Bundling Heterogeneous Classifiers with Advisor Perceptrons

Sauchi Stephen Lee

Division of Statistics

University of Idaho

Moscow, ID 83843

(208) 885-7701

steve@laplace.mathstat.uidaho.edu

John F. Elder IV

Elder Research

1006 Wildmere Place

Charlottesville, VA 22901

(804) 973-2372

elder@dataminingLab.com

October 14, 1997

Abstract

Suppose you have multiple advisors, each trained in a different specialty, whom you consult about a decision. You must decide categorically – an up or down call, say (no hedging allowed) – so you require the same of your advisors. Your group has faced similar questions before, and you've recorded their advice and the subsequent outcome for each case. If you can trust that no advisor is working against you, then you only need examine a finite set of advisor weight vectors to discover those which would have been optimal, historically. This new, robust way to integrate multiple trained classifiers, Advisor Perceptrons (Elder 1992), is demonstrated on six datasets, using five heterogeneous classifiers – which we believe combine more fruitfully than homogeneous ones. It is then compared to some common alternative combination techniques and shown to improve generalization accuracy.

Keywords: classifier combination, ensemble methods, bagging, boosting, bumping, stacked generalization, decision tree, neural network, projection pursuit regression, learning vector quantization, logistic regression

1 Introduction

The recent decade of work on modeling algorithms has witnessed great creativity and ferment from several fields as researchers have extended and successfully applied powerful nonlinear techniques – from decision trees and neural networks, to adaptive splines and polynomial networks – to a number of forecasting, classification, and diagnosis challenges (Ripley 1993; Cheng & Titterington 1994; Cherkassky, Friedman, & Wechsler 1995; Elder & Pregibon 1996; Ripley 1996).

Still, the number of distinct techniques one can employ to inductively create a classification model is actually much smaller than it at first appears; many methods are actually reinventions of, or slight deviations from, others. For instance, case-based reasoning is a form of nearest-neighbor modeling, and polynomial neural networks are a type of kernel technique. (Reducing the apparent diversity however, would require the hardship of communicating outside one's specialty discipline!)

Given a plethora of algorithms, a natural question to ask is *Which works best?* A good study toward that end is described in (Michie, Spiegelhalter, & Taylor 1994; reviewed in Elder 1996b). Recently however, several researchers have found that an ensemble of models can be more accurate (on new data) than the best single model (*e.g.,* Jacobs et al. 1991; Wolpert 1992; Perrone & Cooper 1993; Hashem, Schmeiser, & Yih 1994).

This process of *bundling* models together can be thought of as a approximation of the theoretical Bayesian approach of fitting all possible models and combining them by their prior weights. Bundling requires two stages: 1) generating sufficiently diverse models from the training data, and 2) combining their outputs.

Most researchers have generated multiple classifiers from the same family (Xu & Jordan 1993; Romero & Obradovic 1995; Tumer & Ghosh 1995; Skalak 1997), though some use heterogeneous classifiers (Elder 1996a). When generating homogeneous classifiers, diversity can come from training the models with different guiding parameters (*e.g.,* varied initial neural network weights (Battiti & Colla 1994; Hansen & Salamon 1990), different model structure, different stopping and pruning criteria), iterative changes to the case weights (*boosting*, Freund & Schapire 1996), and/or bootstrapping the training data, either with noise (Raviv & Intrator 1995) or without noise (*bagging*, Breiman 1994). We believe that the greatest useful diversity will come from employing diverse, but individually accurate, methods from the neural network, machine learning, and statistics communities.

For combination, the most common approaches are averaging, weighted averaging, and voting. (And, one recent method of bundling, *bumping* (Tibshirani and Knight 1995) doesn't combine at all, ignoring all but the best of the bootstrapped alternative models.)

We here formally introduce, and demonstrate the utility of, a new method of combination: *Advisor Perceptrons* (APs). They are a type of weighted voting model where the possible weight sets can be exhaustively enumerated in a straightforward manner. Invented by Elder (1992), they were first applied to combining heterogeneous techniques by Lee (1996). This paper strengthens and extends those preliminary results.

2 Advisor Perceptrons

Formalizing the analogy of the abstract, suppose we have the outputs of c classifiers $y_1, \cdots, y_c$ and want to combine them to form an integrated output z. For APs, we first discretize $y_1, \cdots, y_c$ into binary ± 1 and input them into the "unbiased" perceptron (weighted sum with a threshold)

$$z = \begin{cases} -1 & \text{if } \sum_{j=1}^{c} w_j y_j < 0 \\ +1 & \text{if } \sum_{j=1}^{c} w_j y_j > 0, \end{cases}$$

where the weights $\mathbf{w} = \{w_1, \cdots, w_c\}$ are such that $\sum_{j=1}^{c} w_j y_j \neq 0$. Consistent with assuming no classifier is misleading, we further constrain the weights $\mathbf{w}$ to be non-negative; *i.e.*, one should, at worst, ignore a classifier's advice.

Given discrete inputs and outputs, infinitely many weight vectors form identical functions.

For example, in a binary AP with three inputs, the weight set $\{3, 5, 7\}$ behaves like $\{1, 1, 1\}$. As the weight space can be partitioned into regions, the possible APs are finite in number, and we can represent each region by a single point (say, that closest to the origin). The APs can be enumerated, and examined on the (training) data, to find the best for combination.

To enumerate the APs for c dimensions, one can solve a linear program, minimizing the sum of weights $\sum_{j=1}^{c} w_j$ subject to the above constraints. For simplicity, one may also add the restriction that the weights be nonincreasing. This generates *AP templates* which may be permuted to form the possible combinations. *New templates* use all inputs; *i.e.*, have no zero weights, and the others may be generated from lower-dimensional new template permutations.

The sets of new templates for $c = 1$ to 5 are $\{1\}, \{\}$, $\{111\}$, $\{2111\}$, and $\{11111, 22111, 31111, 32211\}$. Permute, (and padded), they enumerate the weight space. For example, the four weight sets of a three-input perceptron have the two templates $\{100\}$ and $\{111\}$ which generate the four APs $\mathbf{w}_1^* = \{1, 0, 0\}$, $\mathbf{w}_2^* = \{0, 1, 0\}$, $\mathbf{w}_3^* = \{0, 0, 1\}$, and $\mathbf{w}_4^* = \{1, 1, 1\}$. The perceptron with $\mathbf{w}_4^*$ corresponds to majority voting, and the other three ignore the advice of two inputs and follow the third.

Interestingly, Muraga et al. (Muroga, Tsuboi, & Baugh 1970), working on an early chip layout problem which happened to be similar to APs, showed that for weight sets involving 7 or fewer inputs, all APs are unique and employ only integer values. However, some functions of 8 inputs have fractional optimum weights; for example: (14.5, 12.5, 9.5, 7.5, 6, 4, 4, 1.5, 1.5) – and others have multiple solutions. Further new classes of weight sets start at 9 inputs, they confirmed, providing what may be read as an example of the *curse of dimensionality*, where low-dimensional behavior does not extrapolate cleanly to higher dimensions. Nevertheless, the APs remain finite in number and, although they grow rapidly with dimension, c, as shown in Table 1, they seem to be a very reasonable subset to explore of the 2^{2^c} binary logic functions it is possible to consider.

Table 1: Number of distinct perceptrons

c	Perceptrons	New "templates"
1	1	1
2	2	0
3	4	1
4	12	1
5	81	4
6	1,684	14
7	123,565	114
8	33,207,256	2,335
9	34,448,225,389	172,958

Here, we integrate 5 different classifiers, leading to only 81 Advisor Perceptrons, so clearly exhaustive enumeration is not only more reliable than searching over the space of voting weights, but faster.

3 Classification Algorithms

We now briefly outline the 5 algorithms employed, which were implemented in S-PLUS Version 3.4 (Chambers & Hastie 1992; Venables & Ripley 1994). Logistic Regression, Decision Trees, and Projection Pursuit Regression are standard S-PLUS functions. The Neural Network and Learning Vector Quantization classifiers come from Brian Ripley's nnet and classif libraries from the "StatLib" archive.

Let x_{ji} represent the j-th attribute of pattern $\mathbf{x}_i$ whose true class is $y_i = -1$ or 1. Let $P(y_i = 1|\mathbf{x}_i)$ denote the probability of $y_i = 1$ given $\mathbf{x}_i$. Let $\hat{P}(y_i = 1|\mathbf{x}_i)$ denote the estimated probability of $y_i = 1$ given $\mathbf{x}_i$.

Logistic Regression (LR) The model is

$$logit(P(y_i = 1)) = \beta_0 + \sum_{j=1}^{p} \beta_j x_{ji},$$

where $logit(z) = log\frac{z}{1-z}$, and β's are the parameters to be estimated via maximum likelihood (Myers 1990). If $\hat{P}(y_i = 1|\mathbf{x}_i) \geq .5$, then pattern $\mathbf{x}_i$ is classified as "class +1", otherwise as "class -1".

Decision Tree (Tree) Tree algorithms recursively partitions the feature space into locally constant regions, usually hypercubes parallel to the feature axes. A fully grown tree was first fitted to the training data, then pruned back to a subtree with a lesser number of terminal nodes to avoid overfit. Overtraining and pruning, rather than stopping – as introduced in CART (Breiman *et al.* 1984), seems to help avoid getting trapped in a type of local minima. We modified the S-plus algorithm to behave a bit more like CART by also using 10-fold cross-validation (10-CV) performance on the training data to determine the cut point.

Projection Pursuit Regression (PPR) This statistical procedure proposed by Friedman and Stuetzle (1981) scores low-dimensional projections of high-dimensional data by how well a scatterplot smoother can estimate the output remaining after previous smoothes have been removed. The model is of the form

$$logit(P(y_i = 1)) = \beta_0 + \sum_{m=1}^{k} \beta_m \phi_m (\sum_{j=1}^{p} \alpha_{jm} x_{ji}),$$

where three sets of parameters, projection directions α_m's, the unknown smooth activation functions ϕ_m's, and the projection "weights" β's are estimated in a sequential manner, using *backfitting*

Neural Network (NN) We used the supervised feedforward single-hidden-layer neural network

$$\hat{P}(y_i = 1|\mathbf{x}_i) = \phi_0(w_0 + \sum_{h} w_h \phi_h (w_{0h} + \sum_{j=1}^{p} w_{jh} x_{ji})),$$

where ϕ_0 and ϕ_h's are some fixed monotonic differentiable functions. Such networks are very general and have been shown by many authors to be theoretically capable of approximating any continuous function of the input variables given sufficiently large numbers of hidden units h. Unlike Projection Pursuit Regression, NN training algorithms estimate the weights $\mathbf{w}$'s simultaneously (although with a randomized gradient search procedure that is relatively slow).

Learning Vector Quantization (LVQ) This method originated as a *self-organizing map*, which has traditionally been labeled a neural network method, although it is rather different from the supervised feedforward multi-layer perceptron above. The basic idea is to replace the example cases by a "representative" set of *codebook* vectors. To be used as a classifier, these points are assigned classes randomly; then, the LVQ algorithm adjusts their location in a feedback manner to converge to a stable state. The classification rule for a new case is to use the class of the nearest codebook vector, so moving them implicitly adjusts the class boundaries.

For LR, the datasets used in the experiments (described next) did not have large numbers of uninformative variables, so all inputs were used. For the other four methods, decisions were required as to the degrees of freedom (model complexity) to employ. 10-CV tests were performed on the Trees, to decide where to stop, and on the Neural Networks, to suggest the number of hidden units, h, and on the Projection Pursuit Regression models, to indicate the best number of smoothes, k. As their complexity was considered roughly equivalent, and the CV tests are only indicative, not fool-proof, the *larger* of the two numbers, h and k, was used for both of those values in the experiments. That number was also used to determine the number of LVQ codebook vectors, as cross-validation is cumbersome to use with that method by itself.

The output vectors of these 5 classifiers serve as inputs to the AP for integration. Whereas **LVQ** provides binary -1 or +1 output directly, the other four produce real-valued outputs which must be discretized to binary values. At first, it seems a loss to reduce the precision of the probability information in the estimates in this manner, but forcing the advisor methods to make a binary decision (vote) also seems to add robustness in many situations. In any case, alternative methods of combination were tracked as well.

4 Data Sets

Four of the six data sets employed are from the UCI repository; the last two are generated as described here. A brief overview:

Hypothyroid The full data set has many qualitative and quantitative input variables, and several missing values. Here, we just considered the five quantitative variables denoted by TSH, T3, TT4, T4U, and FTI, and removed cases with remaining missing values. This leaves 2000 cases, of which 1878 are class -1 (negative), and 122 are class 1 (positive).

Waveform This simulated data set consists of 5000 cases and 3 types of waves denoted 0, 1, and 2, each having probability 1/3, as described by Breiman *et al.* (1984). A C subroutine for generating the data is in the UCI repository. To make it binary, we grouped the 3343 waves in groups 0 and 1 to form the class -1, leaving the 1657 waves in group 2 to be class 1.

Credit This data is from a German bank tracking applicants for credit. Here, we use only the 7 numeric variables, and ignore the 13 qualitative variables. Of the total of 1000 cases, 700 of them are "good" customers (class 1) and 300 are "bad" customers (class -1).

Diabetes This data was gathered among the Pima Indians by the National Institute of Diabetes and Digestive and Kidney Diseases. It consists of 768 cases and 8 input variables. The input variables are medical measurements and pregnancy information on each patient. For the response, 268 cases tested positive for diabetes (class 1) and 500 cases tested negative (class -1).

Investment The data are generated from a rule base adapted from (Luger & Stubblefield 1989), and shown in Table 2, which addresses some of the issues involved in investment advising. The data-generating system consists of 5 input variables, shown in bold in Table 2, and a binary response variable *invest-stocks*: -1 represents advice to put off investing in stocks and 1 represents advice to invest in stocks. We generated 10000 cases with approximately equal number in each class.

Table 2: Investment rule base

(1)	if (saving-adequate and income-adequate)	then	*invest-stocks*
(2)	if dependent-saving-adequate	then	saving-adequate
(3)	if assets-high	then	saving-adequate
(4)	if dependent-income-adequate	then	income-adequate
(5)	if debt-low	then	income-adequate
(6)	if (**saving** $\geq$ **dependents**$\times$5000)	then	dependent-saving-adequate
(7)	if (**income** $\geq$ 2500 + 4000$\times$**dependents**)	then	dependent-income-adequate
(8)	if (**assets** $\geq$ **income** $\times$ 10)	then	assets-high
(9)	if (**annual-debt** $<$ **income** $\times$ 0.3)	then	debt-low

Note, that although there is an exact correspondence between a rule-base such as this and a Decision Tree model, the latter must be induced from the sample data. Here, the several diagonal conditions make it difficult for an axes-parallel tree to capture the structure as well as might first be thought.

Normal

The last dataset is generated from two 3-dimensional Normal distributions, skewed with respect to one another and overlapping. Their mean vectors are {0,0,0} and {1,1,1}, with covariance matrices equal to Identity along the diagonal, with zero off-diagonal correlations for the first, and correlations of 0.9, 0.8, and -0.7 for the second. Half of the 100,000 cases used were from each distribution (class) in both training and evaluation. Using a Quadratic Discriminant analysis (whose assumptions of structure exactly match the problem, but whose parameters were estimated), we got 21.3% error on the training data and 21.5% on the evaluation data.

5 Experiments

The finite datasets were first split randomly, so that 2/3 was used for training, and the remaining 1/3 for evaluation. Key user-defined model parameter settings, (the number of terminal nodes in the pruned Tree, the number of projections in PPR, and the number of hidden units in NN) were recorded using 10-fold cross validation on the training data set. (There, 90% of the data is used for training a sub-model, which is tested on the remaining 10%; those results are accumulated after 10 runs so every case has its turn as an out-of-sample case.) We tested values ranging from 1 to 12 and the winner is shown in the first rows of Table 3. The largest of the NN and PPR parameter counts was used for three methods (NN, PPR, LVQ), as described above.

The second phase was to find the optimal Advisor Perceptron (AP) from the out-of-sample predictions on the training data using another round of cross-validation (CV). One 10-CV produces 10 sub-models for a method, which lead to one vector (the size of the original dataset) of out-of-sample estimates. If this was not sufficient to lead to a single AP (some may tie on performance) the CV process was repeated until the best AP stabilized.

In a third phase, the full training set was used to fit the five different classifiers, using the same user-defined model parameter settings found in phase 1. Lastly, the evaluation set was employed to test the whole system: individual trained classifiers and the AP, as well as alternatives: voting, averaging, and using the AP weights without voting). The resulting misclassification rates of the methods and the combining techniques are shown in Table 3.

6 Discussion & Conclusion

The single best technique in these tests is the Neural Network, winning over its competitors 4 of 6 times. But each algorithm does the best of all, or nearly so, on at least one dataset. However, it is hard to tell beforehand – whether from the problem description or the training performance – which algorithm will be the leader.

Similarly, the various combining methods take turns as best. Still, AP beats simple averaging (probably the most common combination approach) 5 of the 6 times, and the best single method, NN, the same proportion.

To get a clearer picture of the relative merits of the individual methods and the combining techniques, for each data set, we scale the range of the observed misclassification rates (from just smaller than the lowest misclassification rates to just larger than the highest misclassification rates) to the unit interval (0,1). A good method or technique should has a relative error rate close to 0, for all data sets. We plot the relative error rates versus the data sets for all five individual methods, as well as averaging, voting, APs, using the AP weights without voting. It is clear from the side-by-side plots in Figure 1 that APs achieve the best overall performance, and using the AP weights without voting is in the second place. APs are more stable in performance than any individual method, so would be more reliable to use when the problem does not have an obvious solution structure. These preliminary results suggest that Advisor Perceptrons should be a candidate combination technique to explore when considering the very useful strategy of bundling models.

7 References

Battiti, R. and Colla, A. M. (1994). Democracy in neural nets: Voting schemes for classification, *Neural Networks*, vol. 7, pp. 691–707.

Breiman, L. (1994). Bagging Predictors, *Machine Learning*, vol. 26, no. 2, pp. 123–140.

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and Regression Trees.* Monterey, California, Wadsworth & Brooks.

Chambers, J. M. and Hastie, T. (1992). *Statistical Models in S.* Pacific Grove, California, Wadsworth & Brooks.

Cheng, B. and Titterington, D. M. (1994). Neural Networks: A Review from a Statistical Perspective, *Statistical Science*, vol. 9, pp. 2–54.

Cherkassky, V., Friedman, J. H., and Wechsler, H. (1995). *From Statistics to Neural Network: Theory and Pattern Recognition Applications.* Springer.

Elder, J.F. (1992). Optimal Discrete Perceptrons for Graded Learning, International Systems, Man, and Cybernetics Conf., Chicago, Illinois, October 18-21.

Elder, J.F. (1996a). Heuristic Search for Model Structure: the Benefits of Restraining Greed, Chapter 13 in *Learning from Data: Artificial Intelligence and Statistics V*, Lecture Notes in Statistics, eds. D. Fisher and H.-J. Lenz, Springer-Verlag: New York.

Elder, J.F. (1996b). A review of *Machine Learning, Neural and Statistical Classification*, (eds. Michie, Spiegelhalter and Taylor; Ellis Horwood, 1994), *J. American Statistical Association* 91, no. 433: 436-437.

Elder, J. F. and Pregibon, D. (1996). A Statistical Perspective on Knowledge Discovery in Databases, Ch. 4 of *Advances in Knowledge Discovery and Data Mining*, eds U.M.Fayyad, G.Piatetsky-Shapiro, P.Smith, and R.Uthurusamy. AAAI/MIT Press.

Freund, Y. and Schapire, R. E. (1996). Experiments with a new boosting algorithm, *Machine Learning: Proceedings of the 13th International Conference*, July, 1996.

Friedman, J. H. and Stuetzle, W. (1981). Projection pursuit regression, *Journal of the American Statistical Association*, vol. 76, pp. 817–823.

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

Hashem, S., Schmeiser, B., and Yih, Y. (1994). Optimal Linear Combinations of Neural Networks: An Overview, *Proceedings of the 1994 IEEE International Conference on Neural Networks*, vol. 3, pp. 1507–1512.

Hosmer, D. W. and Lemeshow, S. (1989). *Applied Logistic Regression.* New York, John Wiley and Sons.

Jacobs, R. A., Jordans, M. T., Nowlan, S. J., and Hinton, G. E. (1991). Adaptive Mixtures of Local Experts, *Neural Computation*, vol. 3, pp. 79–87.

Kohonen, T. (1995). *Self-organizing Maps.* Springer-Verlag, Heidelberg.

Lee, S. S. (1996). Combining Neural and Statistical Classifiers via Perceptron, *Working Notes of the Thirteenth National Conference on Artificial Intelligence Workshop: Integrating Multiple Learned Models for Improving and Scaling Machine Learning Algorithms*, August, 1996.

Luger, G. F. and Stubblefield, W. A. (1989). *Artificial Intelligence and the Design of Expert Systems.* Benjamin/Cummings.

Michie, D., Spiegelhalter, D. J., and Taylor, C. C. (1994). *Machine Learning, Neural and Statistical Classification.* New York, Ellis Horwood.

Muroga, S., Tsuboi, T., and Baugh, C. R. (1970). Enumeration of Threshold Functions of Eight Variables, *IEEE Transcations on Computers*, September.

Myers, R. H. (1990). *Classical and Modern Regression with Applications.* Boston, PWS-KENT.

Perrone, M. P. and Cooper, L. M. (1993). Learning from What's been Learned: Supervised Learning in Multi-Neural Networks Systems, *Proceedings of World Congress on Neural Networks*, vol.III, pp. 354–357.

Quinlan, J. R. (1993). *C4.5: Program for Machine Learning.* San Mateo, Morgan Kaufmann.

Raviv, Y. and Intrator, N. (1995). Bootstrapping with Noise: An Effective Regularization Technique, Technical Report, Tel-Aviv University.

Ripley, B. D. (1993). Statistical aspects of neural networks, In *Networks and Chaos – Statistical and Probabilistic Aspects*, eds O.E.Barndorff-Nielsen, J.L.Jensen and W.S.Kendall. pp. 40-123. London: Chapman and Hall.

Ripley, B. D. (1994). Neural Networks and Related Methods for Classification (with discussion), *Journal of Royal Statistical Society, B*, vol. 56, pp. 409–456.

Ripley, B. D. (1996). *Pattern recognition and neural networks.* Cambridge University Press.

Romero, P. R. and Obradovic, Z. (1995). Comparison of Symbolic and Connectionist Approaches to Local Experts Integration, *Proceedings of IEEE Technical Application Conference.* pp. 105–110, Portland, OR.

Skalak, D. B. (1997). *Prototype Selection for Composite Nearest Neighbor Classifiers.* Ph.D. Dissertation, Dept. of Computer Science, Univ. Massachusets, Amherst.

Tibshirani, R. and Knight, K. (1995). Model searching and inference by bootstrap bumping, Univ. Toronto Dept. Statistics Tech. Rpt., Nov.

Tumer, K. and Ghosh, J. (1995). Theoretical foundations of linear and order statistics combiners for neural pattern classifiers, Unpublished Manuscript, University of Texas at Austin.

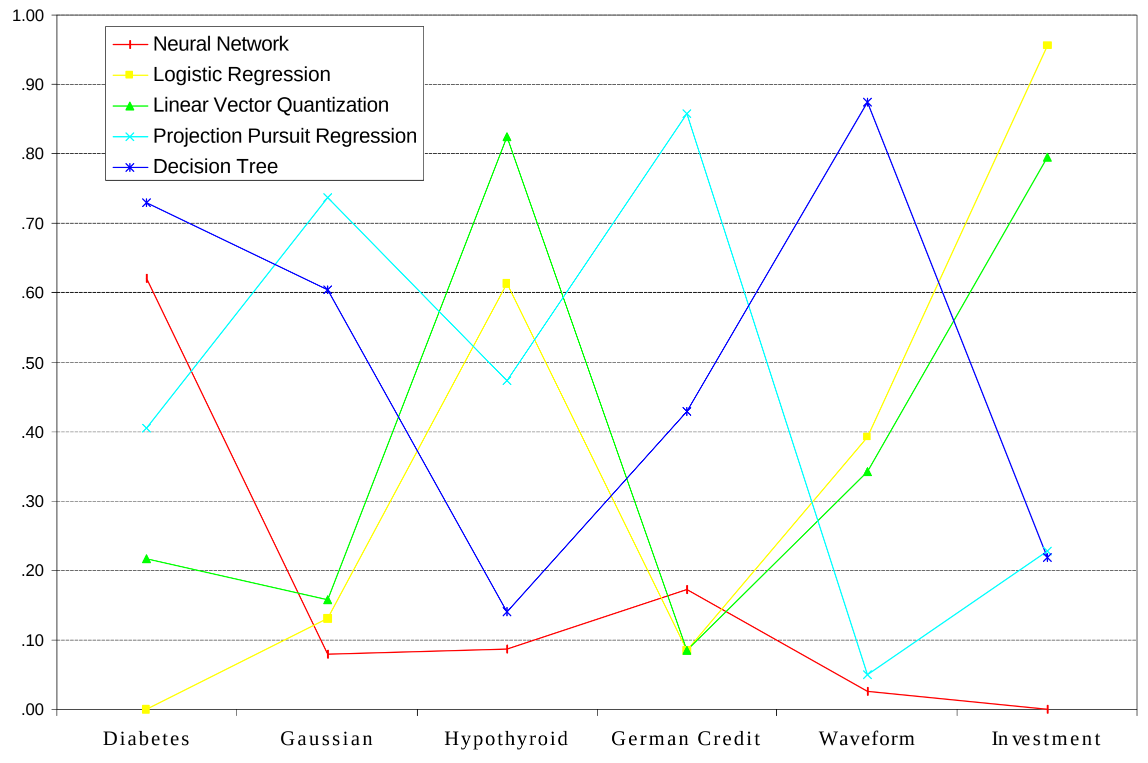
Venables, W. N. and Ripley, B. D. (1994). *Modern Applied Statistics in S-PLUS.* New York, Springer.

Wolpert, D. (1992). Stacked Generalization, *Neural Networks*, vol.5, pp. 241–259.

Xu, L. and Jordan, M. I. (1993). EM Learning on a Generalized Finite Mixture Model for Combining Multiple Classifiers, *Proceedings of World Congress on Neural Networks*, Portland, OR, vol.IV, pp. 227–230.

Table 3: Misclassification rates on evaluation and training data for the 6 data sets

	Hypothyroid	*Waveform*	*Credit*	*Diabetes*	*Investment*	*Normal*
On Evaluation Data						
# nodes	9	19	5	5	41	4
# proj.	8	2	2	3	8	2
# hidden units	7	3	3	3	11	2
LR	6.0	14.4	30.6	21.5	22.9	22.4
Tree	3.3	18.2	33.0	24.2	6.3	24.2
PPR	5.2	11.7	36.0	23.0	6.5	24.7
NN	3.0	11.5	31.2	23.8	1.4	22.2
LVQ	7.2	14.0	30.6	22.3	19.3	22.5
Avg	3.1	11.9	30.6	23.0	3.0	22.5
Vote	4.5	12.0	30.0	22.3	2.6	21.9
AP avg	2.5	11.8	30.6	22.3	1.5	22.4
AP vote	2.7	11.3	30.9	22.3	1.7	22.0
AP	01110	11232	31111	11010	01121	11021
On Training Data						
LR	5.0	14.0	28.8	22.7	24.4	25.6
Tree	1.8	12.8	27.7	20.3	4.8	23.0
PPR	4.5	10.2	23.4	19.5	5.6	21.1
NN	0.6	9.1	24.3	20.3	0.5	25.0
LVQ	7.0	13.1	28.5	23.4	18.0	26.2
Avg	2.6	10.6	28.3	23.4	2.0	26.2
Vote	3.0	9.5	27.1	20.3	1.7	24.2
AP avg	1.1	10.3	28.1	23.2	0.8	26.0
AP vote	1.3	9.3	28.0	20.3	0.7	24.2



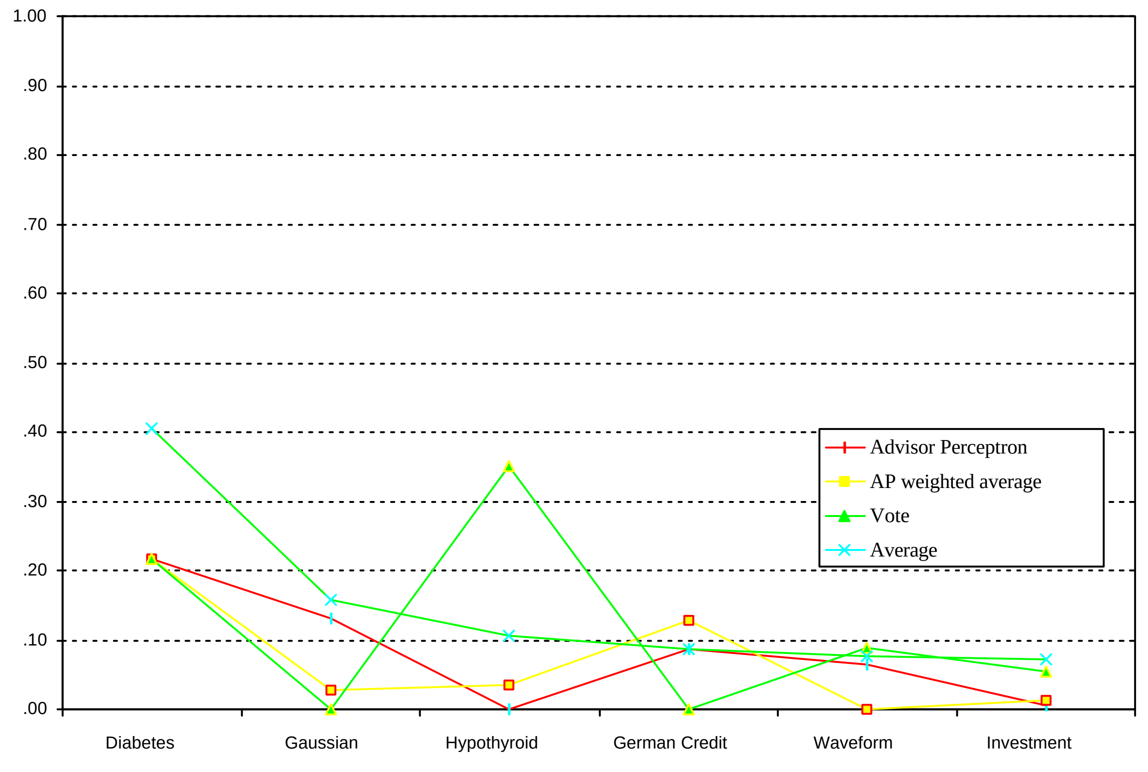


Figure 1: Model Performance on Six Data Sets: Individual Models and Bundles