Building Combined Classifiers

Mark Eastwood, Bogdan Gabrys

Computational Intelligence Research Group,
School of Design, Engineering and Computing,
Bournemouth University
Email: {meastwood,bgabrys}@bournemouth.ac.uk

Abstract. This chapter covers different approaches that may be taken when building an ensemble method, through studying specific examples of each approach from research conductded by the author. A method called Negative Correlation Learning illustrates a decision level combination approach with individual calssifiers trained co-operatively. The Model level combination paradigm is illustrated via a tree combination method. Finally, another variant of the decision level paradigm, with individuals trained independently instead of co-operatively, is discussed as applied to churn prediction in the telecommunications industry.

Keywords. Ensemble Methods, Combining, Classification, Negative correlation learning, churn prediction.

1 Introduction

Classifying and recognising patterns is an important part of many areas of human endeavour, and our brains are excellently adapted to this task. However some such tasks, particularly very repetitive ones, we would much rather automate. This is the problem pattern recognition deals with. We can define a general pattern recognition problem as follows.

Define an input space X and output space Y. The input space may be categorical or continuous, and is the space of features that will be used to describe the objects of interest for a given problem. The output space is categorical for a classification problem, and will be a segment of the real line for a prediction problem. Objects are given as pairs (X, Y) distributed according to some unknown distribution D over the space $\mathsf{X} \times \mathsf{Y}$. We have a sequence of observations (X_i, Y_i) from this distribution, and from these we wish to construct a function $g : \mathsf{X} \to \mathsf{Y}$ which will predict Y from X. In building this function we aim to achieve the lowest possible error on unseen examples (generalisation error). This is usually done by choosing a model, from some class of models, which minimises the error on a training set. To increase the chances of the chosen model generalising well, the class of models should not be too large (there are various measures of the 'size' of a class of models which will not be discussed here, but can be found in [1]). Some form of regularisation may also be used, i.e a modification of the error to penalise some measure of model complexity. A good introduction to statistical learning theory can also be found in [1]. There are many ways to construct this function, each resulting in a slightly different mapping. Given these many different functions that can be built, which one should be chosen for best performance?

Rather than choosing just one, another option is to acknowledge that they may all provide useful, complementary information. We may instead try to use them all, combining them in some way (see [2, 3] for examples).

The combination of classifiers has been shown to be an effective way of reducing generalisation error in both the classification [4] and prediction [5] domains. There are many experimental examples illustrating this, though there is less often a solid theoretical explanation of a methods success. Some methods with by now a fairly solid theoretical basis are the very successful bagging [6], boosting [7] and random forests [8] (see these also for experimental examples). However given a set of classifiers it is not clear how they should best be combined, or what properties they must have to be combined effectively. Intuitively we would expect the combined performance to depend on the accuracy of the individuals, and the extent and nature of the differences between the predictions of the classifiers. For anything more than ad-hoc ideas however, what is needed is a concrete theoretical framework identifying the parameters of an ensemble which control performance, and quantifying the dependence of the performance on these parameters. Understanding of how well these parameters will generalise if we enforce them on a training set is also essential for relating any theory to generalisation errors. Ideally we would like the framework to be so well defined and reliable that we could encapsulate it in a computer program thus automating the selection/combination/tuning procedure.

The goal of this work is to take a few more steps towards this ideal, along these lines:

- Developing the framework to predict the ensemble parameters that will result in a well-performing combined system. We will see later that the relevant parameters are the individual error rates ε_i, size of the ensemble N and some parameter describing the correlations between or complementary nature of individual error distributions.

- Developing the algorithms which will create ensembles whose parameters match those specified by the framework.

- Combining these in a suitable manner.

There are a few differing approaches that can be taken with respect to the second of these points. The ensemble can be generated by selecting from a large pool of classifiers which were generated completely independently of each other, with differences induced in some random manner. This will be referred to as the 'independent' ensemble generation paradigm. We can find a complementary subset for combination via a search using some criterion. Or, we can incrementally build an ensemble taking into account properties of the classifiers already in the ensemble in order to directly generate complementary classifiers for combination. This can be done either by training in parallel or sequentially, and will be referred to as the 'co-operative' paradigm.

There are also two approaches to the third point. Combination of classifiers is usually performed at the decision level; that is the decisions of each classifier in the ensemble are obtained separately and independently, and then some function of the individual decisions is calculated to give a final ensemble decision. A review of some of these methods can be found in [2, 3]. These methods have the disadvantage of needing multiple (often many) different models to be stored and evaluated to give a decision. This often results in greater computational requirements and loss of interpretability (especially important in the business environment).

Another possibility for combining multiple classifiers is model level combination (MLC). The decisions of a classifier are calculated according to some model, and often the model will have a structure such that parts of it can be removed, modified or aggregated. A good example of this is a tree classifier. Pruning of a tree classifier involves the removal of a subtree rooted at a certain node, or equivalently of aggregating the component nodes of that subtree into one node with a single label. This opens up the possibility of combining parts of a multitude of models into a single model. The hope is that by combining components from a number of models into a single model of similar type, some of the benefits of combination methods can be gained, while retaining most of the simplicity and interpretability of a single model.

There has been much less work on model level combination, in comparison to decision level combination. One of the reasons for this is that different classification methods usually give models with differing internal structure, so while decision-level combination methods can usually be used on any base classifier, model level methods are much harder to generalize and will often only apply to one particular base method. What then is the advantage of model level combination? The main advantage is the fact that a single classifier is output at the end, with the advantages of smaller memory requirements and faster implementation over calculating the decisions of a large ensemble of classifiers, with subsequent combination. The decisions of a single classifier are also much more understandable, an advantage especially in a business setting. Thus there has been a little work in this area, which we will review below.

In [9] trees are converted into rulesets, and these rules are combined into a single ruleset by merging similar rules, and resolving conflict between competing rules. Another method that is worth mentioning which, while not a model level combination method, achieves a similar goal (ie combining many models to give a single model of the same type) is [10]. Here a voted ensemble of trees classifies many synthetic examples and then a single tree is trained on these synthetic examples. This has the advantage that it can also be used with any base classifier, but has the disadvantage that many artificial points would need to be generated to approximate well a given ensemble, especially in problems with high dimensionality. This would have an impact on performance, possibly making such an approach infeasible for some problems.

There are other model classes which are amenable to MLC schemes too, which we will briefly mention below. A class of models, such as the parzen classifier, which approximates the class posterior probabilities through the sum of a collection of functions could be combined in an MLC scheme. In this case the individual components are functions defined on the input space, often gaussians. Radial basis function networks also fall into this category of model. It is also possible to combine kernels within Support Vector Machines, as shown in [11, 12].

A similar representation, hyperbox fuzzy sets, also has potential for model level combination. The individual components are hyperboxes, together with the membership function describing how quickly membership drops off with distance from the hyperbox of full membership. Hyperbox Fuzzy Sets have been combined at the model level [13], in a way similar in some respects to the rule-combination method for decision trees described above. Overlaps between hyperboxes are found and the conflict resolved by contracting one of the boxes, and similar boxes may be aggregated.

Neural networks also hold some potential for model level combination. In this case the individual components are the nodes of the network, and nodes can be added or removed, or have their weights modified. In this case however it is difficult to localise the effects of a given node to one area of the input space, and thus it is difficult to associate two nodes from different networks in order to meaningfully combine them. However the possibility is still worth considering. One piece of work which tries to do this is the method in [14], which gets around the problem of associating nodes by training a single network partially, and then building an ensemble using this as a base, with a small amount of additional training on resampled data. The resulting networks are peturbations of the base network, and thus the nodes can be expected to be still doing a similar job in each network allowing an averaging of the weights to be performed to give a final combined network.

The goal of this chapter is to give an overview of the work done towards these goals within the current project. This includes a neural network based method called Negative corellation Lerning (Section 3), a model level combination technique for decision trees (Section 4), and a fairly simple industrial application of a combination method to telecom churn prediction (Section 5). The first of these, NCL, is an example of the the second 'co-operative' approach to ensemble generation, i.e. the other members of the ensemble are constantly taken into account during training of each network in order to directly generate a complementary ensemble. The combination method used there is simple averaging, an example of decision level combination. In last two the ensemble members are generated independently of each other, the second illustrating the concept of MLC. The third is another example of a decision level combination scheme, and also stands as an example of the potential use in industry of ensemble methods, and pattern recognition in general. The next section will give some very brief theoretical background which will be useful, especially for our discussion of NCL in Sect. 3.

2 Some Theoretical background

There is still no theory which tells us how the ensemble error depends on the correlation between the individuals for general loss functions. There has been much progress made in the case of squared error, and much work has also been invested in the very important case of zero-one loss (this is generally the loss function used in classification problems). The problem in this case has turned out to be much harder however. Classification problems can always be re-expressed as a regression problem, so we will take a look at the theoretical error decompositions available for squared loss.

2.1 Bias-Variance Decomposition

Starting with a single predictor, we have the bias-variance decomposition [15]. Assume our training data $T = \{X_i, Y_i\}$ is sampled from an underlying distribution D. We want the average error of our predictor, not an error for one particular sampling, so we consider the expectation $E_T(\varepsilon)$ over all possible training sets T sampled from D.

$$\begin{aligned} E_T(\varepsilon) &= E_T(f-d)^2 \\ &= E_T(f+E_T(f)-E_T(f)-d)^2 \\ &= E_T[(f-E_T(f))^2+(E_T(f)-d)^2+2(f-E_T(f))(E_T(f)-d)] \\ &= (E_T(f)-d)^2+E_T(f-E_T(f))^2 \end{aligned}$$

The first term is the bias, indicating the loss when using the expected value of f to predict d. The second term is variance, and gives us the expected added loss of using one particular f whose average squared deviation from $E_T(f)$ is the variance. When we have an ensemble of N predictors, f becomes $\sum_{i=1}^{N}\omega_i f_i$, a weighted average of the outputs of the predictors. For an unweighted average $\omega_i = \frac{1}{N}$ the expression reduces to:

$$E\left[\left(\frac{1}{N}\sum_i f_i\right)-d\right]^2 = \overline{bias}^2 + \frac{1}{N}\overline{var} + \left(1-\frac{1}{N}\right)\overline{covar}$$

with

$$\overline{bias} = \frac{1}{N}\sum_i (E(f_i) - d)$$

$$\overline{var} = \frac{1}{N}\sum_i E(f_i - E(f_i))^2$$

$$\overline{covar} = \frac{1}{N(N-1)}\sum_i \sum_{j \neq i} E\{(f_i - E(f_i))(f_j - E(f_j))\}$$

Thus we have a decomposition which is dependent on the components of the individual errors, and an interaction term (the covariance). The first term is the ensemble bias, the other two terms together are the ensemble variance. If the predictions of all the ensemble members are independent the interaction term is zero. In this case the variance component of the ensemble error is reduced by a factor of $\frac{1}{N}$ compared to the average variance of the individuals. For dependent (correlated) predictors the variance is reduced by a different factor which has been shown [16] (assuming a common variance for all classifiers) to be:

$$E_{add}^{ave} = E_{add}\left(\frac{1+\delta(N-1)}{N}\right)$$

Where δ is the average correlation between predictions over all pairs of predictors. The implication of this is that if we have predictors whose error is dominated by variance, then by combining we can potentially gain large improvements over any one individual. Larger improvements are gained for smaller δ (lower correlations) and higher N. The difficulty of course is the generation of uncorrelated predictors. We can attempt to generate N uncorrelated predictors while maintaining individual accuracy, but this gets more difficult as N increases. Some methods of trying to do this will be covered in later sections.

Unfortunately, in classification tasks decomposing the error is not so easy. Here the final output of a classifier is one of a few discrete class labels. It is either the right label, or not. There is no concept of `distance'. The labels could be numbers, but could just as easily be strings or anything else, so it is not clear how to define bias and variance. Certainly the standard definitions are of no use as they assume the space of possible output is closed under addition/multiplication/division. This is not true for the classification case even if we use numeric labels (which we can always do). In cases where the output label is based on some continuously varying value with a specific target value, such as classifiers which approximate the posterior probabilities of the classes, progress can be made. As Tumer and Ghosh have shown [17], the squared error of the posterior estimates can be linearly related to the squared error of the classifier decision boundary in approximating the true boundary. This is done by assuming that the posterior probabilities are monotonic in the boundary region, and that the approximated boundary is close to the true boundary. Under these assumptions, the estimated posteriors at the point where they are equal (on the estimated boundary) can be linearly expanded around the true boundary. This results in:

$$b = \frac{\varepsilon_i(x_b) - \varepsilon_j(x_b)}{p'_j(x^*) - p'_i(x^*)}$$

where b is the distance between x_b the estimated boundary, and x^* the true boundary. The denominator is a constant over different training sets and so does not need to be known. In turn b^2 can be shown to be directly proportional to the classification error rate. Thus the framework described above can be used in this case as we have related the misclassification rate to a squared error. Many classifiers (such as the decision tree classifier) cannot approximate the posterior probabilities in this way. Definitions of bias and variance suitable for use with general loss functions do exist, for example [18-20], but there are many different definitions and it is not clear at the moment which, if any, would be of use in guiding the building of a classifier ensemble.

2.2 The Ambiguity Decomposition

Another extremely important result due to Krogh and Vedelsby [21] for combining predictors in the regression context is the ambiguity decomposition:

$$(f_{ens} - d)^2 = \sum_i \omega_i (f_i - d)^2 - \sum_i \omega_i (f_i - f_{ens})^2$$

This gives us a direct decomposition of the ensemble error into the average of the individual errors and a second term containing all interactions, called the ambiguity. It is reached via similar manipulations to the bias-variance decomposition. Because the second term is positive definite, in the case of regression problems we are guaranteed an improvement over the average of the individual errors when combining. It also shows us that, keeping the average error of the predictors in the ensemble constant, we can improve ensemble error simply by increasing the second term, making our predictors spread as widely about the ensemble mean as possible. The negative correlation learning (NCL) method described in the next section is grounded in this decomposition of the error.

3 Decision level combination: Negative Correlation Learning

In the NC method [22], an ensemble of M MLP neural networks are trained in parallel using back-propagation. The error function for each network, in addition to the usual squared error term, contains a penalty term p_i proportional to the correlation of the network predictions with those of all the other networks, making the error for a network:

$$\begin{aligned} E_i &= \frac{1}{N}\sum_{n=1}^{N} E_i(n) \\ &= \frac{1}{N}\sum_{n=1}^{N}\frac{1}{2}\left[f_i(n)-d(n)\right]^2 + \frac{1}{N}\sum_{n=1}^{N}\lambda p_i(n) \end{aligned} \tag{1}$$

where the sum is over the N training examples, f_i is an individual output, and d is the target. For simplicity of notation we will consider the error function at just a single point from now on, removing the necessity for the index n and the sum over points. The penalty term is:

$$p_i = (f_i - f)\sum_{j \neq i}(f_j - f) \tag{2}$$

where f is the ensemble output. This measures and penalises correlations between predictors. In fact, if as in [23] we use the fact that the sum of a set of values around their mean is zero, $\sum_i (f_i - f) = 0$, we can write:

$$p_i = (f_i - f)[-(f_i - f)] = -(f_i - f)^2 \tag{3}$$

which is the ambiguity [21] of the predictor, making the error function

$$E_i = \frac{1}{2}(f_i - d)^2 - \lambda(f_i - f)^2. \tag{4}$$

From the expression for the ambiguity decomposition of the ensemble error with equal weights, we have:

$$\frac{1}{2}(f-d)^2 = \frac{1}{M}\sum_i\left[\frac{1}{2}(f_i-d)^2 - \frac{1}{2}(f_i-f)^2\right] \tag{5}$$

so we can see that if we set $\lambda = \frac{1}{2}$ in ((4)) then the error function we are using to train each network is actually its contribution to the ensemble error as given in the ambiguity decomposition. This is the theoretical grounding of the method; it works because it takes the whole of the contribution of the network to the ensemble error into account, not just the component due to the individual error but that due to the correlations also. It allows us to adjust the relative importance of the two terms, though we will argue later that this freedom should not be exercised as the form of the ambiguity decomposition decides the optimal choice of lambda.

NC has been quite successful in both regression and classification problems [22, 24], and is an attractive method due to its explicit link with the ambiguity decomposition. The success of NC has led to the proposal of some variations of the method using different penalty terms in ((1)). One method in particular, called root quartic negative correlation learning [24], has been shown capable of outperforming NC on some problems. The penalty term in this method is $p_i = \sqrt{\frac{1}{M}\sum_{j=1}^{M}(f_i - f_j)^4}$, however the choice of penalty term has little grounding in theory at the moment, and its success is not well understood.

An elaboration of NC in [25], called constructive neural network ensembles (CNNE) extends the NC framework to allow the number of hidden nodes in each network to be determined by the algorithm. Differing numbers of training epochs may also be used for different networks.

These derivative methods are valuable contributions, however some aspects of the behaviour of the original NC algorithm have not been well understood, particularly the behaviour as λ is varied. This has made it difficult to know without exhaustively trying many different settings what a good setting of λ is likely to be for a particular problem.

When building an NC ensemble, an important choice for good performance is the setting of λ. As we can see from ((4)) a larger (smaller) value of λ corresponds respectively to a greater or lesser emphasis of the ambiguity term resulting in a larger (smaller) emphasis on the spread of the predictions compared to individual accuracy. A greater understanding of the behaviour of NC is needed in order to guide the choice of λ, which we will look at in the next section.

3.1 Setting the λ Parameter

An initial contemplation of the expression for the error in ((4)), may suggest that a natural choice of λ would be $\frac{1}{2}$. With this choice, the sum of the error functions of the individuals is the error of the ensemble as a whole, expressed in its ambiguity decomposition:

$$\begin{aligned} E_{ens} &= \frac{1}{2}(f-d)^2 \\ &= \frac{1}{M}\sum_i\left[\frac{1}{2}(f_i-d)^2 - \frac{1}{2}(f_i-f)^2\right] = \frac{1}{M}\sum_i E_i \end{aligned}$$

and the individual error functions are simply the contribution of each member to the ensemble error. This turns out not to be the case, but we will come back to this thought later. If we seek to minimise these error functions by gradient descent, it is the gradient of the error function and not its actual value that is important as it is this that informs the algorithm. This was noted in Liu's paper [22], where for $\lambda = 1$ it was shown that $\frac{\partial E_i}{\partial f_i} \propto \frac{\partial E_{ens}}{\partial f_i}$, i.e the gradient of an individuals error function w.r.t f_i is proportional to the gradient of the ensemble error w.r.t f_i. The calculation leading to this however relies on an incorrect assumption, as pointed out in [23]. The original calculation assumed that the ensemble output $f = \frac{1}{M}\sum_{i=1}^{M} f_i$ was constant w.r.t any single individual output f_i, which is not true and in the context used could not even be assumed for large M. Taking this correction into account, the calculation proceeds as follows. Starting from the individual error,

$$\begin{aligned} E_i &= \frac{1}{2}(f_i-d)^2 - \lambda(f_i-f)^2 \\ \frac{\partial E_i}{\partial f_i} &= (f_i-d) - 2\lambda\left(1-\frac{\partial f}{\partial f_i}\right)(f_i-f) \\ &= (f_i-d) - 2\lambda\left(1-\frac{1}{M}\right)(f_i-f). \end{aligned}$$

To gain a better understanding of this, we will re-arrange the above to give

$$\frac{\partial E_i}{\partial f_i} = (f-d) + \left[1-2\lambda\left(1-\frac{1}{M}\right)\right](f_i-f). \qquad (6)$$

From here on, we will write the expression in square brackets as

$\theta = \left[1 - 2\lambda\left(1 - \frac{1}{M}\right)\right]$. When performing gradient descent, the first term causes an individual output to move to reduce the ensemble error (regardless of the direction of the individual error), and the second term acts to move the individual output away or towards the ensemble mean depending on the sign of θ. We will look at the effects of this in more detail later.

We also have $\frac{\partial E_{ens}}{\partial f_i} = \frac{1}{M}(f - d)$, so we see from ((6)) that in order to achieve $\frac{\partial E_i}{\partial f_i} \propto \frac{\partial E_{ens}}{\partial f_i}$ we have to choose λ such that $\theta = 0$. This leads to a choice

$$\lambda^* = \frac{1}{2}\left(1 - \frac{1}{M}\right)^{-1}. \qquad (7)$$

With this setting, by performing gradient descent on the individual error functions, we perform gradient descent on the ensemble error, which is a highly desirable situation. At each iteration we are updating the f_i to decrease the ensemble error E_{ens} even if individual accuracy may suffer. It is the ensemble error that is the important quantity, so it is clear that this is a situation we should aim for. For any other choice of λ we are not minimising the ensemble error. Finally, we note that as $M \to \infty$, $\lambda^* \to \frac{1}{2}$, the value our initial intuition would suggest from the form of the ambiguity decomposition. For smaller M, the extra multiplicative term reflects the fact that when adjusting f_i, f will also track the adjustment to some extent. This provides an explanation of the empirical observation in [23] that the optimal setting of λ decays to 0.5 as we add more networks (note our λ is equivalent to γ as used in their paper).

We will also show that the main effect of choosing $\lambda = \lambda^*$ is to maximise the complexity of the algorithm, and show empirically that as in all such cases the increased complexity is only useful if the model underlying the data is complex too. Otherwise we simply get over-fitting. We will therefore argue that λ provides us with an easy way in which to vary the complexity of the network without changing the architecture itself.

We can see this by looking at the effect of changing f_j on the other error functions E_i for $i \neq j$. The error function for individual i is

$$E_i = \frac{1}{2}(f_i - d)^2 - \lambda(f_i - f)^2$$

so we have

$$\frac{\partial E_i}{\partial f_j} = (f_i - d)\delta_{ij} - 2\lambda(f_i - f)\left(\delta_{ij} - \frac{1}{M}\right)$$

where δ_{ij} is the dirac delta function. For $i = j$ we have

$$\frac{\partial E_i}{\partial f_i} = (f_i - d) - 2\lambda(f_i - f)\left(1 - \frac{1}{M}\right). \qquad (8)$$

we can just note (as in the previous section) that $\frac{\partial E}{\partial f_i} = (f - d)$ and choose $\lambda = \lambda^*$ to make $\frac{\partial E_i}{\partial f_i} = \frac{\partial E}{\partial f_i}$. We showed that λ^* is the value giving maximum co-operation and diversification between the individual networks. We can gain insight into what it is we are actually doing in choosing $\lambda = \lambda^*$ by realising that each f_j is present in each error function E_i, so we should re-consider the way we are training the networks and consider the effects of changing f_i on all the error functions when training. For $i \neq j$ we have

$$\frac{\partial E_i}{\partial f_j} = \frac{2\lambda}{M}(f_i - f) \qquad (9)$$

and so for $\lambda = \frac{1}{2}$ we have from (8) and (9)

$$\sum_i \frac{\partial E_i}{\partial f_j} = (f - d) = \left.\frac{\partial E_i}{\partial f_i}\right|_{\lambda = \lambda^*}$$

as of course we must have because by the ambiguity decomposition when $\lambda = \frac{1}{2}$ we have $E = \sum_i E_i$. When we consider training in this alternate way, we see that choosing $\lambda = \frac{1}{2}$ as our intuition would suggest is equivalent to choosing $\lambda = \lambda^*$ with training performed in the traditional way for NC. Thus choosing $\lambda = \lambda^*$ in the individual error functions is equivalent to including the information about the gradient w.r.t f_i of the error functions E_j for $i \neq j$ into the single error function E_i.

When this is done we are simply using the ensemble error E during training, treating the ensemble as a single network as shown in Fig. 1 to be trained by back-propagation. The other extreme, of $\lambda = 0$, corresponds to treating the ensemble as individual networks as shown by the dotted lines in Fig. 1, each trained independently by back-propagation and then combined. This latter case is equivalent to diversifying the individuals simply via different weight initialisations. Thus we can see that λ provides a convenient way of adjusting the complexity of the network between these two extremes without the need for changes in the architecture of the network.

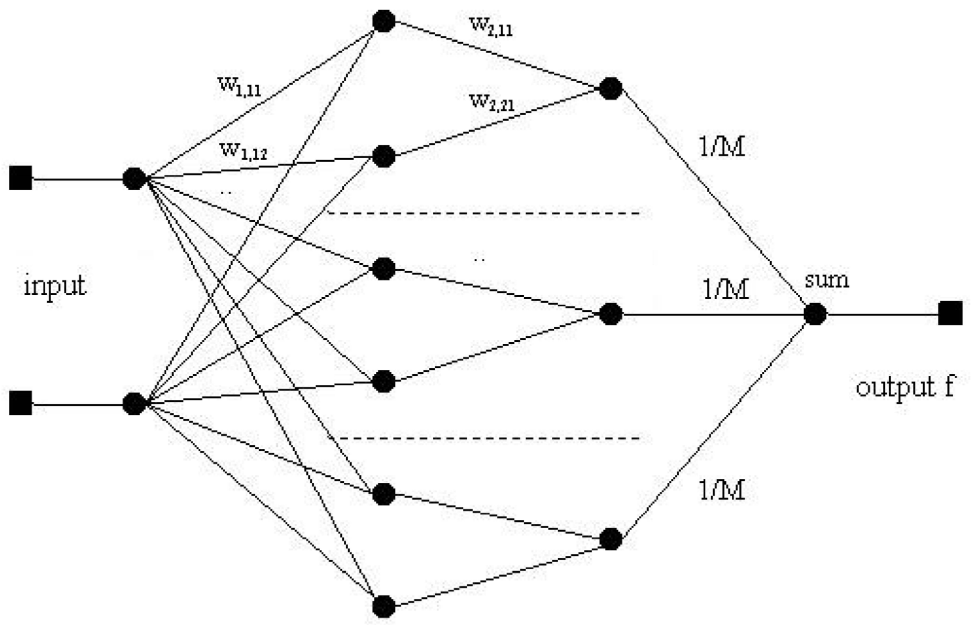

Figure 1. An illustration of the architecture of an NC network.

The weights shown as $\frac{1}{M}$ are fixed. For $\lambda = 0$ the networks are trained as individuals as indicated by the dotted lines. For $\lambda = \lambda^*$ the network is trained as a whole.

Apart from this theoretically optimal setting (in terms of maximising complexity), we can also try to set limits on the value a sensible choice of λ would take. A negative value would defeat the point of NC learning, and of an ensemble method in general as it would encourage the individuals to be very similar, thus removing all advantage from combining. We can also show, by looking at the dynamics of the f_i (i.e how they change over time) during training, that $\lambda \le \lambda^*$ is needed for stability of the algorithm. Further details, which will not be covered here, can be found in [26]

As an illustration of this, we can look at some results from a well-known dataset from the UCI database, the synthetic dataset. The results are shown in Fig. 2.

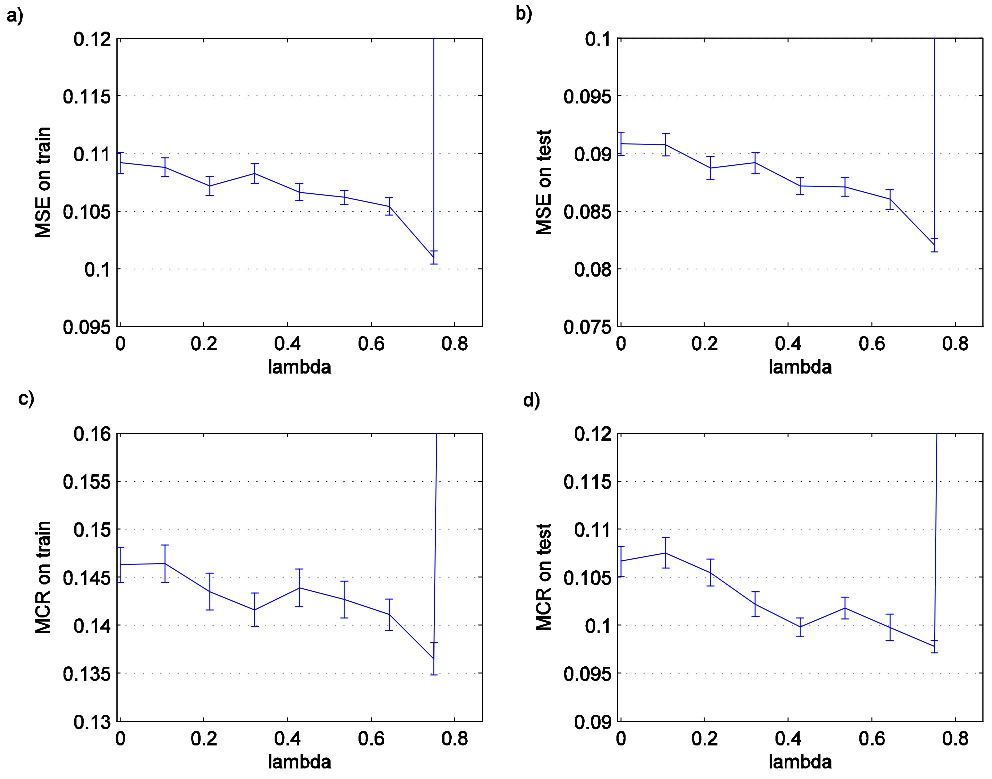

Figure 2. MSE and MCR on both training and testing sets as λ increases on the synthetic dataset

The experimental setup was as follows. An ensemble of three MLP neural networks was used, each with 5 hidden nodes with the tansig transfer function. Linear output nodes were used, and the networks were trained for 5000 epochs at learning rate $\eta = 0.05$. Weights and biases are initialized via the Nguyen-Widrow algorithm. The targets are in one-of-k form. λ^* for this setup is 0.75. A decrease in the generalisation error can be seen up to λ^*, with a large increace beyond that due to instability. The choice λ^* is optimal in the sense that for this value the individual networks will co-operate, and their outputs be de-correlated, to the greatest extent compatible with stability. This co-operative adjustment of the weights allows more complex functions to be fit, leading to improved performance if greater complexity is needed (as in the example above) but also increasing the potential for overfitting if it is not. In some sense the value of λ acts to control the complexity of the ensemble classifier. Thus our choice of λ^* is not optimal in an absolute sense but must be chosen in conjunction with a suitable number of hidden nodes and ensemble size.

This sort of combination method has the disadvantage that it is very hard to interpret the resulting model. A single neural network is not easily interpretable; with multiple networks (or other models more generally) combined at the decision level interpretability is completely lost. The approach to building an ensemble method covered in the next section is one possible way of returning interpretability into an ensemble method.

4 Model level combination: A tree pruning method

The model class we will look at as an illustration of a MLC method are decision trees, as these have an easily decomposable substructure which can be decomposed in a variety of ways. Firstly, a decision tree can always be viewed in terms of its terminal nodes, which gives a decomposition into a number of labeled hyper-boxes. These can equivalently be considered as rules. Secondly, subtrees rooted in given nodes can be considered as separate entities, and can be pruned or grafted onto a similar node in another tree. A tree could also be decomposed into a number of hyper-planes defining the decision surface. This is illustrated in 2-D in fig. 3, where the leaves of the tree can be seen as distinct rectangles, or alternately the lines along which splits have been made can be seen.

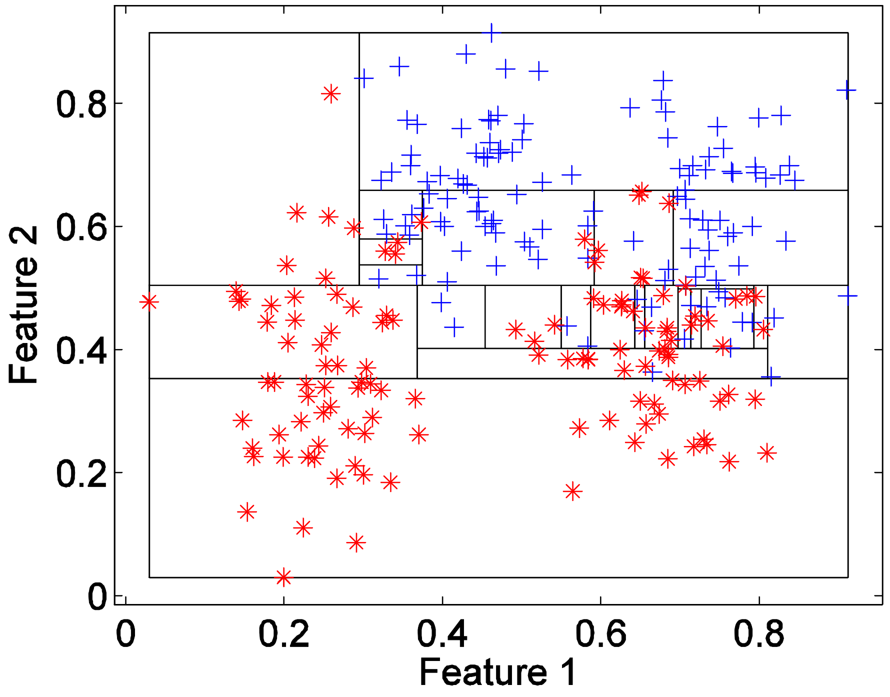

Figure 3. An illustration of the individual leaves into which a tree can be decomposed

The MLC scheme we will present here is an extension of bottom-up pruning which allows the operation of grafting a subtree from one tree onto another tree, in addition to the usual operations of pruning a subtree and leaving a subtree as is. The grafting operation will be described in more detail below. Firstly the method will be described verbally, followed by some psuedocode.

The first stage of the process involves initializing the N trees using any standard decision tree induction method (We used the functions from prtools and from the matlab stats toolbox). The trees can be diversified by any of the usual methods, such as resampling, feature resampling, and use of different bases. As such it is an example of the independent ensemble generation paradigm.

The nodes of the trees are converted to hyperboxes, and are ordered according to their volume (an alternative is to order according to number of points contained within the hyperbox). For each node ν and each tree i, a node $\eta_i(\nu)$ is found such that it is the smallest node in tree i which contains ν. A node can be defined as containing another node if the volume of the first node contains the entire volume of the second node, or if the first node contains all the points of the second node.

Pruning/combination proceeds from the bottom up, in the heuristic sense of smallest to largest hyperboxes, regardless of which tree they belong to, or what level in the tree they are. At a node, there are three possible actions for the algorithm:

• Prune the node to a leaf

• Do nothing, and leave the subtree rooted at the node unchanged

• Replace the subtree rooted at that node with the subtree rooted at one of the nodes $\eta_i(\nu)$

The decision will be based on a pruning criterion which will be calculated for each. Any pruning criterion which can be calculated for a subtree can be used, for example the pessimistic pruning criterion [27]. The pessimistic pruning criterion penalises complexity of a subtree in terms of the number of leaves in the subtree as follows:

let $e^*(\nu) = \frac{1}{2} l(\nu) + e(\nu)$, and $n(\nu)$ be the number of points in node ν, and calculate the pruning criterion for the subtree rooted at ν as follows (dropping the ν's):

$$c = e^* + \sqrt{\frac{e^*(n - e^*)}{n}}$$

the pruning criterion for the leaf is:

$$c = \frac{1}{2} + e_{leaf}$$

Other possibilies are, for example, the criterion defined in [28]. When considering a node with a depth of d and which is the root of a subtree of size n, this criterion is a function of the VC dimension of a subtree with size n, and the VC dimension of the set of paths to a node of depth d. An error bound is also given for this criterion. The error of a subtree on a test set, or any other criterion which can be used to rate the 'quality' of a subtree can also be used. The pessimistic pruning criterion and testset pruning can both be used in our implementation.

The output of the algorithm is a single tree, with the advantages of a single tree classifier - small memory requirements, fast calculation of predictions, and understandable decisions.

4.1 Psuedocode

```
For N trees
    Diversify {via bootstrap resampling, feature resampling, or differing bases};
    Train tree;
end

concatenate nodes of trees;
fix child node indices;
order nodes by size;
fix child node indices;

for each tree i
    for each node n in tree i
        for each other tree j
            find smallest node in j containing {in terms of points or
            volume} node n (itself if i=j);
        end
    end
end

for each node (starting at smallest)
    for each tree
        extract subtree defined on node by tree;
          calculate pruning criterion for subtree {pessimistic pruning criterion or
        testset error};
    end

    calculate criterion for pruning node to leaf;

    if min(criteria) is for leaf
        prune to leaf;
    elseif min(criteria) is subtree already rooted at node
        leave unpruned;
    else
        replace subtree with winning subtree;
    end
end

extract and return final tree;
```

4.2 Discussion

The performance of this method is very little better than a single tree pruned using the same criterion, and certainly much less than decision level combination schemes using the same base classifiers, such as boosting. However it serves as an interesting example of how a number of models may be synthesized into a single model. Future attempts may provide more of a performance boost. It may be that the performance of this method is limited by the quality of the pruning criterion used. The extension to allow multiple trees to be used as the base with which to build the final pruned tree allows a much more diverse range of trees to be built, however the quality of tree that is actually built will depend heavily on the suitability of the pruning criterion used. This criterion was taken from the single-tree pruning literature, future work could be to use a different criterion more geared towards this sort of model level combination scheme.

To illustrate the ideas presented so far and put them in a context of real-world pattern recognition applications, the next section will give an example of an ensemble method applied to a real industrial problem. The chosen problem domain is churn prediction, an important problem in the telecommunications industry.

5 Pattern Recognition and Ensemble Methods in industry: Churn Prediction

In the telecommunications industry, it has been estimated [29] that on average it can cost between 5-8 times more to gain a new customer than it would to keep an existing customer (for example by offering a small incentive). However this incentive is wasted if it is not offered to someone who, in the near future, is likely to churn. The high churn rate prevalent in this area means that fairly small improvements in the accuracy of churn prediction can mean significant cost savings. Thus the problem of predicting customer churn is an important one.

It is a very difficult problem, for a number of reasons. Firstly, people do not always make decisions logically or motivated by any easily definable reason. It may simply be an 'impulse' decision to switch providers. Even those decisions motivated by an easily understandable reason can be very difficult to predict, because people are individuals and react to events in different ways. Secondly, we have only limited data, and many possible reasons for churn will likely leave no imprint in this data, for example competitor's offers, or changes in personal circumstances.

We can expect, however, that in some cases the reason for the decision to churn will leave an imprint in the data prior to the event. This could be in the form of certain patterns of complaints, or repairs, or other warning signs in the pattern of customer behaviour. In these cases we may be able to model and therefore detect situations which will likely result in churn.

Historical data concerning a customer can be considered as a sequence of events (such as repairs, orders, complaints etc) each associated with some features describing that event. An issue which has to be considered when building models for prediction based on historical data is the concept of a relevance 'horizon'. How far back can one go before the historical data becomes obsolete and irrelevant as regards to prediction? This is one question we will attempt to address in this section.

One class of method that has seen wide use and success on sequential data are Hidden Markov Models (HMMs) (see for example [30]). For a review of machine learning methods for sequential data see [31]. This class of models will be described in more detail later. The most basic form of HMM assumes each event in the sequence is described by discrete features, and this type of model has been applied to the churn prediction problem in [32]. However many of the features describing the events (for example time periods) are more naturally expressed as real numbers. They can of course be discretised, but at the cost of losing some information.

A more sophisticated type of HMM model allows for continuous features via a mixture of gaussians approach. This has the advantage of allowing us to retain and use all the information available to us. The majority of this section will look at the application of this sort of HMM to the problem of churn prediction. HMMs can be used to predict the future state probabilities of the system, and can also be used to give probabilities of membership of different classes of sequences. Information gained using HMMs in both of these ways can be used to predict churn.

The focus of this section will be on a combination scheme that while simple provides a significant performance boost. The main reason for using a combination method is to free the resulting model from the details of any specific initialization, as performances can vary quite widely over different initializations for the same model parameters. The combined method should be a little more robust. In terms of the bias-variance decomposition described in section 2 we aim to reduce the variance term using this scheme.

Hidden Markov Models (HMMs) are a form of finite state machine, i.e. the system is assumed to be at any time in one of a finite number of distinct states, and the system may undergo transitions between these states. A sequence of observations is produced over time, and the distribution of these observations depends on the state the system was in at the time the observation was made. They are highly suitable for problems in which the data is essentially sequential in nature, and have been used widely in the areas of speech and handwriting recognition, and in some areas of medical research [33, 34]. We will first describe this class of models in an informal way, and give a more formal definition afterwards. The simplest form of HMM assumes discrete outputs. For each event only certain discrete outputs can be produced, with the probability of each output depending only on the hidden state of the system. As the data we have consists of four continuous features and one categorical feature, it is more naturally represented in a continuous space so a more flexible model called Mixture of Gaussians HMM (or MGHMM) which allows for this is more useful. The basic assumptions of the model are as follows:

• At each time-step t of the sequence the system in question is in one of a limited number of states $q_t \in \{q_1,...,q_Q\}$. It cannot be observed directly which state the system is in.

• The state the system is in at time t depends only on the state at time $t-1$. There are extensions to the HMMs which allow dependence on the state at other times too, but we will not consider these.

• Observed are a time-ordered sequence of feature vectors $\mathbf{x}_t$, one for each timestep.

• The distribution of the values of the observed output features depends only on the state of the system at that time.

Each state is associated with a mixture of M gaussians, and the feature vector output is distributed over the continuous feature space according to the mixture of the state at that time-step.

A little more formally, define Q states, and a matrix $\mathbf{A}$ of transition probabilities a_{ij} giving the probability of a transition from state i to state j, with $\sum_j a_{ij} = 1 \forall i$.

Each state has associated with it a mixture of M gaussians, of the same dimensionality as the feature space. These gaussians have means and covariance matrices μ_{qm} and $\mathbf{\Sigma_{qm}}$.

A mixture matrix $\mathbf{B}$ gives the probability b_{qm}, with $\sum_m b_{qm} = 1$ for all $q = 1,...,Q$, that given the system is in the $q'th$ state the observed feature vector will be generated from the gaussian with parameters μ_{qm} and $\mathbf{\Sigma}_{qm}$.

HMM's of this form will be generated from the customer data, trained iteratively via the usual EM (expectation maximisation) algorithm [35]. In order to have a dataset suitable for the application of this method, we have taken raw data about customer events and, for each unique customer, have constructed a sequence of time-ordered events. Each event is described by 5 features. One of these features is more naturally categorical; it denotes the event as one of four different types one of these being churn. These categories have been expressed numerically for use in the MGHMM, whereas the other features are naturally real-valued.

When constructing these sequences for the churn prediction problem, we can take a variety of approaches depending on how much historical information we include. One intuitive way of doing it is to extract every subsequence (starting at t=1) of these sequences of greater length than three events. So, for each sequence of events $S = \{s_1, ..., s_T\}$, we will extract the $T-2$ sequences $S_j = \{s_1, ..., s_{j+2}\}$ for $j = 1, ..., T-2$. This form of training data could be considered appropriate, because over a customers lifetime it would be exactly these sequences that would become available. It is more likely, however, that only the more recent events are really relevant in predicting future events. In order to discover the timeframe over which it is best to take events when constructing a customer history, we have constructed training sets in which only the most recent N events are considered, for $N = 3:10$. When necessary, a subscript will denote the lengths of sequences allowed, so as an example TR_{any} or TR_N for $N = 3:10$.

The models constructed and trained as described above are highly sensitive to the initialization of the model. One way of reducing this dependency on a specific initialization is to train a number of models using different initializations, and then combine their predictions. We have done this in a relatively simple, rank based manner. For a given individual set of models, after calculating for each sequence the probability of churn, the sequences are ranked in order of descending probability. For each sequence, then, we have the ranks $r = r_1, ..., r_N$ where N is the number of models to be combined. We define a function to map this vector of values onto the real numbers, and rank them again according to this new value. We tried a variety of simple functions, and settled on an inverse square function $s = \sum_i \frac{1}{r_i^2}$ though performance is not too sensitive to the form of this function so long as it increaces sufficiently quickly for small r_i.

We then take the top N sequences as our predictions. Here we have a trade-off to decide between. A larger N means we detect more of the actual churn events, but at a higher error rate. This trade-off is summed up in Fig. 4. For example, if we choose to take the top 0.4% as churn predictions (the percentage of sequences which are churn in the training set), we can expect a correct identification rate of just over 0.3. However if we choose to take the top 0.8% as predictions, we can expect to predict more churn events correctly (about 33% more) but at the lower recognition rate of 0.2.

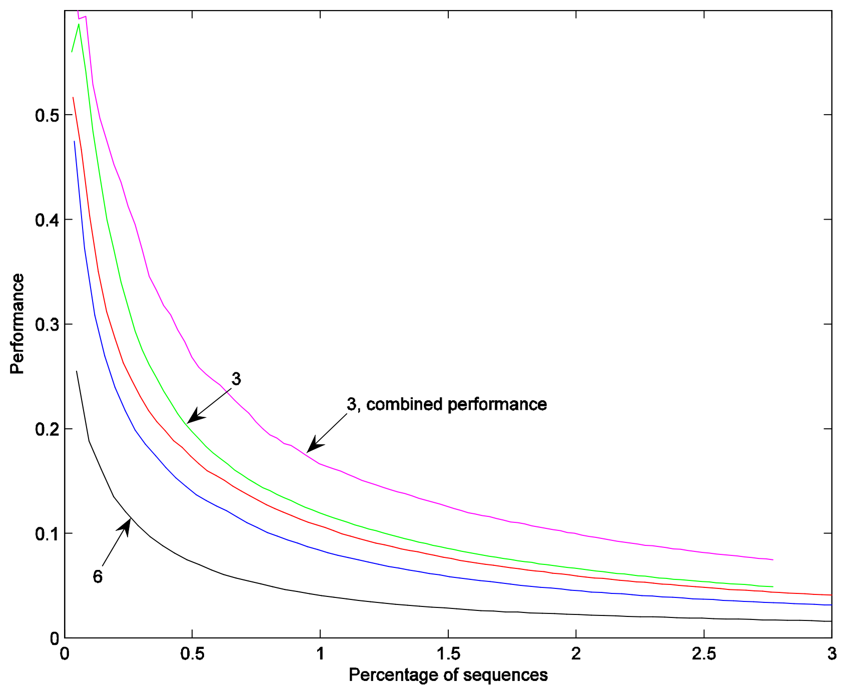

Figure 4. The top line shows the combined performance using training sequences of length 3. Average performance of individual models plotted against percentage of sequences taken as predictions, for training sequences of length 3,4,5, and 6 are the lower plots (from top to bottom line).

As can be seen in Fig. 4, the combination method improves performance quite significantly. This serves to illustrate that even quite simple combination methods can provide a large benefit in real world applications. The challenge is to find the best ways of building these ensemble methods, and to find a theoretical framework to take some of the guesswork out of which methods will work well, and which will not, for a given problem. The length of sequence used in the histories can also be seen to have a large impact on performance, with shorter sequences of only the most recent historical events resulting in much better performance. This illustrates a point that while even simple combination methods can provide quite large performance boosts, it is still extremely important to choose the data correctly and represent it in the most suitable way.

6 Summary

In this chapter we have looked at some of the different approaches that can be taken when building an ensemble method. Each approach has its own distinct advantages and disadvantages. Decision level combination schemes are often independent of the model class used in generating the individuals (good examples of this would be bagging [6] and boosting [7]), or can be used to combine individuals of disparate model types [36]. However the interpretability is lost in the number of different decisions which are combined, and the fact that many individuals must be stored and must give a decision in order for the ensemble decision to be reached can make these methods computationally expensive. Interpretability is especially important in industrial applications as any business decisions must be justified and explained. Model level combination provides a way to retain some interpretability by combining a number of models into a single model of the same type. A single model has the advantage that it can be much more easily interpreted, and is likely to need less computational resources to store and run. However, the combination method is highly dependent on the specific structure of the model class used, so combination methods of this type are usually limited to a single model class that they were designed for.

In the final section we saw that even simple combination methods hold the potential to increase performance significantly. We also saw that the traditionally important concept of data representation is still highly relevant in the building of an ensemble method with both individual and combined performances being increased significantly by the use of only the most recent, relevant data.

References

1. Bousquet, O., S. Boucheron, and G. Lugosi, *Introduction to Statistical Learning Theory*, in *Advanced Lectures on Machine Learning*, U.v.L. Bousquet O. and G. Rätsch, Editors. 2004, Springer: Heidelberg, Germany. p. 169-207.

2. Brown, G., et al., *Diversity creation methods: A survey and categorisation.* Journal of Information Fusion, 2005. **6**(1).

3. Kuncheva, L.I., *Combining Pattern Classifiers: Methods and Algorithms*. 2004: Wiley-Interscience.

4. Opitz, D. and R. Maclin, *Popular Ensemble Methods: An Empirical Study.* Journal of Artificial Intelligence Research, 1999. **11**: p. 169-198.

5. Hansen, J.V., *Combining Predictors: Comparison of Five Meta Machine Learning Methods.* Information Sciences, 1999. **119**(1-2): p. 91-105.

6. Breiman, L., *Bagging Predictors.* Machine Learning, 1996. **24**(2): p. 123-140.

7. Freund, Y. and R.E. Schapire. *Experiments with a new boosting algorithm*. in *Proceedings of the 13th International Conference on Machine Learning*. 1996: Morgan Kaufmann.

8. Breiman, L., *Random Forests.* Machine Learning, 2001. **45**(1): p. 5-32.

9. Hall, M., *Combining Particles and Waves for Fluid Animation*. 1998. p. 73.

10. Domingos, P., *Knowledge discovery via multiple models*. 1998.
11. Gert Lanckriet, N.C., Peter Bartlett, Laurent El Ghaoui, Michael Jordan,, *Learning the Kernel Matrix with Semi-Definite Programming*.
12. Lee, S.W., S. Verzakov, and R.P. Duin. *Kernel Combination Versus Classifier Combination*. in *Proc. 7th Int. WOrkshop, MCS 2007*. 2007.
13. Gabrys, B., *Learning Hybrid Neuro-Fuzzy Classifier Models From Data: To Combine or not ot Combine?* Fuzzy Sets and Systems, 2004. **147**: p. 39-56.
14. Utans, J., *Weight averaging for neural networks and local resampling schemes*. 1996.
15. Geman, S., E. Bienenstock, and R. Doursat, *Neural Networks and the Bias/Variance Dilemma.* Neural Computation, 1992. **4**(1): p. 1-58.
16. Tumer, K. and J. Ghosh, *Error Correlation and Error Reduction in Ensemble Classifiers.* Connection Science, 1996. **8**(3-4): p. 385-403.
17. Tumer, K. and J. Ghosh, *Analysis of decision boundaries in linearly combined neural classifiers.* Pattern Recognition, 1996. **29**(2): p. 341-348.
18. Kohavi, R. and D.H. Wolpert. *Bias Plus Variance Decomposition for Zero-One Loss Functions*. in *Machine Learning: Proceedings of the Thirteenth International Conference*. 1996: Morgan Kaufmann.
19. Breiman, L., *Bias, Variance, and Arcing Classifiers.* Breiman,L. (1996) Bias, Variance, and Arcing Classifiers, Technical Report 460, Statistics Department, University of California, 1996.
20. Domingos, P. *A Unified Bias-Variance Decomposition and its Applications*. in *Proc. 17th International Conf. on Machine Learning*. 2000: Morgan Kaufmann, San Francisco, CA.
21. Krogh, A. and J. Vedelsby, *Neural Network Ensembles, Cross Validation, and Active Learning.* NIPS, 1995. **7**: p. 231-238.
22. Liu, Y. and X. Yao, *Ensemble learning via negative correlation.* Neural Networks, 1999. **12**: p. 1399-1404.
23. Brown, G. and J.L. Wyatt. *The Use of the Ambiguity Decomposition in Neural Network Ensemble Learning Methods*. in *20th International Conference on Machine Learning (ICML'03)*. 2003. Washington DC, USA.
24. McKay, R. and H. Abbass. *Analyzing Anticorrelation in Ensemble Learning*. in *Proceedings of 2001 Conference on Artificial Neural Networks and Expert Systems*. 2001. Otago, New Zealand.
25. Islam, M.M., X. Yao, and K. Murase, *A constructive algorithm for training cooperative neural network ensembles.* IEEE Transactions on Neural Networks, 2003. **14**(4): p. 820-834.
26. Eastwood, M. and B. Gabrys, *The Dynamics of Negative Correlation Learning.* Journal of VLSI Signal Processing, 2007. **49**: p. 251-263.
27. Quinlan, J.R., *Simplifying Decision Trees*, in *Knowledge Acquisition for Knowledge-Based Systems*, B. Gaines and J. Boose, Editors. 1988, Academic Press: London. p. 239-252.
28. Kearns, M. and Y. Mansour. *A fast, bottom-up decision tree pruning algorithm with near-optimal generalization*. in *Proc. 15th International Conf. on Machine Learning*. 1998:

Morgan Kaufmann, San Francisco, CA.
29. Yan, L., et al., *Improving prediction of customer behaviour in non-stationary environments.* Proc. of Int. Joint Conf. on Neural Networks, 2001: p. 2258-2263.
30. Duda, R., P. Hart, and D. Stork, *Pattern Classification*. 2001: John Wiley and Sons.
31. Dietterich, T.G. *Machine Learning for Sequential Data: A Review*. in *Proceedings of the Joint IAPR International Workshop on Structural, Syntactic, and Statistical Pattern Recognition*. 2002. London, UK: Springer-Verlag.
32. Ruta, D., D. Nauck, and B. Azvine. *K Nearest Sequence Method and Its Application to Churn Prediction.* in *IDEAL*. 2006.
33. Jaakkola, T., M. Diekhans, and D. Haussler, *A discriminative framework for detecting remote protein homologies*. 1998.
34. Tamura, M., et al., *Adaptation of pitch and spectrum for HMM-based speech synthesis using MLLR*. 2001.
35. Bilmes, J., *A Gentle Tutorial on the EM Algorithm and its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models*. 1997.
36. Ruta, D. and B. Gabrys, *Classifier Selection for Majority Voting.* Information Fusion, 2005. **6**: p. 63-81.