MultiStage Cascading of Multiple Classifiers: One Man's Noise is Another Man's Data

Cenk Kaynak CENK.KAYNAK@UNILEVER.COM
Ethem Alpaydin ALPAYDIN@BOUN.EDU.TR
Department of Computer Engineering, Bogazici University, Istanbul TR-80815 Turkey

Abstract

For building implementable and industry-valuable classification solutions, machine learning methods must focus not only on accuracy but also on computational and space complexity. We discuss a multistage method, namely cascading, where there is a sequence of classifiers ordered in terms of increasing complexity and specificity such that early classifiers are simple and general whereas later ones are more complex and specific, being localized on patterns rejected by the previous classifiers. We present the technique and its rationale and validate its use by comparing it with the individual classifiers as well as the widely accepted ensemble methods bagging and Adaboost on eight data sets from the UCI repository. We do see that cascading increases accuracy without the concomitant increase in complexity and cost.

1. Introduction

In recent years, there is an increasing interest in schemes for combining multiple learning systems. The idea is that since learning from a finite sample is an ill-posed problem, each learning algorithm, depending on its inductive bias, finds a different explanation for the data and converges to a different classifier. If these classifiers make errors on different parts of the input space, they complement each other and an ensemble scheme can outperform the individual classifiers.

The more different behaviour individual classifiers exhibit, the more beneficial it is to ensemble since the rate of their complementary decision increases. Using different bootstrap replicates of the training set one can build different classifiers to build up an ensemble. Another alternative is to construct classifiers based on different subsets of input parameters. However, the best way to get different classifiers is by using different learning algorithms, which make different assumptions about the data source, i.e., different inductive biases. For example, one classifier may be a semi-parametric method like the multilayer perceptron (MLP) trained with backpropagation and another may be a nonparametric, instance-based statistical method like the k-nearest neighbor (kNN). MLPs inductive bias is that the class discriminants are nonlinear functions written as weighted sums of sigmoidal basis functions. kNN's bias is that instances that are close in the input space, based on an appropriate metric, belong to the same class.

The ensemble techniques are mainly divided into two categories: multiexpert systems and multistage systems. In multiexpert systems, the classifiers work in parallel. All the classifiers are trained together and given a test pattern, they all give their decisions independently and a separate combiner computes the final decision from these. Examples for multistage systems are voting (Kittler et al., 1998), mixture of experts (Jacobs et al., 1991) and stacked generalization (Wolpert, 1992).

The other approach is multistage systems which use a serial approach where the next classifier is only trained by/consulted for patterns rejected by the previous classifiers. Multistage combination is more interesting in that it is possible not to use costlier classifiers, unless they are actually needed, i.e., when previous simpler classifiers' predictions are not confident and have a high probability of error.

In this work, we aim to build classification systems with high accuracy and low cost. Since our concern is not only accuracy but also cost, combining tens of classifiers may not be feasible due to the increased memory and computation needed. Besides, the time to output is very important for an application to be usable in real life. Therefore we opted for a multistage ensemble system with a small number of classifiers, gaining from accuracy but not losing much in terms of cost.

To implement such a classifier, we have developed a multistage system, namely cascading, where an early simple classifier handles a majority of cases and a more complex classifier is only used for a small percentage thereby not significantly increasing the overall complexity. If the problem space is too complex then a few classifiers may be cascaded, increasing the complexity of the classifier at each step. In order not to increase the number of classifiers, the few points not covered by any are learned by a nonparametric instance-based classifier such as kNN.

An example of a cascading system is as follows. The first classifier is a single layer perceptron (SLP) and the next classifier is a multilayer perceptron (MLP) which is trained by focusing on training patterns not covered by the SLP. The remaining few patterns will be treated as exceptions and covered by an expensive instance-based technique, e.g., kNN. During test, the simple linear classifier will be used for all cases. MLP will be used only when the case is not covered by the SLP. The final kNN will be consulted only at rare circumstances when the MLP also rejects. So, the overall complexity of classification will be slightly above that of the first simple rule while increasing the accuracy for hard patterns.

In Section 2, we describe and discuss the cascading algorithm in detail. The complexity issues are discussed in Section 3. In Section 4, simulation results are given based on eight databases from the UCI repository. These simulation results compare cascading with its component classifiers as well as the widely used aggregation techniques, bagging and Adaboost, in terms of accuracy and complexity.

2. Discussion of the Cascading Algorithm

2.1 Cascading Algorithm

Multistage classifiers are ensembles having individual classifiers with reject option (Pudil et al, 1992). In any stage, a classifier either accepts its output or rejects it based on its confidence in its own decision (Baram, 1998). If it rejects, then the classifier at the next stage is applied to the pattern.

Cascading (Alpaydin & Kaynak, 1998) is in the category of multistage systems. Its inductive bias is that the concept can be explained by a small number of rules with an additional small set of exceptions not covered by the rules. At the first stage it builds a simple rule for the whole training set. This rule is constructed using a parametric classifier, which is a good generalizer. Based on its confidence criterion, it cannot cover some part of the space with sufficient confidence. Therefore, at the next stage, cascading builds a more complex rule focusing on those uncovered patterns. Later at some stage, there will remain few unrelated patterns, which are not covered by any of the prior rules. Those patterns are taken as exceptions to the generated rules. So, no further rule is generated and those exceptions are dealt with by an instance-based nonparametric technique which is good at unrelated, singular points, but not generating rules for a complete domain.

Rules and exceptions of the cascading algorithm can be visualized as in Figure 1. The first classifier, which is the simplest one, generalizes on the whole training set with equal focus on each pattern. So, it constructs a simple rule for the whole space. However, there may be training patterns which are not covered, i.e., learned correctly, (or correctly but not with enough confidence), by this simple rule. Therefore, at the next stage, using a costlier classifier, we build a more complex rule to cover those uncovered patterns of the prior stage. The training of the second classifier is performed on a converted training set, which gives priority to the uncovered patterns of the first rule. As can be seen in Figure 1, after the construction of the second rule, there are some unrelated patterns in the set, which are not covered by any of the two rules. So, those patterns are assumed to be exceptions and treated by an instance-based algorithm, and not by a rule learner.

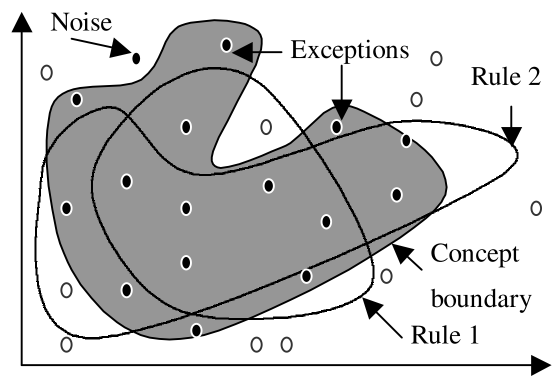

Figure 1. Visualization of how two-rule cascading works. Dark circles are the positive data instances and the shaded region is the concept. Empty circles are negative instances.

Normally, the rule learners are parametric or semi-parametric classifiers, such as perceptrons, which have good generalization properties. Since we increase the complexity of the classifiers at a later stage, if we use an MLP in the first stage then in later stages, we can increase the number of hidden units to increase the complexity. For exceptions, we must use instance-based nonparametric techniques, such as *k*-nearest neighbor, which is good for learning unrelated single patterns although it is expensive in cost.

The sequence of classifiers can be specified as, g_j, where $g_{ji}(x) \equiv P(C_i|x,j)$, the posterior probability estimate of class i for input x by classifier at stage j. Associated with each classifier is a confidence function δ_j such that we say g_j has confidence in its output and can be accepted for decision, if $\delta_j > \theta_j$ where $0 < \theta_j < 1$, is the confidence threshold. The confidence function is calculated at stage j as:

$$\delta_j = \max_i g_{ji}(x) \quad (1)$$

$(1-\delta_j)$ is the probability of error. We consult classifier g_j, if all of its preceding classifiers are not confident in their output.

$$r = \text{argmax}_i \, g_{ji}(x) \quad \text{if } \delta_j > \theta_j \text{ and } \forall k < j, \, \delta_k < \theta_k \qquad (2)$$

During the training of g_{j+1}, we use a converted training set where the patterns are drawn with probability proportional to $(1-\delta_j)$ where

$$\delta_j = g_{jc}(x)$$

for an example belonging to class c. Thus g_{j+1} will focus more on patterns not learned sufficiently well by its predecessor g_j. Note that by θ, we specify a *margin*, distance to the discriminant, in that with larger θ, we enforce a larger margin (Fig. 2)

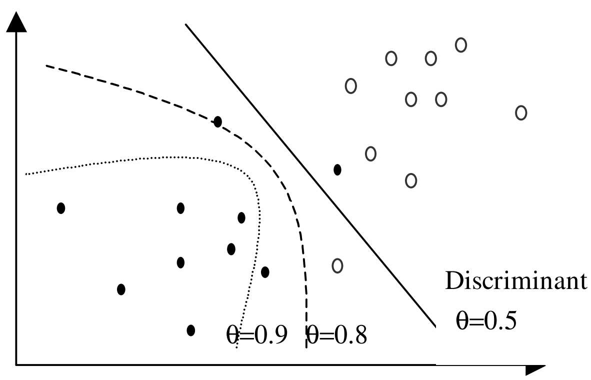

Figure 2. Visualization of the effect of θ on a two-class problem. θ defines the required margin to the discriminant.

This is how we achieve our target of building rules, namely each stage tries to cover the patterns not covered by its predecessors. At stage j+1, the probability that instance with index t is drawn for training is

$$P_{j+1}(x^t) = \frac{1-\delta_j(x^t)}{\sum_{m=1}^{N} 1-\delta_j(x^m)} \qquad (3)$$

Initially, for j=1, these probabilities are equal, that is, $P_1(x^t) = 1/N$ where N is the number of training samples. Since we would like to have better classifiers (less erroneous, more confident) as we go from one stage to the next, we need to have $\theta_{j+1} \geq \theta_j$, that is, the thresholds for classifiers at later stages should be no less than the thresholds at previous stages. At the same time, the classifiers are moving from general to more specific as they get trained on subsets of the original data set.

If we use perceptrons as classifiers with softmax nonlinearity at the output, the values of the output units can be treated as posterior probabilities for each class (Bishop, 1996) and provide us with g_{ji}

$$g_{ji} = \exp(O_{ji}) \,/\, \textstyle\sum_k \exp(O_{jk})$$

where O_{ji} is the i^{th} output neuron of perceptron at stage j.

Therefore, the maximum output value determines the confidence function δ_j, as defined in Eq. (1). Besides, this value also specifies how probable it is that the classification result is correct (Duda & Hart, 1973). At the last stage, we do not need a confidence evaluation, and we use an exception-learner, such as kNN, which just stores the instance as it is.

As stated already, our aim is not only to increase the accuracy but also doing this without much increasing the complexity. The nonparametric instance-based exception learner used at the last stage is a lot costlier than the rules generated at preceding stages. Therefore, we must use it only when we need it. We have already described how we achieve this during test. Parallel to this, we also reduce the unnecessary complexity of the exception learner during training as follows:

Using two-fold crossvalidation, we divide the data set into half. We construct the (n-1) rules using the first (n-1) classifiers on the training half, at each pass reweighting the probabilities using Eq. (3). Then we filter the patterns in the other half through the usual classification tests of the (n-1) classifiers, i.e., $\delta_j > \theta_j$, $j<n$. Only those patterns that fail to pass all of these (n-1) confidence tests are used in the construction of the exception-learner, that is the n^{th} classifier. So, they make up the exceptionary patterns of the original training set and will constitute the lookup table of kNN. This reduced set can be formalized as:

$$\text{TR}' = \{ x^t \mid x^t \in \text{TR and } \delta_j(x^t) < \theta_j \,;\, \forall j, j<n \} \quad (4)$$

where TR is the original training set and TR' is the reduced one. Using two-fold crossvalidation, we then swap the two halves and do this again.

If we use kNN for this purpose, then we do not store the whole training set but only a small subset of it, which fail the confidence tests, thus memory complexity is less. In addition, while classifying a test pattern, we apply the exception-learner only to patterns failing to pass the confidence tests and in kNN, the search for the k closest neighbors is done on a smaller lookup table, thereby decreasing the computational complexity as well.

2.2 Comparison with Bagging and Adaboost

Two very popular ensemble techniques are *bagging* (Breiman, 1994) and *Adaboost* (Freund & Schapire, 1996). Both techniques achieve very good results having MLP as their classifiers. Bagging and Adaboost create differentiated classifiers by differentiating their training sets. Bagging constructs its classifiers by using different bootstrap replicates of the original training set. Adaboost creates those replicates by using different draw probabilities for different patterns of the training set. If a training pattern is misclassified by the prior classifier, then it is weighted higher in training the next classifier.

Although the original *boosting* algorithm (Schapire, 1990) has a multistage nature, both bagging and Adaboost are multiexpert. Adaboost is multiexpert during classification (by taking a vote over all classifiers), even though it is multistage during training.

Cascading algorithm has similar properties to Adaboost. However they also have some major differences. First, cascading is intended to combine a *small* number of classifiers for reduced complexity, whereas Adaboost combines a large number of classifiers.

Second, since cascading uses different learning algorithms with different inductive bias, such as MLP and kNN, and different classifier architectures increasing their complexity at further stages, it makes use of being an ensemble more than Adaboost, which uses the same algorithm and architecture for all classifiers. As already discussed in the introduction section, the more different classifiers an ensemble has, the less correlated they will be and the more beneficial it is to combine them.

Third, cascading is a multistage ensemble, but Adaboost is a multiexpert ensemble. Therefore, cascading uses the benefits of multistage properties and does not consult all classifiers on all instances and thus reduces classification time. Finally, in Adaboost, training replicate construction is based on misclassification whereas in cascading we use the error probability, $(1-\delta_j)$, which gives more information than just misclassification and distributes its focus more appropriately. It can be said that cascading also takes into account the *margin*, the distance to the discriminant, whereas misclassification error, as used by Adaboost, just checks if the input is on the right side of the discriminant or not. In terms of these theoretical aspects, bagging has also the above mentioned disadvantages when compared to cascading.

Therefore, cascading seems to have several advantages over Adaboost and bagging, which are widely accepted and used ensemble methods.

3. Complexity Analysis

Since we aim building a practically implementable and industry-valuable method of classification, complexity and cost are as important as accuracy. Otherwise, even the most accurate algorithm may be infeasible to implement and be wasted.

We have based our cost formulation on computational complexity. We have used single and multilayer perceptrons and kNN in our implementations. For these techniques computational complexity determines both time and space costs.

Say d is the number of input dimensions, c is the number of classes, and h is the number of hidden units. Then the complexity of single-layer perceptron (SLP) and multilayer perceptron (MLP) is as follows:

$$C_{SLP} = O(d \times c) \tag{5}$$

$$C_{MLP} = O(d \times h) + O(h \times c) \tag{6}$$

When N denotes the size of the training set, complexity for kNN is

$$C_{kNN} = O(N \times d) \tag{7}$$

If we combine m MLPs with bagging or Adaboost, the complexity is

$$C_{AdaB} = C_{Bagg} = m \times C_{MLP} \tag{8}$$

Let us consider a cascade of two classifiers, CAS2, with an MLP as the rule-learner and kNN as the exception–learner. During test, if p is the probability that an instance is not rejected by the MLP, kNN is used with probability $(1-p)$. Therefore CAS2's complexity is formalized as

$$C_{CAS2} = C_{MLP} + (1-p) \times C'_{kNN} \tag{9}$$

During training, q denotes the probability that MLP is confident, given a training set of size N, approximately $(1-q) \times N$ of them will be stored as exceptions. Normally we would expect $1-p$, test reject probability, to be close to $1-q$, training reject probability, though in real life q may be an optimistic estimator of p.

So, C'_{kNN}, the complexity of kNN on the reduced set, is only $O((1-q) \times N \times d)$. Substituting this expression into Eq. (9), we get

$$C_{CAS2} = O(d \times h) + O(h \times c) + O((1-p) \times (1-q) \times N \times d) \tag{10}$$

If we increase the level of cascading to three, we end up with three classifiers. CAS3 is the cascading which has two MLPs and kNN. Let h_1 and h_2 be the number of hidden units of the first and second MLPs, respectively. Then substituting those figures to Eq. (6), we get C_{MLP1} and C_{MLP2} respectively. Thus, we get the complexity of CAS3 as follows;

$$C_{CAS3} = C_{MLP1} + (1-p_1) \times C_{MLP2} + (1-p_1) \times (1-p_2) \times C'_{kNN} \tag{11}$$

where p_1 is the test confidence probability by MLP1, and p_2 is the test confidence probability by MLP2 given that MLP1 rejects. In calculation of the complexity of the reduced set, C'_{kNN}, we use the training reject probability of MLP1, q_1, and the training reject probability of MLP2, q_2, given that MLP1 rejects. Thus, in CAS3, we have

$$C'_{kNN} = O((1-q_1) \times (1-q_2) \times N \times d) \tag{12}$$

As a consequence, we can generalize this to a cascade of n stages, having $(n-1)$ MLPs and one kNN.

$$C_{CASn} = C_{MLP1} + \sum_{m=2}^{n-1}\left[\prod_{t=1}^{m-1}(1-p_t)\right] \times C_{MLPm} +$$

$$\left[\prod_{t=1}^{n-1}(1-p_t)\right] \times C'_{kNN} \quad (13a)$$

where

$$C'_{kNN} = O(\prod_{t=1}^{n-1}(1 - q_t) \times N \times d) \quad (13b)$$

From these equations, one can easily see that complex classifiers will be applied rarely, since they are located at later stages.

4. Empirical Results

We have tested the cascading algorithm on eight different databases from the UCI repository (Blake & Murphy, 1999). The tested cascades include perceptrons and kNN and we have investigated whether the cascades construct systems better than their component classifiers in terms of both accuracy and computational complexity. Besides, we have tested Adaboost and bagging on those databases to compare them with our cascades.

4.1 Information on the Data Sets Used

The properties of the data sets are given in Table 1.

Table 1. Properties of data sets used. *C* and *d* are the number of classes and inputs and TRAIN, VALID, and TEST are the number of instances in the training, validation and test sets, respectively.

D. SET	C	d	TRAIN	VALID	TEST
ABALONE	3	8	2700	433	1044
CMC	3	9	700	273	500
LETTER	26	16	10000	5000	5000
OPTDIGITS	10	1024	1934	946	1797
PENDIGITS	10	16	5620	1874	3498
PIMA	2	8	150	50	568
WINE	3	13	100	28	50
WISCONSIN	2	9	250	150	299

4.2 Simulation Results

For some data sets, training set and test set are given separately on the UCI Repository. But for others, a single data set is given and it is split into training set and test set randomly by us once before making the simulations. So for all methods and for all runs, we use the same training set and test set for a given data set.

We have run every algorithm ten times with different random initializations in order to diminish the effect of exceptional circumstances. Only the stand-alone kNN is run once, since its algorithm does not have any randomness. All the results we give here are the averages of those ten runs. We have implemented different cascades by changing the confidence thresholds, θ_j, at different stages. We have also tested Adaboost and bagging ensembles starting from two component MLPs to 50 components (100 for Optdigits).

In order not to clutter the paper with numbers and figures, we give the results on three databases out of eight in detail, which show interesting properties. The best results on all implemented algorithms in complexity versus accuracy scales on all data sets follow. Complexity figures are based on the number of operations performed.

4.2.1 LETTER IMAGE DATABASE

This is the largest database used in this study. Since the size of the training set is large, i.e., 10,000 instances, the application of stand-alone kNN is very costly compared to the application of a perceptron. We have taken k=3 for kNN and MLPs have 10 hidden units.

By calculating the number of operations performed as computational complexity, MLP gets 76.41% classification accuracy with a complexity of 456 operations whereas *k*NN gets 93.38% accuracy with a complexity of 160,000.

Although kNN gets a satisfactory accuracy rate, it is practically too costly to store this training set and the high time complexity may not be afforded. So we need an accurate classifier with an affordable cost. We have also tested Adaboost and bagging using up to 50 MLPs. The most accurate result was from bagging with 50 components, having an accuracy of 80.09% and a complexity of 22,800. Although this cost is less compared to kNN, the accuracy rate is not sufficient.

Then we have tested cascading methods Cas2, which has a single MLP, and Cas3, which has two MLPs in cascade. Of course, both Cas2 and Cas3 is followed by kNN as it is true for all cascadings. To be fair, we used the same MLP architecture with bagging and Adaboost. We have tested with several confidence thresholds. Cas2 has an accuracy of 90.73% with a complexity of 22,878; and Cas3 has an accuracy of 90.89% with a complexity of 22,069. Cascading gets 10.80% more accuracy than bagging with the same complexity.

Moreover, Cas2 got the best accuracy among all the tested methods having 93.73% accuracy with 81,132 complexity. Cas2 gets 0.35% more accuracy than kNN at only half of its cost.

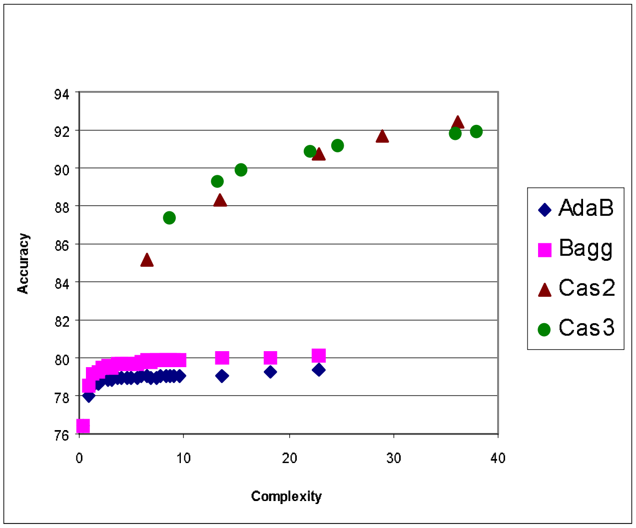

Figure 3. Accuracy vs complexity (×1000 operations) for the letter image recognition application, comparing Adaboost (AdaB), bagging (Bagg) and cascading (Cas2 and Cas3).

It can be seen in Fig. 3 that on letter, Cas3 gets the best results for very low cost values, and Cas2 gets the best results for higher costs. This is because Cas2 stores more instances as exceptions by kNN whereas Cas3 handles some part of them by the simpler second MLP. Both techniques are a lot better than Adaboost/bagging in all aspects.

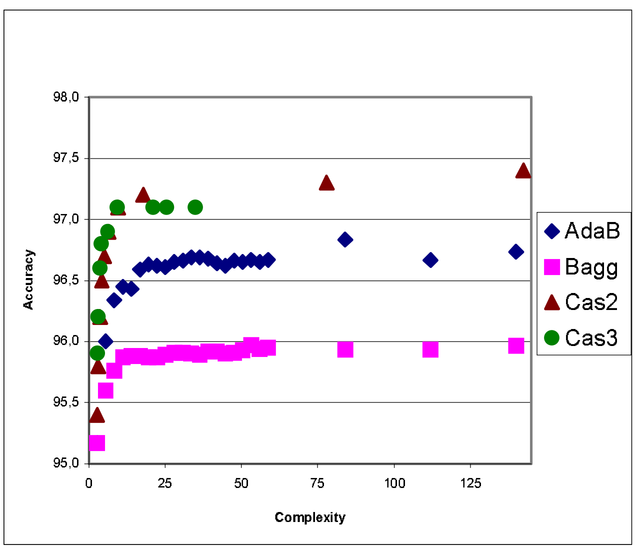

Figure 4. Accuracy vs complexity (×100 operations) for the pen-based digit recognition application, comparing Adaboost (AdaB), bagging (Bagg), and cascading (Cas2 and Cas3).

4.2.2 Pendigits Database

This database contains 5,620 training samples and it can also be considered large. We used kNN with k=3 and MLPs with ten hidden units. On Pendigits, kNN has 97.71% accuracy with a complexity of 89,920; and MLP has 95.17% accuracy with a complexity of only 280.

Adaboost gets the highest accuracy with 96.83% at a complexity of 8,400. Whereas Cas2 gets an accuracy of 97.28% at a cost of 7,792 and Cas3 gets 97.14% at a cost of only 2,557. Therefore, cascading again gets higher accuracy rates than Adaboost/bagging at lower costs.

Fig. 4 shows that Cas2 and Cas3 achieve considerably better results over Adaboost and bagging. We have taken the most accurate result for cascading as 97.35% with 14,238 operations, which is 0.36% less than kNN, but at a cost of only one sixth of kNN (14,238/89,920).

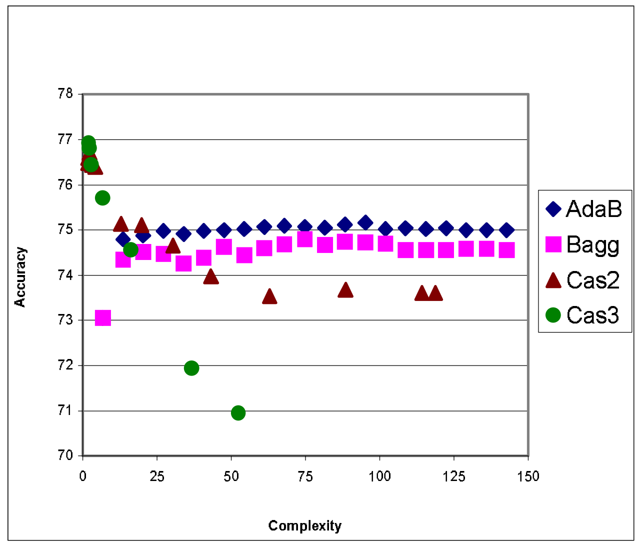

Figure 5. Accuracy vs complexity (×10 operations) for the Pima diabetes recognition application, comparing Adaboost (AdaB), bagging (Bagg), and cascading (Cas2 and Cas3).

4.2.3 Pima Diabetes Database

In comparison to the above discussed two databases, Pima Diabetes has only 768 instances. We used 150 of it for training, 50 for validation of MLPs and 568 for test.

Since our training set is very small, generalizing out of that set is very hard. On this data set, kNN has less accuracy rate (68.10%) than MLP (73.05%), with costs of 1,200 and 68, respectively. This database is different from the other two based on this property that kNN is worse than MLP.

Again, we have tested Adaboost and bagging up to 50 MLPs. Adaboost reaches the best accuracy rate with 75.16% at a cost of 952, which is not far from kNN's cost. Bagging does not reach the 75% accuracy.

On this data set, we have used the single-layer perceptron (SLP) as the first classifier of Cas2 and Cas3. In Cas3, we have increased the complexity at second stage, and used MLP. As usual we used kNN as the last stage. MLPs in

Adaboost/bagging and Cas3 have the same architecture with six hidden units.

On Pima, Cas2 gets 76.71% accuracy at a cost of 23; and Cas3 gets 76.92% accuracy at a cost of 22. Both Cas2 and Cas3 achieve these high accuracy rates at the cost of only one third of a single MLP.

Since increasing the confidence threshold value, θ, increases the usage of kNN, it decreases the accuracy due to the poor performance of kNN (Fig. 5). But for low thresholds, the perceptron is mostly used and those few exceptions are well handled by kNN' (which is the kNN classifier on the reduced set). But if we use high thresholds, the failing patterns will mostly be not exceptions and kNN' cannot deal with them properly. The other interesting point is the fact that Cas3's best score is higher than Cas2's, since it uses kNN fewer times, in fact when it is really needed. Both Cas2 and Cas3 achieve higher accuracy scores than all other algorithms tested; MLP, SLP, kNN, Adaboost/bagging.

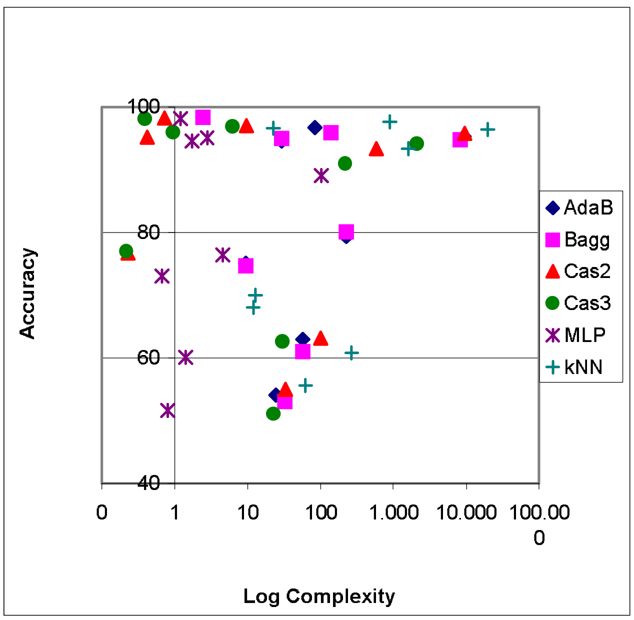

Figure 6. Accuracy vs log-complexity for eight databases using the best results of Adaboost (AdaB), bagging (Bagg), cascading (Cas2 and Cas3), MLP and kNN.

4.2.4 Results On All Databases

In this subsection, results on all eight data sets are presented. In Figure 6, we plot the best results of each method on each database. In the y axis, we have the accuracy percentage and in the x axis, we have the complexity (number of operations) using the same metrics for all databases in logarithmic scale. Each point in the graph represents the best value taken by an algorithm for a database, e.g., best result of Adaboost on pendigits database.

Examining Fig. 6, we see that kNN is the costliest technique and single MLP is the cheapest. Both Cas2 and Cas3 have a good balance of accuracy and complexity. At a glance, it can be easily seen that cascading results are nearer to the top-left corner of the graph, which means that Cas2 and Cas3 get more accurate results with lower costs compared to the other methods we have tested.

In Table 2, there is the numerical comparison in terms of the most accurate results of each method on each data set. The given numbers are accuracy percentages and standard deviations. Looking at Table 2, we see that in four out of eight applications, cascading gives the highest score. In three of them, it immediately follows kNN at lower cost. Only on Wisconsin, cascading gets the second rank after Adaboost/bagging by a difference of 0.11% but at a cost of only one third of them.

Therefore, we conclude that cascading gets higher classification accuracies at less cost than all other tested methods.

Table 2. Classification accuracy comparison in percentages. The first data column, Ada/Bag, shows the highest score of the better of Adaboost and the bagging algorithm.

D. Set	MLP	KNN	Ada/Bag	Casc
Abal	60.05 ±2.2	60.82	63.00 ±0.7	63.51 ±0.7
Cmc	51.58 ±1.7	55.60	54.08 ±0.8	55.08 ±0.8
Letter	76.41 ±0.9	93.38	80.09 ±0.2	93.73 ±0.2
Opt	89.05 ±1.9	96.49	95.33 ±0.3	95.82 ±0.4
pen	95.17 ±0.5	97.71	96.83 ±0.2	97.35 ±0.1
Pima	73.05 ±1.3	68.10	75.16 ±0.5	76.92 ±0.6
Wine	94.60 ±1.3	70.00	95.00 ±1.1	97.80 ±1.7
Wisc	98.16 ±0.3	96.66	98.41 ±0.2	98.30 ±0.4

5. Conclusions

In this paper, we presented a new multistage algorithm, cascading, where there is a sequence of increasingly more complex and specific classifiers. Early classifiers are simple and general. Later ones are more complex and are locally specialized.

Since we begin with a reasonably accurate and simple classifier, most of the patterns will be classified easily at low cost. Only harder patterns are rejected and processed by the later, more complex classifiers. Thus, complex methods are applied to a small percentage of the patterns, and we get higher accuracy rates without significantly increasing the complexity.

The number of cascading stages as well as the threshold values used at each stage, are critical for the accuracy and the cost of the implementation. So, they should be tuned taking into account the constraints on complexity and accuracy as determined by the requirements of the application.

The method has also been generalized to regression and links have been made to radial-basis functions and mixtures of experts (Alpaydin, 1997).

Acknowledgments

This work is supported by Bogazici University Research Funds grants 98HA0101 and 00A101D.

References

Alpaydin, E. (1997). *REx: Learning a rule and exceptions* (Technical Rep. 97-040). International Computer Science Institute, Berkeley, CA.

Alpaydin, E., & Kaynak, C. (1998). Cascading classifiers. *Kybernetika*, *34*, 369-374.

Baram, Y. (1998). Partial classification: The benefit of deferred decision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *20*, 769-776.

Bishop, C. M. (1995). *Neural networks for pattern recognition*. Oxford: Oxford University Press.

Blake, C., & Murphy, P. M. (1999). UCI Repository of Machine Learning Databases, http://www.ics.uci.edu/ ~mlearn/ MLRepository.html

Breiman, L. (1994). *Bagging predictors* (Technical Rep. 94-421). Department of Statistics, University of California, Berkeley, CA.

Duda, R. O., & Hart, P. E. (1973) *Pattern classification and scene analysis*. New York: Wiley.

Freund, Y., & Schapire, R. (1996). Experiments with a new boosting algorithm. *Proceedings of the Thirteenth International Conference on Machine Learning* (pp. 148-156), San Francisco: Morgan Kaufmann..

Jacobs, R. A., Jordan, M. I., Nowlan, S. J., & Hinton, G. E. (1991) Adaptive mixtures of local experts. *Neural Computation*, *3*, 79-87.

Kittler, J., Hatef, M. Duin, R. P. W., & Matas, J. (1998). On combining classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *20*, 226-239.

Pudil, P., Novovicova, J., Blaha, S., & Kittler, J. (1992). Multistage pattern recognition with reject option. (pp. II 92-95). *Proceedings of the Eleventh IAPR International Conference on Pattern Recognition B* (pp. 92-95).

Schapire, R. (1990) The strength of weak learnability. *Machine Learning*, *5*, 197-227.

Wolpert, D. H. (1992) Stacked generalization. *Neural Networks*, *5*, 241-259.