

Local Induction of Decision Trees: Towards Interactive Data Mining

Truxton Fulton¹
Simon Kasif¹
Steven Salzberg¹
David Waltz²

trux@cs.jhu.edu
kasif@cs.jhu.edu
salzberg@cs.jhu.edu
waltz@research.nj.nec.com

Abstract

Decision trees are an important data mining tool with many applications. Like many classification techniques, decision trees process the entire data base in order to produce a generalization of the data that can be used subsequently for classification. Large, complex data bases are not always amenable to such a global approach to generalization. This paper explores several methods for extracting data that is local to a query point, and then using the local data to build generalizations. These adaptively constructed neighborhoods can provide additional information about the query point. Three new algorithms are presented, and experiments using these algorithms are described.

Keywords: Local Learning, Decision Trees, Data Mining.

¹Computer Science Dept., Johns Hopkins U., Baltimore, MD

²NEC Research Institute, Princeton, NJ 08540

1 Introduction

For any large, complex body of data, there is often a need to compute summaries and extract generalizations that characterize the data. Data mining research focuses on processing large databases and computing summaries, detecting patterns, and performing automatic classification on new data. For example, modern medical databases contain enormous quantities of patient data, which can provide great value in the treatment of future patients. In order to gain the maximum value from such databases, data mining tools are essential. One popular and successful data mining technique is the decision tree classifier [BFOS84, Qui93, MKS94] which can be used to classify new examples as well as providing a relatively concise description of the database.

In this paper we describe a notion of interactive data mining where we wait for the user to provide a “query” that specifies a neighborhood to be mined. The query is in the form of a specific example from the space of instances in the database. We then produce information that may contain the classification of the point, the confidence in this classification, a summary or a visual display of the local neighborhood around the point, a comparison of this neighborhood with others, or additional local information.

Our approach is also intended to address a statistical problem that is well known in the decision tree research community as *data fragmentation*. This occurs even in very large databases as the tree induction algorithm recursively splits the data into smaller and smaller subsets. Thus at many leaves of a decision tree, very little data is available to make classification decisions. However, given a specific query point, if one can retrieve all the locally relevant instances in the database, then one should be able to build a better classifier.

This paper describes a local approach to building decision trees: first collecting data in the vicinity of a query example, and then building the tree on the fly. The intuition is simply this: when presented with a new example for classification, retrieve a set of relevant examples from the database, and then build a decision tree from those examples. This approach has the potential to circumvent the data fragmentation problem, if the decision is made using sufficient relevant (local) information.

The idea of building local decision trees is similar in spirit to the standard k-nearest neighbor algorithm. Both ideas face an important implementation question, though, which is: how does one decide the appropriate local neighborhood? This problem is addressed in the algorithms described below, each of which approaches it differently. This paper also explores some more sophisticated methods of choosing a neighborhood. For some databases and some domains, the importance of certain features varies from one part of the space to another. For example, the feature “blood pressure” might be of great importance in one part of a medical database, while genetic factors might be most important elsewhere. To capture this notion more formally, we have devised an algorithm that defines an adaptive neighborhood based upon local characteristics of the data. This extends the usual notion of distance and makes it both domain-dependent and query-dependent.

2 Notation

In this section we give several definitions that are necessary in the algorithm descriptions. For length consideration we will omit formal notation. Let X be a set of instances called the *instance space*. To simplify the presentation we assume that X is the set of points in a multi-

dimensional unit square $[0, 1]^d$. We are given a very large set of instances D (a subset of X), which is the database of objects. (In machine learning D is often referred to as the training set.) With each instance in D we associate a class label. A typical statistical classification problem is to classify new instances in X that are not contained in D .

Decision trees have been established to be a very useful tool in classification and data mining, where they are used to summarize the database D and produce classification of new examples. Geometrically, a decision tree corresponds to a recursive partitioning of the instance space into mutually disjoint regions. Each region is represented by a leaf node in the tree, and associated with a particular value (C_i) giving the class label of all instances contained in the region. It is important to note that geometrically each leaf label corresponds to a hyperrectangle in X . For simplicity we refer to hyperrectangles as rectangles. We define a *monochromatic* rectangle to be a rectangle that contains only points labelled by the same class label.

3 Local Induction Algorithms and Memory-Based Reasoning

Local induction algorithms are based on a simple idea. Instead of building a complex statistical model that describes the entire space, we construct a simpler model that describes the space in a particular neighborhood. Local learning is a special case of memory-based reasoning (MBR). Applications of MBR include classification of news articles [MLW92], census data [CMSW92], software agents [MK93], computational biology [YL93, CS93], robotics [AMS95, Atk89, MAS95], computer vision [Ede94], and many other pattern recognition and machine learning applications. Recent work in statistics addresses the issue of adaptive neighborhood to a given query to improve K-nearest neighbour algorithms [HT94, Fri94]. See also the very relevant theoretical framework for local learning described in [BV92, VB93].

There are three key steps in local learning and MBR algorithms: 1) Given an instance, retrieve a set of instances in the training set that are relevant to the query; 2) Build a model (e.g, classifier or function approximator) using only retrieved points; 3) Use the model to process the query point (classify it or approximate a function value).

In interactive data mining applications the notion of local learning should be extended to provide both visual and statistical query dependent information. Thus, in a particular local neighborhood two features may be sufficient to produce a good classification, whereas the entire domain may need a very complex classifier. In many cases the user may care more about query specific accuracy than about an estimate of the accuracy of a global classifier (see below).

Choosing the appropriate local neighborhood around a query point is a difficult task, especially in high dimensional spaces. In this paper we report three new approaches that are primarily designed to complement standard decision tree algorithms in interactive data mining applications. These methods should be most useful when the database is large and when the user is interested in exploring only a small neighborhood around a query point.

3.1 Choosing a local neighborhood via nearest neighbors

The simplest approach to local learning with decision trees is simply extracting the k nearest neighbors and constructing a decision tree on these instances. If the neighborhood size is 1,

then the decision tree algorithm is equivalent to the 1-nearest neighbor (1-NN) algorithm. If the neighborhood size is N (the full size of the training set), then the algorithm is equivalent to conventional full induction. Nuances in the training set can greatly affect the histogram of accuracy over different values of k between 1 and N . In standard nearest neighbor algorithms the optimal value of K is determined by a specially reserved training set and cross validation techniques. However, it is clear that the best neighborhood size may vary from one query point to another. To make the algorithm more robust, we use a voting scheme as follows. For a given query point, a sequence of k trees is induced using the $1, 2, \dots, k$ nearest points. These trees then vote on the class of the query point. This is what we call the “local induction voting” (LIV) algorithm. In practice this seems to work well, as shown in the experiments below. The voting method can be implemented in many different ways; in these first experiments we use an ad-hoc method. We weigh the vote of each tree by the number of same-class examples in the leaf that is used to classify the example.

3.2 Choosing a local neighborhood via layers of composite rectangles

For the sake of gaining intuition for the algorithm described in this section, assume the target partitioning of the space is actually a decision tree T . Given any query point x it will be contained in some leaf node of T , which is a monochromatic rectangle R . This implies that every other point y in R forms a monochromatic rectangle with x and y at its corners. In other words, all the points in the database D that are contained in the rectangle defined by x and y have the same class label. Next, if we remove all points in R from the database D , we can now form monochromatic rectangles with points that are in regions adjacent to R that are important to the classification of x . We refer to all points in the database that form a monochromatic rectangle with a given query point x as **layer one**. These points are a superset of points in R . Once we remove all points in layer one we can refer to all remaining points that form monochromatic rectangles with a query point as layer two, and so forth. Our algorithm defines the adaptive neighborhood of a query point x as all points in layer one and layer two. The algorithm is given in Figure 2. A two-dimensional example that illustrates this idea geometrically is given in Figure 1.

The algorithm as described in Figure 2 has running time $O(N^2)$, where N is the size of the database. However, it turns out that we can utilize computational geometry techniques to reduce the running time to $O(N \log^{d-1} N)$ obtaining a practical improvement when the dimensionality is small. We provide a very rough sketch of the algorithm for two dimensions below. See [FKSW96] for a complete description. Given a point x we compute the set of monochromatic rectangles that are defined by x and another point in the database. Recall these are the points that we include in the local neighborhood of x . We first sort the points in into four quadrants with the query point x defining the origin. We compute the set of monochromatic rectangles separately in each quadrant. Note that the number of non-empty quadrants is bounded by N irrespective of dimensionality. We now determine for each point in the database if it dominates a point of a different class and therefore should be excluded from consideration. (A point (x_1, x_2) dominates another point (y_1, y_2) if $|x_i| > |y_i|$). We call this type of dominance *monochromatic dominance* that extends the standard notion of dominance ([PS85]). We randomly choose a dimension and project all points on this dimension. We then compute a median on the projected points; the median defines a separator (e.g, vertical line) We recursively solve monochromatic dominance separately for points larger than the median

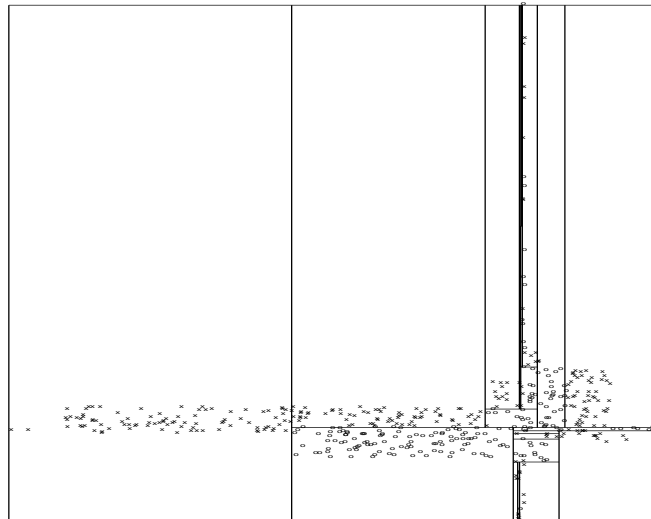


Figure 1: A 2D local decision tree with the local neighborhood defined by two layers of homogeneous rectangles. Only points in the layer one and layer two rectangles are shown, and the local tree is superimposed.

For each query point x

 For each point y_i in the database

 Check if x and y_i form a monochromatic rectangle

 If yes, include y_i in the first layer.

 Remove all points in the first layer.

 Repeat procedure and construct the second layer.

 The neighborhood of x is the set of points in both layers.

 Construct a decision tree on these points.

Figure 2: The multi-layer local decision tree algorithm

```

For a given query point
  The initial neighborhood is the query point itself.
  Find a local neighborhood using neighborhood_search().
  Induce a decision tree  $T$  upon the local neighborhood.
  Classify query point with  $T$ .

Neighborhood_search()
  Expand neighborhood in unblocked dimensions until an obstacle point is reached.
  For each dimensional branch caused by the obstacle point :
    Recursively call neighborhood_search().
  Return the highest scoring neighborhood from among the branches.

```

Figure 3: The adaptive boundary local induction algorithm

and smaller than the median. We then project all points on the line defined by the median. Finally, by climbing up the line we check for monochromatic dominance of points on the left of the median and the points to the right of the median. The algorithm is running in time $O(N \log N)$.

3.3 Choosing a local neighborhood via an adaptive boundary

Our third algorithm was inspired by a method of Hastie and Tibshirani [HT94], who defined a technique for creating an ellipsoidal neighborhood around a query. We have implemented an iterative search for a rectangular neighborhood around a query. Using the obvious greedy algorithm, one would start with a hyperrectangle around the query point and expand outwards until the enclosed region violated some constraint. A reasonable constraint to place on the neighborhood is that it must remain linearly separable; i.e., the examples contained in the rectangle can be classified using a single hyperplane. Once the expansion reaches an obstacle (i.e., a point whose inclusion will violate the constraint), it must limit growth in some dimension. When exploring all possibilities, an obstacle point will create a branch in a search tree. This algorithm is sketched in Figure 3. The size of such a search tree is exponential in the number of dimensions of the feature space, and therefore we experimented with several greedy approximation algorithms (see [FKSW96]).

4 Experiments

In this section we focus on the performance of the local induction voting algorithm used for several scientific domains: breast cancer diagnosis, star/galaxy classification, and identification of coding regions in DNA. We also performed experiments with the other algorithms using artificial datasets. The “dim” astronomy dataset contains 4192 examples in 14 dimensions with 2 classes (stars and galaxies) that occur with approximately equal frequency. The human DNA dataset contains approximately 40,000 examples in 6 dimensions with 2 classes (coding and noncoding) which occur with unequal frequency. Each example represents a piece of human DNA that is 162 bases long and comes either from an exon (a coding region, the part

Data set	num train	num test	prune (%)	full induction (% accuracy)	LIV (% accuracy)
HUMAN DNA	5024	39868	20	78.2	84.8
HUMAN DNA	300	39868	20	75.9	77.2
STAR/GALAXY	2500	1692	20	93.8	95.1
STAR/GALAXY	300	3892	20	89.7	92.1
BREAST CANCER	500	183	20	96.3	96.8

Table 1: Comparison of full induction to local induction voting.

of DNA that is used in genes to produce proteins) or from an intron (a noncoding region). Noncoding DNA is about six times more common than coding DNA in this data set (and is at least that common in the human genome), but for the sake of these experiments, two different training sets of size 300 and 5024 were constructed, each containing equal numbers of coding and noncoding regions. The results from the LIV algorithm on these datasets appear in Table 1. The accuracy reported for the DNA data set is the **unweighted** average of the two class accuracies; because of the much higher frequency of noncoding DNA, we did not want to swamp the estimate of accuracy on coding DNA in this average.

In these experiments, the decision tree induction algorithm is implemented using standard axis-parallel splits and information gain as the goodness criterion. This algorithm is thus very similar to Quinlan’s C4.5 [Qui93]. The decision trees are pruned using standard cost-complexity pruning [BFOS84] with a portion of the training set set aside as a pruning set. The purpose of these experiments is to determine whether local induction voting is an improvement over full induction.

The graphs presented in this section show overall accuracy as a function of the parameter K varies. K is the size in training points of the largest local window around the query point. At each point on the abscissa, all window sizes $1, \dots, K$ contribute their vote to the classification of the query point. In general, accuracy increases and plateaus as K is increased. Accuracy of the decision tree induced upon the full training set is shown as a horizontal rule across each graph. The accuracy of the local induction voting method typically surpasses the accuracy of full induction at a relatively small value of K .

For both of the databases, we experimented with two sizes of training set. The intent of using differently sized training sets was to determine whether local induction was better suited for sparse training sets or for larger training sets.

We used artificial data sets to perform detailed experiments with the multi-layer algorithm and the adaptive neighborhood algorithm. The datasets were generated by creating a random decision tree with approximately 200 nodes in two, four and eight dimensions. We randomly generated 400, 800, and 1600 instances. Each instance was classified using the “true” decision tree. We then used a standard decision tree algorithm, k-NN and the two new methods to classify the same data. Our multi-layer algorithm in two dimensions exhibited similar performance to the standard decision tree algorithm and outperformed k-NN. In four dimensions with 800 points, we obtained better results with the multi-layer algorithm. With 1600 points the results were similar for all methods. We found that the our adaptive neighborhood algorithm is computationally expensive and needs more tuning before it can be used effectively on large datasets.

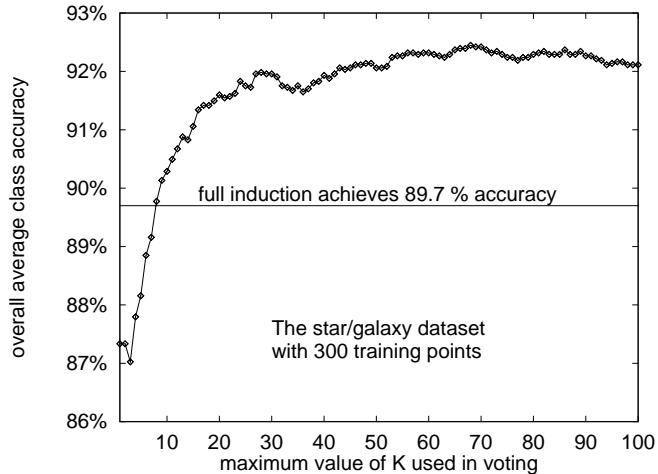
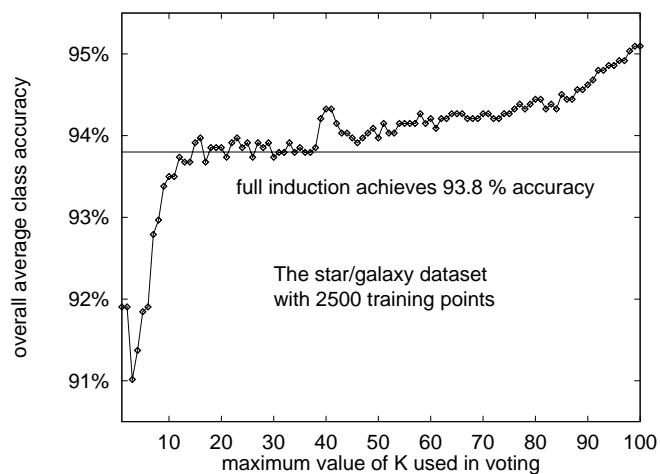
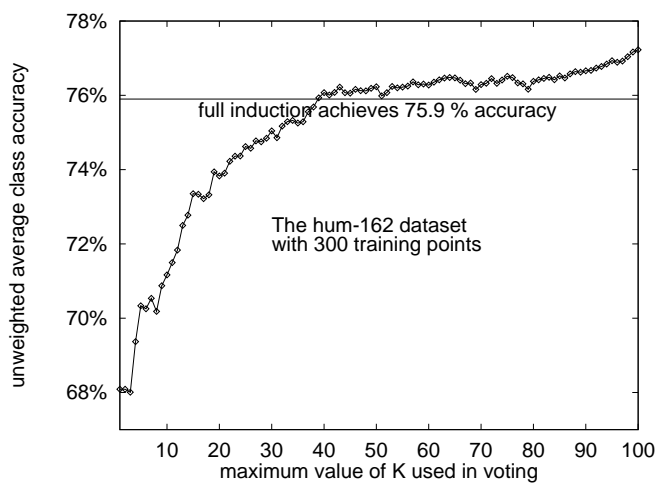
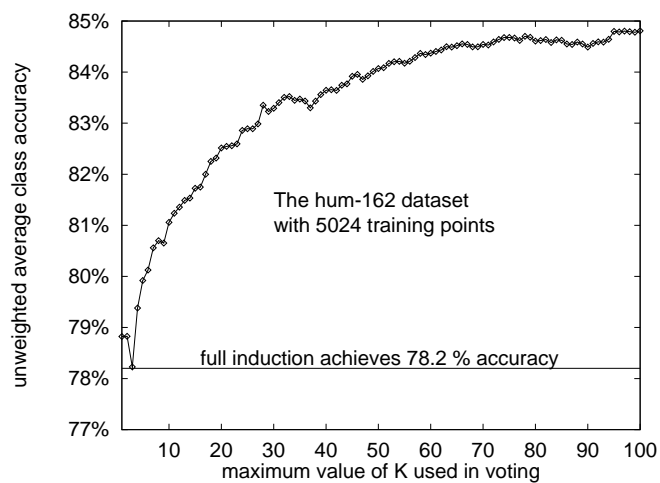


Figure 4: Accuracy of the local induction voting algorithm compared against full induction for the Hum-162 and star/galaxy datasets. The graphs on the left show the performance for large training sets, and on the right for small training sets.

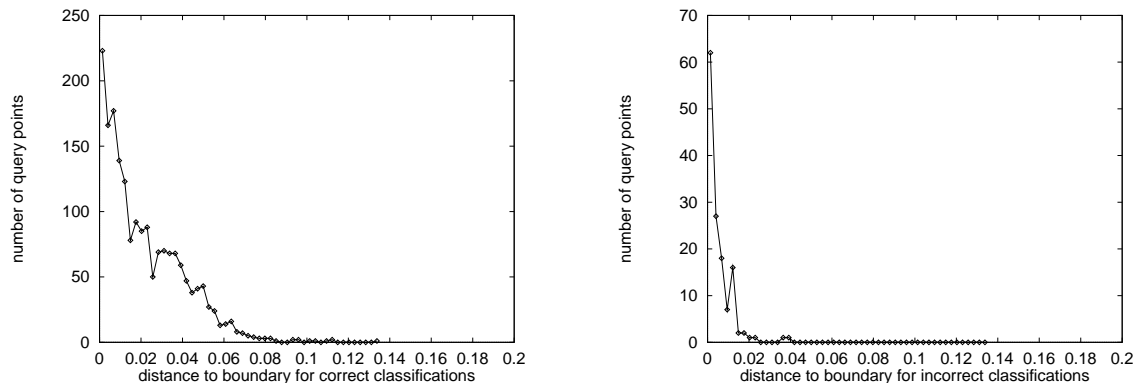


Figure 5: Frequency of distances of query points to the closest decision tree boundary for the star/galaxy dataset.

5 Local Error Analysis

Another useful data mining tool provided by decision trees (local or global) is error analysis. Figure 5 is a histogram of the numbers of query points close to the decision tree boundaries for the standard algorithm on the star/galaxy data set. For any query, we can compute its bin in the histogram and report our confidence in the prediction basen on the query’s distance to the boundary.

Note that the figure also reveals a difference in the distances of correct and incorrect points to a boundary: incorrectly classified points are likely to be much closer than average to a boundary.

6 Conclusions

In this paper we described three new algorithms for local induction decision trees and reported on some initial experimental data using these techniques. The first algorithm, local induction voting, is the simplest and so far the best in our experiments. The multi-layer composite neighborhood algorithm is more complex, especially in the efficient $O(N \log^{d-1} N)$ version. Its main design goal is to battle data fragmentation and to provide an intuitive feel for the distribution of points in the local region of the query point. For each query point we enumerate all of the monochromatic rectangles that include it. The distribution of these rectangles can be helpful for constructing new confidence measures to use with classifiers. The third method, which created axis-parallel adaptive neighborhoods, has so far shown inferior experimental performance (which is therefore not reported here) and needs further refinement. However, it has an interesting potential for applications where the concept boundaries are locally axis-parallel.

The main drawback of the algorithms reported here is the time to perform single classification. The running time of local induction voting per query is similar in practice to standard k-NN algorithms and is dominated by the size of the training set. However, in many situations (e.g., medical diagnosis), accuracy is of utmost importance, and it is worth the extra computation to identify the most accurate method.

The experiments we performed indicate that local induction voting does outperform full

induction in terms of accuracy. The similarity of the graphs of accuracy vs. K in experiments with very different databases indicates that local induction voting is likely to behave similarly for other databases. This makes local induction voting a potentially good candidate for improving classification tools in general.

References

- [AMS95] Christopher Atkeson, Andrew Moore, and Stefan Schaal. Locally weighted learning. submitted for publication, April 1995.
- [Atk89] C. Atkeson. Using local models to control movement. In *Neural Information Processing Systems Conf.*, 1989.
- [BFOS84] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth Internatl. Group, 1984.
- [BV92] L. Bottou and V. Vapnik. Local learning algorithms. *Neural Computation*, 4:888–900, 1992.
- [CMSW92] R. Creecy, B. Masand, S. Smith, and D. L. Waltz. Trading MIPS and memory for knowledge engineering. *Communications of the ACM*, 35(8):48–64, 1992.
- [CS93] S. Cost and S. Salzberg. A weighted nearest neighbor algorithm for learning with symbolic features. *Machine Learning*, 10(1):57–78, 1993.
- [Ede94] S. Edelman. Representation, similarity and the chorus of prototypes. *Minds and Machines*, 1994.
- [FKSW96] T. Fulton, S. Kasif, S. Salzberg, and D. Waltz. Local induction of decision trees: Towards interactive data mining. Technical report, Johns Hopkins University, Baltimore, MD 21218, 1996.
- [Fri94] J. H. Friedman. Flexible metric nearest neighbor classification. Technical report, Stanford University, Statistics Dept., November 1994.
- [HT94] T. Hastie and R. Tibshirani. Discriminant adaptive nearest neighbor classification. Technical report, Stanford University, Statistics Dept., December 1994.
- [MAS95] A. Moore, C. Atkeson, and S. Schaal. Memory-based learning for control. *Artificial Intelligence Review (to appear)*, 1995.
- [MK93] P. Maes and R. Kozierok. Learning interface agents. In *Proc. of the Eleventh National Conf. on Artificial Intelligence*, pages 459–465, Washington, D.C., 1993. MIT Press.
- [MKS94] Sreerama K. Murthy, Simon Kasif, and Steven Salzberg. A system for induction of oblique decision trees. *Journal of Artificial Intelligence Research*, 2:1–33, August 1994.
- [MLW92] B. Masand, G. Linoff, and D. L. Waltz. Classifying news stories using memory based reasoning. In *Proceedings of the SIGIR*, pages 59–65, 1992.
- [PS85] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, New York, NY, 1985.
- [Qui93] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, San Mateo, CA, 1993.
- [VB93] V. Vapnik and L. Bottou. Local learning algorithms for pattern recognition and dependency estimation. *Neural Computation*, 5:893–909, 1993.
- [YL93] T.-M. Yi and E. Lander. Protein secondary structure prediction using nearest-neighbor methods. *Journal of Molecular Biology*, 232:1117–1129, 1993.