Using Weighted Nearest Neighbor to Benefit from Unlabeled Data

Kurt Driessens[1], Peter Reutemann[2], Bernhard Pfahringer[2], and Claire Leschi[3]

[1] Department of Computer Science, K.U.Leuven, Belgium
[2] Department of Computer Science, University of Waikato, Hamilton, New Zealand
[3] Institut National des Sciences Appliquees, Lyon, France

Abstract. The development of data-mining applications such as text-classification and molecular profiling has shown the need for machine learning algorithms that can benefit from both labeled and unlabeled data, where often the unlabeled examples greatly outnumber the labeled examples. In this paper we present a two-stage classifier that improves its predictive accuracy by making use of the available unlabeled data. It uses a weighted nearest neighbor classification algorithm using the combined example-sets as a knowledge base. The examples from the unlabeled set are "pre-labeled" by an initial classifier that is build using the limited available training data. By choosing appropriate weights for this pre-labeled data, the nearest neighbor classifier consistently improves on the original classifier.

1 Introduction

The combination of supervised and unsupervised learning [1] is a growing subfield of Machine Learning. Applications such as text- or image-mining and molecular profiling have revealed application areas that yield very little (and often expensive) labeled data but often plenty of unlabeled data. As traditional machine learning algorithms are not able to use and benefit from the information available in the unlabeled data, custom built algorithms should be able outperform them. Current research in semi-supervised learning using algorithms such as Co-Training [2] or more recent approaches based on graph representations [3] confirms that this is indeed possible.

Most of the semi-supervised learning approaches use the labeled and unlabeled data simultaneously or at least in close collaboration. Roughly speaking, the unlabeled data provides information about the structure of the domain, i.e. it helps to capture the underlying distribution of the data. The challenge for the algorithms can be viewed as realizing a kind of trade-off between robustness and information gain [1]. To make use of unlabeled data, one must make assumptions, either implicitly or explicitly. As reported in [3], the key to semi-supervised learning is the prior assumption of consistency, that allows for exploiting the geometric structure of the data distribution. Close data points should belong to the same class and decision boundaries should lie in regions of low data density; this is also called the "cluster assumption".

In this paper, we introduce a very simple two-stage approach that uses the available unlabeled data to improve on the predictions made when learning only from the labeled examples. In a first stage, it uses an of-the-shelf classifier to build a model based on the small amount of available training data, and in the second stage it uses that model to transform the available unlabeled data into a weighted "pre-labeled" data-set that together with the original data is used in a nearest neighbor classifier. We will show that the proposed algorithm improves on the classifier built in stage 1, especially in cases where much more unlabeled data is available compared to the labeled data.

The rest of the paper is structured as follows: in section 2 we describe a few related semi-supervised learning techniques. Section 3 introduces the proposed algorithm in detail. In section 4 we show experimental results using an array of different classifiers used in the first stage. Section 5 concludes and presents some directions for future work.

2 Learning from Labeled and Unlabeled Data

Early methods in semi-supervised learning were using mixture models (in which each mixture component represents exactly one class) and extensions of the EM algorithm [4]. More recent approaches belong to one of the following categories: self-training, co-training, transductive SVMs, split learning, and graph-based methods. In the self-training approach, a classifier is trained on the labeled data and then used to classify the unlabeled ones. The most confident (now labeled) unlabeled points are added to the training set, together with their predictive labels, and the process is repeated until convergence [5]. Approaches based on co-training [2] assume that the features describing the objects can be divided in two subsets such that each of them is sufficient to train a good classifier, and that the two sets are conditionally independent given the class attribute. Two classifiers are iteratively trained, each on one set, and they teach each other with a respective subset of unlabeled data and their highest confidence predictions. The transductive SVMs [6] are a "natural" extension of SVMs to the semi-supervised learning scheme. They aim at finding a labeling of the unlabeled data so that the decision boundary has a maximum margin on the original labeled data and on the (newly labeled) unlabeled data.

Graph-based methods attempt to capture the underlying structure of the data with a graph whose vertices are the available data (both labeled and unlabeled) and whose (possibly weighted) edges encode the pairwise relationships among this data. Examples of recent work in that direction include Markov random walks [7], cluster kernels [8], and regularization on graphs [3]. The learning problem on graphs can generally be viewed as estimation problem of a classifying function f which should be close to a given function y on the labeled data and smooth on the whole graph. Different graph-based methods mainly vary by their choice of the loss function and the regularizer [9]. For example, the work on graph cuts [10] minimizes the cost of a cut in the graph for a two-class problem, while [11] minimizes the normalized cut cost and [12, 3] minimize a quadratic

cost. As noticed in [9], these differences are not actually crucial. What is far more important is the construction and the quality of the graph, which should reflect domain knowledge through the similarity function which is used to assign edges and their weights.

Collective classification [13] is an ILP approach that uses the relational structure of the combined labeled and unlabeled data-set to enhance classification accuracy. With relational approaches, the predicted label of an example will often be influenced by the labels of related examples. The idea behind collective classification is that the predicted labels of a test-example should also be influenced by the predictions made for related test-examples. The algorithm presented in this paper is closely related to this, but works on non-relational data by using a distance and the nearest neighbor relation that results from it.

3 YATSI

The YATSI algorithm[4] that we present in this paper will incorporate ideas from different algorithms that were discussed in the previous section. Since we really like the idea of giving the user the option to chose from a number of machine learning algorithm (like it is possible in co-training), we will develop a technique that builds on top of any standard machine learning algorithm. To incorporate the general idea behind collective classification, we use a nearest neighbor approach and the distance between as a way of relating them to each other.

Algorithm 1 High level pseudo code for the two-stage YATSI algorithm.

Input: a set of labeled data D_l and a set of unlabeled data D_u, an of-the-shelf classifier C and a nearest neighbor number K; let $N = |D_l|$ and $M = |D_u|$
Step 1:
  Train the classifier C using D_l to produce the model M_l
  Use the model M_l to "pre-label" all the examples from D_u
  Assign weights of 1.0 to every example in D_l
    and of $F \times \frac{N}{M}$ to all the examples in D_u
  Merge the two sets D_l and D_u into D
Step 2:
  For every example that needs a prediction:
    Find the K-nearest neighbors to the example from D to produce set NN
    For each class:
      Sum the weights of the examples from NN that belong to that class
    Predict the class with the largest sum of weights.

[4] YATSI was developed during a time when we were experimenting with a number of multi-stage classifiers that dealt with labeled and unlabeled data. At the time, we referred to the presented algorithm as: "Yet Another Two-Stage Idea", hence the name YATSI

The YATSI classifier (See Algorithm 1 for high-level pseudo-code) uses both labeled and unlabeled data in a two-stage set-up. In the first stage a standard, of-the-self, classifier (or regression-algorithm) is trained on the available training data. Since this kind of data is limited in the specific application areas we are looking at, it is best to choose an algorithm that can learn a model well using only a small amount of learning data.

In the second stage, the model generated from the learning data is used to "pre-label" all the examples in the test set. These pre-labeled examples are then used together with the original training data in a weighted nearest neighbor algorithm. The weights used by the nearest neighbor classifier are meant to limit the amount of trust the algorithm puts in the labels generated by the model from the first step. As a default value, we set the weights of the training data to 1.0 and the weights of the pre-labeled test-data to N/M with N the number of training examples and M the number of test-examples. Conceptually, this gives equal weights to the whole train- and the whole test-set. By adding a parameter F to the algorithm that will cause the weight of the test-examples to be set to $F * (N/M)$, it becomes possible to vary the influence one wants to give to the unlabeled data and the classifier built in step 1. Values of F between 0.0 and 1.0 will lower the influence on the test-data and the learned model from the first step, values larger than 1.0 will increase their influence. In the experiments, we will test values ranging from 0.01 to 10. An F-value of 10.0 will adjust the weights of the individual examples such as to give the total test-set 10 times the weight of the total training set.

3.1 Weighted Nearest Neighbor

In the previous section we stated the way we add a label and a weight to every example in the dataset that will be used for nearest neighbor classification. There are different ways in which to use weights for nearest neighbor classification. One way is to make the distance dependent on the weight of the examples. An obvious way would be to divide the standard distance by the weight of the example [14]. This would make it harder for examples with a small weight to influence the prediction. However, when using k-nearest-neighbor prediction, this approach will change the identity of the k selected examples and in a set-up like the one provided by YATSI , where only 2 different weights are available, it could prevent the examples with the lower weight to ever be part of the k closest examples.

Another way of incorporating weights in nearest neighbor predictions is that once the k nearest neighbors are selected, we choose to use the weights of the examples as a measure for their influence on the total vote. Instead of counting the number of neighbors that belong to each class, we sum their weight and predict the class with the largest weight. By normalizing the sums of the weights, so that they all add up to 1, we can even get an indication of the probability for each of the available classes. Note though, that the distance to an example does not influence its contribution in the vote. Once an example makes it into the set of the k closest examples, the only factor that influences its contribution is its weight.

For continuous class-values, where predictions usually are made using the sum

$$\frac{\sum_j \frac{t_j}{dist_{ij}}}{\sum_j \frac{1}{dist_{ij}}}$$

over all examples in the with dataset t_j being the target value of example j and $dist_{ij}$ being the distance between examples i and j, both ways of incorporating the weights of examples are equivalent. As such, although we have not yet implemented this and do not have any experimental results, YATSI can be used for predicting continuous target values as well without major changes.

3.2 Other Nearest Neighbor Issues

For our experiments, we fixed the number of nearest neighbor to 10. However, this is not a requirement for the YATSI algorithm. Cross-validation on the labeled training examples would be a way adapting the number of nearest neighbors. However, the resulting values of k obtained in this fashion might be misleading because of the large amount of extra examples that will be available in the second step of the YATSI algorithm.

Since the algorithm is designed to work in applications where the amount of labeled training data is limited, one can in most cases get away with less efficient algorithms in the first step. As we expect the amount of test data to greatly exceed that of the training data, most of the computational complexity will lie in the search for nearest neighbors, as this search spans the combined sets of training and test examples.

YATSI will therefore greatly benefit from using efficient nearest neighbor search algorithms. Currently, we use KD-trees [15] to speed up the nearest neighbor search. However, recently a lot of research effort has gone into the development of more efficient search strategies for nearest neighbors, which can be directly applied to the YATSI algorithm. Examples of such search strategies are cover trees [16] and ball trees [17].

4 Experimental Results

We evaluated YATSI using a number of datasets from the UCI-repository. We created labeled and unlabeled set by splitting the available data into randomly chosen subsets. We ran experiments with 1%, 5%, 10% and 20% of the available data labeled (the training set) and the rest available as the test-set. In general, we collected results from 29 different data set, except for the 1%-99% case split, where the 8 smallest data-set were removed because a 1% sub-set was not large enough to train a classifier on.

The design of YATSI does not specify any specific algorithm to be used in the first step. We ran experiments with an array of algorithms that are all available in WEKA consisting of:

AdaBoostM1 : This is a straightforward implementation of the AdaBoostM1 algorithm. In the experiments reported we used J48 both with default parameter settings and without pruning as a base learner, and performed 10 iterations.

J48 : This is Weka's reimplementation of the original C4.5 algorithm. Default parameter settings were used except for the confidence, which was set to the values 0.25, 0.50 and 0.75. We also ran experiments without pruning the trees.

Logistic : A straightforward implementation of logistic regression run with Weka's default parameter settings.

RandomForest : An implementation of Breiman's RandomForest algorithm, but based on randomized REPTrees (instead of CART). At each split the best of $log(n_attrs)$ randomly chosen attributes is selected. The ensemble size was set to 10 and 100.

SMO : Weka's implementation of the SMO algorithm for training support vector machines. Linear, quadratic and cubic kernels and a cost value of 1.0 were used.

IB1 : A standard nearest-neighbor algorithm using Euclidean distance with all attributes being normalized into a $[0, 1]$ range.

We also collected results for different values of the weighting parameter F ranging from 0.1, i.e., giving 10 times as much weight to the training set as to the test-set, to 10.0 which does the exact opposite. We also ran some experiments that used no weights at all. These experiments treat all the "pre-labeled" test-set examples exactly like training examples. Therefore, in the 1%-99% split case, the total weight of the test-set would be almost 100 times as big as that of the training-set.

We expect the performance of Yatsi to go down with the performance of the classifier trained on the labeled data in stage 1 as the amount of available training data decreases, but we expect (and will show) that the performance degrades slower, i.e., that Yatsi is able to improve on the results obtained by only learning from the labeled data. To get statistically sound results, we repeated every experiment 20 times.

Table 1 shows the number of statistically significant wins, draws and losses of Yatsi versus the classifier trained on the training-set in stage 1. For J48, we show the results for the experiment with the confidence set to 0.75. This is higher than normal so this setting generates slightly larger trees, which seems to be appropriate for the very small training sets that we use. Higher levels of pruning could even lead to empty trees in extreme cases. Overall, all the J48 experiments showed the same trend. The results shown for the RandomForest experiments are those with an ensemble size of 100. The ones with ensemble size 10 were similar with a slightly bigger advantage for Yatsi . On the SMO experiments, we show the results for the linear kernel experiments. For quadratic and cubic kernels, Yatsi produces less of an advantage, mostly due to the fact that the SMO predictions get better and Yatsi is not able to improve on them, but performs equal to the SMO algorithm more often. For AdaBoost, the shown

results are obtained with the standard settings for J48; a range of different parameter values for AdaBoost produced almost identical results.

Overall, the results show that YATSI often improves on the results of the base classifier. Especially when very little of the data is labeled, YATSI gains a lot from having the unlabeled data available. When the percentage of labeled data increases, YATSI loses some of its advantage, but for the most part performs comparable if not better than the base classifier. The exception seems to be when one uses Random Forests. The weighted nearest neighbor approach of YATSI seems to lose some of the accuracy obtained by voting over the ensemble of trees.

Table 1. Number of statistically significant wins, draws and losses (in that order: W/D/L) in predictive accuracy of YATSI vs. the classifier trained in stage 1, for different values of the weighting parameter. (Tested with a paired t-test, confidence level 0.05, two tailed)

Base Classifier	% labeled data	$F = 0.1$	$F = 1.0$	$F = 10.0$	No Weights
J48	1%	14/7/0	14/7/0	13/8/0	6/15/0
	5%	15/13/1	16/12/1	15/9/5	14/9/6
	10%	16/8/5	16/7/6	15/7/7	16/7/6
	20%	18/4/7	18/4/7	13/6/10	15/6/8
RandomForest	1%	10/10/1	10/10/1	9/11/1	7/12/2
	5%	9/11/9	9/11/9	10/10/9	10/10/9
	10%	6/10/13	10/7/12	9/6/14	10/6/13
	20%	5/9/15	9/8/12	7/5/17	10/13/16
Logistic	1%	13/7/1	13/7/1	13/7/1	11/8/2
	5%	17/9/3	15/11/3	15/12/2	15/11/3
	10%	17/8/4	18/7/4	12/13/4	14/11/4
	20%	13/8/8	15/9/5	12/6/11	14/7/8
SMO	1%	11/8/2	11/8/2	11/8/2	10/9/2
	5%	8/19/2	7/20/2	9/15/5	9/12/8
	10%	5/17/7	8/17/4	9/12/8	10/11/8
	20%	6/14/9	9/12/8	8/5/16	7/11/11
AdaBoost (J48)	1%	13/8/0	13/8/0	13/8/0	6/15/0
	5%	15/13/1	15/13/1	13/12/4	12/13/4
	10%	12/10/7	14/7/8	15/7/7	12/10/7
	20%	11/10/8	13/8/8	12/7/10	12/8/9
IB1	1%	6/12/3	6/12/3	7/11/3	7/11/3
	5%	12/12/5	12/12/5	12/9/8	13/9/7
	10%	13/13/3	14/11/4	11/7/11	15/4/10
	20%	12/10/7	13/9/7	12/6/11	13/7/9

To give more of an indication of the actual improvements reached by YATSI in terms of predictive accuracy, Table 2 shows the actual predictive accuracies

from the experiments with 5%-95% splits when one uses J48 as the classifier in stage 1.

Table 2. Predictive accuracies of J48 and YATSI using J48 as the stage 1 classifier averaged over 20 runs of the experiments. The data-sets were split into training- and test-set with a 5%-95% ratio. Significant improvements or degradations were tested with a two-tailed 5% confidence interval.

Dataset	J48	$F = 0.1$	$F = 1.0$	$F = 10.0$	No Weights
iris	75.73	87.18 ○	87.15 ○	84.52 ○	83.79 ○
ionosphere	76.63	74.60	74.60	72.21 ●	72.23 ●
lymphography	62.73	63.37	63.41	60.99	60.77 ●
labor	60.27	65.88 ○	66.25 ○	60.27	60.27
hungarian-14-heart-disease	76.73	75.74	75.74	77.00	76.69
cleveland-14-heart-disease	68.72	73.83 ○	73.79 ○	73.37 ○	72.56 ○
hepatitis	73.77	78.16	78.03	77.93	77.89
heart-statlog	68.50	71.28 ○	71.34 ○	70.91 ○	70.66 ○
vote	93.80	91.60 ●	91.58 ●	91.86 ●	91.83 ●
vehicle	52.41	55.07 ○	55.08 ○	53.94 ○	53.24 ○
zoo	57.27	72.31 ○	72.37 ○	59.84 ○	59.79 ○
vowel	33.69	33.55	33.55	29.65 ●	28.65 ●
sonar	60.83	62.43	62.48	60.76	60.33
primary-tumor	19.72	24.55 ○	23.75 ○	20.31	20.06
soybean	47.49	65.56 ○	65.59 ○	53.25 ○	52.77 ○
balance-scale	68.74	74.08 ○	74.07 ○	69.72 ○	69.53 ○
autos	38.51	40.26	40.31	38.72	38.46
wisconsin-breast-cancer	90.52	94.65 ○	94.62 ○	94.56 ○	94.43 ○
breast-cancer	64.69	66.52	67.11 ○	67.69 ○	67.69 ○
anneal.ORIG	76.38	77.22	76.88	74.80	74.67
anneal	87.69	87.70	87.70	86.81 ●	86.81 ●
audiology	43.63	43.50	43.69	40.50 ●	40.36 ●
pima-diabetes	66.86	68.18 ○	68.53 ○	68.60 ○	68.14 ○
german-credit	65.18	67.55 ○	67.53 ○	68.27 ○	68.46 ○
Glass	42.94	48.74 ○	48.74 ○	44.74 ○	44.32
ecoli	65.49	73.31 ○	73.34 ○	71.70 ○	71.65 ○
horse-colic.ORIG	62.79	63.19	63.26	64.08	64.20
horse-colic	75.57	76.12	76.22	78.13 ○	78.30 ○
credit-rating	79.69	81.53 ○	81.51 ○	82.72 ○	82.52 ○

○, ● statistically significant improvement or degradation

For additional insight into the results we have selected a few typical datasets and plot in Tables 3 and 4 error rates for J48 and YATSI(J48) using different values for the weighting parameter F. The percentage of labels available for training varies between 1% and 20%. General trends are very obvious in these plots, like the fact that more labels usually lead to globally better results, or that with a very small number of labels J48 usually performs worse than YATSI , but

that J48 can outperform YATSI when given more labeled data. With regard to the weighting parameter F we see that values of 0.1 and 1.0 consistently perform better than a value of 10 or the `no-weight` option, which clearly indicates the advantage of taking a cautious approach which puts more trust into the originally supplied labels over the labels generated by the first stage classifier.

As already stated, all previous experiments were run with the number of nearest neighbor for the second stage fixed to 10. Because of the use of weights and the large difference in weights between training and test examples, we thought it might make sense to use a larger number of nearest neighbors, so we also performed experiments with 20 and 50 nearest neighbors in the 1% labeled training data case. Overall, these experiments showed very little difference with the 10 nearest neighbor ones. When there was a difference, there was a little improvement for low values of F (0.1 or 1.0) and a small loss for the cases where a high weight was given to the test-examples ($F = 10.0$ or no weights used at all).

5 Conclusions and Further Work

We have presented a simple two-stage idea that benefits from the availability of unlabeled data to improve on predictive accuracies of standard classifiers. YATSI uses an of-the-shelf classification or regression algorithm in a first step and uses weighted nearest neighbor on the combined set of training data and "pre-labeled" test data for actual predictions. Experimental results obtained from both a large array of different classifiers used in the first step, different amounts of available unlabeled data and a relatively large selection of data-sets show that YATSI will usually improve on or match the predictive performance of the base classifier used generated in the first stage. These improvements are largest in cases where there is a lot more unlabeled data available than there is labeled data.

The YATSI algorithm in its current form is quite simple and therefore a number of further improvements are possible. Some ideas have already been presented in section 3 such as the inclusion of a more efficient nearest neighbor search algorithm or the use of cross validation to determine the best number of nearest neighbors to use. More elaborate extensions could include some sort of EM-algorithm that tries to match the "pre-labels" of test-examples with the eventually predicted values. Distance functions different to simple Euclidean distance could encode specialized domain knowledge and thus help improving classification performance. These directions would relate YATSI more closely to both graph-based and kernel-based methods of semi-supervised learning.

References

1. Seeger, M.: Learning with labeled and unlabeled data. Technical report, Edinburgh University (2001)
2. Blum, A., Mitchell, T.: Combining labeled and unlabeled data with co-training. In: COLT: Proceedings of the Workshop on Computational Learning Theory, Morgan Kaufmann (1998) 92–100

3. Zhou, D., Bousquet, O., Lal, T., Weston, J., Schölkopf, B.: Learning with local and global consistency. In: Proceedings of the Annual Conf. on Neural Information Processing Systems, NIPS. (2004)
4. Nigam, K., McCallum, A., Thrun, S., Mitchell, T.: Text classification from labeled and unlabeled documents using em. Machine Learning **39** (2000) 103–134
5. Rosenberg, C., Hebert, M., Schneiderman, H.: Semi-supervised self-training of object detection models. In: 7th IEEE Workshop on Applications of Computer Vision / IEEE Workshop on Motion and Video Computing (WACV/MOTION 2005), 5-7 January 2005, Breckenridge, CO, USA, IEEE Computer Society (2005) 29–36
6. Joachims, T.: Transductive inference for text classification using support vector machines. In Bratko, I., Džeroski, S., eds.: Proceedings of ICML99, 16th International Conference on Machine Learning, Morgan Kaufmann (1999) 200–209
7. Szummer, M., Jaakkola, T.: Partially labeled classification with markov random walks. In Dietterich, T., Becker, S., Ghahramani, Z., eds.: Advances in Neural Information Processing Systems 14 [Neural Information Processing Systems, NIPS 2001, December 3-8, 2001, Vancouver and Whistler, British Columbia, Canada], Cambridge, MA, MIT Press (2001) 945–952
8. Chapelle, O., Weston, J., Schölkopf, B.: Cluster kernels for semi-supervised learning. In Becker, S., Thrun, S., Obermayer, K., eds.: Advances in Neural Information Processing Systems 15 [Neural Information Processing Systems, NIPS 2002, December 9-14, 2002, Vancouver, British Columbia, Canada], Cambridge, MA, MIT Press (2002) 585–592
9. Zhu, X.: Semi-supervised learning with graphs. PhD thesis, Carnegie Mellon University, School of Computer Science, Pittsburgh, Pennsylvania (PA), USA (2005)
10. Blum, A., Chawla, S.: Learning from labeled and unlabeled data using graph mincuts. In Brodley, C., Pohoreckyj Danyluk, A., eds.: Proceedings of the Eighteenth International Conference on Machine Learning (ICML 2001), Williams College, Williamstown, MA, USA, June 28 - July 1, 2001, Morgan Kaufmann (2001) 19–26
11. Joachims, T.: Transductive learning via spectral graph partitioning. In Fawcett, T., Mishra, N., eds.: Machine Learning, Proceedings of the Twentieth International Conference (ICML 2003), August 21-24, 2003, Washington, DC, USA, AAAI Press (2003) 290–297
12. Zhu, X., Ghahramani, Z., Lafferty, J.: Semi-supervised searning using gaussian fields and harmonic functions. In Fawcett, T., Mishra, N., eds.: Machine Learning, Proceedings of the Twentieth International Conference (ICML 2003), August 21-24, 2003, Washington, DC, USA, AAAI Press (2003) 912–919
13. Neville, J., Jensen, D.: Collective classification with relational dependency networks. In: Proceedings of the Second International Workshop on Multi-Relational Data-Mining. (2003)
14. Blum, A., Chawla, S.: Learning from labeled and unlabeled data using graph mincuts. In: Proceedings of the Eighteenth International Conference on Machine Learning, Morgan Kaufmann (2001)
15. Friedman, J., Bentley, J., Finkel, R.: An algorithm for finding best matches in logarithmic expected time. ACM Transactions on Mathematical Software **3** (1977) 209–226
16. Beygelzimer, A., Kakade, S., Langford, J.: Cover trees for nearest neighbor. In pre-print, available from www.cs.rochester.edu/u/beygel/publications.html (2005)
17. Omohundro, S.: Efficient algorithms with nearal network behavior. Journal of Complex Systems **1** (1987) 273–347

Table 3. Exemplary plots of error rates for J48 and Yatsi(J4) using different values for the weighting parameter F for some selected datasets with between 1% and 20% known labels (I).

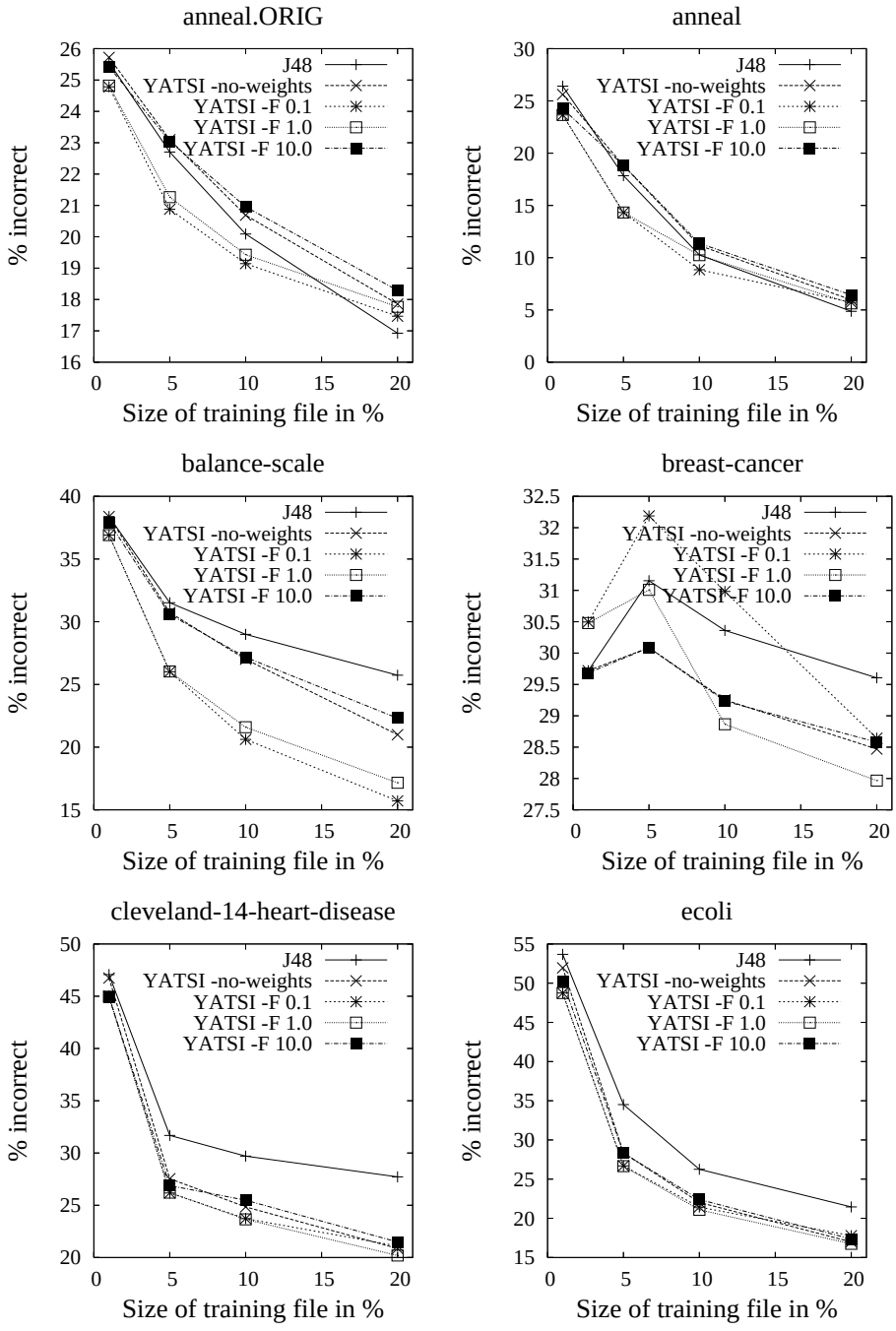

Table 4. Exemplary plots of error rates for J48 and Yatsi(J4) using different values for the weighting parameter F for some selected datasets with between 1% and 20% known labels (II).