Proceedings of Engineering of Intelligent Systems '98 Conference (Ed. E Alpaydın), Vol 2, 6–12, ICSC Press, 1998.

Techniques for Combining Multiple Learners

Ethem Alpaydın
Department of Computer Engineering
Boğaziçi University
TR-80815 Istanbul, Turkey
alpaydin@boun.edu.tr

Abstract—Learners based on different paradigms can be combined for improved accuracy. Each learning method assumes a certain model that comes with a set of assumptions which may lead to error if the assumptions do not hold. Learning is an ill-posed problem and with finite data each algorithm converges to a different solution and fails under different circumstances. Our previous experience with statistical and neural classifiers was that classifiers based on these paradigms do generalize differently, fail on different patterns and to a certain extent complement each other and thus we look for ways to combine them for higher accuracy. One way to get complementary classifiers is by using different input representations. The methods we investigate are voting, mixture of experts, stacking and cascading. We do experiments on three real-world applications: optical handwritten digit recognition, pen-based handwritten digit recognition and the estimation of road travel distances which is a regression problem in operations research. Our results indicate that by combining several learners/representations, we get more accurate systems without increasing much the overall memory and computational requirements.

I. Introduction

Many different paradigms have been proposed in the past for machine learning. Examples are statistical, neural, syntactical, fuzzy, bayesian methods. Though these are generally successful, there is not one single paradigm that is always most accurate. The usual approach is to try many and choose the one that performs best on a separate validation set. However, each paradigm dictates a certain model that comes with a set of assumptions which may lead to high bias if the assumptions do not hold. Learning is an ill-posed problem and with finite data, each algorithm converges to a different solution and fails under different circumstances. For example in pattern recognition, it is generally the case that the set of patterns misclassified by different classifiers do not fully overlap, implying that classifiers (or paradigms that they are based on) to a certain extent complement each other.

Recently with the cost of computation and memory getting cheaper, it has been possible to build systems that are composed of multiple learners based on different learning paradigms, especially for optical character recognition [27], [28]. The idea is that by suitably combining those, indeterminacy can be removed and performance can be improved.

Separate learners may also be using different representations of the input thus making it possible to integrate different types of sensors/measurements or features. Different representations make different characteristics explicit allowing better identification.

In Section II., we review techniques for combining multiple learners. Section III. contains a brief report of our results on three real-world applications and Section IV. concludes.

Supported by Tübitak Grant EEEAG-143. This paper was prepared while the author was on sabbatical leave at the International Computer Science Insitutute, Berkeley CA as a Fulbright scholar. Section III. reviews results that are joint work with respectively, Cenk Kaynak, Fevzi Alimoğlu, and Kuban Altınel-Necati Aras.

II. Techniques for Combining Multiple Learners

A. Introduction

The performance of a learner may be finetuned to get the highest possible accuracy on a validation set but this finetuning is a complex task and still there are patterns on which even the best learner is not accurate enough and some other learner may be accurate on these. By suitably combining those, performance can be improved. As there is no point in combining multiple learners that always make similar decisions, the aim is to be able to find a set of learners who differ in their decisions to complement each other.

Combination approaches can be divided into two depending on the representational methodology:

- In *Uni-Representation*, all learners use the same representation. Thus the learners themselves should be different to cause them make different decisions.
- In *Multi-Representation*, different learners use different representations of the same input. This is either due to using different sensors or different features extracted from the same input data.

From an algorithmic point of view, there are two possibilities:

- The learners may use the same technique, e.g., a multilayer perceptron, but care must be taken to get different final learners: (i) They may use inputs in different representations, i.e., multi-representation, (ii) Training sets may be distinct, e.g., by bootstrapping, (iii) Learning may start from different initial conditions, e.g., in gradient-descent, (iv) Other parameters may be different, e.g., number of basis functions, cost function minimized, etc.
- The different learners may be based on different techniques. For example, one can be statistical and one neural, or one may be programmed as opposed to a trained system.

Orthogonal to these two dimensions, there are two architectural methodologies:

- In *Multiexpert* methods, the learners work in parallel. All of them are trained on all patterns and given a pattern, they all give their decisions and a separate combiner computes the final decision using these. Examples are *voting* (aka ensembles, linear opinion pools), *mixture of experts*, and *stacked generalization.*
- *Multistage* methods use a serial apprach where the next classifier is trained/consulted only for patterns rejected by the previous classifier(s). Multistage methods suit well to multi-representation in that earlier classifiers use simple to extract features and handle a majority of the patterns. Complex, but more informative, features are only extracted for a small percentage without much increasing the overall complexity. Example is *cascading.*

The gain to be had from combining learners depends on how much they are correlated or dependent. The multiexpert methods assume independence whereas the multistage methods actively try to generate such learners.

B. Notation

Let us say that we have m learners. We denote by d_j the estimate of learner j given input x. (In the case of multirepresentation, each d_j sees a different input representation x_j). For now we ignore how d_j are calculated and instead we concentrate on the ways to combine them to find r, the final estimate

$$r = g(d_1(x), d_2(x), \ldots, d_m(x)|\psi) \tag{1}$$

where g is the combining function with ψ denoting its parameters.

When there are c outputs, each learner gives c outputs which are combined. If we are doing classification, assuming that r_i estimates the posterior probability of class i, and that the zero-one loss function is used, to minimize Bayesian risk, we choose the class with the highest probability

$$i^* = \arg\max_i r_i \tag{2}$$

C. Voting

The simplest way to combine multiple classifiers is by *voting* which corresponds to taking a linear combination of the learners. This is also known as *ensembles* and *linear opinion pools.* Let us denote by g_j the weight of learner j. Then the final output is computed as

$$r = \sum_{j=1}^{m} g_j d_j \tag{3}$$

satisfying

$$\forall j, g_j \geq 0 \text{ and } \sum_{j=1}^{m} g_j = 1 \tag{4}$$

In regression, this corresponds to a weighted average of outputs. The name voting comes from its use in classification, where d_j is the vote of learner j and g_j the weight of its vote. If each voter just indicates the class it thinks the input belongs to, then one can only have *simple voting* where all voters have equal weight, i.e., $g_j = 1/m$. If the voters can also supply the additional information on how much they vote for each class, then these can be converted to certainties and used as weights in a *weighted voting* scheme, for example proportional to the difference of the two highest outputs [32], [3]. Another possibility is to assess the accuracies of the classifiers on a separate cross-validation set and use that information to compute the weights.

It has been shown by Hansen and Salamon [13] that given independent classifiers with success probability higher than 1/2, by taking a majority vote, success increases as the number of voting classifiers increases. Mani [21] has shown that in the case of simple voting, variance decreases as the number of independent voters increase. It has been shown by Jacobs [15] that m dependent experts are worth the same as k independent experts where $k \leq m$. Ali and Pazzani [1] show that there is a substantial correlation between the amount of error reduction due to the use of multiple models and the degree to which the errors made by individual models are correlated.

Xu et al. [32] discuss various ways in which the outputs of several classifiers are combined. To compute the weights, they propose to use a *belief* measure or the Dempster-Schafer theory (See also Rogova [26]). Lincoln and Skrzypek [20] propose a way to learn the weights in a voting scheme. Model complexities can also be taken into account in a Bayesian framework by giving larger weights to simpler models [3]. Perrone [24] gives a number of didactic examples that depict the advantage of voting. He also shows that for minimum square error, when the learners are unbiased and uncorrelated, weights should be inversely proportional to variances. Benediktsson and Swain [9] propose to use a voting scheme for multisource sensing where data from different sources are integrated based on consensus theory. Tumer and Ghosh [30] propose to use order statistics combiners as an alternative to linear combiners.

It is also possible to use regression to compute the weights during voting by minimizing error on a training set. Lagrange optimization can be used to enforce that weights sum up to one [24]. Or it can be unconstrained and an intercept term may also be added. Stacked generalization (Section E.) is an extension in that it allows this combination to be any function of the outputs.

Adapting what Perrone [24] stated for regression to classification, if we view each learner as a random noise function added to the true class discriminant function and if these noise functions are uncorrelated with zero mean then the averaging of the individual estimates is like averaging over the noise. In this sense, the voting method is smoothing in the functional space and can be thought of as a regularizer with a smoothness assumption on the true discriminant function.

Krogh and Vedelsby [18] define "ambiguity" as the variation of the output of voters averaged over unlabelled data to quantify the disagreement among the voters and show that for minimum error, we need to maximize the ambiguity. If the voters are strongly biased, the ambiguity will be small because voters implement very similar functions and thus agree on inputs even outside the training set. If on the other hand there is a large variation, the ambiguity is high and in this case the generalization error will be smaller than the average generalization error. They also note that one way to increase the ambiguity is to train the voters on different data sets. Meir [23] showed for linear regression that by training experts on disjoint data sets and using voting, large decrease in variance can be achieved due to the independence of experts. The increase in bias is not as high and hence error too decreases. For the case of nearest neighbor classification too, significant improvements in accuracy can be achieved through voting on subsets of training set [6]. Depending on the size of the training set, the subsets may be chosen by drawing patterns with replacement (bootstrap) or without.

Bagging takes a vote over learners trained with bootstrap replicates of a training set [10]. Adaboost [11] modifies the probability of drawing a training pattern when new learners are generated in such a way that patterns misclassified by the current learner has higher chance of being drawn to train the next learner.

Voting schemes can be seen as approximations under a Bayesian framework with weights approximating prior model probabilities and model decisions approximating model-conditional likelihoods [3] and under certain circumstances, linear opinion pools and Bayesian techniques will yield identical results [15]. The priors are in turn modeled as distributions with hyperparameters and in the ideal case, one should integrate over the whole model-parameter space. This approach is generally infeasible in practice as modeling the priors and likelihoods analytically is not straightforward; Gibbs sampling is done to get a small number of models with high priors. With advances in Bayesian statistics, these supra Bayesian techniques may become more important in the near future.

D. Mixture of Experts

In voting, the weights g_j are constant over the input space. In the mixtures of experts architecture proposed by Jacobs et al. [14], they are a function of the input. Thus the learners, or experts in the original terminology, are allowed to specialize on local regions of the input space. This is dictated through the used cost function which is based on the likelihood of a normal mixture where to maximize the likelihood, experts are forced to compete and not overlap. This has the effect that the experts partition the input space among them and the gating system decides on the weights $g_j(x)$, i.e., which expert to use given input x.

$$r = \sum_{j=1}^{m} g_j(x) d_j \tag{5}$$

The local experts and the gating expert are trained all in parallel and are based on linear models. Structures similar to the mixture of experts have also been proposed by Stokbro et al. [29] and Martinetz et al. [22] where the training of the gating expert and those of local experts are uncoupled. Alpaydın and Jordan [5] show that the coupled approach as done by the mixture of experts is superior to the uncoupled. Jordan and Jacobs [16] generalize the approach to learn tree-like hierarchies of experts.

E. Stacked Generalization

This is a technique proposed by Wolpert [31] that extends voting in that, the way the output of the learners is combined need not be linear but is learned through a combiner system which is also trained. In the original terminology, the learners are called level 0 generalizers and the combiner is the level 1 generalizer. In our case, level 0 generalizers are the d_j and the level 1 generalizer is the combining function $g(\cdot|\psi)$ specified up to a parameter vector ψ. For example $g(\cdot)$ may be a multilayer perceptron and ψ its connection weights

$$r = g(d_1, d_2, \ldots, d_m|\psi) \tag{6}$$

The level 1 generalizer learns what the correct output is when level 0 generalizers give a certain output combination. Thus level 1 needs be trained on data unused in training the level 0 generalizers. Wolpert proposes to use leave-one-out though this is too costly and k-fold cross validation is more efficient when a larger sample exists. Note that there is no restriction on $g(\cdot)$ and unlike voting, the combination can be nonlinear.

LeBlanc and Tibshirani [19] make the link between stacking and statistical model mixing. Zhang et al. [33] use stacking for protein secondary structure prediction with significant improvement in accuracy. In their study, the level 0 generalizers are a statistical model, a memory-based learner and a one hidden layer neural network. The level 1 generalizer is another neural network with one hidden layer.

F. Cascading

The idea in cascaded learners is to have a sequence of learners d_j [7], [8]. It is the case that d_{j+1} is a costlier method than d_j or uses features that are costlier to extract. Associated with each learner is a confidence function g_j such that we say d_j is confident of its output and can be used if $g_j > \theta_j$ where $0 < \theta_j \leq \theta_{j+1} < 1$ is the confidence threshold. We use learner d_j if all the preceding learners are not confident

$$r = d_j \text{ if } g_j > \theta_j \text{ and } \forall k < j, g_k < \theta_k \quad (7)$$

In classification we use a short-cut in that we say learner j is confident if the highest posterior is larger than a threshold, a strategy used for rejection in pattern recognition.

Starting with $j = 1$, given a training set, we train d_j. Then we find all patterns on which d_j's performance is not acceptable (covering both regression/classification) and these constitute the training set of d_{j+1}. The patterns on which d_j's performance is acceptable constitutes the training set of g_j which estimates $p(x|j)$.

In our experiments, we use two-level cascading where the first one learns a "rule" that hopefully covers a large percentage of the cases and the second learns the "exceptions" not covered by the "rule." The rule-learner is a simple linear model or a multilayer perceptron with a small number of hidden units and the exception-learner is is a more accurate but expensive nonparametric memory-based model.

III. Applications

A. Optical Handwritten Digit Recognition

`digit` was created using the set of programs made available by NIST [12] for optical handwritten digit recognition [17]. 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

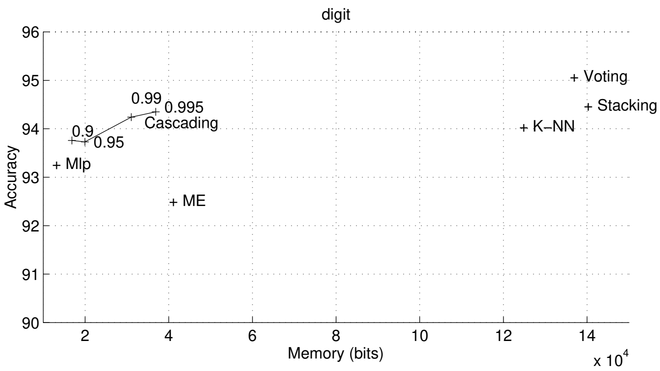

Fig. 1. On `digit`, accuracy vs memory requirement of MLP, k-NN and four ways of combining the two (voting, stacking, mixture of experts and cascading) is shown. The results by cascading are given for different θ values. We use eight bits to store each real value.

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 and a writer-dependent set. The other 14 forms containing examples from distinct writers make up the writer-independent (WI) test set which we use as the real test set. As the final writers can never be known in advance, the aim is to have high accuracy on the writer independent test set containing samples from writers unseen during training. This dataset is available over the web[1].

We tried different ways to combine a multilayer perceptron (MLP) type neural network with 20 hidden units and the statistical method k-nearest neighbors (k-nn) with $k = 9$. In Fig. 1, we see accuracy on the writer independent test set vs 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 (Fig. 2).

B. Pen-Based Handwritten Digit Recognition

`pen` is a dataset on pen-based handwritten digit recognition. We collect 250 samples from 44 writers [2]. 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. The database in UNIPEN format is available over the web.

The primary goal is to construct a high-performance system that exploits the difference between two different representations of the same handwritten digit (Fig 3). One representation is dynamic, that is, the movement of the pen as the digit is written on a pressure-sensitive tablet. The other representation is static, that is, the image generated as a result of the movement of the pen

[1] `ftp://ftp.cmpe.boun.edu.tr/people/ethem/datasets/{digit,pen}`

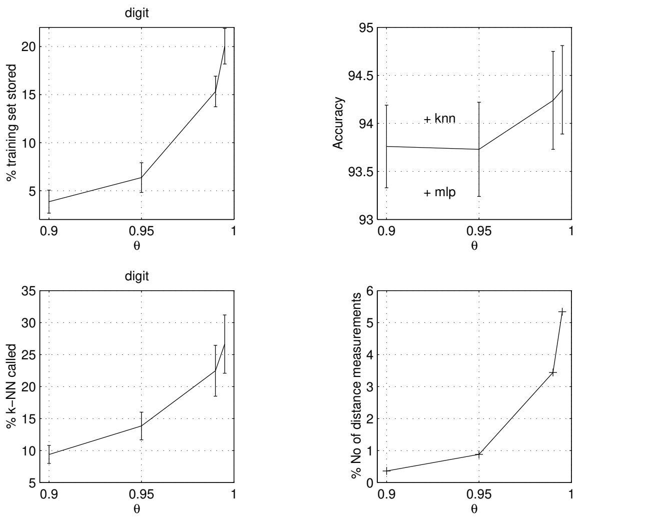

Fig. 2. On the **digit** dataset with cascading, percentage of exceptions stored during training and accuracy on the test set as a function of θ are shown. Accuracy of MLP and k-NN alone are also shown. On the second line, percentage of calls to the exception learner during testing and the number of 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.

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.

In many applications, there are multiple sources of information and it is desirable to use all of these data to extract more information and achieve higher accuracy in prediction. This is similar to *multisensor fusion* where the data from different sensors are integrated to extract more information for a specific application.

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. Moreover, larger input dimensionalities make the systems more complex and require larger samples for the estimators to be accurate. For example in a statistical classifier assuming normal distributions, the covariance matrices have $O(d^2)$ parameters and with small samples, they may be singular. The sources also may have different reliabilities which should be taken into account.

The approach we take is to make separate predictions using different sources and then combine these predictions. We train two MLPs with 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 $[0..16]$ and thus is 64 dimensional. We notice that the two classifiers make mistakes for different samples and investigate

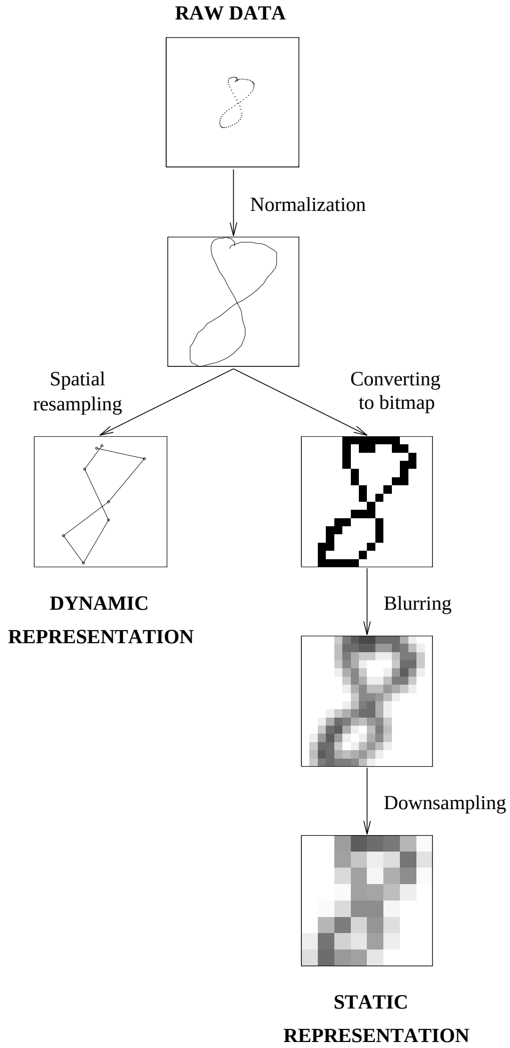

Fig. 3. The stages of preprocessing to get the dynamic (pen-movements) and static (image) representations.

ways to combine them efficiently (Table I).

We take a simple vote on the two classifiers. This is as accurate as concatenating but uses less parameters. We use a mixture of experts architecture where the experts are the two MLP-based classifiers and the gating network is another MLP with 20 hidden units which as input takes a concatenation of the two representations. This works better than the ME proper with linear experts and gating.

In stacked generalization, the level 0 classifiers are the two MLPs. As level 1 generalizer, we use an MLP with ten hidden units and this performs a little better than a linear combiner (equal to a voting approach with trained unconstrained g_j with an intercept). We use 2-fold cross-validation to train the level 0 and level 1 generalizers which makes this approach longer to train.

In cascading, we use the dynamic MLP as the simple classifier and static MLP as the complex one. This is because the preprocessing required for the static MLP is time consuming and cascading is a way to avoid doing

TABLE I

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

Set	Dynamic	Static	Both
TR	1.02	1.24	0.25
CV	1.67	3.76	0.46
WD	1.74	4.27	0.70
WI	4.74	5.75	1.70

TABLE II

For the road travel distance estimation problem, average error of three estimators and the result of voting and stacking.

Estimators	Training error	Test Error
Parametric model	3.61	8.10
Multilayer perceptron	2.63	7.91
Locally weighted averaging	2.38	8.49
Voting	2.26	7.63
Stacking	3.81	7.41

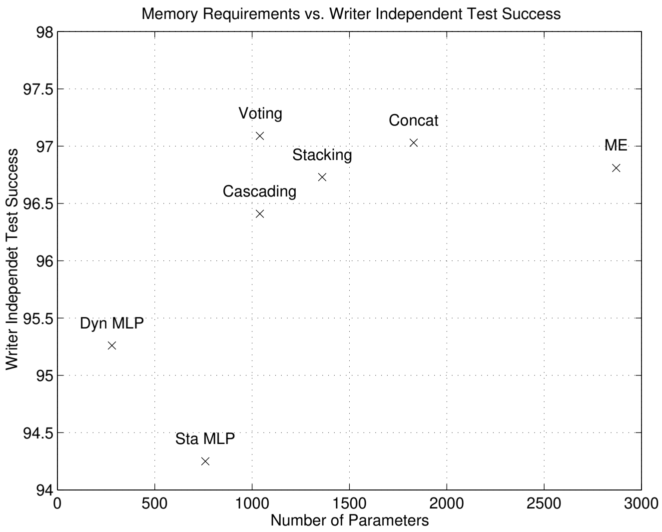

Fig. 4. Average accuracy on writer independent test set versus the number of parameters.

this for all patterns. We again use 2-fold cross-validation to train the two classifiers with $\theta = 0.99$ where 30% are rejected by the dynamic MLP and train the static MLP. When θ is less, success on writer dependent set is higher but there are not enough patterns to train the static MLP and thus success on writer independent set is less.

C. Road Travel Distance Estimation

The actual distance between any two points on the earth surface is the length of the shortest road connecting them travellable with the given means of transportation. Since it is often not feasible to measure the actual road travel distances for all pairs of points, it is a common practice to use distance estimators. This has many applications in location and distribution theory, e.g., travelling salesman problem.

Let us say that $\boldsymbol{x} = (\boldsymbol{x}_1, \boldsymbol{x}_2)^T$ where $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are two points on the Cartesian plane with coordinates $\boldsymbol{x}_1 = (x_{11}, x_{12})^T$ and $\boldsymbol{x}_2 = (x_{21}, x_{22})^T$. The aim is to build an estimator $d(\boldsymbol{x}|\Phi)$ of the actual distance between $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$. The parameters Φ are found to minimize a normalized error measure on a training set, so this is a regression problem

$$E(\Phi; \mathcal{X}) = \frac{1}{N} \sum_{i=1}^{N} \left[\frac{d(\boldsymbol{x}^i|\Phi) - r^i}{\sqrt{r^i}} \right]^2 \tag{8}$$

where $\mathcal{X} = \{\boldsymbol{x}^i, r^i\}_{i=1}^N$ is the set of pairs of coordinates and the associated given actual distances.

In the operations research literature, the approach is to define parameterized distance functions as

$$d(\boldsymbol{x}|k, p, s) = k(|x_{11} - x_{21}|^p + |x_{12} - x_{22}|^p)^{1/s} \tag{9}$$

and find the optimal k, p and s values that minimize Eq. (8). When $p = s = 2$ this is a scaled version of the L_2 norm where k is the scaling factor.

In our work [4], we used two other estimators: a multilayer perceptron and locally-weighted averaging. We also used voting and stacking to combine the three estimators. In voting, weights are equal and in stacking, the combiner is a multilayer perceptron with two hidden units. We have collected two distinct samples by pairing respectively 28 and 23 cities of Turkey for training and test sets. The third dimension of altitude is ignored as its effect has been shown to be almost null. In Table II, we report average normalized error per pair in kilometers on both the training and test sets where we see that combining reduces error also in the case of this regression problem.

IV. Conclusions

In machine learning, the aim is to develop a learner with high accuracy, fast learning and use, and low number of free parameters. There is a trade-off between these in that successful methods generally tend to work slowly and use large amounts of memory. Our experiments on three real-world applications where two are on classification and one on regression indicate that by combining we get more accurate systems without increasing much the overall memory and computational requirements.

We conclude that having several learning paradigms is a source of richness that can be exploited by suitable combination. We are still in early stages in exploiting this source and more future work is needed to make its best use.

References

[1] K. M. Ali, M. J. Pazzani (1995) *Error Reduction through Learning Multiple Descriptions*, TR 95-39, Dept of ICS, Univ of Ca, Irvine.

[2] F. Alimoğlu (1996) *Combining multiple classifiers for pen-based handwritten digit recognition* MSc Thesis, Institute of Sciences and Engineering, Boğaziçi University. http://www.cmpe.boun.edu.tr/~ alimoglu.

[3] E. Alpaydın (1993) "Multiple Networks for Function Learning," *IEEE International Conference on Neural Networks*, March, San Francisco: CA, vol. 1, pp. 9–14.

[4] E. Alpaydın, İ. K. Altınel, N. Aras (1996) "Parametric Distance Functions vs Nonparametric Neural Networks for Estimating Road Travel Distances," *European Journal of Operations Research*, vol. 93, no. 2, pp. 230–243.

[5] E. Alpaydın, M. I. Jordan (1996) "Local Linear Perceptrons for Classification," *IEEE Transactions on Neural Networks*, vol. 7, no. 3, pp. 788-792.

[6] E. Alpaydın (1997) "Voting over Multiple Condensed Nearest Neighbor Subsets," *Artificial Intelligence Review*, **11**, 115–132.

[7] E. Alpaydın (1997) *REx: Learning A Rule and Exceptions*, International Computer Science TR-97-040.

[8] E. Alpaydın, C. Kaynak (1998) "Cascaded Classifiers," *Kybernetika*, to appear.

[9] J. A. Benediktsson, P. H. Swain (1992) "Consensus Theoretic Classification Methods," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 22, pp. 688–704.

[10] L. Breiman (1994) *Bagging Predictors*, TR-421, Department of Statistics, University of California, Berkeley.

[11] Y. Freund., R. E. Schapire (1996). Experiments with a New Boosting Algorithm. *Proceedings of the Thirteenth International Conference on Machine Learning*, 148–156.

[12] M. D. Garris, J. L. Blue, G. T. Candela, D. L. Dimmick, J. Geist, P. J. Grother, S. A. Janet, C. L. Wilson (1994) *NIST Form-Based Handprint Recognition System*, NISTIR 5469.

[13] L. K. Hansen, P. Salamon (1990) "Neural Network Ensembles," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, pp. 993–1001.

[14] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, G. E. Hinton (1991) "Adaptive Mixtures of Local Experts," *Neural Computation*, vol. 3, pp. 79–87.

[15] R. A. Jacobs (1995) "Methods for Combining Experts' Probability Assessments," *Neural Computation*, vol. 7, pp. 867–888.

[16] M. I. Jordan, R. A. Jacobs (1994) "Hierarchical mixtures of experts and the EM algorithm," *Neural Computation*, vol. 6, pp. 181–214.

[17] C. Kaynak (1995) *Methods of Combining Multiple Classifiers and Their Applications to Handwritten Digit Recognition*, MSc Thesis, Institute of Graduate Studies in Science and Engineering, Boğaziçi University.

[18] A. Krogh, J. Vedelsby (1995) "Neural Network Ensembles, Cross Validation, and Active Learning," in *Advances in Neural Information Processing Systems 7*, G. Tesauro, D. S. Touretzky, T. K. Leen (Eds.), MIT Press, pp. 231–238.

[19] M. LeBlanc, R. Tibshirani (1994) *Combining Estimates in Regression and Classification*, Department of Statistics, University of Toronto.

[20] W. P. Lincoln, J. Skrzypek (1990) "Synergy of Clustering Multiple Back Propagation Networks," *Advances in Neural Information Processing Systems 2*, D. Touretzky (Ed.), Morgan Kaufmann, pp. 650–657.

[21] G. Mani (1991) "Lowering Variance of Decisions by using Artificial Neural Network Ensembles," *Neural Computation*, vol. 3, pp. 484–486.

[22] T. M. Martinetz, S. G. Berkovich, K. J. Schulten (1993) " 'Neural-Gas' network for vector quantization and its application to time-series prediction," *IEEE Transactions on Neural Networks*, vol. 4, pp. 558–569.

[23] R. Meir (1994) *Bias, Variance and the Combination of Estimators: The Case of Linear Least Squares*, Department of Electrical Engineering, Technion.

[24] M. P. Perrone (1993) *Improving Regression Estimation: Averaging Methods for Variance Reduction with Extensions to General Convex Measure Optimization*, PhD Thesis, Department of Physics, Brown University.

[25] P. Pudil, J. Novovičová, S. Bláha, J. Kittler (1992) "Multistage Pattern Recognition with Reject Option," *11th IAPR International Conference on Pattern Recognition*, Vol II, IEEE Computer Society Press, pp. 92–95.

[26] G. Rogova (1994) "Combining the Results of Several Neural Network Classifiers," *Neural Networks*, vol. 7, pp. 777–781.

[27] S. N. Srihari (1992) "High-performance reading machines," *Proceedings of the IEEE*, vol. 80, no. 7, pp. 1120–1132.

[28] C. Y. Suen, C. Nadal, R. Legault, T. A. Mai, L. Lam (1992) "Computer recognition of unconstrained handwritten numerals," *Proceedings of the IEEE*, vol. 80, no. 7, pp. 1162–1180.

[29] K. Stokbro, D. K. Umberger, J. A. Hertz, (1990) "Exploiting neurons with localized receptive fields to learn chaos," *Complex Systems*, vol. 4, pp. 603–622.

[30] K. Tumer, J. Ghosh (1995) *Theoretical Foundations of Linear and Order Statistics Combiners for Neural Pattern Classifiers*, Dept of ECE, Univ of Texas, Austin.

[31] D. H. Wolpert (1992) "Stacked Generalization," *Neural Networks*, vol. 5, pp. 241–259.

[32] L. Xu, A. Krzyżak, C. Y. Suen (1992) "Methods of Combining Multiple Classifiers and Their Applications to Handwriting Recognition," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 22, pp. 418–435.

[33] X. Zhang, J. P. Mesirov, D. L. Waltz (1992) "Hybrid System for Protein Secondary Structure Prediction," *Journal of Molecular Biology*, vol. 225, 1049–1063.