3 Results We show the performance measures for the optimal tree we obtained for each value of L_0 in Fig. 3. The x-axis of the plot show the values of L_0 an the three curves correspond to the precision, recall and F performance measures. We see that the value of L_0 that maximizes the F measure is 0.6. We see that as L_0 increases we see an increase in the precision, because our classification tree classifies only the pure nodes as CCO and they tend represent approximately 85% of the total number of CCO dockets. The change of the weight L_0 cannot change the classification decision for these regions, sine there are no misclassified dockets that are not CCO. On the other hand, recall decreases because we misclassify more and more positive examples as negatives in non-pure nodes, since this is not penalized because of the increase of L_0 with respect to L_1.

In Eq. 5.7 we provide the performance measures for the optimal tree. We see that with the appropriate tuning of this simple classifier we get an improvement of +68.5% with respect to our base case.

$$\begin{aligned} \text{Precision}_{\text{tree}} &= 0.79 \\ \text{Recall}_{\text{tree}} &= 0.86 \\ \text{F}_{\text{tree}} &= 0.8 \\ \text{Improvement} &= +68.5\% \end{aligned} \tag{4.6}$$

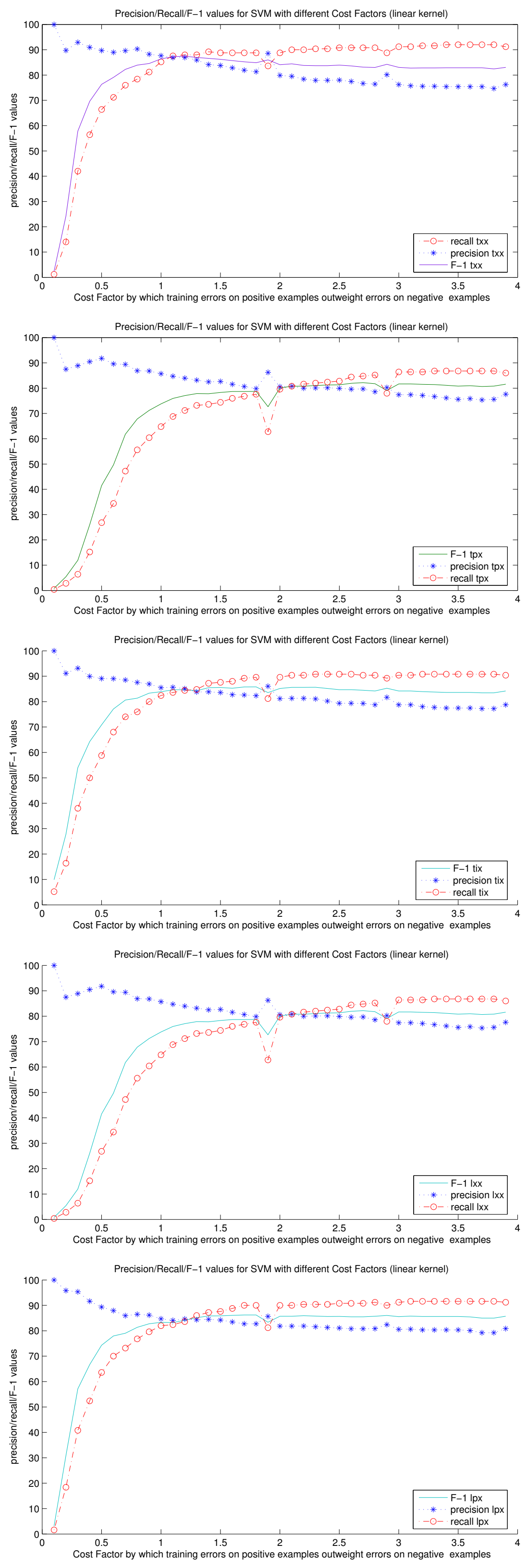


Figure 4: Comparing Precision/Recall for SVM classifiers with different cost factor and for five different indexing schemes ((i) term frequency (txx) (ii) term frequency/probabilistic inverse (tpx) (iii) term frequency / inverse document frequency (tix) (iv) logarithmix (lxx) (v) logarithmix/probabilistic inverse)

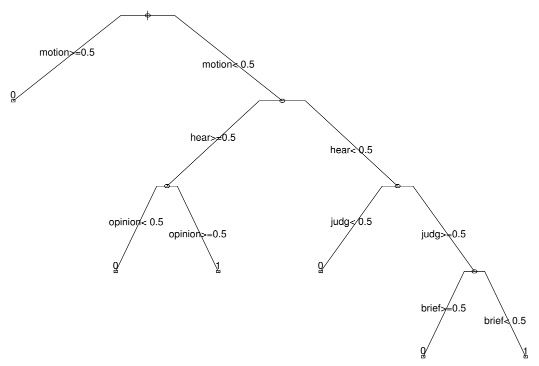


Figure 5: Pruned Optimal Tree

We show a pruned version of the optimal tree in Fig. 5. The words that were actually used in the construction of the optimal tree are: brief, constru, coordin, deni, disput, file, hear, hrg, judg, motion, manag, notic, opinion, parti, patent, plaintiff, schedul, set, staff, stipul, strike and summari. We see that out of the approximately 5000 dimensions of our feature vectors, classification trees achieve an 68.5% by utilizing only 22 of them.

5 Support Vector Machines (SVM)

5.1 Description Support vector machines (SVMs) are a set of related supervised learning methods used for classification and regression. They belong to a family of generalized linear classifiers. A special property of SVMs is that they simultaneously minimize the empirical classification error and maximize the geometric margin; hence they are also known as maximum margin classifiers. Support vector machines map input vectors to a higher dimensional space where a maximal separating hyperplane is constructed. Two parallel hyperplanes are constructed on each side of the hyperplane that separates the data. The separating hyperplane is the hyperplane that maximizes the distance between the two parallel hyperplanes. An assumption is made that the larger the margin or distance between these parallel hyperplanes the better the generalisation error of the classifier will be.

5.2 Tuning SVMs We performed a set of experiments using support vector machines, and specifically Thorsten Joachims SVMlight package, as binary classifiers for dockets. We used a linear kernel function and we varied the cost-factor; the factor by which training errors on positive examples outweight errors on negative examples. We report results for cost factors in the interval $[0.1, 4]$ with a step of 0.1. We did not modify any of the parameters of the SVMs we trained based on the test set results, so we did not do a second level development set/test set split.

Also, for each docket we used all combinations of local and global term weighting schemes from Table 1 to derive the vector space representation of a docket.

5.3 Results Figure 4 illustrates the precision, recall and F-1 measure for five different combinations of term weighting schemes and variations of the cost factor used in SVMs. Overall we noticed that simple term frequency can yield the maximum F-1 value. Specifically the performance results in that case are:

$$\begin{aligned} \text{Precision}_{\text{tree}} &= 0.8696 \\ \text{Recall}_{\text{tree}} &= 0.88 \\ \text{F}_{\text{tree}} &= 0.87 \\ \text{Improvement} &= +79\% \end{aligned} \tag{5.7}$$

This could be justified from the relatively small size of the content of each docket. Notice that overall SVMs improve the rules-based baseline by a factor of 79%.

6 Conclusions

In this paper, we focused on building classifiers for the dockets belonging in a specific class. This was done, since IPLC provided us training data for only this class. We provided experimental evidence that we can efficiently classify dockets of non-trivial classes using only the docket text. Our optimal classifier yielded an improvement in the current classification method +79%. However, we think that we can build upon our classification techniques and further improve the classification accuracy by taking advantage of docket feature that we were not provided with. These features include the names of the attached documents to each docket and the sequence of docket classes that are formed in a patent case.

Future work includes the investigation of classification methods that take into consideration the aforementioned additional features.

References

[1] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees.* Wadsworth and Brooks, Monterey, CA, 1984. new edition [**?**]?

[2] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[3] J. Friedman. Greedy function approximation: a gradient boosting machine, 1999.

[4] Mark A. Lemley and J. H. Walker. Intellectual Property Litigation Clearinghouse: Data Overview. *SSRN eLibrary*, 2007.