Chapter 4

Variance reduction techniques

Bias and variance are two sources of error reacting in opposite way to the model complexity. There thus exists a tradeoff between these two sources of error. After a short introduction, we review in Section 4.2 the "classical" way this tradeoff is handled in the context of different learning algorithms. Section 4.3 then considers a class of variance reduction techniques based on the aggregation of several models (also called "ensemble techniques") which have been proposed in the recent years, and which are potentially able to reduce variance without increasing bias. The last section provides examples of such techniques and, in particular, introduces our "dual perturb and combine" algorithm which we further develop in the context of decision tree induction in chapter 8.

Note that in this chapter we provide no empirical comparisons of the variance reduction techniques, our aim being essentially to provide a broad and synthetic view of the methods. Many of these methods will be studied more in detail later on in the context of decision tree induction.

4.1 Introduction

Automatic learning methods like decision/regression trees or neural networks have the nice property of providing a very flexible hypothesis space. They have the universal approximation property which means that they are able to model with arbitrary accuracy any practically relevant input/output function, however complex, provided that sufficiently complex tree or neural network structures are used. The main problem of these automatic learning methods is thus not bias (which, in fact, can be made arbitrarily small) but variance which arises when such complex models are needed and must be chosen on the basis of finite samples. The increase in interest in variance reduction has been recently strengthened by two things. First, the success of automatic learning techniques makes people want to apply them in more and more complex domains, e.g. in bioinformatics, cryptography, or, for temporal data modeling. Even if traditional algorithms are in principle applicable in these domains, they are not really adapted to these new kinds of data (see the third part of this thesis for time series data) and even more complex model spaces have to be designed. Second, at the same time, the very fast increase in computer power makes it possible to search larger and larger hypothesis spaces. These new improvements in representational power of the learning algorithms will be useless if they are not combined with some variance reduction techniques.

Variance reduction techniques can be classified into two main families:

- techniques that control the bias/variance tradeoff in the context of one particular learning algorithm. These methods do not change the bias/variance configuration

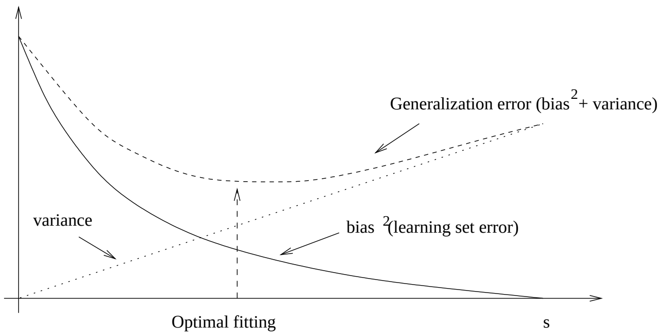

Figure 4.1: Bias and variance with respect to the learning set fitting (neglecting residual error).

of the learning algorithm but rather adapt the optimization part of the algorithm in order to find the best tradeoff between bias and variance.

- techniques that change the bias/variance configuration of a learning algorithm. These are general techniques which can be applied to any learning algorithm; all of them work by averaging several (unstable) predictions.

The next two sections will describe both classes of techniques. The fourth section will discuss among other things Bayesian learning paradigms and relate them to variance reduction techniques.

4.2 Controlling the bias-variance tradeoff of a method

4.2.1 General framework

In every method which uses a learning set to select one model, there is a need to control the bias/variance tradeoff to avoid overfitting. The more you give means to an algorithm to fit the learning set, the more it will suffer from variance. Figure 4.1 shows the typical evolution of the bias and variance of a machine learning algorithm with respect to a parameter s measuring the means given to the learning algorithm to fit the learning set (e.g. number of decision tree nodes, number of weights for neural networks). Bias generally is a monotonic decreasing function of s and variance on the other hand is a monotonic increasing function of the same parameter. The error thus reaches a minimum for some given value s^* of s. Since, as we commented above, variance decreases with sample size and bias is often not at all dependent of sample size, the best value of s should be an increasing function of the sample size. Also since bias is always dependent on the problem (by means of the Bayes model), the best value s^* is also problem dependent.

In practice, there are three ways of controlling a parameter like s to avoid overfitting:

1. First, a reasonable value of s can be fixed according to some prior knowledge (a thumb rule) and the best model selected on the learning set taking into account this value. For example, in the neural network literature one can read rules like "the number of weights of a neural network should not exceed 10% of the learning sample size N".

   Note that, even if this procedure is fast as it needs only one optimization, the best value of the parameter is most of the time unknown a priori and intrinsically dependent on the problem. Thus, in practice, this approach turns out to be suboptimal.

2. The second approach aims at identifying the optimal value of s on the basis of the available sample of objects. It works as follows :

   For increasing values of s:

   - use the learning set to find a model taking into account the current value of s;
   - evaluate the generalization error of this model and stop the iteration when this error is not decreasing anymore.

   As bias and variance are difficult to estimate from one learning set, a generalization error estimate is used to select the best model. To evaluate the generalization error, we can use either cross-validation or an independent validation set (see section 2.5). Apart from the increase in computational burden, the main drawback of this technique is that it can be trapped in a local minimum of the generalization error. This leads to the third algorithm.

3. For increasing values of $s < s_{max}$:

   - use the learning set to find a model taking into account the current value of s;
   - evaluate the generalization error of this model;

   Among the resulting sequence of models, select the one which yields the best generalization error. When there is no natural maximum value s_{max}, iterations can be stopped either when the learning set error vanishes or at least when this error stops decreasing. The problem of local minimum is thus avoided (or at least mitigated) but at the price of higher computation times.

Note that the computation of the optimal model on the learning set for increasing values of s can be very complex in some cases. However, sometimes, it is possible to compute efficiently the optimal model for one value of s from the previous model in the sequence. If this is impossible, one solution is to design heuristics that compute an approximation of this model from the previous one.

4.2.2 Examples in the context of different learning algorithms

In the context of a particular machine learning algorithm there are often many possible para-meters s, each one regulating a different bias/variance tradeoff. One general way to control the goodness of fit to the learning set is to control the size of the hypothesis spaces. The parameter s thus defines a sequence of nested hypothesis spaces $\mathcal{H}_1 \subset \mathcal{H}_2 \subset \ldots \subset \mathcal{H}_s \subset \ldots$ which are all included in the original space $\mathcal{H}$. For a fixed value of s, learning consists in searching $\mathcal{H}_s$ for the best model on the learning set. Almost all parameter choices discussed below comply with this framework of nested hypothesis spaces. Usually, to limit the size of the hypothesis space, a complexity measure $C(h)$ of models is defined and the subspace $\mathcal{H}_s$ is restricted to models whose complexity is less than a threshold s, i.e. $\mathcal{H}_s = \{h \in \mathcal{H} | C(h) \leq s\}$. From the theory of Lagrange multipliers, the sequence of models obtained by minimizing the error $\hat{Err}(h, ls)$ in $\mathcal{H}_s$ for increasing values of s is equivalent to the sequence of models obtained by minimizing $\hat{Err}(h, ls) + \lambda \cdot C(h)$ for decreasing values of λ. This second formulation often makes the problem easier to solve.

Variance controlling techniques are reviewed here in the context of the five learning algorithms described in the first chapter.

Linear regression

Even if this algorithm generally does not suffer from very high variance, the first variance reduction techniques were devoted to it. Ridge regression, for example, consists in bounding the norm of the weight vector or equivalently in adding a penalty term to the learning set error which is proportional to the norm of this vector. More precisely, the full hypothesis space $\mathcal{H}$ contains all linear models and the hypothesis subspace $\mathcal{H}_s$ for some s contains only those functions which satisfy

$$h(\underline{a}) = w_0 + \sum_{i=1}^{n} w_i \cdot a_i \text{ with } \sum_{i=0}^{n} w_i^2 < s. \tag{4.1}$$

Note that, as we have argued above, minimizing the empirical risk

$$\hat{Err}(h, ls), \tag{4.2}$$

in $\mathcal{H}_s$, is equivalent to minimizing in $\mathcal{H}$ the criterion

$$\hat{Err}(h, ls) + \lambda(s) \cdot \sum_{i=0}^{n} w_i^2, \tag{4.3}$$

for some suitably chosen decreasing function $\lambda(s)$. Once λ is fixed, the minimization of (4.3) can be carried out solving a set of linear equations derived by slightly modifying the normal equations of linear least squares.

Irrespectively of variance reduction, another advantage of ridge regression is that even small values of λ stabilize the analytical solution of equation (4.3) when the solution for $\lambda = 0$ may be ill-conditioned (e.g. in the case of singularities resulting from quasi-linear relationships between attributes).

Neural networks

There exist many parameters controlling the bias/variance tradeoff in neural networks. First, the complexity of the neural networks (in terms of the number of neurons and hidden layers) is certainly a valid choice for s. This is a typical case where heuristics have to be designed to select the right complexity. Indeed, as optimization of neural networks can be very time consuming, exhaustive search on structure is essentially intractable. Growing (starting with a simple network and iteratively adding new units to it) and pruning strategies (starting from complex networks and gradually removing connections or units) both have been explored.

Another obvious parameter is the number of iterations of the optimization algorithm. If we let this algorithm drive the resubstitution error to zero, the model will certainly overfit the data and hence suffer high variance. So, stopping the optimization earlier could reduce the generalization error (the combination of this parameter and one of the two ways of finding the optimal value of s is called early stopping in the neural network community).

The equivalent of ridge regression for neural networks is called "weight decay" in the neural network literature. Like (4.3), weight decay consists in adding to the error function a term which penalizes large weights. The parameter s is the inverse of the constant multiplying the norm of the weights and is called the decay constant. Weight decay is based on the idea that very large weights in a neural network can cause the output function to be very irregular and thus to fit the noise in the data. Techniques penalizing irregular output functions in favor of smoother ones are called regularization techniques and weight decay is only one of them. Another alternative is to use a penalty term involving various derivatives of the output function.

Decision/regression trees

The obvious parameter regulating the bias/variance tradeoff is the complexity of the tree measured for example by its number of terminal nodes. In practice, to control the complexity, a quality measure $Q(\mathcal{T}, ls)$ for a tree $\mathcal{T}$ is defined by:

$$Q(\mathcal{T}, ls) = R(\mathcal{T}, ls) - \frac{1}{s}.C(\mathcal{T}), \tag{4.4}$$

where $R(\mathcal{T}, ls)$ is some measure of the reliability of the tree on the learning set (resubstitution error and information quantity have been proposed in the literature). Small values of s favor simple trees while large values of s favor more complex ones. Fixing a priori the value of s and optimizing the quality is called stop splitting (or pre-pruning). Optimization of s following the third procedure of subsection 4.2.1 is called pruning (or post-pruning). Note that these ideas will be explained in detail in Chapter 6 of the second part of this thesis, focusing on variance reduction in decision tree induction.

K-Nearest Neighbors

The only parameter in K-nearest neighbors is precisely K the number of neighbors which are taken into account when doing a prediction. A parameter s satisfying the conditions is $N - K$ where N is the size of the learning set. When $s = 0$, the bias is maximum and the variance minimum. Indeed, the prediction is constant allover the input space but may still change from one learning set to another. On the other hand, variance is maximum and bias is minimum when K is equal to 1. Note that in the context of K-NN, leave-one-out is an obvious candidate for estimation of the generalization error during optimization of K. Indeed, the learning phase is very efficient and one leave-one-out test for some value of K can take into account the distances between objects which have already been computed for testing previous values of K.

Naive Bayes

In the Naive Bayes method, a bias/variance tradeoff appears when it comes to learn a model for probability distributions $P(A_i|C)$. If the model for this distribution is too complex, it will not generalize. On the other hand, if the model is too simple, bias will be high. For example, if we use histograms to model such a distribution, the number of cells of the histogram (bins) is a parameter which controls a bias/variance tradeoff. If this number is too large, some subregions will be empty of any cases from the learning set and thus the histogram will not generalize. On the other hand, too few subregions will hide some information contained in the learning set.

General techniques

There exist also some general model independent techniques to regulate some bias/variance tradeoff. For example, attribute selection is a way to reduce the variance. Indeed, if there are some useless or redundant attributes, they will give some further means to the learning algorithm to overfit the data. The number of attributes is thus a valid choice of parameter s and a lot of work has been done in this field (see for example [KJ97] for a model-independent wrapper technique). Note that tree based methods intrinsically are doing some attribute selection and actually can be used as an attribute selection method for other learning algorithms.

Another general variance reduction technique is jitter ([HK92, Bis95]). Jitter consists in generating new pseudo-samples cases obtained by adding a small amount of random variations (noise) to copies of the input attribute vectors of existing learning cases while leaving the output attribute unchanged. Assuming the goal functions are mostly smooth and injecting a small amount of noise, expanding the learning set in such a way will not change very strongly the goal function (and hence bias) but will certainly force the learning algorithm to build smoother models (and hence decrease variance). In the context of linear regression, it can be shown that jitter with white Gaussian noise is strictly equivalent to ridge regression. In the context of neural networks, although not strictly equivalent, jitter and weight decay have been shown to produce very similar results.

4.2.3 Related approaches for model selection

Several general frameworks have been proposed in the machine learning literature in order to formalize the compromise between complexity of model structure and empirical risk, which are more or less strongly related to the approaches discussed above. Below, we briefly discuss three of them, namely the Bayesian approach to model selection, the Minimum Description Length (MDL) principle, and the structural risk minimization approach. Rather than providing a detailed theoretical development of these approaches, our aim is to give an intuitive description of how they work and how they are related to the other techniques presented in this chapter.

Bayesian model selection

We have seen that the classical approach to machine learning (inherited from classical statistics) can be summarized as follows: define an error measure and a set of candidate models, then apply the empirical risk minimization principle in order to select one model on the basis of the learning sample. Note that under some additional hypotheses on the underlying data generation process, most of the approaches following this principle can be viewed as a *maximum likelihood* approach. In other words, the model they select is the model (within the set of candidate models) which maximizes the likelihood of the learning sample. Under some restrictive conditions on the hypothesis space (essentially, finite VC-dimesion) one can show that this approach is optimal asymptotically, i.e. when the sample size tends towards infinity. On the other hand, as we have argued in the preceding sections, when sample size is small or moderate compared to the size of the hypothesis space, this approach is essentially suboptimal.

In the Bayesian approach, rather than considering the model as a deterministic object which has to be identified from the data, we consider the model as a random variable and the observable samples as random variables which have a conditional distribution given the model. Then, the prior distribution over the hypothesis space represents our prior beliefs about the possible explanations of a sample, and these beliefs can be updated once a learning sample has been observed, yielding a posterior distribution. In practice, to apply the approach we thus need to specify two distributions:

- a prior distribution, $P(H)$, defined on the hypothesis space $\mathcal{H}$ which measures how plausible are a priori the models of the hypothesis space (without any knowledge of the data);

- a conditional distribution $P(LS|H)$ which measures the likelihood (also called the evidence) of a learning sample under the assumption that it is generated according to the model H.

From these two distributions, Bayes theorem allows us to compute the posterior distribution of the model given the data:

$$P(H|LS) = \frac{P(LS|H)P(H)}{P(LS)} \tag{4.5}$$

$$\propto P(LS|H)P(H), \tag{4.6}$$

where $P(LS) = \sum_H P(LS|H)P(H)$ is the probability of observing a particular sample, and can be viewed here as a normalization constant which ensures that the distribution $P(H|LS)$ over $\mathcal{H}$ sums to 1.

From a technical point of view, the Bayesian approach to machine learning essentially consists in computing the posterior distribution $P(H|ls)$ from the priors and from the observed learning sample ls. The output of the approach is thus a *distribution* over $\mathcal{H}$, rather than a particular model $H(ls)$. Thus, unlike classical techniques, the Bayesian approach gives information about the uncertainty which remains about the model when the data is known. The posterior probability distribution gives us a ranking of the plausible models of the hypothesis space for our data and may be used for several inference tasks: parameter estimation, prediction, and model selection.

As concerns the use of $P(H|LS)$ for model selection, this approach thus constitutes an alternative to the other automatic learning methods described earlier. The most direct way to select a model from data is to select the most probable one according to the posterior probability distribution:

$$f_{LS} = \arg\max_{h\in\mathcal{H}} P(H = h|LS) = \arg\max_{h\in\mathcal{H}} P(LS|H = h)P(H = h). \tag{4.7}$$

This formulation of model selection allows to take into account the bias/variance compromise in two ways.

First, it is possible to choose the prior distribution of models in order to penalize too complex models. For example, under some hypotheses, it is possible to give a Bayesian interpretation of the weigth decay approach for learning neural networks (see [CM98]). Indeed, let us assume that the hypothesis space is composed of all possible parameterizations of a neural network of a given complexity and the goal of learning is to find the most probable choice of parameters given the learning sample. A model is then defined by the vector of parameters $\underline{W}$ and the Bayesian approach aims at computing $P(\underline{W}|LS)$. In the previous section, we have seen that large weights imply irregular functions and hence should be avoided. A way to penalize large weights is to assume that small values of the weights are a priori more probable, e.g. taking:

$$P(\underline{W}) = \frac{1}{Z_\beta}\exp(-\frac{\beta}{2}\sum_{i=1}^{d} W_i^2), \tag{4.8}$$

where d is the number of parameters of the neural network, β is some parameter of the distribution, and Z_β is a normalization constant. On the other hand, the likelihood of the learning sample ls (of size N) for some value of the parameter w is given by:

$$P(LS = ls|\underline{W} = w) = \prod_{i=1}^{N} P(\underline{A} = \underline{a}_i, Y = y_i|\underline{W} = \underline{w}) \tag{4.9}$$

$$= \prod_{i=1}^{N} P(Y = y_i|\underline{A} = \underline{a}_i, \underline{W} = \underline{w}) \prod_{i=1}^{N} P(\underline{A} = \underline{a}_i), \tag{4.10}$$

since parameters $\underline{W}$ does not predict input values. Assuming that the output values are generated according to

$$Y = f(\underline{A}; \underline{w}_o) + \varepsilon,$$

where $f(\underline{A}; \underline{w}_0)$ is the prediction of our neural network with some parameters $\underline{w}_0$ and ε is a Gaussian random variable, we have:

$$P(Y = y_i | \underline{A} = \underline{a}_i, \underline{W} = \underline{w}) = \frac{1}{Z_\alpha} \exp(-\frac{\alpha}{2}(y_i - f(\underline{a}_i; \underline{w}))^2), \tag{4.11}$$

where α depends on the variance of ε and Z_α is some normalization constant. The most probable choice of parameters is the one which maximizes $P(\underline{W}|LS) \propto P(LS|\underline{W})P(\underline{W})$ and hence which minimizes $-log(P(\underline{W}|LS))$. Using equations (4.8), (4.10), and (4.11), one can show that:

$$-log(P(\underline{W} = \underline{w}|LS = ls)) = \frac{\alpha}{2}\sum_{i=1}^{N}(y_i - f(\underline{a}_i; \underline{W}))^2 + \frac{\beta}{2}\sum_{i=1}^{d} W_i^2 + C, \tag{4.12}$$

where C is some constant which does not depend on the parameters $\underline{w}$. Hence, minimizing $P(\underline{W} = \underline{w}|LS = ls)$ according to $\underline{w}$ is equivalent to minimizing an expression like (4.3) and so, the weight decay approach of the previous section has been given a Bayesian interpretation. In the context of decision tree induction, the "growing+pruning" strategy can be viewed as the selection of the a posteriori most probable model in a Bayesian framework. Indeed, the quality measure (4.4) using information quantity may be derived from a posterior distribution of decision trees where prior distribution penalizes large trees, see [Weh93].

Even without adopting a specific choice of prior distribution to penalize complex models, the Bayesian approach also encompasses some built-in prior preference for simpler models and hence implicitly penalizes complex models (see [Mac92]). More precisely, let us consider a hypothesis space of models which are defined by two sets of variables: a structure S (for example, for the hypothesis space of single hidden layer neural networks, the structure would denote the number of nodes in the hidden layer) and the parameters characterizing a model of this structure $\underline{W}$ (the exact value of the weights of the neural network). Note that the prior distribution over the hypothesis space can thus be written in the form

$$P(H) = P(S, \underline{W}) = P(S)P(\underline{W}|S),$$

and the Bayesian approach may be used in order to determine the most probable structure given the observed data by computing:

$$P(S|LS) \propto P(LS|S)P(S). \tag{4.13}$$

Depending on the assumptions made on the prior distribution $P(H)$ this criterion will favor more or less complex models. Note that one can argue that a natural assumption is to consider that $P(S)$ is a uniform distribution (so called uninformative prior distribution) over the range of structures covered by $\mathcal{H}$. As concerns $P(\underline{W}|S)$ a similar argument would then also lead to a uniform distribution. However, since the dimensionality of $\underline{W}$ is an increasing function of the complexity of the structure, $P(\underline{W}|S)$, as well as $P(S, \underline{W})$, will necessarily be a decreasing function of the model complexity. On the other hand, the assumption that all the structures are a priori equiprobable, implies that the a posteriori most probable structure is the one which maximizes the evidence $P(LS|S)$ which may be written:

$$P(LS|S) = \int P(LS|S, \underline{W})dP(\underline{W}|S), \tag{4.14}$$

where the integration is over all possible parameterizations of the model. The question is whether this criterion will favor simple models or complex ones. Actually, while the maximum likelihood approach (or the empirical risk minimization approach) which selects structures maximizing

$$\arg\max_{\underline{W}} P(LS|S, \underline{W}),$$

will systematically select the most complex models of the hypothesis space, the criterion (4.14) is not systematically biased towards the more complex structures. This is essentially due to the assumption of uniform conditional distribution $P(\underline{W}|S)$ which implies an exponentially fast decrease of $P(\underline{W}|S)$ with increasing complexity. This effect is generally strong enough to overcompensate (for complex enough models) the increase of $\arg\max_{\underline{W}} P(LS|S, \underline{W})$. We refer the interested reader to [Sch78] and [Mac92] for a more in depth discussion of this natural complexity penalization property.

Minimum description length (MDL) principle

The MDL principle, and its cousin, the *minimum message length* (MML) principle, are based on a data compression argument. Let us assume that we want to send output values corresponding to the learning cases to a receiver which already knows the input values. Then we have several alternatives, e.g. we could send an explicit description of the output values of each object, or we could determine and send to the receiver a model able to recompute the output values from the inputs, or we could send a model computing approximations of the output values together with error corrections. The MDL principle [Ris78] states that the best model for the learning sample is the one which minimizes the length of the message which has to be sent to the receiver:

$$L(ls, h) = L(ls|h) + L(h) \tag{4.15}$$

where $L(h)$ is the minimal number of bits necessary to describe the model to the receiver and $L(ls|h)$ is the minimal number of bits necessary to describe the errors made by this model on the learning sample. Intuitively, the lower is the empirical risk of the model, the lower will be the length $L(ls|h)$; in the limit if the model perfectly fits the ls this term vanishes. On the other hand, the number of bits used to code a model is certainly an increasing function of the number of parameters used to represent this model and hence of its complexity. Thus the two terms making up the description length are essentially representing a tradeoff between the empirical risk and the model complexity. The MDL principle thus embodies a preference for simpler models and is of the form $Err(f) + \lambda.C(h)$ which was proposed in Section 4.2.2.

Notice that in eqn. (4.15) the two terms defining $L(ls, h)$ are in theory defined as the Kolmogorov complexities of the corresponding objects. These quantities are essentially non computable [CT91] and hence various approximations have been proposed in the literature, leading to various practical versions of the MDL and MML principles.

On the other hand, the link which exists between the Bayesian approach and these principles can be made on the basis of Shannon's information theory [CT91]. Indeed, Shannon's first theorem says that the optimal encoding of long sequences of a random variable X amounts to taking codeword lengths $l_i = -\log_2(P(X = x_i))$ for the i^{th} value of this variable. So, by taking the length $L(ls|h) = -\log_2 P(LS = ls|H = h)$ and $L(h) = -\log_2 P(H = h)$ in (4.15), we get minus the logarithm of (4.6) and hence minimizing (4.15) is equivalent to maximizing (4.6). We refer the interested reader to [ZRF99] for a more detailed discussion of the MDL and MML principles and their relationship with Bayesian model selection, in the specific context of decision tree induction.

Structural risk minimization

The main benefits of the Bayesian and MDL principle is that they provide a language (using either a probabilistic interpretation or a encoding scheme) which facilitates the design of optimization criteria which take into account the tradeoff between error and complexity. However, we do not have assurance that minimizing the posterior probability or the number of bits to encode data and model will give low generalization error (which is the final goal of learning). Actually, it is easy to find a very bad choice of prior probability distribution or encoding scheme which will lead to inappropriate models for many problems. Furthermore, both approaches will often rely on some parameters (like α and β in (4.12)) which would require to use one of the techniques presented in Section 4.2.1 to be fixed automatically.

Structural risk minimization on the other hand is based on the bounds on generalization error given by Vapnik (see Section 3.5.4 or [Vap95]). This bound is the sum of the empirical risk and an increasing function of the VC-dimension, denoted h (which is some measure of the expressiveness of the hypothesis space). Hence, when the VC-dimension increases, empirical risk is decreasing like the bias in Figure 4.1 and the bound on the difference is increasing like the variance. All in all, the generalization error is bounded by a function which follows a trajectory similar to the one of the mean error in the same figure. Vapnik proposes structural risk minimization to minimize the error bounds. It works as follows:

- Structure the hypothesis space into nested subsets $\mathcal{H}_i$ such that $\mathcal{H}_1 \subset \mathcal{H}_2 \subset \ldots \subset \mathcal{H}_i \subset \ldots$ and the VC-dimension h_i of each $\mathcal{H}_i$ is finite. Hence, we have $h_1 \leq h_2 \leq \ldots \leq h_i \leq \ldots$.

- Find the function f_i^* which minimizes the empirical risk in each subspace $\mathcal{H}_i$.

- Among these functions, choose the one which together with its subspace minimize the error bound.

Structural risk minimization is thus strictly equivalent to our approach where the parameter s is the VC-dimension. However, this method uses the derived bound on the generalization error to select the optimal value of complexity and hence does not require an estimate of the generalization error. The practical application of structural risk minimization is however limited by the fact that often the VC-dimension is very difficult to compute.

4.3 Aggregation techniques

The techniques discussed in the previous section are dependent on the intrinsic bias/variance curves of the learning algorithms they are applied to. For a given value of the parameter s, bias and variance are determined by the learning algorithm and the optimization of s is tied to respect these values. The techniques described in the present section, on the contrary, try to attack the intrinsic bias/variance tradeoff of a learning algorithm essentially by shifting its variance curve towards zero without changing too much its bias curve. To this end, the common idea of all these techniques is to generate by some means a set of predictions using the original (unstable) learning algorithm, and then to aggregate these predictions using a voting or an averaging scheme, in order to yield a more stable final prediction. These aggregation techniques essentially differ in the way they operate on the original algorithm in order to build a set of predictions, but the usual way is to build an ensemble[1] of *models* whose predictions are aggregated.

This section is organized as follows : we start by presenting "Bagging", one of the most popular aggregation technique; we then proceed by analyzing the effect of averaging[2] techniques applied to randomized algorithms; we conclude by proposing a general framework encompassing all these averaging techniques and show how bagging can be viewed as a particular case of this general framework.

4.3.1 Bootstrap aggregating (Bagging)

Underlying idea

By definition, the average model $\overline{f}_{LS}(\underline{a})$ corresponding to a learning algorithm has zero variance and it has the same bias as the original algorithm. So if we could find an algorithm producing on the basis of the learning sample a good approximation of this average model, this would reduce variance without too much altering bias. Let us study this idea in the case of regression and in an ideal situation, i.e. assuming that we can draw as many learning sets as we want from the distribution $P(LS) = P^N(\underline{A}, Y)$. An approximation of $\overline{f}_{LS}$ is then simply built by drawing several learning sets ls_i, $i = 1, \ldots, T$, from $P(LS)$, inducing a model f_{ls_i} from each of the ls_i and then computing the average predictor by:

$$f_{ls^T}(\underline{a}) = \frac{1}{T}\sum_{i=1}^{T} f_{ls_i}(\underline{a}), \tag{4.16}$$

for some value of T. This expression defines a new model whose predictions are also random. Since the ls_i are independently drawn from the same distribution $P(LS)$, the average model corresponding to the new learning algorithm, which generates $f_{LS^T}(\underline{a})$, is nothing but the average model of the original algorithm:

$$E_{LS^T}\{f_{LS^T}(\underline{a})\} = \frac{1}{T}\sum_{i=1}^{T} E_{LS_i}\{f_{LS_i}(\underline{a})\} = \overline{f}_{LS}(\underline{a}). \tag{4.17}$$

On the other hand, its variance is given by:

$$var_{LS^T}\{f_{LS^T}(\underline{a})\} = var_{LS^T}\{\frac{1}{T}\sum_{i=1}^{T} f_{LS_i}(\underline{a})\} = \frac{1}{T^2}\sum_{i=1}^{T} var_{LS_i}\{f_{LS_i}(\underline{a})\} = \frac{1}{T}\mathrm{var}_R(\underline{a}) \tag{4.18}$$

since the f_{ls_i} are independent and have the same variance as the original learning algorithm. Thus, averaging over T learning sets divides the regression variance by a factor T. All in all, the average prediction is less variable than each individual one and its bias remains unchanged. The result is obviously a decrease of the mean squared error at point $\underline{a}$. In this ideal case, it is possible to compute the value of T which leads to a value of error satisfying some initial requirements.

Equivalently, in classification, a set of models can be induced from random learning sets ls_i and the aggregated prediction at each point is the majority class among the set of predictions:

$$f_{ls^T}(\underline{a}) = \arg\max_{c}\sum_{i=1}^{T} 1(f_{ls_i}(\underline{a}) = c). \tag{4.19}$$

[1] The term "ensemble methods" is used in the literature to denote techniques which combine different models produced by one or several different automatic learning algorithms [Die00b].

[2] Although in the strict sense "averaging" is a particular case of "aggregation" we will use the two terms interchangeably.

Numbers of interest in this case are the probabilities that this model outputs one particular class c. Let us denote by $P_{LS^T}(c|\underline{a})$ this probability for one class c. Assuming a sufficiently large value of T, we know from the central limit theorem that the proportion:

$$P_c(\underline{a}) \triangleq \frac{1}{T}\sum_{i=1}^{T} 1(f_{ls_i}(\underline{a}) = c), \tag{4.20}$$

of instances of class c appearing in the set of predictions is distributed according to a Gaussian distribution with mean:

$$E_{LS^T}\{P_c(\underline{a})\} = P_{LS}(c|\underline{a}), \tag{4.21}$$

and variance:

$$var_{LS^T}\{P_c(\underline{a})\} = \frac{P_{LS}(c|\underline{a}) \cdot (1 - P_{LS}(c|\underline{a}))}{T}. \tag{4.22}$$

In a two class case, the probability $P_{LS^T}(c|\underline{a})$ is the probability of $P_c(\underline{a})$ being greater than 0.5, which yields:

$$P_{LS^T}(c|\underline{a}) = \Phi(\frac{0.5 - P_{LS}(c|\underline{a})}{P_{LS}(c|\underline{a}) \cdot (1 - P_{LS}(c|\underline{a}))} \cdot T), \tag{4.23}$$

where Φ is the upper tail of the standard normal distribution (see Figure 3.7, p. 41). From the definition of Φ, it thus appears that this probability tends towards 1 with increasing values of T if c is the majority class at $\underline{a}$, $f_{LS}^{MAJ}(\underline{a})$, and towards 0 otherwise. So, the classifier f_{LS^T} converges to the majority vote classifier. Asymptotically, the error of f_{LS^T} is thus equal to Tibishirani's bias and its variance vanishes. Contrary to the regression case, however, as mentioned in the previous chapter, this majority vote classifier is not always better than individual predictions for biased points (in the mean). However, assuming that most of the learning algorithms are mainly unbiased (at least the ones which need such techniques because of their high variance), aggregation is beneficial most of the times. Actually, in the two class case, it can be shown that we have $P_{LS^T}(c|\underline{a}) > P_{LS}(c|\underline{a})$ if $P_{LS}(c|\underline{a}) > 0.5$, whatever the value of $T > 2$ (see [ZC01]). Hence, if the learning algorithm is unbiased at this point, aggregated models $f_{LS^T}(\underline{a})$ are always better than single ones.

The bootstrap approximation

In practice, of course, we do not have access to the true distribution $P(LS)$ but only to a single learning set ls. The most direct idea to find an approximation of $P(LS)$ is to use bootstrap sampling to generate several learning sets from the original one. This is the key idea underlying bagging (for bootstrap aggregating, [Bre96b]). Bagging proceeds as follows:

1. From ls (of size N), derive different bootstrap samples bs_i $(i = 1, \ldots, T)$ of size N by random resampling with replacement.

2. From each bs_i produce a model f_{bs_i} using the learning algorithm.

3. Produce a single model, f_{bag}, by aggregating the T models f_{bs_i}:

$$f_{\text{bag}}(\underline{a}) = \text{aggr}\{f_{bs_1}(\underline{a}), f_{bs_2}(\underline{a}), \ldots, f_{bs_T}(\underline{a})\} \tag{4.24}$$

The aggregation operator in the context of regression models is of course averaging.

For classification problems, we have the choice between computing the majority class among the T predictions (like (4.19)) or, if the learning algorithm provides class probability estimates, we can average these conditional class probability estimates of individual models and then select the class corresponding to the largest aggregated probability estimate. Note that several authors have carried out experiments with these two bagging variants in classification (e.g. [BK99]) and even if aggregating class-probability estimates appears to be slightly better, there is no clear evidence that this improvement is significant.

4.3.2 Randomized algorithms

Let us assume that we dispose of a randomized learning algorithm which on different runs with a fixed learning sample may output different models. For example, typical neural network learning algorithms behave in this way since they start iterations from random initial conditions which are re-drawn on every run of the algorithm. Similarly, automatic learning methods using genetic algorithms in the optimization part typically will produce randomized results. Also, as we will see, many existing deterministic automatic learning algorithms may easily be modified in order to randomize them. Our aim in this section is to show that the variance introduced by such randomizations should be reduced by using the idea of averaging described in the preceding section in order to improve performance.

Bias/variance decomposition

Let us assume that the randomness of our algorithm can be fully captured by a random vector of parameters which we will denote by ε (e.g. the final model given by neural network learning is a function of the random initial weights). The model produced by this algorithm is thus a function of the two random variables LS and ε and will be denoted by $f_{LS,\varepsilon}$. The true mean error of this algorithm at a point $\underline{a}$ is given by $E_{LS,\varepsilon}\{Err(f_{LS,\varepsilon}(\underline{a}))\}$. Taking into account the additional expectation according to ε, the regression bias/variance decomposition of the mean error becomes (see (3.15)):

$$E_{LS,\varepsilon}\{Err(f_{LS,\varepsilon}(\underline{a}))\} = \sigma_R^2(\underline{a}) + (f_B(\underline{a}) - \overline{f}_{LS,\varepsilon}(\underline{a}))^2 + var_{LS,\varepsilon}\{f_{LS,\varepsilon}(\underline{a})\}, \tag{4.25}$$

where:

$$\overline{f}_{LS,\varepsilon}(\underline{a}) = E_{LS,\varepsilon}\{f_{LS,\varepsilon}(\underline{a})\} \tag{4.26}$$

$$var_{LS,\varepsilon}\{f_{LS,\varepsilon}(\underline{a})\} = E_{LS,\varepsilon}\{(f_{LS,\varepsilon}(\underline{a}) - \overline{f}_{LS,\varepsilon}(\underline{a}))^2\}. \tag{4.27}$$

From the total variance theorem, the variance term may be further decomposed into two terms:

$$var_{LS,\varepsilon}\{f_{LS,\varepsilon}(\underline{a})\} = var_{LS}\{E_{\varepsilon|LS}\{f_{LS,\varepsilon}(\underline{a})\}\} + E_{LS}\{var_{\varepsilon|LS}\{f_{LS,\varepsilon}(\underline{a})\}\} \tag{4.28}$$

The first term in (4.28) is the variance with respect to the learning set randomness of the average prediction according to ε while the second term is the expectation over all learning sets of the variance of the prediction with respect to ε.

Averaging out the effect of ε

Let us suppose that instead of building one single model $f_{ls,\epsilon}$ for a given learning set with our original random algorithm, we consider a hypothetical algorithm which would produce for the given learning set ls, the expected model $E_{\varepsilon|ls}\{f_{ls,\varepsilon}(\underline{a})\}$.

The bias of this method would actually be identical to the bias of the original random algorithm, since we have:

$$\overline{f}_{LS,\varepsilon}(\underline{a}) = E_{LS,\varepsilon}\{f_{LS,\varepsilon}(\underline{a})\} = E_{LS}\{E_{\varepsilon|LS}\{f_{LS,\varepsilon}(\underline{a})\}\}. \tag{4.29}$$

On the other hand, the variance of this algorithm would be equal to the first term of eqn. (4.28).

In practice however, it may be that $E_{\varepsilon|LS}\{f_{LS,\varepsilon}(\underline{a})\}$ is difficult to compute exactly. This suggests the following approximation algorithm: Given a learning set ls, draw several values ϵ_i, $i = 1, \ldots, T$ from the distribution $P(\varepsilon|ls)$, build a model f_{ls,ϵ_i} according to the learning algorithm for each value ϵ_i and compute the average model:

$$f_{ls,\epsilon^T}(\underline{a}) = \frac{1}{T}\sum_{i=1}^{T} f_{ls,\epsilon_i}(\underline{a}). \tag{4.30}$$

Transposing the development of the previous section, it is easy to show that the bias of this model is equal to the bias of the original learning algorithm and that its variance is given by:

$$var_{LS,\varepsilon^T}\{f_{LS,\varepsilon^T}(\underline{a})\} = var_{LS}\{E_{\varepsilon|LS}\{f_{LS,\varepsilon}(\underline{a})\}\} + \frac{E_{LS}\{var_{\varepsilon|LS}\{f_{LS,\varepsilon}(\underline{a})\}\}}{T} \tag{4.31}$$

So averaging over T values of ε reduces the second part of the variance by a factor T and leaves the remaining parts of the mean error unchanged. The global reduction of error by averaging is thus proportional to the term $E_{LS}\{var_{\varepsilon|LS}\{f_{LS,\varepsilon}(\underline{a})\}\}$:

$$E_{LS,\varepsilon^T}\{Err(f_{LS,\varepsilon^T}(\underline{a}))\} = E_{LS,\varepsilon}\{Err(f_{LS,\varepsilon}(\underline{a}))\} - (1-\frac{1}{T})\cdot E_{LS}\{var_{\varepsilon|LS}\{f_{LS,\varepsilon}(\underline{a})\}. \tag{4.32}$$

The more variable are the predictions with respect to ε, the more useful it is to use averaging.

Ambiguity decomposition

While equation (4.32) describes the mean impact of averaging, it is possible to relate the error of an average model to the individual errors of its components. Indeed, this latter term is given by:

$$\frac{1}{T}\sum_{i=1}^{T} Err(f_{ls,\epsilon_i}(\underline{a})) = \frac{1}{T}\sum_{i=1}^{T} E_{Y|\underline{a}}\{(Y - f_{ls,\epsilon_i}(\underline{a}))^2\}, \tag{4.33}$$

which may be decomposed into:

$$\frac{1}{T}\sum_{i=1}^{T} Err(f_{ls,\epsilon_i}(\underline{a})) = E_{Y|\underline{a}}\{(Y - f_{ls,\epsilon^T}(\underline{a}))^2\} + \frac{1}{T}\sum_{i=1}^{T}(f_{ls,\epsilon_i}(\underline{a}) - f_{ls,\epsilon^T}(\underline{a}))^2. \tag{4.34}$$

Inverting the decomposition, we obtain:

$$Err(f_{ls,\epsilon^T}(\underline{a})) = \frac{1}{T}\sum_{i=1}^{T} Err(f_{ls,\epsilon_i}(\underline{a})) - \frac{1}{T}\sum_{i=1}^{T}(f_{ls,\epsilon_i}(\underline{a}) - f_{ls,\epsilon^T}(\underline{a}))^2. \tag{4.35}$$

So, the error of the average model is the difference between the mean error of individual models minus the variance of their predictions. This decomposition is due to Krogh and Vedelsby [KV95] and is called the ambiguity decomposition. The last term of equation (4.35) is called the ambiguity:

$$A_r(\underline{a}) = \frac{1}{T}\sum_{i=1}^{T}(f_{ls,\varepsilon_i}(\underline{a}) - f_{ls,\epsilon^T}(\underline{a}))^2. \tag{4.36}$$

As ambiguity is always positive, the main conclusion is that the average model is always better than the individual models in the mean. How much better depends on the ambiguity: the more individual predictions disagree with each other, the better will be the improvement by consensus. Taking the expectation of both sides of (4.35) according to the learning set and ε^T, we retrieve equation (4.32) and by identifying the different terms, we get:

$$(1-\frac{1}{T})\cdot E_{LS}\{var_{\varepsilon|LS}\{f_{LS,\varepsilon}(\underline{a})\}=E_{LS,\varepsilon^T}\{\frac{1}{T}\sum_{i=1}^{T}(f_{LS,\varepsilon_i}(\underline{a})-f_{LS,\epsilon^T}(\underline{a}))^2\} \tag{4.37}$$

which shows that the reduction of mean error is equivalently the mean (over all learning sets and parameters choice) variance of the prediction inside the ensemble produced for each learning sample.

The case of classification problems

While the effect of model averaging according to ε is simple to study in the context of regression, the problem is much more complex in classification. Assuming a random learning algorithm, Tibishirani's bias/variance decomposition of the mean error rate can be written:

$$E_{LS,\varepsilon}\{Err(f_{LS,\varepsilon}(\underline{a}))\}=Err(\mathrm{maj}_{LS,\varepsilon}\{f_{LS,\varepsilon}(\underline{a})\})+\mathrm{var}_T(\underline{a}) \tag{4.38}$$

where $\mathrm{maj}_{LS,\varepsilon}\{f_{LS,\varepsilon}(\underline{a})\}=\arg\max_c P_{LS,\varepsilon}(c|\underline{a})$ computes the majority class over all values of LS and ε and the term $\mathrm{var}_T(\underline{a})$ is defined such that this additive decomposition holds. The relationship between the mean error of $f_{LS,\varepsilon}$ and the one of the aggregated classifier $\mathrm{maj}_{\varepsilon|LS}\{f_{LS,\varepsilon}(\underline{a})\}$ is difficult to characterize. First, contrary to the regression case, the bias is not necessarily conserved by aggregation. Indeed, in general, we have:

$$\mathrm{maj}_{LS,\varepsilon}\{f_{LS,\varepsilon}(\underline{a})\}\neq\mathrm{maj}_{LS}\{\mathrm{maj}_{\varepsilon|LS}\{f_{LS,\varepsilon}(\underline{a})\}\} \tag{4.39}$$

and hence,

$$Err(\mathrm{maj}_{LS,\varepsilon}\{f_{LS,\varepsilon}(\underline{a})\})\neq Err(\mathrm{maj}_{LS}\{\mathrm{maj}_{\varepsilon|LS}\{f_{LS,\varepsilon}(\underline{a})\}\}). \tag{4.40}$$

Second, even if the bias is unchanged, as variance can be negative, it is also possible for the error to increase by averaging. The only thing which is certain in classification is that $E_{\varepsilon|LS}\{1(f_{LS,\varepsilon}(\underline{a})=c)\}$ are better estimates than $1(f_{LS,\varepsilon}(\underline{a})=c)$ for class probabilities $P(c|\underline{a})$ in the sense of least square since their bias terms are the same and the variance of the latter is reduced by averaging. However, there is no guarantee that this will yield lower misclassification error.

Focusing on non average results, it is also difficult to rely the error of the majority vote classifier to the mean error of its individual constituents. For example, trying to transpose the ambiguity decomposition from the regression case, we get the following definition for the ambiguity in classification:

$$A_c(\underline{a})=Err(\mathrm{maj}_{i=1\ldots T}\{f_{ls,\epsilon_i}(\underline{a})\})-\frac{1}{T}\sum_{i=1}^{T}Err(f_{ls,\epsilon_i}(\underline{a})) \tag{4.41}$$

which positiveness is unfortunately not ensured.

4.3.3 General perturb and combine algorithm

Generic algorithm

If we average different models built from several runs of a random algorithm, we know from the previous section that we obtain a more stable algorithm. This idea can not be applied directly to a deterministic algorithm which always returns the same model in response to a fixed learning set. However, a deterministic algorithm can be made random by the introduction of some perturbations in the learning procedure. This gives the idea of the following generic algorithm:

- From a learning set ls, for i going from 1 to T:
  - perturb the learning procedure (e.g. by perturbing its data or settings of its parameters) with a set of random parameters ϵ_i drawn independently from a distribution $P(\varepsilon|ls)$ and apply the perturbed versions to get the models f_{ls,ϵ_i}
- Build the aggregated predictor f_{ls,ϵ^T} defined by:

$$f_{ls,\epsilon^T}(\underline{a}) = \text{aggr}\{f_{ls,\epsilon_1}(\underline{a}), f_{ls,\epsilon_2}(\underline{a}), \ldots, f_{ls,\epsilon_T}(\underline{a})\}, \tag{4.42}$$

  where, like for bagging, "aggr" stands for average in regression and majority vote or average of probability estimates in classification.

Following Breiman [Bre96a], we will call an averaging technique following this strategy a *perturb and combine* (P&C) algorithm. For example, bagging is a perturb and combine algorithm. The random vector ε is in this case a vector of N (the learning set size) integers randomly and uniformly drawn in $[1, N]$, where each component of ε indexes one element of the learning set selected in the particular random bootstrap sample of size N.

When does it work ?

For the sake of the present discussion let use the following terminology

- "original algorithm" denotes any given automatic learning algorithm (we assume, without any limitation, throughout the discussion that this algorithm is deterministic, say like the standard decision tree induction method);
- "perturbed algorithm" denotes the randomized version of the original algorithm, according to some perturbation technique (say a decision tree method selecting tests in a random fashion);
- "combined algorithm" denotes the algorithm producing models built by aggregating T different models obtained with the perturbed algorithm.

Our discussion aims at giving conditions under which the *combined* algorithm would perform better (in the usual sense of average error) than the *original* algorithm. Given the formal difficulties that we have already encountered in the context of classification methods when discussing the effect of aggregation, we will focus here on regression.

Figure 4.2 shows the expected evolution of bias and variance during one perturb and combine procedure and is explained below. The perturbed algorithm provides a model $f_{LS,\varepsilon}$ which depends on the original algorithm and on the way the perturbation ε interacts with the induction process of this latter. In regression, the variance of this new algorithm can be decomposed into the two terms given in (4.28):

$$var_{LS,\varepsilon}\{f_{LS,\varepsilon}(\underline{a})\} = var_{LS}\{E_{\varepsilon|LS}\{f_{LS,\varepsilon}(\underline{a})\}\} + E_{LS}\{var_{\varepsilon|LS}\{f_{LS,\varepsilon}(\underline{a})\}\}. \tag{4.43}$$

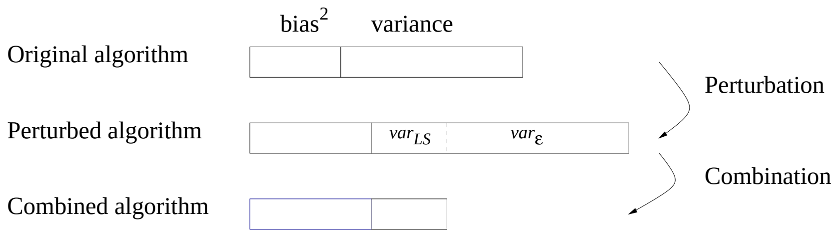

Figure 4.2: Expected evolution of bias and variance by perturbation and combination

At the same time, the square bias of the perturbed algorithm becomes:

$$(f_B(\underline{a}) - \overline{f}_{LS,\varepsilon}(\underline{a}))^2. \tag{4.44}$$

Using the average operator to aggregate T regression models produced in this fashion, we obtain the combined algorithm which, as we already know, produces a model f_{LS,ε^T} with the same bias and lower variance than the perturbed algorithm. As T increases, the variance of the perturbed algorithm converges towards

$$var_{LS}\{E_{\varepsilon|LS}\{f_{LS,\varepsilon}(\underline{a})\}\}.$$

Thus, asymptotically with respect to T, the whole process of perturbation and aggregation decreases the variance with respect to our original algorithm if:

$$var_{LS}\{E_{\varepsilon|LS}\{f_{LS,\varepsilon}(\underline{a})\}\} < var_{LS}\{f_{LS}(\underline{a})\}, \tag{4.45}$$

i.e. if the perturbation artificially introduced in the learning procedure captures some part of the intrinsic variance of the initial learning algorithm, given that this artificial variance is canceled by the averaging process.

However, in order to assess the average error of the perturbed algorithm and compare it with that of the initial algorithm, we need also to take into account the effect on bias. Indeed, in general,

$$E_{LS,\varepsilon}\{f_{LS,\varepsilon}(\underline{a})\} \neq E_{LS}\{f_{LS}(\underline{a})\} \tag{4.46}$$

and hence bias could possibly be increased by the perturb and combine process. Actually, when the original learning algorithm explicitly tries to minimize empirical risk (like decision tree induction without pruning which builds trees of minimal learning set error), it is likely that the perturbed version will disturb the algorithm in this goal. Consequently, both the perturbed and the combined algorithms are likely to have higher bias than the original algorithm. All in all, a perturb and combine approach will be effective if the increase in bias is not larger than the reduction of variance.

It is possible to draw similar conclusions from the ambiguity decomposition (4.35). From this decomposition, Krogh and Vedelsby have emphasized the need for small mean errors of models $f_{LS,\varepsilon}$ and high ambiguity between them for averaging to be efficient. Small mean errors mean small values of $E_{LS,\varepsilon}\{Err(f_{LS,\varepsilon}(\underline{a}))\}$ and high ambiguity means a large variance term $E_{LS}\{var_{\varepsilon|LS}\{f_{LS,\varepsilon}(\underline{a})\}\}$. From the decomposition (4.25) which combines these two terms via (4.28), these two requirements can only be satisfied at the same time if the bias of $f_{LS,\varepsilon}$ is kept small and the variance term $var_{LS}\{E_{\varepsilon|LS}\{f_{LS,\varepsilon}(\underline{a})\}\}$ is also small. So the randomization of the original algorithm should be carried out in order to meet these requirements.

In practice this discussion implies that for the perturb and combine method to be effective in reducing errors, the randomization part of it should essentially produce a highly diverse set of models of small empirical risk. These models will be likely to have small bias, and at the same time have a reduced dependence on the learning sample.

Randomization in classification

Of course, the discussion about classification problems at the end of the previous section can be transposed here. Very little can be said in theory about the effect of perturb and combine algorithms in classification. Nevertheless, several authors have searched for a relationship in classification between the diversity of ensemble members and the improvement obtained by averaging. For example, Ali [AP95] and Dietterich [Die00c] separately define a measure of the diversity of the classification obtained by the models $f_{LS,\varepsilon}(\underline{a})$ for random ε and empirically show a correlation between their diversity measure and the improvement obtained by aggregation. Ho in [Ho98] defines a measure of disagreement between a set of classification models and also attributes better performance of various perturb and combine algorithms to the increase of disagreement they yield in the context of decision trees. Amit and Geman [AG97, AB01], as well as Breiman [Bre01], give an upper bound on the error $E_{\underline{A}}\{Err(E_{\varepsilon|LS}\{f_{LS,\varepsilon}(\underline{A})\})\}$ which is a decreasing function of the "strength" of the individual models (which is some measure of their accuracy) and an increasing function of the "correlation" between them (which is some measure of the dependence between their predictions). They also show empirically that strong uncorrelated models give the best improvement when aggregated.

Bias/variance tradeoff

From the preceding discussion as well as from experimental studies[3], it appears that in the P&C algorithms there is a bias/variance tradeoff regulated by the amount of perturbation injected in the learning procedure:

- the stronger the effect of randomization, the lesser the dependence of the perturbed models on the learning sample and the lesser the variance of the combined models;

- the stronger the effect of randomization, the lesser the dependence of output predictions on the input attributes and the higher bias.

Hence, suppose that we can measure the strength of the random effect of the perturbation by a parameter s, then the general techniques described in the previous section to fix s can be transposed here to fix the right level of perturbation.

Note that the increase of variance resulting from the second term of (4.43) does not influence the tradeoff since this term is the one which is canceled by averaging. Nevertheless, larger values of this term will imply slower convergence with respect to the number T of ensemble terms, and thus lead to higher computational requirements in practice.

Discussion

Note that the preceding analysis does not give a practical way to design valid perturb and combine algorithms. In practice, it is difficult to assess otherwise than empirically the effect of the introduction of a perturbation in a learning algorithm. However, by interpreting the effect of this perturbation in terms of bias and variance, it is somewhat possible to predict its behavior. For example, bagging which perturbs the algorithm by building a model from a bootstrap sample of the actual learning set is likely not to increase very much bias and, at the same time, should capture by this perturbation some of the variance of the initial algorithm. This kind of reasoning will allow us to design at Chapter 7 an extreme perturb and combine algorithm in the context of decision tree induction.

[3] See, for instance chapters 7 and 8 for detailed results in the context of decision tree induction.

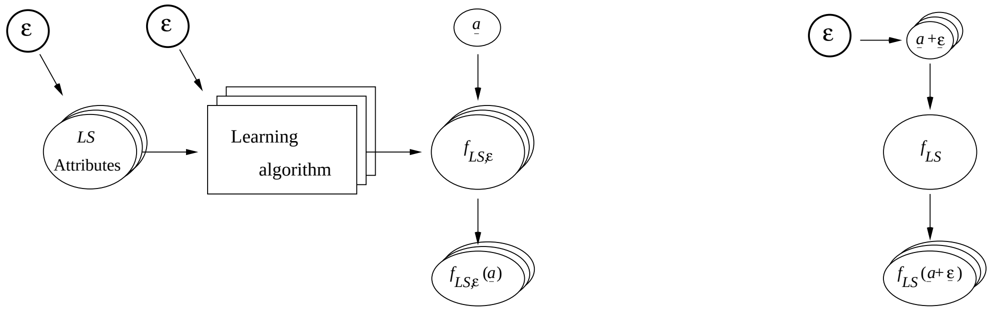

Figure 4.3: Three places where to introduce noise in the learning protocol. Left, with traditional perturb and combine algorithms. Right, the dual perturb and combine approach

4.3.4 Examples of P&C algorithms

Several algorithms have been proposed in the literature which satisfy the description given in the preceding section. They all introduce some noise in the learning process so as to generate several models. Algorithms differ in the place where the random noise is injected during the learning process. The left part of Figure 4.3 shows two places where perturbations are usually introduced: the learning sample or the learning algorithm. The first type of techniques generate several models by feeding the original learning algorithm with modified versions of the learning sample and hence may be potentially applied to any learning algorithms. The second type of techniques directly perturb the learning algorithm to generate the set of models with the same learning sample. We review below some algorithms which have been proposed in these two categories.

Eventually, we introduce a new idea of perturb and combine method which contrasts with other methods by the fact that perturbations are introduced at the prediction stage. The right part of Figure 4.3 illustrates this idea. In order to generate a set of perturbed predictions, the attribute vector of the tested object is randomized and each of the resulting vectors is propagated in the same model, giving a set of predictions which are then aggregated. In this category, we briefly discuss our "Dual Perturb and Combine" algorithm which will be presented in more detail in Chapter 8.

Perturbing the learning set : bagging and its variants

The most direct idea to emulate the variance coming from the learning set randomness is to perturb the learning set itself. The most direct idea to do this is then to use bootstrap sampling to generate several learning sets from the original one. Starting with a learning set ls of size N, this yields the following general procedure:

1. From ls, derive different "bootstrap" samples ls_i, $i = 1, \ldots, T$, of size M by random resampling with or without replacement.

2. From each ls_i, produce a model f_{ls_i} using the learning algorithm.

3. Produce a single model by aggregating the T predictions of the models f_{ls_i}

Bagging is a particular instance of this algorithm with $M = N$ and resampling with replacement. Its generalization has been studied for example by Sollich and Krogh [SK96] for linear models and by Friedman and Hall [FH00b] in the nonlinear case. Although better results are possible with smaller values of M and/or without replacement sampling, bagging has proven to be very efficient in many applications, and although its efficiency varies from one application to another it almost never decreases accuracy.

Wagging [BK99] is a variant of bagging which requires that the learning algorithm can handle weights associated to learning cases. These weights give differing relative importance to learning cases. All the algorithms presented here can deal with such weights. For example, in neural networks, weights are introduced naturally in the error criterion which becomes a weighted sum of the square difference between the output and model. In decision trees, frequency counts at nodes also can easily take into account such weights. While original learning algorithm uses unit weights for each instance of the learning set, wagging (for weight aggregating) perturbs the learning set by assigning random weights to the learning set cases. For example, in the original formulation of wagging, Bauer and Kohavi [BK99] generate random weights from a Gaussian distribution of mean zero and a fixed standard deviation. Webb's version [Web00] of wagging draws weights from a continuous Poisson distribution. In both variants, wagging has been shown to give very similar results to classical bagging.

Another way to perturb the learning set is to perturb the attribute vector describing each instance. Along this idea, Breiman proposed an approach he calls *output smearing* [Bre00], in which only the output attribute values are perturbed when the auxiliary learning sets are generated. In the case of regression, an instance $(\underline{a}, y) \in LS$ thus becomes $(\underline{a}, y + \sigma_y \cdot \varepsilon)$ where ε is the realization of a Gaussian random variable of mean zero and unit standard deviation and σ_y is the standard deviation of the output Y estimated in the learning set. Classification problems can be handled in this framework, either by transforming them into a regression problem (the output variables of which will correspond to class-indicator variables), or by *flipping* output classes. This latter option consists in changing the output class of an instance for another one according to a probability distribution. The probability $p(j|k)$ of flipping class k to class j is given in his implementation by:

$$p(j|k) = \begin{cases} w \cdot c(j) & \text{if } j \neq k \\ 1 - w(1 - c(k)) & \text{if } j = k \end{cases} \tag{4.47}$$

where w is a parameter and $c(k)$ is the proportion of element of class k in the learning set. The experiments detailed in [Bre00] shows that randomizing output (output smearing or output flipping) actually gives consistently better results than bagging.

Another way to perturb the attribute vector is to randomly hide some attributes to the learning algorithm. This approach is adopted for example in [CC00] in combination with K-NN or in [Ho98] in combination with decision trees.

Perturbing the learning algorithm

If the learning algorithm is random, i.e. may output different models in different runs on the same learning set, we have seen that some part of the variance can be reduced simply by running the algorithm several times on the same learning set and aggregate the predictions. For example, in neural networks, a large part of variance is due to the fact that initial weights are random. A good way to get rid of this variance is to build several models with different random initial weights and aggregate the results. This procedure has been called "simple" in [Han00] and shows improvements comparable to bagging or boosting on several problems. These results are confirmed by experiments in [OM99].

If the learning algorithm is deterministic (or not), several predictions can still be produced by injecting perturbations during learning. In the context of neural networks for example, several models can be produced by using different sets of learning parameters [Alp93], or by considering different architectures [Has97]. To be efficient in terms of variance reduction, these perturbations should be consistent with natural perturbations which comes from the learning set randomness. For example, in decision tree induction, at each node, a test is selected among a set of candidates. In the next chapter, it is shown that the choice of a particular test is highly variable from one learning set to another. This variability can be artificially expressed by randomly selecting one test among the best ones in the set of candidates. This approach is adopted by Dietterich in [Die00c] to build random trees, by selecting at each node a test at random among the twenty best candidate tests. A set of trees is built from the learning set using this random algorithm and their predictions are aggregated. The experiments reported in [Die00c] show that this type of randomization compares favorably to bagging. Several other ways to perturb decision tree induction proposed in the literature will be discussed in more details in Chapter 6.

Dual perturb and combine

The two previous algorithms precompute a set of models by injecting perturbations at the learning stage; for prediction these models are used one by one on a new object so as to yield a corresponding number of output values which are then aggregated. In contrast to this "perturbation at the learning stage", one can imagine to build one single model with the initial learning algorithm and learning sample and then perturb the predictions made by this model by introducing at the prediction stage some random variations. For example, in the "dual perturb and combine method" proposed in [Geu01a], perturbations of the predictions of a model f_{ls} at a point $\underline{a}$ are obtained by

$$f_{ls,\underline{\epsilon}}(\underline{a}) = f_{ls}(\underline{a} + \underline{\epsilon}), \tag{4.48}$$

where $\underline{\epsilon}$ is a noise vector of same dimensionality than the input space, generated according to some appropriately chosen distribution, and where the $+$ operator denotes in a generic way the combination of two attribute vectors.

Based on this idea, we propose a new averaging technique which proceeds as follows: A certain number of perturbed versions of the attribute vector $\underline{a}$ of a test instance are produced. The model (induced traditionally from the original learning set) is applied to these vectors resulting in a set of predictions which are aggregated to obtain the final prediction. Considering numerical attributes only, a natural choice to perturb the attribute vector is to add a zero-mean Gaussian noise to it. Thus, dual perturb and combine makes a prediction at $\underline{a}$ with a model f_{ls} according to the following formula:

$$f_{ls}^{DPC}(\underline{a}) = \text{aggr}\{f_{ls}(\underline{a} + \underline{\epsilon}_1), f_{ls}(\underline{a} + \underline{\epsilon}_2), ..., f_{ls}(\underline{a} + \underline{\epsilon}_T)\}, \tag{4.49}$$

where the vectors $\underline{\epsilon}_i$ are obtained from a simple diagonal Gaussian distribution normalized according to the variance of each input variable. In other words, for each input attribute A_i a noise term is generated according to a normal distribution of zero-mean and standard deviation equal to $\lambda\sigma_i$, where σ_i is the standard deviation of the attribute A_i in the learning set and λ is a constant which magnitude reflects the intensity of perturbation.

If the model is smooth, the value of $f_{ls}^{DPC}(\underline{a})$ will be close to the prediction $f_{ls}(\underline{a})$ for small values of λ. On the other hand, if the model is rough, dual perturb and combine makes it smoother. The amount of noise on attributes, measured by λ, thus regulates some bias/variance tradeoff and the best value can be determined by cross-validation for example.

Although less efficient than bagging, we will see in Chapter 8 that this technique reduces very strongly the variance of decision trees. Furthermore, one of its main advantages with respect to other averaging techniques is that it does not require to learn several models and hence intrinsically saves the interpretability and efficiency of the learning algorithm to which it is applied.

4.3.5 Other aggregation techniques

We have described a general framework encompassing perturb and combine algorithms like bagging. Several other techniques have been proposed in the machine learning literature which also rely on the combination of the output of several models. We describe some of them in this section and relate them to the perturb and combine approach. We first review Bayesian model averaging, then boosting, and eventually, several approaches where the combination of the outputs is learned from the data in contrast with perturb and combine algorithms which give equal importance to all models in the ensemble.

Bayesian model averaging

The full Bayesian approach does not advice to select one model from the posterior probability distribution (for example the maximum a posteriori) but rather to compute the average predictions at a point $\underline{a}$ according to the posterior distribution, i.e.:

$$f^*_{LS}(\underline{a}) = \int_{\mathcal{H}} h(\underline{a})dP(H = h|LS), \tag{4.50}$$

In practice, this integration over all possible models is only tractable in specific cases and, usually, it is replaced by approximate subsampling from the posterior distribution $P(H|LS)$ (for example by Monte-Carlo methods [Nea96]).

With respect to the classical selection of one model, this approach takes into account the uncertainty about the model which remains when we have seen the data. From this point of view, Bayesian model averaging draws similarities with bagging and other perturb and combine approaches. The ultimate goal of perturb and combine algorithms (as discussed in this section) is to capture and eliminate the uncertainty about the model which are caused by the uncertainty about the learning sample. To this end, this uncertainty is approximated by some probability distribution on models $P_{P\&C}(H|LS) = P(H|\varepsilon, LS).P(\varepsilon|LS)$ where $P(H|\varepsilon, LS)$ is often deterministic. This distribution is then sampled to build the models which are finally aggregated. We have seen that in our approach, randomization should not decrease too much the bias which means that the error of the model should remain low on the learning sample. So, the posterior probability distribution with P&C would ideally be centered around the best models on the learning sample. On the other hand, the posterior distribution $P(H|LS)$ with uniform priors is proportional to $P(LS|H)$ and hence should also be centered on very good models (from the empirical risk viewpoint). So, intuitively, perturb and combine may be considered as an approximation of the full Bayesian approach. However, whether or not this relationship gives some explanation of the good behavior of either perturb and combine algorithms or Bayesian approaches is a difficult question. Some discussions about the links between Bagging and Bayesian model averaging may be found in [Dom97b] and [HTF01] for example.

Boosting

Boosting is still another approach which combines the predictions of several classification models. Originally, it has been proposed by Schapire [Sch90] as a way to "boost" the performance on the learning sample of a "weak" learning algorithm, i.e. a learning algorithm which is only slightly (but certainly) better than chance. After several improvements, this algorithm has evolved into the now popular AdaBoost algorithm (for "adaptive boosting", see [FS95]). The "M1" version of this algorithm introduced in [FS96] is given in Table 4.1.

Contrary to perturb and combine algorithms, Boosting builds models sequentially. It assumes that the weak learning algorithm can handle weighted objects in the learning

Table 4.1: Boosting algorithm ("AdaBoost.M1")

AdaBoost.M1(ls, LA):
(Build a boosted ensemble of models from a learning sample ls using a learning algorithm LA)

- set $w_k = 1/N \ \forall k = 1, \dots, N$
- For $i = 1$ to T
  - Build a model, $f_{ls,i}(.)$, with LA from the learning sample ls using weights w_k.
  - Compute
$$Err_i = \frac{1}{N}\sum_{k=1}^{N} w_k.1(c_k \neq f_{ls,i}(\underline{a}_k)),$$
    and $\alpha_i = 1/2 \log((1 - Err_i)/Err_i)$.
  - Set $w_k = w_k.\exp(\alpha_i.1(c_k \neq f_{ls,i}(\underline{a}_k)))$, $k = 1, \dots, N$, and normalize these weights such that $\sum w_k = 1$.
- Return the model
$$f_{Boost}(\underline{a}) = \arg\max_c \sum_{i=1}^{T} \alpha_i 1(f_{ls,i}(\underline{a}) = c).$$

sample (this is the case of all algorithms discussed in this thesis) and sequentially applies this algorithm to the original learning sample with modified weights. Starting with uniform weights, the algorithm sequentially changes them by increasing the weights of the misclassified instances by the previous model in the sequence. So, the learning cases which are difficult to classify by the type of model under consideration are receiving higher and higher weights as the iteration proceeds and the subsequent models are thus forced to focus on these "difficult" instances. Eventually, predictions are carried out by a weighted majority vote among the models according to weights α_i which depend on the accuracy of the corresponding model on the learning sample.

The efficiency of boosting at improving classification learning algorithms has been highlighted in many empirical studies, mainly in combination with decision trees (e.g. [DC96], [BK99]). Actually, boosting with trees has been termed by Breiman [Bre98] as the "best off-the-shelf classifier of the world". Several variants of this original algorithm have been proposed, for classification problem [Bre98] but also for regression problems [FHT00].

Although boosting and perturb and combine algorithms have been compared in many empirical studies (e.g. [Qui96, BK99, Die00c]), the two algorithms are very different. Indeed, boosting is a deterministic algorithm while bagging and perturb and combine algorithms are random. While the main goal of perturb and combine approaches is to reduce the variance of a learning algorithm, the goal of boosting is to reduce the bias as well by focusing on misclassified instances in the learning sample. From our discussion about bias and variance, it is clear that such an algorithm which focuses more and more on misclassified instances in the learning sample should suffer at some point from overfitting. Although it is possible to find situations where the error of boosting starts increasing when further models are added to the ensemble (see [FHT00]), boosting seems nevertheless very resistant to overfitting in practice. Because of its success, many researchers have come with some theory to explain the good behavior of boosting algorithms ([SFBL97, Bre99, FHT00]) but there still remain several open questions to be solved before we can really explain the good behavior of boosting algorithms.

As this algorithm has become a state of the art algorithm especially in combination with decision trees, in subsequent chapters, we will sometimes analyze our results with respect to those obtained by boosting decision trees (using the AdaBoost.M1 version of Table 4.1).

Mixture of models, stacking

In our formulation of perturb and combine, models are averaged using uniform weights. In boosting, even if the weights are changing from one model to another, they are not learned globally to improve the accuracy of the ensemble of models. There exist however several techniques in machine learning which aim at learning the way to combine several models to improve the accuracy of the ensemble. For example, in a mixture of experts [JJNH91], the predictions of several models (the experts) are aggregated by using weights which are function of the points of the input space at which we want to make a prediction. The idea is that some experts may be better than others to predict the output variable at some point of the input space and hence they should receive more importance at this point. The input space is divided by a model which is related to soft decision trees (see Section 8.4.3 of Chapter 8). Another method to learn the combination step is stacking [Wol92]. Stacking consists in using a learning algorithm to build a model which combines the predictions given by several other models (which may be built all with the same algorithm or with different ones). If the combining algorithm learns linear models, then we come up with the weighted predictions of several models where the weights are learned from the data.

With respect to the perturb and combine algorithms, the idea underlying such algorithms is somewhat different. It is argued that the models are complementary and hence that accuracy will be improved by combining their prediction taking into account their complementarity. The idea behind perturb and combine algorithms is that several models are equally good (or bad) given the available learning sample and we have no argument to prefer one over the others. So, learning the weights of the aggregation from the learning sample should reduce the bias but also increase the variance with respect to the uniform aggregation of the same models.

4.4 Conclusion

In this chapter, we have introduced several means to control the variance of a learning algorithm. The first class of methods are means to control the complexity of the hypothesis space so as to find the best tradeoff between bias and variance. The second class of methods, termed perturb and combine algorithms, consist in aggregating the predictions provided by several models built from the same learning sample. The dual perturb and combine algorithm contrasts from other perturb and combine algorithms by the fact that it generates its perturbed predictions at the prediction stage with only one model.

We will exploit these techniques in the subsequent parts of this thesis. In the context of decision tree induction, the first kind of method will be described and validated in Chapter 6. Chapters 7 and 8 are respectively devoted to perturb and combine algorithms and the dual perturb and combine approach. Many of the ideas developed here will also be used in the third part of this thesis in the specific context of learning from temporal data.