Automated Patent Classification

CS229/CS229A: Machine Learning

Ian Christopher • Sydney Lin • Sigurd Spieckermann

December 17, 2011

Abstract

The goal of this project is to automate the process of classifying patents under the hierarchical International Patent Classification System (IPC). In this age of innovation, the war over intellectual property (IP) becomes common between companies and even individuals. Although a patent protects the ownership of IP, its application process is costly and slow. Moreover, most patent lawyers nowadays manually classify patent applications based on their knowledge, experience and individual research. Therefore, automation on patent classification not only helps to reduce human error that might lead to expensive cost, but also accelerate the application process.

1 Introduction

Our system is aiming to categorize a query patent under a five-leveled hierarchical classification structure of different sections, classes, subclasses, groups and main groups or subgroups. In the training step, our classification algorithm analyzes the abstract, title, author (plus company, if applicable) and citations of a given patent and learns patterns between these features and its assigned class(es). In a fair amount of cases, patents can fall under different categories and it may be difficult for a human expert to accurately classify a patent.

Due to our inexperience with patent patent law, we met with a patent lawyer in the area in order to get a better understanding of common practice and computer-aided tools in this business. After the conversation with him, we were surprised that for new patents applications, most patent lawyers mostly rely on their knowledge and research on the World Intellectual Property[1] Organization website with key words. This increases our motivation on applying machine learning to solve the multi-class classification problem.

2 Background

Text classification is a well-known area of pattern recognition and information retrieval. Though there has been a sufficient research in the area, we tried to keep our effort largely original. That being said we did use a number of publications to guide us when results started to stall or there were too many possible next steps to choose from.

There were a number of basic text classification papers that we read to better understand the space. [SL01] was a well cited survey of

[1] `http://www.wipo.int`

sorts into hierarchical text classification. Their metrics for classification performance was especially interesting. [RS02] used neural networks to address the problem. [WL09] used support vector machines and produced competitive results. [QHLZ11] helped introduce the use of latent features and their importance to the problem. [XXYY08] had very good results using a two step classification scheme for each patent that involved pruning the hierarchy before actually classifying a document. We even found a paper that was also trying to classify patents using text [CIW] but it ignored the hierarchical nature of the problem.

In addition to browsing text classification papers, we considered other types of hierarchical classification problems. In particular [ima] provided background of a classification competition on the Imagenet data set and algorithms that seemed more successful on image data. Google research's [SZYW10] provided information about classifying videos.

3 Patent Data

Our data source is bulk downloads of patent data from Google Patents, which originates from the United States Patent and Trademark Office (USPTO). Google hosts a number of different data sets within the patent legal space including patent applications, patent grants, and maintenance events related to the sets. We use weekly bibliographic data on granted patents, which Google provides for all patents since 1976.

3.1 Structure

One of the challenges of dealing with this data is the sheer size of the data sets. A typical week in 2010 comprises several thousand granted patents. Each of these weekly sets takes typically under 10 MBytes compressed and approximately 90 MBytes uncompressed in size. Consequently, a single year of patents takes up about 4.5 GBytes of space. Relatively speaking this is not a massive amount of data but on commodity hardware it is a nontrivial amount as we are intending to use many years of data. It is also worth noting that the number of patents each year roughly increases as time moves on so earlier data sets are smaller. Another issue with Google's data turns out to be the inconsistency in its structural representation across different years. More specifically, the data over the last ten years is provided according to two different XML schemata and one prefix-based format.

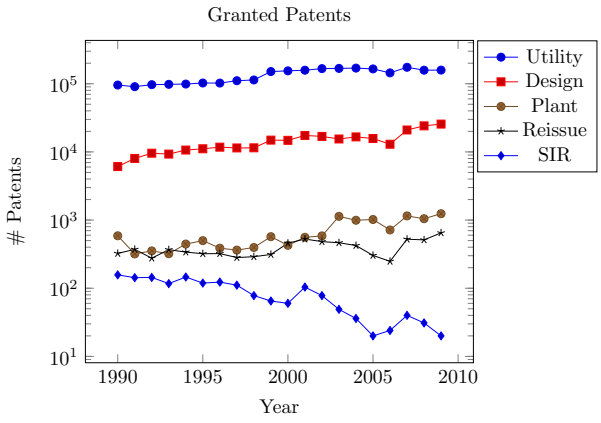


Figure 1: Granted Patents: Frequency of Types

3.2 Content

There exist different types of patents—utility patents, design patents, reissue patents, statutory invention registrations, and plant (like corn/beans/trees) patents—with different properties regarding their information schemata. Utility patents are the most common types of patents and the ones we are focusing on, but the data set contains patents of all types. They include the information that we anticipate to be meaningful features as opposed to other types—in particular design patents that make

the largest fraction of the non-utility patents, but merely consist of very few key words and no classes. We wonder about their usefulness, but do not dive deeper into this question.

Further, there exist various classification system in different parts of the world besides the International Patent Classification System (IPC). In general, there is no one-to-one mapping between the different systems, but United States patents are classified by a national classification system as well as by the IPC. This has lasting effects on the rest of our work as we decide to use the IPC as the basis of our classification effort due to its deeper hierarchy.

3.3 Preprocessing

The first step in handling the data is to parse it into a more appropriate format. The data set provided by Google contains a significant amount of redundant information represented in the notoriously verbose XML format. As a result, the data set is largely space-inefficient for our purposes and hence wastes storage capacity as well as computation time in subsequent processing steps. We decided to convert the data into a comma-separated file format that only contained information relevant to our problem. This step resulted in a compression ratio of the order ten to one making storage of patent data across many years much more feasible.

In the second step, we preprocess the patent text data in multiple ways to improve the performance of our learning algorithms.

- **Stop-word and punctuation removal** Stop-words do not contribute to the richness of information of a text and punctuation is difficult to handle properly by machines. By removing both, the next is normalized and better suited for our purposes. We remove punctuation by means of the string-library in Python that contains a complete set of punctuation characters and replace all occurrences of those in our text data. Stop-words are identified and removed by means of the Natural Language Toolkit Development (NLTK) library in Python using the English corpus.

- **Stemming** In order to achieve invariance with respect to inflected forms, we reduce words to their stem using the Porter stemmer provided in the NLTK library.

- **Mutual information and frequency count** Only a subset of words in text corpora are often indicative of the content of a text. In order to retain the most relevant words and thereby limit the dimensionality of our dictionary, we combine the mutual information (MI) metric and the frequency counts words by intersecting both sets, which are each sorted in descending order of their values. Although MI is only defined to relate words with one particular label—in our case patent classes—, we obtain an overall scalar metric that attempts to generalize the relevance of a word to all classes by summing the quantities for a word with respect to a particular class over all classes. However, it is prone to give high weight to rare words that are greatly nonuniformly distributed across classes while they are in fact not particularly indicative in general.

- **Latent Semantic Analysis** Our above-described steps only address cosmetic changes and statistical filtering. By applying Latent Semantic Analysis (LSA) to our bag-of-words matrix, we attempt to identify similar semantic meanings of words and project onto a lower-dimensional subspace of abstracted semantics. This step will

prove vastly beneficial in later sections of this report.

In the third step, we generate additional features by computing the joint probabilities between an assignee as well as the assignees country of origin and each class we are considering in our classification task. Finally, we construct the the binary-valued matrix of classes, that a patent is categorized by and export all data to a MATLAB data file using the SciPy Python library.

4 Classification

Our first goal is to accurately classify patents into the first level of the classification hierarchy. In the second step, we consider two levels of the hierarchy and flatten out the tree structure, hence, we attempt to classify to approximately 150 different subclasses. Third, we perform a hierarchical classification task by training a model for each level and passing patents with active predictions on to the next level. In these steps, we compare a number of different learning algorithms:

- Logistic Regression
- Linear L2-regularized L2-loss soft-margin Support Vector Machine
- Multi-layer Feedforward Neural Network (+ GPU implementation)

We implemented all algorithms, except for the SVM, ourselves in MATLAB and the GPU Neural Network in C++/CUDA by means of the NVIDIA linear algebra library CUBLAS and the open source library Thrust which is the CUDA-equivalent of the STL in C++. The cost function of the Logistic Regression algorithm and the Neural Network are optimized using the L-BFGS optimization algorithm. The GPU implementation of the Neural Network uses a gradient descent optimization method.

4.1 Algorithms

All algorithms except for the Neural Network are binary classifiers. In order to achieve classification with multiple simultaneous activations, we follow the one-vs-all methodology and learn a model for each class separately. Neural Networks are naturally capable of performing a multi-class multiple activations classification task. In consequence of only few patents being assigned to a certain class our data quite skewed which especially diminishes the performance of the binary classifiers. In order to account for the data skew, we up-sample the patents with positive class label giving a noteworthy improvement. In order to comprehensively assess the performance of our algorithms, we utilize four common metrics: accuracy, precision, recall and F-Score. The accuracy alone may be misleading in some cases, e.g. when the data we train on is skewed. The other three metrics give a better insight into the actual performance and are in fact still relevant after up-sampling the data because this step only introduces balance artificially.

During early stages of testing our various algorithms on a 10 weeks data set of patents from 2011, we observe that the bag-of-words feature matrix with raw frequency counts is suboptimal because documents with a larger text corpus cause larger values in their corresponding row. In order to account for this issue, we normalize the document word frequencies and see a significant improvement in the classification quality.

4.1.1 Logistic Regression

In a first step, we start with a one-vs-all regularized Logistic Regression classifier as a baseline

because it is simple to implement and because it provides a good basis for the later evaluation of our more advanced learning algorithms. Logistic Regression performs well for its implementation complexity and performs best for a bag-of-words feature matrix reduced to 200 dimensions using LSA. The result of this setting yields an F-Score of approximately 0.9 on the first level of our test set after training the classifier with L-BFGS. In the following steps, we use the Logistic Regression implementation to verify the performance of the other learning algorithms.

Because the regularization term does not improve the F-score, we suspect logistic regression might under fit the data set. Consequently, we implement a weighted logistic regression to improve the F-score. At early stage of developing algorithm, we defined accuracy with the assumption that there is only one active label. However, the accuracy was below what we had hoped for. We stopped further work on this direction because the low accuracy was very likely to be a result of overfitting the data set.

4.1.2 Backpropagation Neural Network

Neural Networks are capable of classify patents into multiple active classes simultaneously which makes them an attractive algorithm to use. In a first implementation in MATLAB, we implement a fully vectorized Backpropagation Neural Network with one hidden layer and choose the number of hidden units close to the number of output units. Coincidentally this architecture gives us some of our best results compared to others architectures with a single hidden layer. Beyond adjusting the number of hidden units in a single hidden layer, we take the following steps to arrive at our final version of this algorithm:

- **Normalized Bag-of-Words** The normalized bag-of-words feature matrix yields an F-Score of approximately 0.9 on the test set and slightly above that for the training set after tuning the network parameters.
- **Latent Semantic Analysis** A further approach to improve on the classification quality drives us towards applying LSA to our bag-of-words in order to capture semantic similarity between words. However, our scores for both, training and test set, drop by about 15% which is indicative of a bias problem. Different configurations of the single hidden layer Neural Network do not seem to improve results.
- **Latent Semantic Analysis + 2 Hidden Layers** We extend our implementation to handle multiple hidden layers in order to realize a more complex model. The number and sizes of the hidden layers are specified by a vector whose entries denote the number of hidden units for each layer. A 100-dimensional feature space using 200 hidden units for the first, 20 units for the second hidden layer and a regularization parameter of 0.3 give best results across all our tests.

The convergence plot shown in Figure 2 gives a better understanding of the relationship of the number of iterations and the performance of the algorithm. Overall, we notice that the cost functions of our networks are difficult to optimize and even though we use the L-BFGS algorithm, we often terminate early in local minima and have to repeat the optimization process with a new set of random initial values or slightly modified network parameters.

4.1.3 Backpropagation Neural Network (GPU)

Heavy training of the Neural Network with two hidden layers requires several thousand iterations of the L-BFGS algorithm. Unfortunately

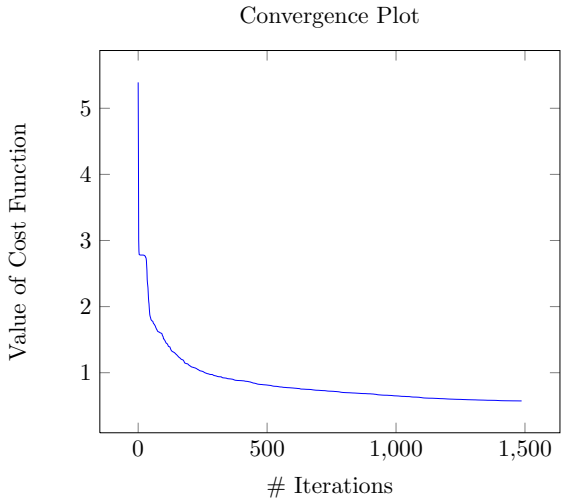


Figure 2: Neural Network Convergence Plot (1 Level)

this takes hours to run on the cluster we are using. Luckily our group has some experience with general purpose GPU programming, so we decide to port the Neural Network to CUDA. For simplicity and feasibility in the context of this project, we implement the gradient descent optimization algorithm to minimize the cost function instead of L-BFGS.

One of our tools for this implementation is the Thrust Library, which provides a high level interface for GPU programming. "High-level" here is relative term as the code is still low level. Regardless it helps speed up development with matrix multiplication methods and a number of other high level utilities.

In our implementation, matrices are represented using Thrust vectors and managed by a custom matrix class which handles the dimensions and wraps required linear algebra operations. Matrix multiplications are generally executed using CUBLAS whereas element-wise matrix multiplications and reduction operations—e.g. required to compute the cost function—utilize optimized CUDA implementations of the Thrust library. We also implement a few specialized kernels in order to combine smaller operations in one kernel launch.

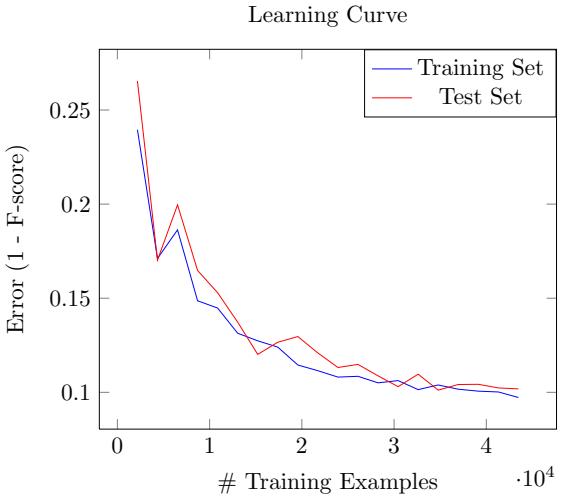
Our GPU implementation of the Backpropagation Neural Network approximately yields a factor 20 speed-up over the vectorized MATLAB implementation that we modify to use gradient descent as well for fair comparison. However, we notice that the gradient descent optimization algorithm is vastly inferior to the L-BFGS algorithm and is in fact unable to optimize the cost function enough to train the Neural Network. We make this observation for both implementations—MATLAB and C++/CUDA. Nevertheless, we appreciate the speed-up and refer to future work to implement an advanced optimization algorithm on the GPU.

4.1.4 Linear L2-regularized L2-loss soft-margin Support Vector Machine

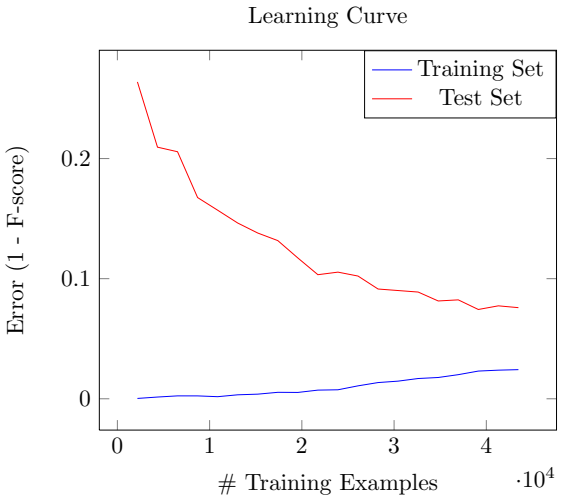
Support Vector Machines are a typical learning algorithm for text classification in many of the papers we looked at so we decide to apply them to our problem. Conveniently linear support vector machines are available through the popular library liblinear so testing them is a relatively quick and easy task.

Though plugging the SVM library into our code is simple and compare to the Neural Network there are fewer parameters to tweak. It turns out that we get best results for a normalized bag-of-words matrix reduced through LSA to a 750-dimensional feature space plus the assignee and assignees country of origin features as described above. The performance of the SVM is very good and compares to our optimal results using the Neural Network. In terms of execution time, it is noticeably faster although we train one model per class in consequence of the one-vs-all method we use. We also tried to use nonlinear SVMs through the library but unfortunately

they do not seem to perform well and training is slow.



(a) Logistic Regression



(b) Linear SVM

Figure 3: Learning Curves

4.1.5 Hierarchy Discussion

After getting sufficient results on the first layer of the hierarchy, we move on to tackle the hierarchy. Taking baby steps, we decide to concentrate on just the first two levels until we have good enough accuracy to move on to the others. Nonetheless we try to develop methods that would scale to more levels if we choose to do so.

We approach the hierarchy problem in two distinct ways. Because we are trying to classify each patent to at least one leaf node, we flatten the hierarchy to a single level that just consists of leaf nodes. The other approach is to classify recursively on each level of the hierarchy until we reach the bottom of the tree. Ultimately we are left with comparable results between the two which actually seem to agree with a few results we found in literature before implementing.

The flat approach is the easiest to implement because it only means changing the labels matrix in our preprocessing. After this we can just run algorithms for dealing with just the first level on the data set. As such, this approach is often called for training for each of the one hundred plus labels on the second level. Consequently this approach becomes much slower than the first level code so we are unable to optimize parameters as well.

The standard recursive approach is slightly more difficult to implement but does not take too long. We actually have a number of different versions of this approach depending on classification parameters and how often we want to run latent semantic analysis (top level, every level, etc.). Because many of the classification algorithms are one vs all, we actually have more train runs here than in the flat approach, but here the training sets are smaller as we only train on patents that can be classified by the current node.

5 Results

6 Future Work

There are a number of ways that we could move forward with our work in the future.

(a) Classification on 1 Level

	Accuracy	Precision	Recall	F-Score
Logistic Regression	0.973908 / 0.973689	0.848143 / 0.847178	0.962301 / 0.861350	0.901623 / 0.900660
Neural Network	0.984336 / 0.983452	0.985644 / 0.982583	0.902030 / 0.898394	0.941985 / 0.938604
Linear SVM	0.986899 / 0.981362	0.922070 / 0.901788	0.990729 / 0.973069	0.955167 / 0.936073

(b) Classification on 2 Levels

	Accuracy	Precision	Recall	F-Score
Neural Network	0.995011 / 0.994597	0.957578 / 0.933084	0.594714 / 0.573634	0.733735 / 0.710483
Linear SVM (flat)	0.998595 / 0.988762	0.892609 / 0.508447	0.998661 / 0.818083	0.942662 / 0.627128
Linear SVM		0.6160	0.8590	0.6968

Table 1: Results for Training/Test Set

- **Classify further down the hierarchy** Much of our time has been spent on classifying on just the first level. We held off on classifying on the second level until we had sufficiently accurate results because we were worried that otherwise we would get bad results.

- **Larger Datasets** The size of the raw XML data is one the order of several gigabytes per year. Though we can effectively compress these files by picking out pertinent metadata from XML, this is still a large set without the help of a database if we want to hold decades worth of data without a database. A larger dataset would help us with the uncommon classes and dive deeper in to the classification hierarchy (which has over sixty thousand leaf nodes).

- **Additional features** There were a number of fields in the raw XML data that we ignored. Though most of them do not seem useful, one in particular could be very helpful; patent citations. Unfortunately only using only a year of data makes it hard to resolve these citations, but if we had a database we could construct numerous features from them (citation classes, graph relationships, titles, etc.).

- **Hierarchy pruning** A number of the more successful papers we read, used a two step classification approach. During the first step a lightweight similarity metric was used to prune the tree, leaving only plausible categories remaining. After it would classify in this pruned hierarchy. Of course this would mean more training, but the results might be worth it.

- **GPU optimization techniques** At the moment, we use gradient descent in our GPU neural network propagation algorithm. We use this algorithm due to its implementation simplicity but we might be able to use a more powerful optimization algorithm to speed up our results there. In particular, BFGS seems like a prime candidate to implement.

References

[CIW] Ioana Costantea, Radu Ioan, and Bot Gert Wanka. Patent document classification based on mutual information feature selection.

[ima] Large Scale Visual Recognition Challenge 2010. `http://www.image-net.org/challenges/LSVRC/2010/pascal_ilsvrc.pdf`.

[QHLZ11] X. Qiu, X. Huang, Z. Liu, and J. Zhou. Hierarchical text classification with latent concepts, 2011. `http://www.aclweb.org/anthology/P/P11/P11-2105.pdf`.

[RS02] Miguel E. Ruiz and Padmini Srinivasan. Hierarchical text categorization using neural networks. *Information Retrieval*, 5:87–118, 2002. 10.1023/A:1012782908347.

[SL01] Aixin Sun and Ee-Peng Lim. Hierarchical text classification and evaluation. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pages 521 –528, 2001.

[SZYW10] Y. Song, M. Zhao, J. Yagnik, and X. Wu. Taxonomic classication for web-based videos. 2010.

[WL09] X L Wang and B L Lu. Improved hierarchical svms for large-scale hierarchical text classification challenge. *Large scale hierarchical text classification*, (60903119), 2009.

[XXYY08] G.R. Xue, D. Xing, Q. Yang, and Y. Yu. Deep classification in large-scale text hierarchies. In *Proceedings of the SIGIR conference on Research and development in information retrieval*. ACM Press, 2008.