CASCADING MULTIPLE CLASSIFIERS AND REPRESENTATIONS FOR OPTICAL AND PEN-BASED HANDWRITTEN DIGIT RECOGNITION

E. ALPAYDIN, C. KAYNAK, F. ALİMOĞLU
Department of Computer Engineering
Boğaziçi University
TR-80815 Istanbul, Turkey
E-mail: alpaydin@boun.edu.tr

Abstract

We discuss a multistage method, cascading, where there is a sequence of classifiers ordered in terms of complexity (of the classifier or the representation) and specificity, in that early classifiers are simple and general and later ones are more complex and are local. For building portable, low-cost handwriting recognizers, memory and computational requirements are as critical as accuracy and our proposed method, cascading, is a way to gain from having multiple classifiers, without much losing from cost. Simulation results on optical and pen-based handwriting digit recognition indicate that when compared with voting, mixture of experts and stacking, our proposed method, cascading, does stand out as the most realistic combination scheme.

1 Introduction

In recent years with computation getting cheaper, 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 assumptions, finds a different explanation for the data and converges to a different classifier. If these classifiers make errors on different cases, they complement each other and an ensemble scheme can outperform the individual classifiers.

To get different classifiers, one may use *different learning algorithms*, which correspond to making different assumptions about the data source, i.e., the inductive bias of the learning method. For example, one classifier may be a neural network method like the multilayer perceptron (MLP) trained with backpropagation and another may be a nonparametric, instance-based, statistical method like the k-nearest neighbor (k-NN). MLP's inductive bias is that the class discriminants, separating the instances of one class from the instances of other classes, is a nonlinear function written as a weighted sum of nonlinear sigmoidal basis functions. MLP is attractive in that it has a small number

In: L.R.B. Schomaker and L.G. Vuurpijl (Eds.), Proceedings of the Seventh International Workshop on Frontiers in Handwriting Recognition, September 11-13 2000, Amsterdam, ISBN 90-76942-01-3, Nijmegen: International Unipen Foundation, pp 453-462

of parameters (low memory complexity) and once it has been trained, it is very fast to use (low computational complexity), and it is a good generalizer. k-NN's inductive bias is much simpler: It only assumes that instances that are close in the input space (according to an appropriate metric) belong to the same class. k-NN is also quite accurate; 1-NN has been proved to be never worse than twice the Bayesian risk [4], implying that half of the classification information in an infinite sample set is contained in the nearest neighbor, which supports the inductive bias of k-NN. But being a nonparametric method, it requires the storage of the whole training set (high memory complexity) and given a test instance, its distance to all the training instances is computed to choose the k closest (high computational complexity).

Another way to get different classifiers is by using *different representations* of the same input. Different representations make different features apparent and provide supplementary information. For example in pen-based handwriting recognition, there is the *dynamic* movement of the pen tip while writing the character, represented as a temporal sequence of tablet coordinates. There is also the *static* image once the writing of the character is over, represented as a two dimensional binary image.

Orthogonal to these algorithmic and representational dichotomies, there are two architectural methodologies: In *multiexpert* methods, the classifiers work in parallel. All the classifiers are trained together and given a test pattern, they all give their decisions and a separate combiner computes the final decision from these. Examples are *voting* [9], mixture of experts [7,2] and *stacked generalization* [11]. *Multistage* methods [10] use a serial approach where the next classifier is only trained by/consulted for patterns rejected by the previous classifier(s). Multistage combination is more interesting in that costlier classifiers are not used, or costlier representations are not extracted, unless they are actually needed, i.e., when previous classifiers' predictions are not ambiguous.

In this work, we aim building a low-cost, possibly portable, system for optical or pen-based handwriting recognition and thus accuracy is not the only concern but complexity (as it determines the cost) is also very important. In such a scenario, combining tens of classifiers may not be feasible, due to the increased memory and computation needs. Thus we aim for a combination of small number of — two or three — classifiers, gaining from accuracy, but not losing much in terms of cost.

We advocate multistage methods where an early simple classifier handles a majority of cases and a complex classifier is only used for a small percentage, thereby not significantly increasing the overall complexity. A classifier may be simpler than another due to its algorithm; for example, MLP is simpler than k-NN because given an instance to classify, the memory and computational

complexity is less. A classifier may also be simpler than another one due to the representation it uses. In pen-based recognition, as we will see shortly, the static image representation is more costly than the dynamic pen movement representation as the image is two dimensional and preprocessing the image through several filters takes more time and computation than the one (temporal) dimensional signal of pen coordinates.

This paper is organized as follows: In Section 2, we formalize cascading and in Section 3, we detail two variants of cascading on the two tasks of optical and pen-based handwritten digit recognition. We conclude in Section 4.

2 Cascading

Multistage classifiers are ensembles having individual classifiers with reject option [10]. In any stage, a classifier either accepts its output or rejects it based on its certainty of its own decision. If it rejects, then the classifier at the next stage is applied to the pattern. The next classifier in the cascade is a more accurate but costlier classifier, maybe due to using another representation.

We propose two variants of cascading:

1. In *cascaded-algorithms* [3], the first classifier is a simple classifier learning a general rule over the input space. The next classifier is a costly classifier learning local exceptions to the general rule. In our implementation, the first rule-learner is MLP which is fast and has small number of parameters and the second exception-learner is k-NN. Thus during test, a large percentage of the cases are handled by the rule-learner resorting to the costlier k-NN only in a small number of cases. The algorithm has the following inductive bias: *Underlying the data, there is a simple rule and there are also a small number of exception instances, not covered by the rule with enough confidence.* It is also possible to cascade a sequence of increasingly more complex rule-learners before the final exception-learner is employed [8].

2. In *cascaded-representations* [1], the first classifier uses a simple representation and the second classifier uses a representation that is costlier to extract. There is the following inductive bias: *A large percentage of the examples from different classes are separable in the space defined by the simple representation; a smaller percentage of ambiguous instances are distinguishable in the space defined by the more complex representation.* One can also envisage a setting where a sequence of increasingly costlier representations is extracted from the input each feeding to a different

classifier. Thus costlier representations are not extracted unless previously extracted simpler representations do not suffice to classify with confidence.

There is a sequence of learners g_j which provide $g_{ji}(x) \equiv P(C_i|x, j)$, the posterior probability estimate of class i for input x by learner j [3,8]. It is the case that g_{j+1} is a costlier method than g_j or uses features that are costlier to extract. Associated with each learner is a confidence function δ_j such that we say g_j is confident of its output and can be used if $\delta_j > \theta_j$ where $0 < \theta_j < 1$ is the confidence threshold. We use the output of learner g_j (class with max posterior) if all the preceding learners are not confident

$$r = \arg\max_i g_{ji} \text{ if } \delta_j > \theta_j \text{ and } \forall k < j, \delta_k < \theta_k \tag{1}$$

The confidence in classification is the maximum posterior probability

$$\delta_j(x) = \max_i g_{ji}(x) \tag{2}$$

Note that $1 - \delta_j$ is the probability of error and thus the higher δ_j is, the more confidence we have in the prediction. During training, after having trained learner g_j, g_{j+1} should be trained on a training set where the patterns are drawn with probability proportional to $1-\delta_j$. Thus g_{j+1} will focus more on patterns misclassified, or ones classified without enough confidence. At stage $j + 1$, the probability that instance with index t is drawn for training is

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

For $j = 1$, these probabilities are all equal, i.e., $P_1(x^t) = 1/N$ where N is the number of training examples. Note also that as we would like to have better (less erroneous, thus more confident) learners as we go from stage j to $j + 1$, we need to have $\theta_{j+1} \geq \theta_j$. At the same time, the classifiers are moving from general to more specific as they get trained on subsets of the original dataset.

The cascading method seems similar to the AdaBoost [5] algorithm but there are certain differences. First, cascading is intended to combine a *small* number of *different* classifiers whereas AdaBoost combines a large number (typically 50-100) classifiers using all the same learning method or representation. Second, AdaBoost is multistage during learning but uses voting for decision and thus is multiexpert during test, whereas cascading is multistage both during training and test. Third, 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.

3 Applications

3.1 Cascaded Algorithms on Optical Handwritten Digit Recognition

`optdigits` was created using the set of programs made available by NIST [6] for optical handwritten digit recognition. The 32 by 32 normalized bitmaps were low-pass filtered and undersampled to get 8 by 8 matrices where each element is an integer in the range 0 to 16. 44 people filled in forms which were randomly divided into two clusters of 30 and 14 forms. From the first 30, three sets were generated: A training set, a validation set on which to tune hyperparameters (number of hidden units of the MLP, k of k-NN, etc) and a writer-dependent set. The other 14 forms containing examples from distinct writers make up the writer-independent test set which we use as the real test set. We have recently donated the dataset to the UCI repository.

We used cascaded-algorithms where a multilayer perceptron (MLP) with 20 hidden units is the first rule-learner and k-nearest neighbors (k-NN) with $k = 9$ is used as the exception-learner. We use two-fold cross-validation in training as follows: We divide the training set into two parts and first train the MLP with the first half of the training set. Then we take the second part and feed it to the MLP and all the patterns that the MLP rejects (the highest posterior is less than θ) are taken as exceptions and stored. We then do the same with the role of the two halves exchanged. The final list of exceptions define the lookup table of the k-NN. During test, an instance is first given to the MLP and if the highest posterior is greater than θ, it is taken as the output. Otherwise, k-NN is done over the table of exceptions to give the output.

Let us say that d is the number of input dimensions and c is the number of classes. If there are h hidden units, the time and space complexity of MLP is

$$C_{MLP} = O(d \times h) + O(h \times c) \tag{4}$$

If N denotes the size of the training set, the time and space complexity of k-NN is

$$C_{kNN} = O(N \times d) \tag{5}$$

If p is the probability that an instance is not rejected by the MLP during test, $1 - p$ is the probability that k-NN is used; MLP is always used. The

complexity of a two-stage cascade of MLP and k-NN is therefore

$$C_{cas2} = C_{MLP} + (1 - p) \times C'_{kNN} \tag{6}$$

During training, if q denotes the probability that MLP is confident, given a 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.

Then C'_{kNN}, the complexity of k-NN on a reduced set, is $O((1-q) \times N \times d)$. Substituting the complexities, we have

$$C_{cas2} = O(d \times h) + O(h \times c) + O((1 - p) \times (1 - q) \times N \times d) \tag{7}$$

On our dataset, the accuracy of cascading, the percentage of exceptions stored, and complexities as a function of θ are given in Fig. 1. We see for example that 85% of the cases are classified with 0.99 confidence by the MLP, storing 15% as exceptions ($q = 0.85$). During test, only 23% of the cases are not classified with 0.99 confidence by the MLP ($p = 0.77$) and we use k-NN. 23% of 15% makes 3%, which is the complexity of cascading as opposed to doing k-NN over the whole set, all the time. In a combination scheme like voting and stacking where both classifiers are always used, the complexity is $C_{MLP} + C_{kNN}$!

In Fig. 2, we compare the single classifiers and various combination schemes in terms of their accuracies on the writer independent test set and the number of bits required to store the parameters. MLP is fast and simple but is not as accurate as 9-NN which is costly in terms of memory and computation. In voting and stacking, both are used so the memory (and computational) requirement is high. Cascading here seems ideal as it gives high accuracy with only a small increase in memory without much additional complexity.

3.2 Cascaded Representations on Pen-Based Handwritten Digit Recognition

`pendigits` is a dataset on pen-based handwritten digit recognition. We collect 250 samples from 44 writers [1]. The samples written by 30 writers are used for training, validation and writer dependent testing, and the digits written by the other 14 are used for writer independent testing. We have also donated this dataset to the UCI repository.

The primary goal is to construct a high-performance system that exploits the difference between two different representations of the same handwritten digit. One representation is dynamic; the movement of the pen as the digit

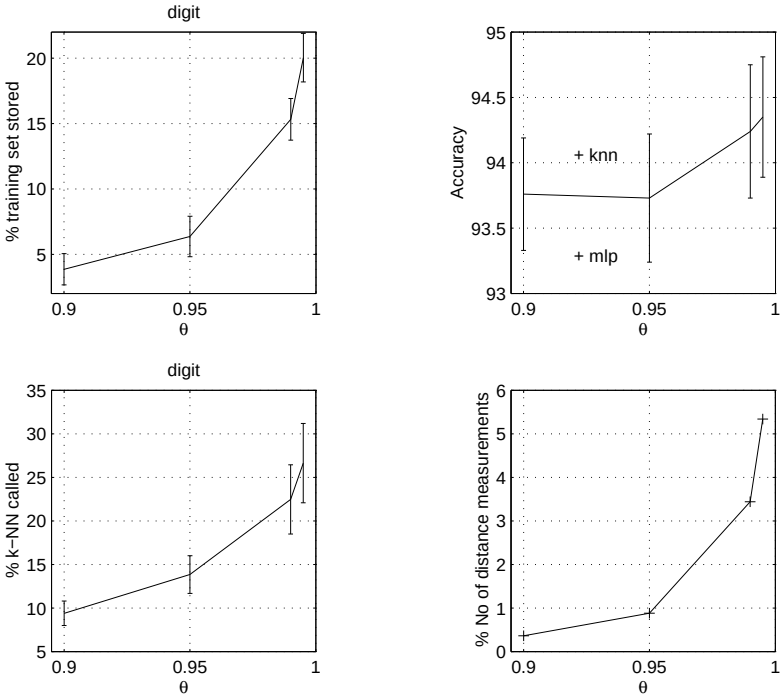

Figure 1: On the `optdigits` dataset with cascading, average percentage of exceptions stored during training and accuracy on the test set as a function of θ are shown with one standard deviation error bars. Accuracy of MLP and k-NN alone are also shown. On the second line, percentage of calls to the exception learner during testing and the number of distance measurements done as a function of θ are shown. The number of distance measurements is the product of the percentage of calls to the exception learner and the number of exceptions.

is written on a pressure-sensitive tablet. The other representation is static; the image generated as a result of the movement of the pen tip. Two different hand movements for the same character may lead to similar images or different images of the same character may be generated by similar hand movements. Therefore, combining the two representations may improve recognition accuracy.

The simplest approach is *concatenating* all data vectors and treat as one large vector from a single source. This does not seem theoretically appropriate as this corresponds to modelling data as sampled from one multivariate statistical distribution. The approach we take is to make separate predictions using different sources and then combine these predictions. We train two MLPs with

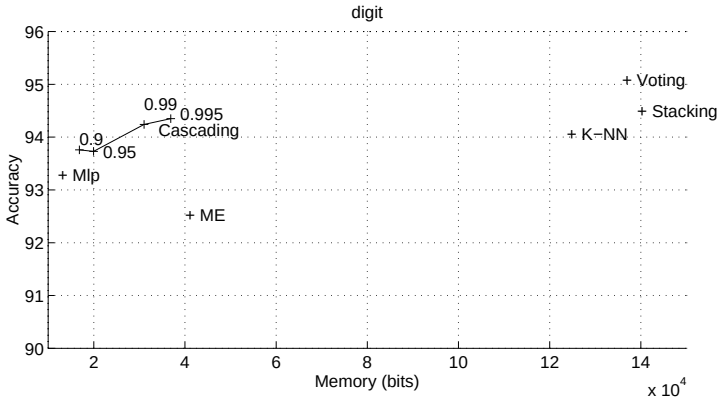

Figure 2: On `optdigits`, accuracy vs memory requirement of MLP, k-NN and four ways of combining the two (voting, stacking, mixture of experts and cascading) is shown. The results by cascading are given for different θ values. We use eight bits to store each real value.

Table 1: Error percentages of the two classifiers and of both. If we have an oracle to choose which classifier to use, error can be reduced to 1.70%.

Set	Dynamic	Static	Both
Training	1.02	1.24	0.25
Validation	1.67	3.76	0.46
Writer-dependent	1.74	4.27	0.70
Writer-independent	4.74	5.75	1.70

ten hidden units on each of the representations. Dynamic representation has eight (x, y) coordinates thus is 16 dimensional. Static representation is a 8×8 matrix of elements in the range 0 to 16 and thus is 64 dimensional. We notice that the two classifiers make mistakes for different samples and investigate ways to combine them efficiently (Table 1).

We test voting, mixture of experts and stacking to combine the two MLPs and compare with cascading. In cascading, we use the dynamic MLP as the simple classifier and static MLP as the complex one. This is because the preprocessing required to form the image, blur and downsample is time consuming. Cascading is a way to avoid doing this for all patterns.

Let us denote by C_{MLPdyn} and C_{MLPsta}, the complexities of the dynamic and static MLP classifiers. Note that MLPsta is more complex because it has four times as many inputs. Let us denote by $C_{prepsta}$, the complexity of the

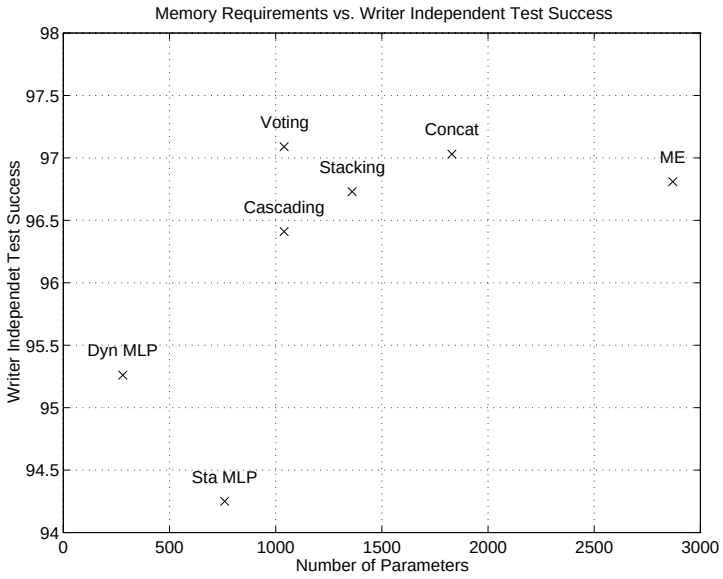

Figure 3: Average accuracy on writer independent test set versus the number of parameters.

preprocessing done to get the image from the dynamic information, blur, and downsample it. If p denotes the probability that dynamic MLP is certain, with probability $(1-p)$, we also need to generate the image and use the static MLP. Then cascading has complexity

$$C_{cas} = C_{MLPdyn} + (1-p) \times (C_{MLPsta} + C_{prepsta}) \tag{8}$$

We again use two-fold cross-validation to train the two classifiers with $\theta = 0.99$ where 30% are rejected by the dynamic MLP ($p = 0.7$) and train the static MLP. When θ is less, the accuracy on the writer dependent set is higher but there are not enough patterns to train the static MLP and thus the accuracy on the writer independent set is less. In Fig. 3, we plot the accuracy vs the number of parameters of different combination methods. Though static MLP has an overall performance worse than the static MLP, because they do not make the same errors (Table 1), combining them the accuracy increases. Among the various combination methods, cascading is the fastest as it uses the costlier static MLP only in 30% of the cases whereas the other combination methods use it all the time.

4 Conclusions

In building low-cost, real-world handwriting recognizers, combining a large number of classifiers may require too much memory and computation. We propose a new multistage combination method, cascading, that takes the advantage of multiple classification algorithms and/or representations without significantly increasing memory and computational complexities. The idea is to cascade a small number of classifiers ordered in terms of complexity (of the classification algorithm or representation) and specificity, in that early classifiers are simple and general and later classifiers are locally accurate. Thus later, complex, classifiers/representations are not used unless actually necessary.

Acknowledgment

This work is supported by Boğaziçi University Research Fund grant 00A101D.

References

1. F. Alimoğlu and E. Alpaydın, "Combining Multiple Representations and Classifiers for Pen-based Handwritten Digit Recognition," *ICDAR 97*, Ulm, Germany, August (1997).
2. E. Alpaydın and M. I. Jordan, "Local Linear Perceptrons for Classification," *IEEE Trans. NN*, **7**, 788-792 (1996).
3. E. Alpaydın and C. Kaynak "Cascaded Classifiers," *Kybernetika*, **34**(4), 369–374, (1998).
4. T. M. Cover, "Estimation by the Nearest Neighbor Rule," *IEEE Trans. IT*, **14**(1), 50–55, (1968).
5. Y. Freund and R. E. Schapire, "Experiments with a New Boosting Algorithm," *Proceedings of the 13th ICML*, 148–156, (1996).
6. M. D. Garris *et al*, *NIST Form-Based Handprint Recognition System*, NISTIR 5469, (1994).
7. R. A. Jacobs *et al*, "Adaptive Mixtures of Local Experts," *Neural Computation*, **3**, 79–87, (1991).
8. C. Kaynak and E. Alpaydın "Multistage Cascading of Multiple Classifiers," *Proceedings of the 17th ICML*, (2000).
9. J. Kittler *et al*, "On Combining Classifiers," *IEEE PAMI*, **20**, 226–239, (1998).
10. P. Pudil *et al*, "Multistage Pattern Recognition with Reject Option," *11th IAPR PatRec* II, 92–95 (1992).
11. D. H. Wolpert, "Stacked Generalization," *Neural Networks*, **5**, 241–259, (1992).