

Frankenstein classifiers: Some experiments

Hendrik Blockeel and Jan Struyf

Katholieke Universiteit Leuven, Department of Computer Science
Celestijnenlaan 200A, B-3001 Leuven, Belgium
{hendrik.blockeel,jan.struyf}@cs.kuleuven.ac.be

Abstract

We present some empirical results on the use of two methods for integrating different classifiers into a hybrid classifier that should perform better than each of its constituent classifiers. The main point of these methods is that instead of combining full classifiers, they combine pieces of them. One of the methods is based on ROC analysis; the other is based on augmenting the data with features derived from earlier learned classifiers. Experimental results are presented that suggest that these approaches can yield improvements over combining full classifiers.

1 Introduction

In some two decades of existence, the field of machine learning has given rise to many different algorithms for learning. These algorithms differ, among other things, with respect to their inductive bias: certain implicit assumptions that they make about the learning task, and which need to be fulfilled in order to guarantee good performance of the algorithm (Mitchell, 1996). In other words, an algorithm will perform well if its inductive bias matches the characteristics of the problem at hand. Unfortunately, choosing the appropriate learner for a given problem is made difficult in practice because of two reasons: the inductive bias of learning algorithms is not always well understood, and the relevant characteristics of the problem may not be known either. Hence, it is not surprising that at some point interest has risen into how one can run several algorithms on the same problem and combine their results, so that the final result is at least as good (and possibly better) than the best of the separate results.

One popular approach towards combining the strengths of different learning algorithms is to use each of these algorithms separately to build

a classifier, then compose these classifiers into a single hybrid one. Such a hybrid classifier could, e.g., consider the predictions of the separate classifiers as votes and use the majority vote as the final prediction; or it could try to identify areas where one classifier is more trustworthy than another one, and base its prediction entirely on that classifier, etc. A variety of more sophisticated schemes exist (see, e.g., (Bauer and Kohavi, 1999)).

We can distinguish two ways in which a hybrid classifier might improve upon its constituent classifiers. One is that the hybrid classifier is simply more accurate in its predictions than any of its constituents, in all cases. A second, weaker form of improvement follows from the fact that the quality of a classifier depends on some characteristics of the environment in which it will be used (e.g., the class distribution among the new instances that are to be classified). The hybrid classifier may then be equally good as its constituents in their own areas of expertise, but by being able to always choose the best one achieves better performance on average, over different deployment environments. Voting schemes may yield improvements of the first kind, whereas techniques such as ROC analysis (Provost and Fawcett, 1998) can yield improvements of the second kind.

In the above we have been talking about combining classifiers. A less popular but possibly interesting approach might be to take *pieces* out of those classifiers and construct a better classifier from those. If classical methods for composing classifiers are compared to having a jury of experts and a judge who makes the final decision based on the expert's opinions, this approach would boil down to taking the brain of the cleverest expert, the eyes of the one who

sees best, etc. and stitch everything together¹ into what we could call Frankenstein’s classifier.

Agreed, this move from the principles of democracy to those of gothic horror may not be pretty; but then again, quoting Victor Frankenstein : “*with how many things are we upon the brink of becoming acquainted, if cowardice (...) did not restrain our inquiries.*” (Shelley, 1818) Taking full responsibility for the outcome, we have therefore investigated a number of ways in which Frankenstein classifiers could be built. By no means is it our intention to be exhaustive; rather, we propose a number of general ideas and report on our practical experiences with some specific instantiations of these ideas. Our results do suggest that further investigations in this direction may be worthwhile.

The rest of this paper is structured as follows. In Section 2 we briefly introduce the Sisyphus problem, on which our experiments were performed. Section 3 briefly reviews the idea behind ROC curves. In Sections 4 and 5 we present two recipes for Frankenstein classifiers: the first one makes use of ROC curves, the second one uses the preprocessing approach. In both cases the ideas are presented, as well as illustrated and evaluated on the Sisyphus dataset. Section 6 summarizes our experimental results, Section 7 discusses some related work and in Section 8 we conclude.

2 The Sisyphus problem

During the first year of the European SolEuNet project (<http://www.soleunet.com>), which among other things investigates methods for collaborative data mining, several data mining groups worked on the so-called Sisyphus problem, provided by the insurance company SwissLife. In this problem, the data are stored in a relational database (10 tables, number of tuples in each table ranging from 184 to 111077). Two classification tasks (A and B) and one clustering task were defined. We here focus on the classification tasks. Task A concerns the classification of partners in a household, Task B concerns classification of households; more precise details on the meaning of the classes were not made available to the data mining groups.

The multi-relational format of the dataset

¹Lightning is not necessary, but can be added for effect.

makes this a relatively difficult problem: most data mining software handles data in a single table only, so that one either needs to preprocess the information in the relational database into a single table, or use a data mining system that can handle the multi-relational format by itself. The two approaches are in some sense complementary, as data preprocessing is often done by humans (using their intuition about which information is important and which is not) whereas, e.g., ILP approaches (Muggleton and De Raedt, 1994) rather use brute force to find interesting features. Both approaches have been reported to yield results the other missed.

(Gärtner, 2001) provides an overview of the approaches that were investigated by several (not all) data mining groups involved in the SolEuNet project. The main tools used were Weka (Witten and Frank, 1999), Kepler (Wrobel et al., 1996) and ACE (Blockeel et al., 2000). All these tools allow the user to run a number of data mining algorithms on a given data set. The data mining approaches that were considered were

- decision trees : J48, the Weka implementation of C4.5 (Quinlan, 1993); and Tilde (Blockeel and De Raedt, 1998), an ILP system that is a first order upgrade of C4.5
- 1R, a simple algorithm that induces decision trees of a single node only (Holte, 1993)
- Naive Bayes, in its Weka implementation (Witten and Frank, 1999)
- Linear support vector machines (Burges, 1998)
- deviating subgroup discovery (Midos (Wrobel, 1997)).

This is a relatively broad range of algorithms, i.e., the algorithms differ quite strongly with respect to each other. Obviously, it makes more sense to build composite classifiers from a selection of widely differing algorithms than from a set of variants of the same algorithm, as in the latter case there will usually be more overlap in the strong and weak points.

3 ROC diagrams

Classifiers are often evaluated on the basis of their predictive accuracy, i.e., the probability

that they will classify an unseen object correctly. Two problems with this kind of evaluation are that (a) accuracy is unstable with respect to changing class distributions, and (b) it assumes a symmetric misclassification cost (i.e., misclassifying an object of class A as B is an equally bad mistake as misclassifying an object of class B as A).

ROC analysis (Provost and Fawcett, 1998) has been proposed as an alternative evaluation criterion that does not suffer from these problems. A ROC diagram shows how a classifier can be expected to perform in an environment with a given class distribution (not necessarily the one of the training set) and given misclassification costs. On a ROC diagram one can easily see under which circumstances one classifier is better than another (or possibly, whether a classifier is better than another classifier in all possible environments). Given that different classifiers may perform better in different environments, it is natural to use ROC analysis to build a hybrid classifier that is optimal in the sense that in each environment it will employ the best classifier.

First we introduce some notation and terminology. We assume a binary classification problem: objects are to be classified as positive or negative. We represent the true classes as $+$ and $-$ and the predictions as pos and neg . $C_{pos|-}$ represents the cost of misclassifying a negative object as positive; $C_{neg|+}$ the cost of misclassifying a positive object as negative. We assume $C_{pos|+} = C_{neg|-} = 0$. $P(+)$ is the probability that an unseen instance is positive; $P(-)$ is the probability that it is negative. $P(pos|+)$ is the probability that an instance is classified as positive given that it actually is positive; we similarly use $P(neg|+)$, $P(pos|-)$, $P(neg|-)$.

The expected misclassification cost for a single object is then

$$EC = C_{pos|-}P(pos|-)P(-) + C_{neg|+}P(neg|+)P(+)$$

The *true positive rate* TP estimates $P(pos|+)$ and is computed from a test set as $n_{pos,+}/n_+$, i.e., the number of actual positives predicted positive over the number of actual positives. The *false positive rate* FP estimates $P(pos|-)$ and is computed as $n_{pos,-}/n_-$. Using these estimates, the formula for expected misclassification

costs becomes

$$EC = C_{pos|-}P(-) \cdot FP + C_{neg|+}P(+)(1 - TP)$$

In a ROC diagram the horizontal axis represents FP , the vertical axis TP . Points with the same expected classification cost (iso-cost lines) are on a straight line with slope $C_{pos|-}P(-)/C_{neg|+}P(+)$. Points with the same expected accuracy are on a straight line with slope $P(-)/P(+)$. Note that given $P(+)$ and $P(-)$, expected accuracy can be computed from TP and FP , but not the other way around. Also note that these slopes cannot be negative, assuming positive costs.

The upper left corner of a ROC diagram is the ideal situation ($FP = 0$, $TP = 1$; i.e., all positives and no negatives found). A classifier A is better than B in a certain situation if there exists an iso-cost line for that situation such that A is above the line and B is below it. A dominates B if no situations exist where B is better than A; on the diagram A is then to the left and above B. A set of classifiers S dominates a classifier B if in any situation there exists a classifier in S that is better than B; on the diagram B is below the convex hull of S.

For each set of classifiers S, a minimal subset S' can be defined such that no elements e of S' are dominated by $S' - \{e\}$; this is the set of all the classifiers that lie on the convex hull of S. S' can be seen as a hybrid classifier of minimal complexity that still performs at least as good as any of its component classifiers.

4 Dr. Frankenstein's ROC Recipe

4.1 Method

A classifier A can sometimes be decomposed in such a way that from the pieces new classifiers can be obtained. A trivial example of this is a rule set $S = \{R_1, \dots, R_n\}$ with rules R_i ; any subset of S is a new classifier. As the body of each rule is a set of conditions, the same principle could be used on those. Note that, if a rule for instance predicts pos , then dropping the rule decreases TP and FP , whereas dropping a condition in the body of the rule increases these parameters. Similarly, a decision tree gives rise to subtrees that are valid classifiers by themselves, but have different TP and FP values. This will be discussed further on.

This derivation of new classifiers from the original one bears some resemblance to pruning decision trees or rules sets. Note however that the aim is quite different: pruning a classifier aims at finding a new classifier that is simpler but has comparable or better predictive accuracy, and in this sense improves upon the original one. In our context we are looking for classifiers with quite different properties than the original one.

When a classifier A gives rise to a set of derived classifiers A_1, \dots, A_n , it makes sense to include these classifiers as well on the ROC diagram. In the worst case A dominates all the A_i , but it is probable that some of the A_i are not dominated by A ; in fact, the method for deriving classifiers from A could be constructed in such a way that it is unlikely that all A_i are dominated by A .

For decision trees we propose the following method. Given a tree with positive and negative predictions, order the leaves according to some estimated probability of an unseen example sorted into that leaf being positive. This probability estimate could e.g. be the m -estimate (Cestnik, 1990), a linear interpolation between the ratio of positives in the leaf and some a priori probability p , which is given a weight m : $e = \frac{n_+ + pm}{n_+ + n_- + m}$. As a value for p it makes sense to use the proportion of positives in the training set. We further choose $m = 1$, which means the estimate is mainly influenced by the proportion of positives in the leaf and only slightly by the size of the leaf.

Now from the decision tree A new decision trees can be constructed as follows: A_i is A with for the first i leaves according to the ordering just described, *pos* substituted for the leaf's prediction, and for all other leaves *neg* substituted for their prediction. A_0 always predicts *neg*, A_l (with l the number of leaves in the tree) always predicts *pos*. The A_i can be pruned separately (for instance, A_1 is essentially just one rule).

This idea is not really new; it was used, e.g., in (Blokkeel et al., 1999) (although not with an m -estimate), from the point of view that decision trees can be considered rank classifiers (i.e. classifiers that give a classification together with some degree of confidence; such classifiers give rise to a curve in the ROC diagram). A similar technique has been used by Gärtner (Gärtner,

2001) in his experiments with the Naive Bayes classifier; different versions of that classifier are constructed, with varying a priori probabilities of having a positive. Since the probability estimates used by Naive Bayes need not be recomputed, we can consider this another example of deriving a set of classifiers from a single classifier without further access to the data.

Thus, there have been approaches similar to the technique we propose here; but to our knowledge this technique has not been described as a methodology as such. Our main point here is that the approach seems quite generally applicable, and should probably be applied in a systematic way. It seems worth investigating, not only for decision trees but also other representations, what is the best way of deriving a set of classifiers from a single one such that the set will exhibit maximal performance on a ROC diagram. (Note that even for decision trees, what we describe is a rather simple approach and better ones might be devised.)

4.2 Experimental results

Figure 1 shows the results obtained on Task A when performing ROC analysis according to the above method (the FP and TP are here measured on an independent test set, in the same way as in (Gärtner, 2001)). It is clear that the convex hull of the derived classifiers is much better than the one of the classifier itself; it actually dominates the convex hull reported in (Gärtner, 2001), where a variety of methods was employed but without decomposing any of the classifiers into derived classifiers (except for Naive Bayes, as discussed above).

We did the same exercise on Task B, where the decision tree induced by Tilde was not on the ROC convex hull. Figure 2 shows that some of the derived classifiers do lie on the convex hull. It is likely, however, that if the same decomposition was done on the J48 tree (which we could not do because we do not have the tree), the convex hull would mainly consist of J48 derived classifiers.

5 Dr. Frankenstein's Bootstrap Recipe

5.1 Method

The second method for integrating classifiers is in a sense a bootstrapping method: classifiers

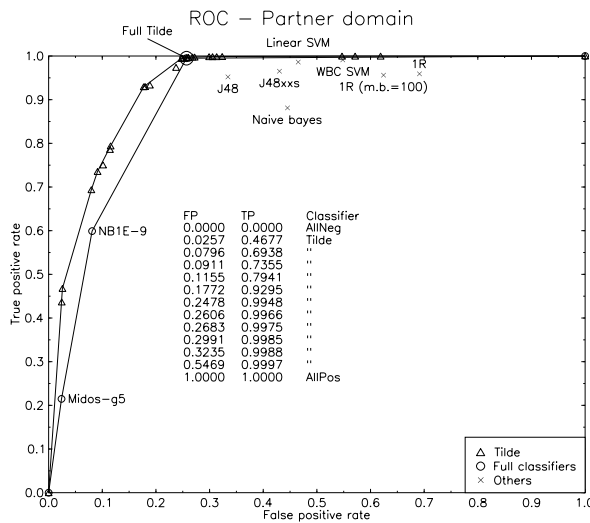


Figure 1: Task A: ROC convex hull formed by classifiers derived from a single decision tree, compared with the convex hull formed by another set of classifiers as reported in (Gärtner, 2001).

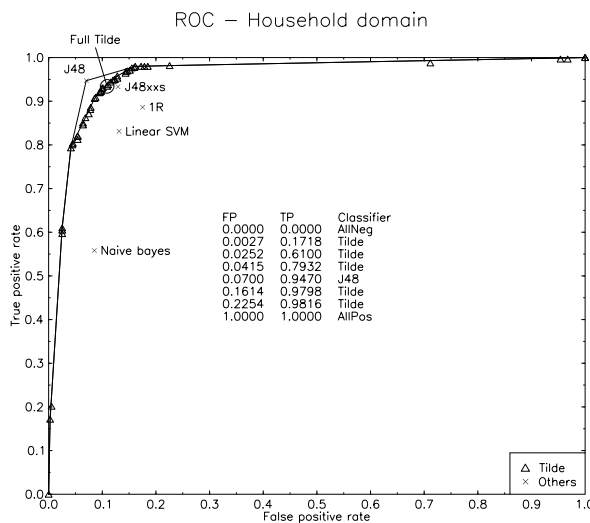


Figure 2: Task B: ROC convex hull, formed by both original and derived classifiers.

are induced one by one in a certain order; each result serves as a bootstrap for the next run. In general such a bootstrap method to learning Frankenstein classifiers could go as follows:

```

while stopping criterion not reached
  choose some learning algorithm
  induce from the current data a classifier C

```

augment the data with attributes derived from C

This generic description leaves a number of parameters to be instantiated, such as:

- In what order are learning algorithms tried? Can a learning algorithm be tried more than once?
- when do we stop: when the list of learning algorithms (or variants thereof) to try is exhausted; when no improvement was achieved with the previous algorithm; ...
- exactly how do we define the new attributes that reflect the results of the induction process?

It is clear that there are many possibilities, and an in-depth study of a large set of them, while interesting, is outside the scope of this paper. Instead, we focus on one approach that was explored in the Sisyphus context.

5.2 Experimental results

The method followed in this experiment is a very simple version of the approach described above:

- The learners are ACE's Tilde and Weka's J48. Both are run once, in that order (i.e., J48 gets to use the Tilde results).
- The tree induced by Tilde is represented as one single attribute that indicates the leaf an example is sorted into by the tree.

Note that this approach differs from a number of classical approaches for combining classifiers, in that not the prediction of Tilde is given, but instead the actual leaf that made the prediction. This enables the second classifier to distinguish high quality predictions (e.g., from a large and pure leaf) from low quality predictions. From the "Frankenstein" point of view, the second classifier can decide to trust certain parts of the first one and disregard others, and just picks out the pieces it wants.

It may seem strange to twice use a decision tree learner. Note, however, that Tilde is a first-order system and J48 is not; because of this the two approaches are still sufficiently different to possibly yield improved results when combined. Also note that the capability to combine different types of classifiers makes this approach

essentially different from approaches such as boosting (Freund and Schapire, 1996). The question springs to mind how these approaches compare, but such a comparison is outside the scope of this paper however (note that similar comparisons could be done with bagging, voting, stacking, etc.)

The proposed approach might seem prone to overfitting, given that the resulting theory can become relatively complex, and following the minimal description length principle that states that simpler theories are less prone to overfitting. In this respect we wish to mention Domingos' (Domingos, 1998) excellent discussion on model selection, where it is argued that not the size of a theory, but the number of hypotheses that are considered during the search (i.e., the size of the actual search space) determines how likely overfitting is. The combination of two models in the way proposed here just doubles the number of hypotheses looked at, which has only a small influence on the possibility of overfitting.

In a previous report (Struyf and Blockeel, 2001), this approach was evaluated by looking at its predictive accuracy; but obviously, since J48 produces a tree, we can evaluate it by performing a ROC analysis in just the same way as described in the previous section, i.e., by constructing the convex hull of the classifiers derived from the J48 tree using the m -estimate. On Figure 3 we compare the convex hull of the J48+Tilde derived classifiers with the convex hull reported by Gärtner (Gärtner, 2001) and with the hulls generated by the classifiers derived from Tilde alone, respectively from J48 alone.

The optimal classifier is formed by taking the convex hull around these convex hulls. In this case it turns out it coincides almost entirely with the J48+Tilde convex hull; it contains one point generated by Tilde but the improvement to the convex hull caused by this one point is insignificant.

6 Discussion

The results we have obtained on the Sisyphus data set can be summarized as follows. We denote the convex hull generated by a set of classifiers S with $CH(S)$ and the set of classifiers derived from a single classifier C with $S(C)$.

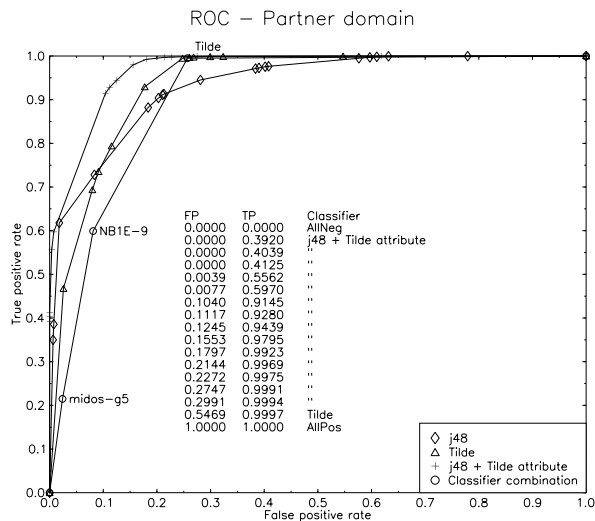


Figure 3: Convex hulls generated by Tilde, J48, J48+Tilde and the original convex hull as reported in (Gärtner, 2001). The global convex hull almost entirely coincides with J48+Tilde's.

The general idea behind combining classifiers C_i with ROC analysis is that the hybrid classifier defined by $CH(\{C_1, C_2, \dots, C_n\})$ may be better than any individual classifier, if its performance is averaged over a range of misclassification costs or class distributions.

Our initial idea was that for a given C_i , $CH(S(C_i))$ should similarly improve over $CH(\{C_i\})$. This was confirmed by our experiments; it was even the case that a C_i could be found for which $CH(S(C_i))$ dominated $CH(\{C_1, C_2, \dots, C_n\})$.

Our second experiment, using one classifier as attribute to learn another classifier, yielded a clear improvement of the ROC convex hull over both original classifiers. When deriving classifiers from this classifier, the resulting classifiers were all on the convex hull.

These results may seem to diminish the importance of ROC analysis as a means for constructing hybrid classifiers, since most of the convex hulls we found were generated by a single classifier. I.e., it could be hypothesized that $Acc(C_1) > Acc(C_2)$ (with Acc predictive accuracy) usually implies $CH(S(C_1))$ dominates $CH(S(C_2))$. It should be kept in mind, however, that our results are probably not very representative in this respect. More specifically, it was already apparent from earlier experiments

(Struyf and Blockeel, 2001) that one classifier performed clearly better than the others on Task A (and only there), probably because of its relational nature. It is not so surprising that if C_i has clearly better performance than C_j , also $S(C_i)$ turns out to be overall better than $S(C_j)$. It would be more interesting to perform similar experiments in a more balanced context, where different classifiers each have their own domain of expertise.

In any case, when there are cheap methods for computing $S(C)$ from some C , it is clear that this opportunity for improving the convex hull should not be ignored.

7 Related work

As mentioned, the technique of deriving a set of classifiers from a single one has also been used in (Blockeel et al., 1999) and (Gärtner, 2001), although it was not explicitly advocated as a methodology. The latter is much more the case in Srinivasan’s work on extraction of multiple models in ILP (Srinivasan, 2001). Srinivasan discusses how an ROC convex hull can be optimized by running an ILP system repeatedly on different sets of background knowledge, and proposes an algorithm to actively search for a subset of background knowledge that will yield a theory on the convex hull. The idea is similar to ours; Srinivasan searches for suitable subsets of background knowledge to learn from, we search for suitable subtheories of a single learned theory. Besides this difference, Srinivasan’s approach is computationally more expensive because several full theories are integrated, each of which is relatively expensive to induce, instead of pieces of theories which are cheap to obtain.

The idea of using more information from a classifier than just its prediction was also promoted by (Ting and Witten, 1999); they propose to employ certainty measures for predictions, instead of just predictions, when stacking multiple classifiers. Our bootstrap approach can be seen as a generalisation of this.

8 Conclusions

We have discussed two ways in which better classifiers can be built from *pieces* of existing classifiers, instead of from full classifiers. In both cases it was shown that significant im-

provements to the performance of classifiers can be achieved in this way. The exact extent to which this can be exploited remains to be investigated.

Acknowledgements

Hendrik Blockeel and Jan Struyf are respectively post-doctoral fellow and research assistant of the Fund for Scientific Research of Flanders. The Sisyphus dataset was made available by SwissLife, in the context of the EU’s Fifth Framework Project IST-1999-11495 (SolEuNet). Mary Wollstonecraft Shelley provided inspiration for the title. The authors also thank Steve Moyle, Peter Flach and Thomas Gärtner for stimulating discussions.

References

- E. Bauer and R. Kohavi. 1999. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 36:105.
- H. Blockeel and L. De Raedt. 1998. Top-down induction of first order logical decision trees. *Artificial Intelligence*, 101(1-2):285–297, June.
- H. Blockeel, L. Dehaspe, K. Driessens, N. Jacobs, R. Kosala, J. Ramon, and W. Van Laer. 1999. The Leuven submission to the Benelearn-99 competition. In P. van der Putten and M. van Someren, editors, *The Benelearn 1999 Competition*, pages 1–8. Sociaal-wetenschappelijke Informatica, Universiteit van Amsterdam.
- H. Blockeel, B. Demoen, L. Dehaspe, G. Janssens, J. Ramon, and H. Vandecasteele. 2000. Executing query packs in ILP. In J. Cussens and A. Frisch, editors, *Proceedings of the 10th International Conference in Inductive Logic Programming*, volume 1866 of *Lecture Notes in Artificial Intelligence*, pages 60–77, London, UK, July. Springer.
- C.J.C. Burges. 1998. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167.
- B. Cestnik. 1990. Estimating probabilities: A crucial task in machine learning. In *Proceedings of the 9th European Conference on Artificial Intelligence*, pages 147–149, London. Pitman.
- P. Domingos. 1998. A process-oriented heuristic for model selection. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 127–135. Morgan Kaufmann, San Francisco, CA.
- Y. Freund and R. E. Schapire. 1996. Experiments with a new boosting algorithm. In L. Saitta, editor, *Proceedings of the Thirteenth International*

- Conference on Machine Learning*, pages 148–156. Morgan Kaufmann.
- T. Gärtner. 2001. Roc analysis on sisyphus data. Unpublished.
- R. Holte. 1993. Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, 11:63–91.
- T. Mitchell. 1996. *Machine Learning*. McGraw-Hill.
- S. Muggleton and L. De Raedt. 1994. Inductive logic programming : Theory and methods. *Journal of Logic Programming*, 19,20:629–679.
- F. Provost and T. Fawcett. 1998. Analysis and visualization of classifier performance: comparison under imprecise class and cost distributions. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*, pages 43–48. AAAI Press.
- J. R. Quinlan. 1993. *C4.5: Programs for Machine Learning*. Morgan Kaufmann series in machine learning. Morgan Kaufmann.
- M. Shelley. 1818. *Frankenstein: or, the Modern Prometheus*. Lackington, Hughes, Harding, Mavor, and Jones, London. A.o. at <http://etext.lib.virginia.edu/>.
- A. Srinivasan. 2001. Extracting context-sensitive models in inductive logic programming. *Machine Learning*. To appear.
- J. Struyf and H. Blockeel. 2001. KDD Sisyphus I, Task A: Attribute generated with ACE. Unpublished.
- K.M. Ting and I.H. Witten. 1999. Issues in stacked generalization. *Journal of Artificial Intelligence Research*, 10:271–289.
- I. Witten and E. Frank. 1999. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann.
- S. Wrobel, D. Wettschereck, E. Sommer, and W. Emde. 1996. Extensibility in data mining systems. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*. AAAI Press.
- S. Wrobel. 1997. An algorithm for multi-relational discovery of subgroups. In J. Komorowski and J. Zytchow, editors, *Proceedings of the First European Symposium on Principles of Data Mining and Knowledge Discovery (PKDD '97)*, pages 78 – 87. Springer-Verlag.