Combining Predictors

Meta Machine Learning Methods and Bias/Variance & Ambiguity Decompositions

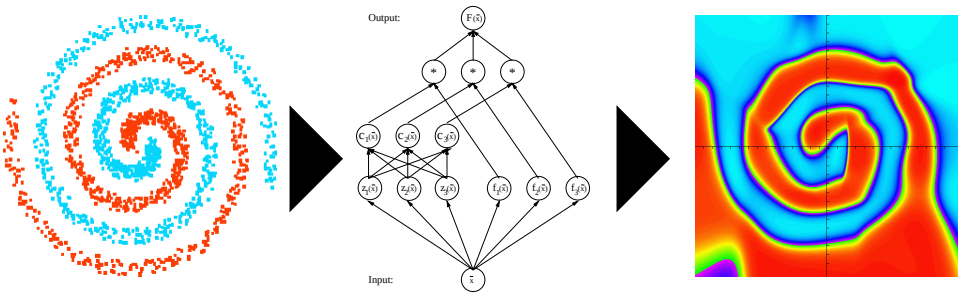
Jakob Vogdrup Hansen

Ph.D. Dissertation

Department of Computer Science
University of Aarhus
Denmark

Combining Predictors

Meta Machine Learning Methods and Bias/Variance & Ambiguity Decompositions



A Dissertation
Presented to the Faculty of Science
of the University of Aarhus
in Partial Fulfillment of the Requirements for the
Ph.D. Degree

by
Jakob Vogdrup Hansen
Handed in: January 31, 2000
Revised: June 26, 2000

Preface

Before I began my four year Ph.D. education, I participated in the courses "Neural Networks" and "Neural Network Study Group", where my interest in the field of artificial intelligence and especially machine learning was founded. The "Neural Network Study Group" resulted in the report "Optimal Brain Construction" [29].

It was therefore natural that Brian Mayoh became my supervisor when I began my Ph.D. education in February 1997.

I used the first part (part A) to take courses and to find the field in which I wanted to specialize. During this process I studied various areas such as artificial life, genetic algorithms, genetic programming, and neural networks. A result was the report "Studies of Optimal Strategies for War & Peace using Neural Networks, Genetic Programming, and Genetic Algorithms" [30].

In the second year of part A, I decided to specialize in the field of ensemble methods. I developed the DynCo method, that turned out to be a mixtures of experts method, so I broadened my field to include mixtures of experts methods as well.

After my part A exam, Anders Krogh became my assistant supervisor. I continued my study of meta machine learning methods as I call ensemble methods and mixture of experts methods. Parallel with empirical tests of different meta machine learning methods, I began, encouraged by Anders Krogh, to study the bias/variance decomposition, which is the central theoretical tool in meta machine learning methods. After I had made some theoretical advances I got in contact with Tom Heskes, with whom I cooperated to find the group of error functions with a natural bias/variance decomposition.

This dissertation is a presentation of my work in the field of meta machine learning methods. All the work has been done at the department of computer science (DAIMI), University of Aarhus, Denmark or at the Center for Biological Sequence Analysis (CBS), Technical University of Denmark, Denmark.

The first version of this dissertation was handed in January 31, 2000. This is the revised version. Only minor errors and the reference list have been corrected.

Acknowledgments

During the four year period I have been in contact with many people that in some positive way have contributed to my work.

First and foremost, I would like to thank my two supervisors Brian Mayoh (DAIMI, AU) and Anders Krogh (CBS, DTU), who taught me the value of supervision. A special thank to Tom Heskes (SNN, KUN) for being both a nice guy and a pedantic. A productive mixture.

In no particular order I would like to thank Torsten Ertbjerg Rasmussen (IMF, AU), Niels Væver Hartvig (IMF, AU), and Jens Ledet Jensen (IMF, AU) for answering all my questions, Simeon Falk Sheye (Cryptomathic) for constructive discussions, writing a PVM based training program, and for proofreading, Mikkel Tjørnfelt-Jensen (DAIMI, AU) for constructive discussions, and for choosing to be my officemate after all, Peter Møller-Nielsen (DAIMI, AU) for introducing me to Hansen and Olsen, Claus Andersen (CBS, DTU) and Ole Lund (CBS, DTU) for the sharing their protein data set, Ole Lajord Munk (PET, AUH) for sharing his brain co-registration data set and for proofreading, Bente Lynge Pedersen for proofreading and for believing in me all the time.

Jakob Vogdrup Hansen,
Århus, June 26, 2000.

Abstract

The advances presented in this dissertation[1] fall into two groups: Improvement of theoretical tools and development of meta machine learning methods.

The most important theoretical tool in connection with meta machine learning is the bias/variance decomposition of error functions. Together with Tom Heskes, I have found the family of error functions with a natural bias/variance decomposition that has target independent variance. It is shown that no other group of error functions can be decomposed in the same way. An open problem in the machine learning community is thereby solved. The error functions are derived from distributions in the one-parameter exponential family. Empirical tests show that there is positive correlation between how well an error function derived from a certain distribution performs and the noise distribution on the training set. The tests also indicate that the error function derived from the Poisson distribution generally outperforms other error functions, among them the commonly used mean square error function.

A bias/variance decomposition can also be viewed as an ambiguity decomposition for an ensemble method. The family of error functions with a natural bias/variance decomposition that has target independent variance can therefore be of use in connection with ensemble methods.

The term "meta machine learning methods" covers both ensemble methods and mixture of experts methods. I have developed the logarithmic opinion pool ensemble method and reinvented the meta machine learning method DynCo, which is similar to earlier published methods [43]. It has been empirically established that the cooperative error function used by DynCo is superior to the formerly preferred competitive error function. The DynCo method bridges the gap between ensemble methods and mixture of experts methods, since it, via the continuous parameter γ, can be set to be either a

[1] Some of the results presented in this dissertation have been published [32, 33], accepted for presentation [34, 36], or are in submission [35]. Since the dissertation has been handed in, the articles [34, 36] have been presented, and the article [35] has been accepted for presentation.

mixture of experts method, an ensemble method or a combination of both. This can also be used to test whether a problem benefits from decomposition or not. The DynCo method has been compared empirically with well-known meta machine learning methods such as AdaBoost, Bagging, and Hierarchical mixtures of experts. DynCo has generally outperformed them. But in some cases the most simple ensemble method, called Simple, outperformed DynCo and the other methods.

The logarithmic opinion pool (LOP) ensemble method has been developed based on the LOP ambiguity decomposition using the Kullback-Leibler (KL) error function. The KL error function compares class probabilities, so the LOP ensemble method is tailor-made for classification. The LOP ensemble method is extended to the cross-validation LOP ensemble method. The advantage of the cross-validation LOP ensemble method is that it can use unlabeled data to estimate the generalization error, while it still uses the entire labeled example set for training. The cross-validation LOP ensemble method has been tested on prediction of the secondary structure of proteins and it compares favorably with other methods.

The cross-validation LOP ensemble method is easily reformulated for another error function, as long as the error function has an ambiguity decomposition with target independent ambiguity.

Bringing the results together, it is indicated that the DynCo method, the cross-validation LOP ensemble method, and the Simple ensemble method are well-performing methods. The LOP ensemble method and the Simple ensemble method are similar except in the error function, and both would benefit from the cross-validation technique. Furthermore, the cross-validation technique applies to all error functions that have an ambiguity decomposition with target independent ambiguity. It is shown exactly which error functions have a natural ambiguity decomposition with target independent ambiguity.

I recommend using the mixtures of experts form of the DynCo method on problems that benefit from decomposition, or the cross-validation Simple ensemble method on problems that do not benefit from decomposition. In both cases the error function can be chosen to fit noise from the one-parameter exponential family of distributions.

Contents

Chapter 1

Introduction

Let's get this party started.
— КоЯn, Issues, Let's Get This Party Started.

In section 1.1 we[1] give a discussion of the nature of machine learning, meta machine learning, and the problems involved at a conceptual level. This is also the motivation. In section 1.2 we give an overview of the contents of the dissertation.

1.1 Motivation

The aim of all science is to give answers to questions, or solve problems. The aim of machine learning is to automate the process of solving problems, and thereby machine learning is a rival to what humans have considered their domain. Not all problems are suitable for machine learning. Let us divide all problems into three groups: The simple mathematical/algorithmic problems, e.g. how to add two numbers or how to sort a list of numbers. These problems are well understood, and - more importantly - a way to find the answer of an instance of the problem is readily available. Another group of problems are what could be called meta-problems, which is problems where no answer can be given, or are subjective, e.g. "Do Humans Beings have a soul?", or "Is this painting beautiful?". Both of these groups cannot benefit from machine learning. In between lies a grey zone of problems where the complete answers are unknown, but partial information is available. Until recently this group of problems has been exclusively for human experts, e.g. medical doctors. Let us use that as an example. It is not always easy to

[1] Throughout this dissertation I will use the plural forms 'we' and 'us' because it narrows the gap between reader and author, and because we are more comfortable with that.

give the right diagnosis for a patient. The symptoms can be faint and irregular, they can be somewhat similar to another disease, or the disease can be unknown. A medical doctor will have been taught about symptoms and the corresponding disease before encountering real patients, but his or her skill should improve by examining patients, diagnosing them, and see the result of the treatment. The doctor will learn from these examples, and become better to diagnose patients in the future. The doctor will learn to generalize. Today medical doctors are not alone in giving diagnosis. Also computers have been used to diagnose people. The first such systems were expert system, where the rules were given the computer by human experts, so the generalization or the learning did not take place in the computer, but still in the mind of the human experts. In recent years this has changed, now the systems learn by themselves in the same ways the doctors do: generalizing from examples, this is why it is called machine learning. Machine learning methods have been used in many situations, e.g. to control fusion reactors [8], speech recognition [28, 62], character recognition [78, 67], speaker identification [16], survival prediction of AIDS patients [58], backgammon [83], document classification [77] and many other things.

In this dissertation we will look into an area of machine learning which is very promising, and already has produced important results: The area of combining predictors. A predictor is here the computer system which is the result of machine learning. The reason behind combining predictors is the same as for gathering a group of human experts: to get better generalization.

Theoretical results tell us that combining predictors improves generalization, and empirical tests support that. But that is just the source of the excitement. There are as many ways of combining predictors as there are researchers in machine learning. Let us illustrate the situation with the group of human experts. They are given an example of something, say a medical record, and must answer a question, like "what is the diagnosis?" What do they do? Let us say they disagree, because that is the interesting case. If most of them agree, then we should trust the majority. But what if the world expert on that disease votes with the minority? Should we give him or her extra votes? The domain of the answer could be continuous instead of classes, then no one would agree with anybody. We could choose the average of the guesses. But what if the guesses come in two distinct groups, so the average is nowhere near any of the groups? The human experts could be specialists in each their subset of the problem domain. Given a question we are faced with the problem of finding the right specialist.

The problems are many, and but potential benefits are even greater, which makes it an exciting research field.

1.2 Overview of the dissertation

Many of the results in this dissertation are from five papers

- HANSEN, J. V. Combining predictors: Some old methods and a new method. In *JCIS '98 Proceedings* (1998), G. Georgiou, Ed., Association For Intelligent Machinery, Inc., pp. 12–16.
- HANSEN, J. V. Accepted for oral presentation at ICCIN2000: The superiority of simplicity. comparison of four meta machine learning methods. Aug. 1999.
- HANSEN, J. V. Combining predictors: Comparison of five meta machine learning methods. *Information Science, an International Journal* (1999).
- HANSEN, J. V., AND HESKES, T. Submitted to 15th international conference on pattern recognition: General bias/variance decomposition with target independent variance of error functions derived from the exponential family of distributions. Dec. 1999.
- HANSEN, J. V., AND KROGH, A. Accepted for presentation at the international conference on artificial neural networks in medicine and biology (ANNIMAB-1): A general method for combining predictors tested on protein secondary structure prediction. Oct. 1999.

The dissertation is in two parts: A theoretical part (chapter 2–7) and an empirical part (chapter 8–11). The theoretical part is mainly about bias/variance decompositions and the corresponding ambiguity decompositions. Also selected literature on the subject is reviewed. In the empirical part different meta machine learning methods are compared, analyzed, and discussed.

- Chapter 2. Framework.

  The framework used in this dissertation is presented.

- Chapter 3. Bias/variance and ambiguity.

  The bias/variance decomposition is a very important theoretical tool in machine learning and especially in meta machine learning. It is presented intuitively and in detail for the mean square error. The connection to the ambiguity decomposition is stated.

- Chapter 4. Bias/variance decomposition in literature.

  An overview of bias/variance decompositions in literature. The claimed connection between two decompositions is shown to be false.

- Chapter 5. General bias/variance decomposition.

  The main theoretical result of the dissertation. The work has been done in cooperation with Tom Heskes. It is shown exactly which error functions have a natural bias/variance decomposition. The error functions are connected to the one-parameter exponential family of distributions. The results is to be published in [35].

- Chapter 6. Machine learning methods.

  Overview of the machine learning methods used in the dissertation.

- Chapter 7. Meta machine learning methods.

  Overview of the meta machine learning methods used in the dissertation.

- Chapter 8. Empirical comparison of deviance error functions.

  The error functions from chapter 5 are connected mathematically to distributions. It is tested empirically if an error function connected to a special distribution is more suitable for training sets with noise from that distributions.

- Chapter 9. DynCo compared with four other methods.

  Presentation and expansion of results from [31, 32, 33]. Five meta machine learning methods are compared empirically on natural example sets and the results are analyzed.

- Chapter 10. Comparison of four meta machine learning methods.

  Presentation and expansion of results from [34]. Four meta machine learning methods are compared empirically on artificial target functions and the results are analyzed.

- Chapter 11. Cross-validation logarithmic opinion pool ensemble and prediction of the secondary structure of protein.

  Presentation and expansion of results from [36]. This work has been done in cooperation with Anders Krogh. The logarithmic opinion pool ensemble method is tested on prediction of the secondary structure of protein.

- Chapter 12. Conclusion and Recommendation

  All the results are brought together in a recommendation on how to solve machine learning problems.

Part I

Theory

Chapter 2

Framework

Machine learning is suitable for a large group of problems, and many different methods and results have been developed in that area. In order to evaluate these different methods and results it is necessary to have a common framework. A framework is a common intuition expressed in suitable and consistent notation and definition of essential concepts in terms of the notation. In the literature the framework is often implicitly assumed, which can lead to difficulties. We will therefore define an appropriate framework - the application framework.

2.1 A general framework

In the book "Mathematics of Generalization" by Wolpert [89], there is an excellent overview of four commonly used frameworks. The overview is in chapter 5 with the describing title "The Relationship between PAC, the statistical Physics framework, the Bayesian Framework, and the VC Framework". Wolpert suggests a general framework called *Extended Bayesian Framework* or EBF (also see [90]), which unifies the four other frameworks.

We will discuss EBF only to give a flavor of the framework, since it will not be used in this dissertation. The EBF consists of an input space X, an output space Y, an example set T containing $X - Y$ pairs, a target function t, which is used to generate the example set, a predictor f, which is used to guess the target, and a cost (error) E. The different terms are connected by distributions, e.g. the learning algorithm is the probability of the predictor f given training set T or $P(f|T)$. An example set is generated with probability $P(T|t)$. The target t is considered to be the outcome of a stochastic variable with probability $P(t)$. The probability of t can also depend on the example set. This probability is denoted $P(t|T)$. It may

seem counter-intuitive, that the target function is conditional dependent on the training set, but it is reasonable when assuming that the target function is unknown. It is intuitive that there is higher probability that the unknown target function resembles the example set, and a lower probability that it does not resembles the example set. The error at a point x in input space is described by a distribution where the probability of error E is given by $P(E|t, f, x)$. So almost everything is expressed in outcomes of stochastic variables and associated distributions. This makes the EBF very broad and strong in the sense of generality of the results, but it also makes the notation somewhat cumbersome to work with. It is possible to "downgrade" EBF to a more limited framework, but there is some overhead in notation complexity.

2.2 The application framework

EBF is very broad and a good choice of framework, but for this dissertation not all of the broadness is required, so a framework well-suited for a less general view will be defined. It will be called the application framework. In EBF the view is on distributions, while the view in this dissertation will be on functions. The two views are interchangeable, e.g. a predictor in EBF is viewed as a distribution on output given an input, so a predictor f is described by the probability $P(y|f, x)$, while we will view a predictor as a function, and use the notation $y = f(x)$. If the predictor is stochastic, the view in EBF is the better. The predictor could still be viewed as a non-deterministic function and the probability of output y would be given by the stochastic variable $\underline{\mathbf{f_x}}$ so $P(y = f(x)) = P_{\underline{\mathbf{f_x}}}(y) = P(y|f, x)$. If the predictor is deterministic, the function view is the most suitable. In EBF the conditional probability of output y would be $P(y|f, x) = \delta_{(y,f(x))}$, where $\delta_{(\cdot,\cdot)}$ is Kronecker delta function.

The intuition of the application framework is

- The aim of machine learning is to learn a problem from examples.
- There is a fixed target function, the "truth", that has generated a finite set of problem examples. The example set could be distorted, e.g. by measurement noise.
- A machine learning method can be used to find an approximation of the target function based on the problem set. The machine learning method is a function that takes the example set as input and outputs an approximation. The machine learning function is not necessary deterministic.
- The approximation from a machine learning method is in the form of a deterministic function, a predictor, that maps input to output.

- The error of a predictor is calculated with an error function.
- The quality of the predictor is measured by the generalization error, which is the error on the target function.

The framework outlined above is application oriented, therefore the name "application framework".

Below is given definitions of the terms used above.

Definition 1 (Function Space)

Let g be a function with input $\vec{x}$ and let it depend on parameters $\vec{w}$, then the function space of g is the set $\mathcal{G}$ consisting of all possible functions $g(;\vec{w})$ obtained by varying $\vec{w}$ in the domain of $\vec{w}$. ■

The function space of $g(x;n) = x^n$, where $n \in \mathcal{N}$ is the set of all single term polynomial functions. The function space of a predictor is all the functions the predictor can learn.

Definition 2 (Predictor)

A predictor is a function f that takes an input $\vec{x}$ and generates an output $\vec{y}$ and depends on some parameters $\vec{w}$. The output of the predictor is denoted $\vec{y} = f(\vec{x})$. If the parameters are of importance the notation is $f(\vec{x};\vec{w})$. All predictors are assumed to be deterministic. A regression predictor is a predictor with continuous, and thereby metric, output. A classification predictor is a predictor with class density output: $f(\vec{x}) = \{f^{c_1}(\vec{x}), \ldots, f^{c_n}(\vec{x})\}$, where $f^{c_i}(\vec{x})$ is the output (the estimated probability) for class c_i. A classifier outputs the class label. ■

A regression neural network nn depends on the weights W, takes continuous input $\vec{x}$ and gives output $nn(\vec{x};W)$. A regression neural network can be turned into a classification predictor by post processing, e.g. using the SOFTMAX function [14]

$$f^c = \frac{\exp[nn^c]}{\sum_{c'} \exp[nn^{c'}]}, \tag{2.1}$$

where nn^c is the c'th output of the neural network. The SOFTMAX function ensures that the f^c's sum to one, and are positive or zero, so the f^c's can be regarded as probabilities

Definition 3 (Estimator)

An estimator is a parameterized function that takes as input an outcome of a stochastic variable and outputs the probability or density of the outcome. ■

As an example an estimator $p(y)$ can be the Normal density function parameterized by the mean μ and variance σ^2:

$$p(y;\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp[-\frac{(y-\mu)^2}{2\sigma^2}].$$

An estimator can be connected to a predictor in two ways: Firstly the output of a predictor can be one or more parameters of the estimator, e.g the predictor f outputs the mean, so the estimator $p(y)$ becomes $p(y;f,\sigma)$. Secondly a predictor can be identical to the estimator, so the parameters of the predictor are equal to the parameters of the estimator. In the discrete case the output of the predictor can be a vector corresponding to the class probability (see chapter 11 for an example). In the continuous case the outcome must be a part of the input to the predictor and the output is the density of the given outcome. When an estimator is connected to a predictor, the probability or density is conditioned on the input of the predictor, i.e. $p(y|\vec{x}) = p(y;f(\vec{x}),\sigma^2)$.

Definition 4 (Machine learning method)

A machine learning function L takes as input a training set T and gives as output a predictor: $f = L(T)$. Often a machine learning function depends on some parameters $\vec{w}$. If the parameters are important the notation will be $f = L(T,\vec{w})$. A group of machine learning functions are called iterative, because they refine a predictor repeatedly. This will be written

$$f^{new} = L(T, f^{old}).$$

■

Machine learning functions can be deterministic, e.g C4.5 [65] and splines (see section 6.4 in [48]) or stochastic. Back propagation (BP) is an example of an iterative, stochastic machine learning function.

Definition 5 (Meta machine learning methods)

A meta machine learning function MML takes as input a training set T and a machine learning method L. It produces as output a predictor: $f = MML(L,T)$. Often a meta machine learning function depends on some parameters $\vec{w}$. If the parameters are important the notation will be $f = MML(L,T,\vec{w})$. ■

A meta machine learning method can use several distinct machine learning methods, but that possibility is not investigated in this dissertation.

Definition 6 (Error and Error functions)

The error of a predictor is a measure of how wrong the predictor is. The error is given by an error function $\bar{E}$. The notation $\bar{E}$ is chosen to indicate that the error can be viewed as an average over a set of errors. An error function takes a predictor f and a function t as input and yields the error $\bar{E}(t, f)$.

One can speak of the error of a point in input space: $E(t(\vec{x}), f(\vec{x}))$ or some subset of input space. The subset can be defined by a set of input points, so one can speak of the error on the training set or the training set error $\bar{E}(T, f)$, where the training set T is viewed as a function that is undefined except for the input-output pairs in the set. The connection between the error $\bar{E}$ for a function and the error E for a point is

$$\bar{E}(t, f) = \langle E(t(\vec{x}), f(\vec{x})) \rangle_{\underline{\mathbf{X}}},$$

where the mean is with respect to the input density in the domain where t is defined, e.g the error for set T is

$$\bar{E}(T, f) = \sum_{(\vec{x},\vec{y}) \in T} E(\vec{y}, f(\vec{x}))P(\vec{x}).$$

The probability $P(\vec{x})$ can be assumed to be $\frac{1}{|T|}$ if the set T is sampled from the real density of the input.[1] *For a function t defined on the entire input space the error becomes*

$$\bar{E}(t, f) = \int_{\vec{x}} d\vec{x} p(\vec{x}) E(t(\vec{x}), f(\vec{x})).$$

■

Definition 7 (Generalization Error)

The generalization error of a predictor is $\bar{E}(t, f)$ where t is the target function. ■

[1] Let us assume that there are only two possible inputs x_1 and x_2. The real probability is $P'(x_1) = 0.8$, and $P'(x_2) = 0.2$. We sample ten times. On average the point x_1 will be sampled eight times and x_2 two times. If we set $P(x) = \frac{1}{10}$ then the examples with x_1 as input is represented by weight 0.8 as it should.

In all practical situations the target function t is not known, therefore the generalization error must be estimated. The error on the training set $\bar{E}(T, f)$ cannot be used since the predictor f often will be biased towards the training set. An independent test set D, which is not used for training can be used to estimate the generalization error. The estimated error becomes

$$\bar{E}(D, f) = \frac{1}{|D|} \sum_{(\vec{x},\vec{y})\in D} E(\vec{y}, f(\vec{x})).$$

There are many different error functions, but there are some characteristics almost all error functions have in common. For a given function g and predictor f there is a lower limit for the value of error function (often zero). The lower limit is reached when the predictor and the function are identical. Furthermore, for an error function to be sensible, a lower value must in some way express that the predictor has become a better approximation of the function.

Two commonly used error functions are the mean square error (MSE) and the Kullback-Leibler entropy error (KL).

Definition 8 (Mean Square Error (MSE))

The MSE is defined as

$$E(\vec{y}, f(\vec{x})) = \frac{1}{2K} \sum_{j}^{K} (y_j - f_j(\vec{x}))^2,$$

where K is the size of the output vector and j is the output index. ■

Note that MSE is defined only for metric values. It is therefore suitable for regression.

Definition 9 (Kullback-Leibler Error (KL))

The Kullback-Leibler error function is defined as

$$E(\vec{y}, f(\vec{x})) = \sum_{i} y^{c_i} \log(\frac{y^{c_i}}{f^{c_i}(\vec{x})}),$$

where y^{c_i} is the target probability for class c_i and $f^{c_i}(\vec{x})$ is the estimated probability for class c_i. ■

The KL error function is well-suited for classification.

Definition 10 (Early Stopping)

To avoid overfitting a part of the training set is not presented to the ML method, but is used to estimate the generalization error. This is the validation set. The ML method stops training when the estimated generalization error is minimal. ■

Since it is generally impossible to determine if the validation error has reached it minimum, training is often continued until the validation error has not attained a minimum for some time. The predictor at the time of lowest validation error is chosen. Training is not independent of the validation set, so the validation set is rightly considered a part of the training set.

Definition 11 (Combined Predictor)

A combined predictor is a predictor F that is defined in terms of a finite group of predictors $\vec{f}$ and a combination rule. The combining rule is a function B that takes the predictors $\vec{f}$ as input and gives the combined predictor as output $F = B(\vec{f})$. ■

Several kinds of combined predictors will be discussed in this dissertation. Among them are the *linear average predictor* (LAP) combined predictor

$$F(\vec{x}) = \sum_i \alpha_i f_i(\vec{x}),$$

and the logarithmic opinion pool (LOP) combined predictor for classification. For a class c the combined predictor is given by

$$F^c(\vec{x}) = \frac{1}{Z} \exp[\sum_i \alpha_i \log f_i^c(\vec{x})].$$

The normalization factor Z is given by $Z = \sum_c F^c(\vec{x})$. The α's are positive and sum to one.

Definition 12 (Average Predictor)

An average predictor is a predictor $\bar{f}$ defined in terms of a finite or infinite group of predictors $\vec{f}$ and an average measure. The average measure is a function B that takes the predictors $\vec{f}$ as input and gives the average predictor as output $\bar{f} = B(\vec{f})$. The average measure often involves an average from a distribution over the set of predictors: A discrete distribution in case of a finite set of predictors and a continuous distribution in case of a infinite set of predictor. ■

Let $\langle \cdot \rangle_{\underline{\mathbf{F}}}$ be a mean operator of the form $\sum_i P(f_i)$ in the discrete case and $\int df p(f)$ in the continuous case. The linear average predictor (LAP) is given by

$$\bar{f} = \langle f \rangle_{\underline{\mathbf{F}}},$$

and the logarithmic opinion pool (LOP) average predictor for classification. For a class c the LOP combined predictor is given by

$$\bar{f}^c(\vec{x}) = \frac{1}{Z} \exp[\langle \log f^c \rangle_{\underline{\mathbf{F}}}].$$

The normalization factor Z is given by $Z = \sum_c \bar{f}^c(\vec{x})$.

Note the similarities between combined predictor (see definition 11) and average predictor. Besides that the combined predictor is only defined for a finite group of predictors, the difference is only in the view.

In appendix A is listed a number of commonly used symbols and notation.

Chapter 3

Bias/variance and ambiguity

The success of a machine learning method can be expressed by the generalization error $\bar{E}(t, f)$. The lower the generalization error, the better the predictor approximates the target function, and predicting the target function is the true aim of machine learning.

The value of the generalization error is therefore of great importance. It is possible to decompose the generalization error into two terms with different "feel". Assume we have a training set generated with noise from target function t. We have a machine learning function that generates a simple predictor, e.g. a neural network with few weights, and a machine learning function that generates a complex predictor, e.g. a neural network with many weights. Furthermore, assume that the function space of the simple predictor does not contain t, while the function space of the complex predictor does. The predictors are trained until the training error is non-decreasing. It is well known that the complex predictor can have a generalization error of the same size as the simple predictor. The problem is that the complex predictor *overfits*.

In figure 3.1 a target function (dashed line) and a training set (points) are illustrated. A complex predictor (left graph) and a simple predictor (right graph) are also illustrated. As can be seen the way the predictors err is different in nature. The complex predictor is confused by the noise,[1] while the simple predictor cannot approximate the target function well.

Let the training of the predictors be repeated a number of times. For each

[1] Even if the training set is noise-free, a complex predictor can overfit. In figure 3.1 the complex predictor is a spline (see section 6.4 in [48]) and is well-behaved between points. A polynomial with the same degree as the number of points, can go through all the points as the spline, but the value between points can be very large or small. This would also be the case for a noise-free training set. The commonly used machine learning method, Neural networks with back-propagation, can also overfit on noise-free training sets.

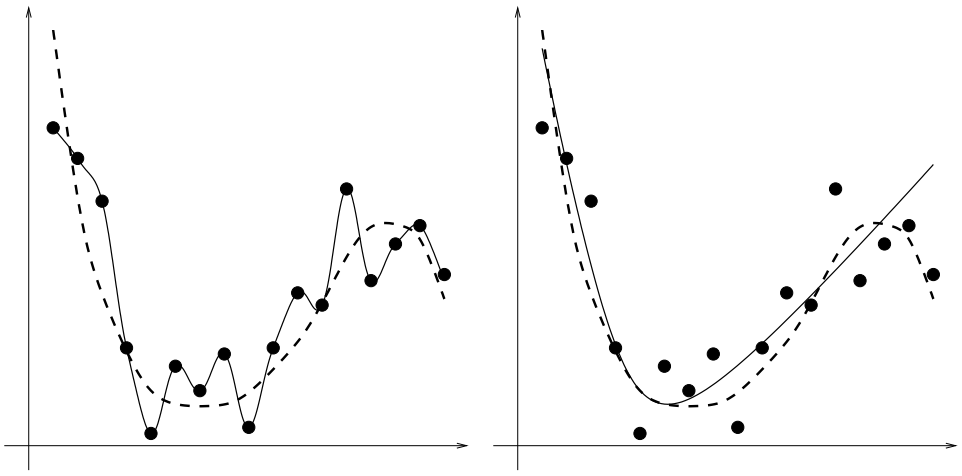


Figure 3.1: A complex predictor (left full line) and a simple predictor (right full line) trained on the same training set (points) generated by a target function (dashed line).

repetition a new training set is generated from the target function with noise. The simple predictor will give approximately the same result for each training session, and will for a given input be biased with respect to the target function. The complex function will on average predict the target function, but will vary around it. The way the simple predictors errs is called *bias* while the way the complex predictors errs is called *variance*.

This can be expressed mathematically for the MSE: Assume a ML function L and a target function t. The target function can generate different training sets, e.g. by adding noise with zero mean[2], varying the number of examples, or sampling the input space for input examples at random. Let $P(T)$ be the probability that training set T is generated and let $\underline{\mathbf{T}}$ be the corresponding stochastic variable. Let f_T denote $L(T)$. The predictor f_T is the outcome of the stochastic variable $L(\underline{\mathbf{T}})$, so it is possible to talk about the mean of f_T i.e. $\langle f_T \rangle_{\underline{\mathbf{T}}}$. Because this is an important concept the mean of f_T will be denoted $\bar{f}$ and called the *average predictor*. The mean of the generalization error with respect to $\underline{\mathbf{T}}$ is $\langle \bar{E}(t, f_T) \rangle_{\underline{\mathbf{T}}}$. This is a sensible measure of error because it expresses what we on average can expect the generalization error to be. It can be written (assuming one-dimensional output and omitting

[2]The noise can be explained by a stochastic target function, but since all target functions in this dissertation are assumed to be deterministic, the noise is associated with the process of generating a training set, e.g. by noise in the measuring process. The assumption about zero mean noise is made for practical reasons. It is desirable that the mean of the training sets equals the target function. Furthermore, a non-zero mean noise will be oblivious for the machine learning function without additional information, and is therefore uninteresting.

input for simplicity)

$$\langle \bar{E}(t, f_T) \rangle_{\underline{\mathbf{T}}} = \langle [t - f_T]^2 \rangle_{\underline{\mathbf{X}},\underline{\mathbf{T}}} \tag{3.1}$$

Using $\langle 2\bar{f}^2 - 2\bar{f}f_T \rangle_{\underline{\mathbf{XT}}} = 0$ and $\langle 2tf_T \rangle_{\underline{\mathbf{XT}}} = \langle 2t\bar{f} \rangle_{\underline{\mathbf{X}}}$ yields

$$\begin{aligned} \langle \bar{E}(t, f_T) \rangle_{\underline{\mathbf{T}}} &= \langle [t - f_T]^2 \rangle_{\underline{\mathbf{X}},\underline{\mathbf{T}}} \\ &= \langle t^2 + f_T^2 - 2tf_T \rangle_{\underline{\mathbf{XT}}} + \langle 2\bar{f}^2 - 2\bar{f}f_T \rangle_{\underline{\mathbf{XT}}} \\ &= \langle t^2 + \bar{f}^2 - 2t\bar{f} \rangle_{\underline{\mathbf{X}}} + \langle \bar{f}^2 + f_T^2 - 2\bar{f}f_T \rangle_{\underline{\mathbf{XT}}} \\ &= \langle [t - \bar{f}]^2 \rangle_{\underline{\mathbf{X}}} + \langle [\bar{f} - f_T]^2 \rangle_{\underline{\mathbf{XT}}} \end{aligned} \tag{3.2}$$

This decomposition can be found in e.g. [27]. Both terms in (3.2) are errors. The first $\langle [t - \bar{f}]^2 \rangle_{\underline{\mathbf{X}}} = \bar{E}(t, \bar{f})$ is the bias, which will be denoted $Bias(t, \bar{f})$. The second $\langle [\bar{f} - f_T]^2 \rangle_{\underline{\mathbf{XT}}} = \langle \bar{E}(\bar{f}, f_T) \rangle_{\underline{\mathbf{T}}}$ is the variance, which will be denoted $Var(f)$. Because both expressions are errors in themselves they obey some nice properties. Both are greater than or equal to zero. Furthermore, $Var(f)$ only depends on the target function through the mean with respect to the training set.

To sum up the error can be written in terms of bias and variance

$$\langle \bar{E}(t, f_T) \rangle_{\underline{\mathbf{T}}} = Bias(t, \bar{f}) + Var(f) \tag{3.3}$$

To get a better understanding of the bias/variance let us return to the simple and the complex predictor. The complex predictor can approximate the different training sets very well, and since the mean of the training sets equals the target function, the average of the complex predictors will be very close to t, so $Bias(t, \bar{f})$, that measures the difference between the target function and the average predictor will be very small. Each of the complex predictors differs to some degree from the average predictor, and will on average give a substantial contribution to the mean of error. This is what $Var(f)$ measures. The situation is the opposite for the simple predictors. For almost all training set generated from the target function the simple predictor will be very similar to the U-graph in figure 3.1, so the average predictor for the simple predictors will also be similar to figure 3.1, and $Var(f)$ will be close to zero. The average predictor will not resemble the target function very well, so $Bias(t, \bar{f})$ will contribute significantly to the mean of the error.

3.1 The bias/variance dilemma

From the first studies of bias and variance it has been thought that there is a trade-off between bias and variance (e.g. see [86]). This is supported by

empirical experiments, where the generalization error, bias and variance are plotted against the "complexity" of a predictor. The "complexity" can be the number of hidden nodes in a neural network, or the number of iteration a neural network has been trained. Both is believed to increase "complexity" of a predictor. A typical graph of error vs. "complexity" can be found in figure 3.2

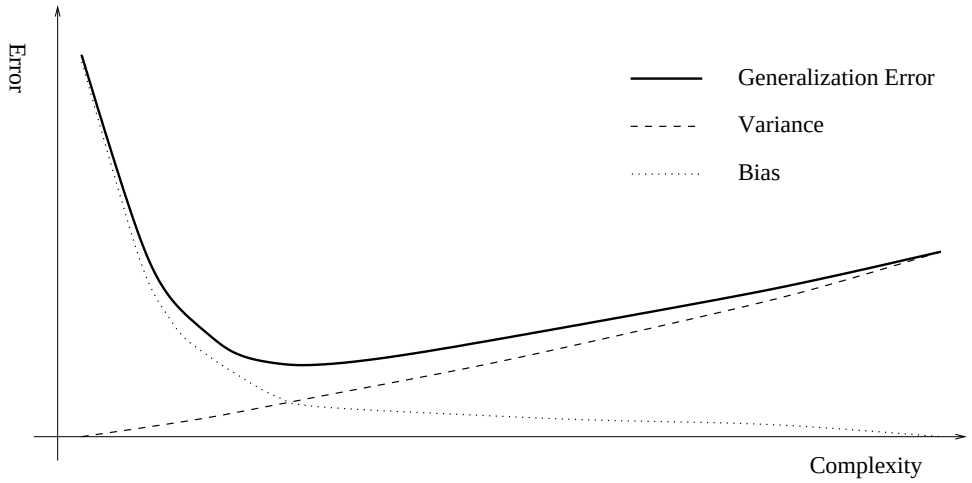


Figure 3.2: Common behavior of generalization error, bias and variance.

This apparent trade-off is called the "*Bias/Variance Dilemma*" [27].

Let the increase in complexity come from iterations by an iterative machine learning function. In the beginning of training the predictor will resemble the simple predictor and have a large bias. If the training is allowed to continue and the predictor has sufficiently large function space the predictor will resemble the complex predictor and have a large variance. The generalization error will drop in the beginning of training where the predictor learns, then reach a minimum and begin to rise. This is overfitting, where the predictor remembers the training set. The optimal predictor will be somewhere in between, where both variance and bias are small. A predictor close to the optimal predictor can be found by estimating the generalization error on a validation set, and stopping the training when the error on the validation set is smallest. This is *early stopping*.

The bias/variance dilemma is also applicable in connection with choosing the optimal architecture. Choose an architecture for the predictor too small and it can be trained forever without the generalization error decreases because the bias continues to be high. Choose an architecture too large and the predictor can overfit, resulting in high variance. Again the optimal predictor is somewhere in between.

This suggests that the bias error and variance error are equal evils. This

is not so. It is possible that a predictor that "knows too little" and a predictor that "knows too much" have comparable generalization errors, but the predictor that "knows too much" has the crucial information built in, so it is potentially possible to extract it, while the predictor that "knows too little" has "forgotten" information, which therefore cannot be extracted. Meta machine learning methods like Bagging [9, 10, 92, 66, 39] are designed to extract information from predictors that "know too much".

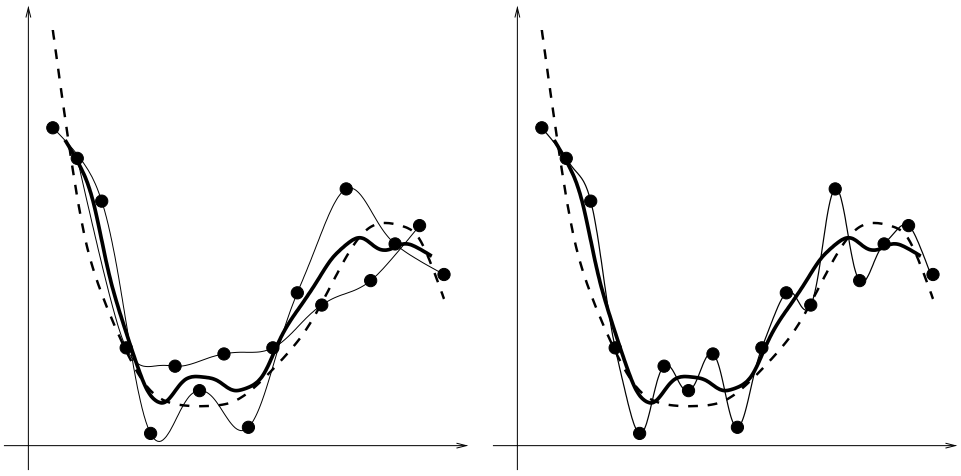


Figure 3.3: Two predictors (thin lines in left graph) trained on different training sets combined into one predictor (thick line). The combined predictor is a better approximation of the target function (dashed line) than a predictor trained on the entire training set (thin line in right graph).

An example of how to extract information from predictors that overfit is given in figure 3.3. In the left graph of the figure the thin lines depict two predictors trained on disjoint subsets of the training set from figure 3.1. Both of them are poor approximations of the target function. The thick line is the uniform linear combination of the two predictors. This combined predictor is a much better approximation of the target function than the two predictors used to form the combination. Furthermore, the combined predictor is a better approximation than the predictor trained on the entire training set (see right graph of figure 3.3). The drop in generalization error is accomplished by approximating the average predictor (see equation 3.2) and thereby lowering the variance. This indicates that variance is a lesser evil than bias, since variance can be reduced.

It should also be noted that the bias/variance decomposition in some frameworks does not hold, e.g. the EBF (see section 2.1 or [90]). If the predictor and the target function are dependent, a covariance term is introduced. In this dissertation it is assumed that the target function is deterministic, and by definition of covariance the predictor and the target function are inde-

pendent.

3.2 Ambiguity decomposition

The ambiguity decomposition is defined in terms of an error function and a finite set of predictors (an ensemble), that is combined to form a combined predictor F. Assume that F is a linear combination of M predictors f_i

$$F = \sum_{i=1}^{M} \alpha_i f_i.$$

Let the coefficients α_i sum to one, then the coefficients can be viewed as the probability of the predictors. The corresponding stochastic variable is called $\underline{\mathbf{F}}$, so $F = \langle f \rangle_{\underline{\mathbf{F}}}$. The combined predictor F is called a *linear average predictor* (LAP). Assume that the error function $\bar{E}$ is the MSE, then the error of F with regard to an arbitrary function d can be decomposed using $\sum_i \alpha_i = 1$. Assuming one dimensional output and omitting input for simplicity we have

$$\bar{E}(d, F) = \langle [d - f]^2 \rangle_{\underline{\mathbf{F}},\underline{\mathbf{X}}} - \langle [f - F]^2 \rangle_{\underline{\mathbf{F}},\underline{\mathbf{X}}} \tag{3.4}$$

Proof

$$\begin{aligned}
\bar{E}(d, F) &= \langle [d - F]^2 \rangle_{\underline{\mathbf{X}}} \\
&= \langle d^2 + F^2 - 2dF \rangle_{\underline{\mathbf{X}}} \\
&= \langle d^2 + F^2 - 2d \sum_i \alpha_i f_i + \sum_i \alpha_i f_i^2 - \sum_i \alpha_i f_i^2 + F^2 - F^2 \rangle_{\underline{\mathbf{X}}} \\
&= \langle d^2 - 2d \sum_i \alpha_i f_i + \sum_i \alpha_i f_i^2 - \sum_i \alpha_i f_i^2 + 2F \sum_i \alpha_i f_i - F^2 \rangle_{\underline{\mathbf{X}}} \\
&= \langle d^2 + \sum_i \alpha_i f_i^2 - 2d \sum_i \alpha_i f_i - [\sum_i \alpha_i f_i^2 + F^2 - 2F \sum_i \alpha_i f_i] \rangle_{\underline{\mathbf{X}}} \\
&= \langle \sum_i \alpha_i (d - f_i)^2 - \sum_i \alpha_i (f_i - F)^2 \rangle_{\underline{\mathbf{X}}} \\
&= \langle [d - f]^2 \rangle_{\underline{\mathbf{F}},\underline{\mathbf{X}}} - \langle [f - F]^2 \rangle_{\underline{\mathbf{F}},\underline{\mathbf{X}}}.
\end{aligned}$$

In the third line of the proof we have added two pairs of terms $\sum_i \alpha_i f_i^2 - \sum_i \alpha_i f_i^2$ and $F^2 - F^2$, that sums to zero.

This decomposition for a linear average predictor (LAP) is due to Krogh & Vedelsby [51]. The first term on the right hand side in (3.4) is the mean (with respect to $\underline{\mathbf{F}}$) of the error of the predictors f_i. The second term measures the error between F and the f_i's. This is called the ambiguity and

is denoted A. Note that A does not depend directly on d, but only on F and f_i. This is a very desirable property. (see sections 5.9 and 4.4.1 for general expressions for ambiguity, and chapter 11 for an application of ambiguity). The decomposition can now be written in this short form

$$\bar{E}(d, F) = \langle \bar{E}(d, f_i) \rangle_{\underline{\mathbf{F}}} - A. \tag{3.5}$$

Since A is an error it obeys $A \geq 0$, which yields $\bar{E}(d, F) \leq \langle \bar{E}(d, f_i) \rangle_{\underline{\mathbf{F}}}$. In words: the error of the LAP is less than or equal to the mean of the error for the members of the ensemble. A result also noted in [51] and [63].

3.3 Bias/variance and ambiguity

We have encountered two different decompositions for MSE. The bias/variance decomposition in chapter 3 and the ambiguity decomposition in section 3.2. These are mathematically equivalent as can be seen by a minor rewriting of the ambiguity decomposition. First the bias/variance

$$\langle \bar{E}(d, f_T) \rangle_{\underline{\mathbf{T}}} = \bar{E}(d, \bar{f}) + \langle \bar{E}(\bar{f}, f_T) \rangle_{\underline{\mathbf{T}}}. \tag{3.6}$$

Then the ambiguity

$$\langle \bar{E}(d, f_i) \rangle_{\underline{\mathbf{F}}} = \bar{E}(d, F) + \langle \bar{E}(F, f_i) \rangle_{\underline{\mathbf{F}}}. \tag{3.7}$$

So $Bias(d, \bar{f})$ is equivalent to the error of the LAP $\bar{E}(d, F)$ and $Var(f)$ is equivalent to ambiguity. If we have a decomposition for an arbitrary error function E on the form:

$$\langle \bar{E}(t, f) \rangle = \bar{E}(t, \mathcal{F}) + \langle \bar{E}(\mathcal{F}, f) \rangle, \tag{3.8}$$

where $\mathcal{F}$ is the "average predictor", which only depends on f, and the term $\langle \bar{E}(\mathcal{F}, f) \rangle$ is independent of the target function t, we have both a bias/variance and an ambiguity decomposition.

The equivalence is more than mathematical, as the equivalence can and have been put to practical use, e.g. the meta machine learning method Bagging (see [9] and figure 3.3) is based upon the equivalence between the LAP F and the average predictor $\bar{f}$. The rational behind Bagging is that predictors outputted by a machine learning method have a low bias if the predictors are complex enough. They will generally have a large variance, but if one could find the average predictor, then the variance of the average predictor will be zero. It is impossible to find $\bar{f}$ exactly since the mean is over all possible training sets, but if the training set is resampled an approximation can be made by averaging over the group of resampled training set. This is a LAP and we have reached the ambiguity decomposition.

For ambiguity to be of practical use, it must obey some properties:

- Ambiguity must always be positive or zero.
- Ambiguity must measure the average difference between the set of predictors and the combined predictor.
  If all predictors are equal the ambiguity should be zero.
- Ambiguity must not depend on the target function.
  This also means that ambiguity must not depend on the training set.

In chapter 5 a general decomposition of errors into bias/variance with target independent variance is presented. Due to the equivalence between bias/variance decomposition and ambiguity decomposition, a general expression for ambiguity, that obeys the properties listed above, is implicitly presented.

Chapter 4

Bias/variance decomposition in literature

The bias/variance decomposition has been studied intensively in the literature, mainly in three different ways, or combinations of those: Empirical studies [27, 12, 25, 92, 72], limited decompositions with desirable properties [10, 39], and general decompositions without guarantee of all of the desirable properties, e.g. target independent variance [44, 10]. In this overview of the literature we will concentrate on the last two. Bias/variance decomposition for both regression and classification will be discussed.

Before the overview, we present a very simple bias/variance decomposition in section 4.1 that exemplifies some of the issues.

4.1 A trivial bias/variance decomposition

The fundamental issue is to decompose the generalization error into two terms

$$\text{Generalization Error} \;=\; \text{Bias} \;+\; \text{Variance}.$$

The bias is attributed the systematic part of the error, while the variance is attributed the stochastic part of the error. The stochastic aspect comes into the picture, since the generalization error is defined as a mean of an error function over different stochastic variables, e.g. the target function, the predictor, and the input. Let an arbitrary error function be $E(t(x), f(x))$, then the generalization error can be defined as $\langle E(t(x), f(x))\rangle_{\underline{\mathbf{T}},\underline{\mathbf{F}},\underline{\mathbf{X}}}$. This is how it is defined in [90]. Often one or more of the stochastic variables are omitted, depending on the perspective. Most bias/variance decomposition involves an average predictor $\bar{f}$ defined on the distribution of predictors $\underline{\mathbf{F}}$.

With an average predictor a general bias/variance decomposition can be defined

$$\langle E(t,f)\rangle = [\langle E(t,f)\rangle - \langle E(\bar{f},f)\rangle] + \langle E(\bar{f},f)\rangle.$$

For notational convenience we omit the dependency on input and the stochastic variables. The variance is $\langle E(\bar{f},f)\rangle$ and the "bias" is $\langle E(t,f)\rangle - \langle E(\bar{f},f)\rangle$. This decomposition holds for all error functions and even for all definitions of average predictors. Also, the variance is independent of the target function. But as a general decomposition it is uninteresting, because in most cases the "bias" is meaningless. Only in specialized cases, e.g. for the MSE function and the KL error function, can the "bias" be given a reasonable interpretation.

A number of such decompositions can be made by rearranging terms. In [44] one of the more useful can be found, and some useful concepts are defined (See appendix C and section 4.2).

4.2 James 1996 [44]

The decomposition in [44] is due to James & Hastie. Note that the terms in the decomposition below are not called bias and variance in [44], but bias-*effect* and variance-*effect*. In [44] separate bias- and variance terms are also defined, but they do not form a general decomposition. We will regard the bias-effect/variance-effect decomposition as a bias/variance decomposition.

The decomposition is given by

$$\langle E[t,f]\rangle = \mathrm{var}(t) + BE(t,Sf) + VE(t,f),$$

where

$$\mathrm{var}(t) = \langle E[t,St]\rangle,$$

is the intrinsic noise,

$$BE(t,Sf) = \langle E[t,Sf] - E[t,St]\rangle,$$

is the bias-effect, and

$$VE(t,f) = \langle E[t,f] - E[t,Sf]\rangle$$

is the variance-effect. The operator S is called the *systematic mean* (see definition 15 in appendix C), and is defined as

$$Sz = \underset{y}{\mathrm{argmin}}\ \langle g(y,z)\rangle.$$

We assume that $Sf = \mathrm{argmin}_t \langle E(t,f)\rangle_{\underline{\mathbf{F}}}$ and $St = \mathrm{argmin}_f \langle E(t,f)\rangle_{\underline{\mathbf{T}}}$, even though it is not clear from [44]. The decomposition holds for any definition

of Sf and St, but is most meaningful with the above definition, e.g. the intrinsic noise $\text{var}(t) = \langle E[t, St] \rangle$ is with the definition of St the lowest error obtainable, because

$$\forall f : \langle E[t, f] \rangle_{\underline{\mathbf{T}}} \geq \langle E[t, St] \rangle_{\underline{\mathbf{T}}}.$$

This also implies that the bias-effect always is greater than or equal to zero. Furthermore, Sf can be regarded as the average predictor, so the bias-effect can be regarded as the average increase in error when changing prediction from the best prediction St to the average predictor Sf. Also the variance-effect can be meaningful interpreted. It is the mean increase in error when changing the prediction from the average predictor to another predictor with regard to the distributions over predictors. Unfortunately there is no guarantee that variance-effect is positive, or that the variance-effect is independent of the target function.

4.3 Zhu [96]

The decomposition in [96] is due to Zhu. It is not a bias/variance decomposition as it is presented. It is a decomposition of the mean generalization error into intrinsic noise (irreducible error) and reducible error. First the decomposition is presented as it is given, then it is shown how to reinterpret the decomposition, so it becomes a bias/variance decomposition in the usual sense. The author claims that the decomposition is a generalization of the already very general decomposition in [39] (see section 4.4). This claim is not correct (see section 4.5 for elaboration) even with the reinterpretation.

The decompositions in e.g. [44, 27] are based directly on predictors (see definition 2), while the decomposition in [96] is based on estimators (see definition 3). The relationship between input and output for an estimator is not described by a function, but by a distribution. The distribution can be discrete or continuous. We will discuss the result in [96] in terms of a continuous distribution, so we have a density $p(x, y)$ that describes the relationship between input and output.

Let $z = \{z_1, \ldots, z_n\}$ be an example set sampled from the distribution p, where $z_i = (x_i, y_i)$. The distribution p is described by the stochastic variables $\{\underline{\mathbf{X}}, \underline{\mathbf{Y}}\}$, so $p(x, y)$ is the likelihood of $\underline{\mathbf{X}} = x$ and $\underline{\mathbf{Y}} = y$. The estimator is the density $q(x, y)$. The goal is to find the most accurate estimator compared with p. In [96] the inaccuracy of an estimator is measured with the *information loss* function $D_\gamma(p, q)$. The scalar $\gamma \in [-1, 2]$ spans the family of information loss functions. The parameters p and q must obey $\int p < \infty, \int q < \infty$ and not necessarily $\int p = \int q = 1$. The family of infor-

mation loss functions is defined by

$$D_\gamma(p,q) = \begin{cases} \int_{x,y} \frac{\gamma p + (1-\gamma)q - p^\gamma q^{1-\gamma}}{\gamma(1-\gamma)} & \gamma \in [-1,2]/\{0,1\} \\ \int_{x,y} q - p + p\log\frac{p}{q} & \gamma = 1 \\ \int_{x,y} p - q + q\log\frac{q}{p} & \gamma = 0 \end{cases} . \qquad (4.1)$$

The information loss function has some desirable properties:

- $D_\gamma(p,q) = D_{1-\gamma}(q,p) \geq 0$
- $D_\gamma(p,q) = 0 \Leftrightarrow p = q$

An estimator $q(x,y)$ depends on the example set z. This we will signify with the notation $q(x,y|z)$ or $q(z)$. The mean generalization error of an estimator is defined as

$$E(q) = \int_z P(z)\langle D(p,q(z))\rangle_z$$

The notation $\langle g(z)\rangle_z$ is the mean operator with regard to the density $\pi(p|z)$, so $\langle g(p)\rangle_z = \int_p \pi(p|z)g(p)$. It is a somewhat unusual to take the mean with regard to $\pi(p|z)$, since p is the "truth". The intuition is: we have a data set z, then $\pi(p|z)$ expresses the density of the different p's under the condition that they generated z. So the mean is over the possible "truths" that could generate z. The notation $\langle\cdot\rangle_z$ from [96] is also unusual. The mean is not over z, but for a fixed z.

If the function I is defined by $D_\gamma(p,q) = \int_{x,y} I(p,q)$ then the mean generalization error of q can be stated as three different means over I

$$E(q) = \langle\langle\langle I(p,q(z))\rangle_{(\underline{\mathbf{X}},\underline{\mathbf{Y}})}\rangle_z\rangle_{\underline{\mathbf{Z}}}$$

The inner mean is over $(\underline{\mathbf{X}},\underline{\mathbf{Y}})$ with uniform density, and the outer mean is over all the possible example sets.

Below is a list of some of the members of the information loss family.

- The Hellinger distance:

$$D_{\frac{1}{2}}(p,q) = \int_{x,y} (\sqrt{p} - \sqrt{q})^2$$

  Note that this can be transformed into the mean square error. let $p' = p^2$ and $q' = q^2$ then $D_{\frac{1}{2}}(p',q') = \int_{x,y}(p-q)^2$. That can be done because $\int p' < \infty \Leftrightarrow \int p < \infty$.

- Kullback-Leibler deviation:

$$D_1(p,q) = D_0(q,p) = \int_{x,y} (q - p + p\log\frac{p}{q}).$$

  If $\int p = \int q = 1$ then $D_1(p,q) = D_0(q,p) = \int p\log\frac{p}{q}$.

- The χ^2 deviation:

$$D_2(p,q) = D_{-1}(q,p) = \int_{x,y} \frac{(p-q)^2}{2q}.$$

The information loss function can be decomposed by the γ-mean. Before the decomposition is shown, more definitions are needed.

- γ-coordinate: $l_\gamma(p) = \frac{p^\gamma}{\gamma}$, $l_0(p) = \log(p)$.
- γ-potential: $\Psi_\gamma(p) = \int \frac{p}{(1-\gamma)}$, $\Psi_1(p) = \int p\log(p)$.
- γ-mean ($\hat{p}_z$): $l_\gamma(\hat{p}_z) = \langle l_\gamma(p)\rangle_z$.

  For $\gamma \neq 0$ the γ-mean $\hat{p}_z$ is given by $(\langle p^\gamma\rangle_z)^{\frac{1}{\gamma}}$. For $\gamma = 0$ the γ-mean $\hat{p}_z$ is given by $\exp\langle\log(p)\rangle_z$. Note that $\int p = 1$ generally does not imply $\int \hat{p}_z = 1$[1].
- γ-variance ($\langle D_\gamma(p,\hat{p}_z)\rangle_z$): $\langle\Psi_\gamma(p)\rangle_z - \Psi_\gamma(\hat{p}_z)$.

  For $\gamma \notin \{0,1\}$ the γ-variance $\langle D_\gamma(p,\hat{p}_z)\rangle_z$ is given by $\langle\int_{x,y}\frac{p}{1-\gamma}\rangle_z - \int_{x,y}\frac{\hat{p}_z}{1-\gamma}$. If $\int p = 1$ then $\langle D_\gamma(p,\hat{p}_z)\rangle_z = \frac{1}{1-\gamma} - \int_{x,y}\frac{\hat{p}_z}{1-\gamma} \neq 0$. For $\gamma \in \{0,1\}$ the γ-variance is given by $\langle D_\gamma(p,\hat{p}_z)\rangle_z = \langle\int_{x,y} p\log(p)\rangle_z - \int_{x,y}\hat{p}_z\log(\hat{p}_z)$.

The decomposition is

$$\langle D_\gamma(p,q)\rangle_z = \langle D_\gamma(p,\hat{p}_z)\rangle_z + D_\gamma(\hat{p}_z,q). \tag{4.2}$$

Proof:

$$\langle D_\gamma(p,\hat{p}_z)\rangle_z + D_\gamma(\hat{p}_z,q) =$$
$$\int_p \pi(p|z)\int_{x,y}\frac{\gamma p + (1-\gamma)\hat{p}_z - p^\gamma\hat{p}_z^{1-\gamma}}{\gamma(1-\gamma)} + \int_{x,y}\frac{\gamma\hat{p}_z + (1-\gamma)q - \hat{p}_z^\gamma q^{1-\gamma}}{\gamma(1-\gamma)} =$$
$$\int_{x,y}\int_p \pi(p|z)\frac{\gamma p + (1-\gamma)\hat{p}_z - p^\gamma\hat{p}_z^{1-\gamma} + \gamma\hat{p}_z + (1-\gamma)q - \hat{p}_z^\gamma q^{1-\gamma}}{\gamma(1-\gamma)} =$$
$$\int_{x,y}\int_p \pi(p|z)\left(\frac{\gamma p + (1-\gamma)q - p^\gamma q^{1-\gamma}}{\gamma(1-\gamma)} + \frac{p^\gamma q^{1-\gamma} + \hat{p}_z - p^\gamma\hat{p}_z^{1-\gamma} - \hat{p}_z^\gamma q^{1-\gamma}}{\gamma(1-\gamma)}\right) =$$
$$\langle D_\gamma(p,q)\rangle_z + \int_{x,y}\frac{\hat{p}_z^\gamma q^{1-\gamma} + \hat{p}_z - \hat{p}_z^\gamma\hat{p}_z^{1-\gamma} - \hat{p}_z^\gamma q^{1-\gamma}}{\gamma(1-\gamma)} =$$
$$\langle D_\gamma(p,q)\rangle_z$$

[1] Let $\langle p^\gamma\rangle_z$ be a discrete average over two densities p_1 and p_2 each with probability $\frac{1}{2}$. Furthermore, let $p_2 = p_1 + \delta$, where δ is small and integrates to zero over input and output. Then $\langle p^\gamma\rangle_z$ is approximately $p_1(1 + (\frac{\gamma}{2}\frac{\delta}{p_1})^{\frac{1}{\gamma}})$ which is not normalized.

There are two problems with the decomposition in (4.2). The γ-mean is defined on a mean with regard to the density of p - the "truth". Furthermore, the γ-mean is not normalized even if the p is normalized.

Let us begin with the definition of the γ-mean. It is not equivalent with the average predictor, since it is defined on the distribution of p - the "truth" (the target). The γ-mean can be regarded as the average target for a fixed example set z. The decomposition in (4.2) is then a decomposition of the generalization error into an intrinsic noise term (the γ-variance) and the reducible part of the generalization error $D_\gamma(\hat{p}_z, q)$. This cannot be seen as a bias/variance decomposition.

By interchanging the meaning of p and q we get a new interpretation of the decomposition, so it can be seen as a bias/variance decomposition. The γ-variance must be equivalent with the variance, and the term $D_\gamma(\hat{p}_z, q)$ must be the bias. The former estimator q must be interpreted as the "truth" or the target density, while p must be interpreted as the estimator. That is in agreement with the definition of the γ-mean. With the new interpretation the γ-mean is defined on a mean with regard to the density of p - the predictors. Only the definition of the average generalization error must be changed. Now it is p that depends on z:

$$E'(p) = \int_z P(z) \langle D(p(z), q) \rangle_z$$

The second problem is that $\int p = 1$ generally does not imply $\hat{p}_z = 1$. Furthermore, the decomposition does not hold for the normalized γ-mean $\hat{p}'_z = \frac{\hat{p}_z}{\int \hat{p}_z}$.

4.4 Heskes 1998 [39, 40]

The decomposition in [39, 40] is due to Tom Heskes. It is based on the Kullback-Leibler (KL) divergence, that compares density or probability functions. If the density or probability functions are predictors the KL divergence can be viewed as an error function (see chapter 11 or section 7.1.4), but if the density or probability functions are estimators parameterized by predictors (see definition 3), the KL divergence is not in it self an error function, but is a way of deriving error functions from density or probability functions.

The combination rule is the logarithmic opinion pool (LOP). We repeat the definition of the LOP from definition 12:

The logarithmic opinion pool is an average measure of a (possibly infinite) group of estimators $\hat{p}$. The estimators are probability or density functions.

The group of estimators have associated a mean operator $\langle \cdot \rangle_{\underline{\hat{\mathbf{P}}}}$[2]. *The LOP for an outcome* y *is given by*

$$\bar{p}(y) = \frac{1}{Z} \exp \langle \log \hat{p}(y) \rangle_{\underline{\hat{\mathbf{P}}}}, \tag{4.3}$$

where Z *is a normalization factor satisfying for the continuous case* $\int dy \bar{p}(y) = 1$ *and the discrete case* $\sum_i \bar{p}(y_i) = 1$

The LOP is intimately connected to the Kullback-Leibler (KL) error

$$E_{KL}(t, p) = \sum_i t(y_i) \log \left(\frac{t(y_i)}{p(y_i)} \right), \tag{4.4}$$

where $\vec{y} = \{y_1, \ldots, y_n\}$ is the vector of classes, and t is the target probability function. In the continuous case t and p are continuous density functions of y and the summation in (4.4) is replaced by integration over y. Note that the KL error always is positive or zero. The proof is by Jensen's inequality

$$\sum_i \lambda_i \log(x_i) \leq \log(\sum_i \lambda_i x_i),$$

where $\lambda_i \geq 0$, and $\sum_i \lambda_i = 1$. Setting $\lambda_i = t(y_i)$ and $x_i = \frac{p(y_i)}{t(y_i)}$ we get

$$\begin{aligned}
&\sum_i t(y_i) \log \left(\frac{p(y_i)}{t(y_i)} \right) \leq \log \left(\sum_i t(y_i) \frac{p(y_i)}{t(y_i)} \right) \Leftrightarrow \\
&\sum_i t(y_i) \log \left(\frac{p(y_i)}{t(y_i)} \right) \leq \log \sum_i p(y_i) \Leftrightarrow \\
&\sum_i t(y_i) \log \left(\frac{p(y_i)}{t(y_i)} \right) \leq 0 \Leftrightarrow \\
&\sum_i t(y_i) \log \left(\frac{t(y_i)}{p(y_i)} \right) \geq 0
\end{aligned}$$

The variance is defined analogously to the definition of variance in [44] as the smallest mean distance measured by the KL error function between the estimators $\hat{p}(y)$ and an average estimator. This definition of the average estimator is equal to the systematic mean (see definition 15 in appendix C). There is a normalization constraint on the average estimator, since it is a probability function. We arrive at

$$\text{var}(\hat{p}) = \min_z \langle E_{KL}(z, \hat{p}) \rangle_{\underline{\hat{\mathbf{P}}}} = \langle E_{KL}(\bar{p}, \hat{p}) \rangle_{\underline{\hat{\mathbf{P}}}}, \tag{4.5}$$

[2]The mean operator is not with regard to the density or probability given by the estimators viewed as density/probability functions, but is with regard to the density or probability of the estimator in the group of estimators.

under the constraint $\sum_i \bar{p}(y_i) = 1$. The average estimator $\bar{p}(y)$ found this way is the LOP. The variance will be called the *logarithmic variance*. Note that the logarithmic variance always is positive or zero, since the KL error always is positive or zero.

The bias is the difference between the target function and the average estimator

$$\text{bias}(t, \bar{p}) = E_{KL}(t, \bar{p}).$$

Note that the bias always is positive or zero, since the KL error always is positive or zero.

The bias and variance constitute a proper decomposition of the mean of the KL error

$$\langle E_{KL}(t, \hat{p}) \rangle_{\underline{\hat{\mathbf{p}}}} = \text{bias}(t, \bar{p}) + \text{var}(\hat{p}). \tag{4.6}$$

If it is assumed that the target function is a discrete class probability vector and that the individual target probabilities are either one or zero, the expression for the error and the bias becomes simpler. This is the case if the target function is associated with a set of class examples. Because probabilities sum to one, there is only a single of the target probabilities that can be one. Let the index of this be c. Now the error decomposition becomes

$$\langle E_{KL}(t, \hat{p}) \rangle_{\underline{\hat{\mathbf{p}}}} = -\langle \log(\hat{p}(y_c)) \rangle_{\underline{\hat{\mathbf{p}}}} = -log(\bar{p}(y_c)) + \langle E_{KL}(\bar{p}, \hat{p}) \rangle_{\underline{\hat{\mathbf{p}}}}. \tag{4.7}$$

For a continuous target function we get a similar result up to an irrelevant constant[3] with Dirac's delta function (see e.g. [70] p. 614-618) and integration.

$$\langle E_{KL}(t(y), \hat{p}) \rangle_{\underline{\hat{\mathbf{p}}}} = -\langle \log(\hat{p}(y')) \rangle_{\underline{\hat{\mathbf{p}}}} = -log(\bar{p}(y')) + \langle E_{KL}(\bar{p}, \hat{p}) \rangle_{\underline{\hat{\mathbf{p}}}}, \tag{4.8}$$

where $t(y)$ is defined as the Dirac's delta function $\delta_{y,y'}$.

In the discrete case the estimator $\hat{p}$ can be viewed as a vector of class probabilities, which can be equated with the output of a predictor, i.e $(\hat{p}(y_1|\vec{x}), \ldots, \hat{p}(y_n|\vec{x})) = (f^1(\vec{x}_1), \ldots, f^n(\vec{x})) = \vec{f}(\vec{x})$. An example of this can be seen in chapter 11. The general way is to view the estimators are as parameterized density/probability functions. As an example assume that the density is given by the continuous density function of the Normal distribution: $p(y; \mu, \sigma)$. The outcome y is the outcome of a continuous stochastic variable $\underline{\mathbf{Y}}$ and the output of a predictor f is associated with the mean parameter μ, so we have $\hat{p}(y) = p(y; f, \sigma)$. Note that the mean operator $\langle \cdot \rangle$ is

[3]The constant goes towards infinity as t goes towards the Dirac's delta function, but the constant appears on both sides of the right-hand equality sign in (4.8) and is therefore irrelevant.

now with regard to the probability/density of predictors f, and only indirectly with regard to the probability/density of the estimators $\hat{p}$. Therefore the symbol for the stochastic variable is changed to $\underline{\mathbf{F}}$.

The estimator associated with average predictor becomes

$$\begin{aligned}\bar{p}(y) &= \frac{1}{Z}\exp\langle \log\hat{p}(y)\rangle_{\underline{\mathbf{F}}} \\ &= \frac{1}{Z}\exp\langle -\log(\sigma\sqrt{2\pi}) - \frac{(y-f)^2}{2\sigma^2}\rangle_{\underline{\mathbf{F}}} \\ &= \frac{1}{Z}\exp[-\frac{\langle f^2\rangle_{\underline{\mathbf{F}}}}{2\sigma^2}]\exp(-\log(\sigma\sqrt{2\pi}) - \frac{y^2-2y\langle f\rangle_{\underline{\mathbf{F}}}}{2\sigma^2}).\end{aligned} \tag{4.9}$$

The constraint on $\hat{p}$ gives

$$\begin{aligned}Z &= \int dy \exp\langle -\log(\sigma\sqrt{2\pi}) - \frac{(y-f)^2}{2\sigma^2}\rangle_{\underline{\mathbf{F}}} \\ &= \exp[-\frac{\langle f^2\rangle_{\underline{\mathbf{F}}}}{2\sigma^2}]\int dy \exp(-\log(\sigma\sqrt{2\pi}) - \frac{y^2-2y\langle f\rangle_{\underline{\mathbf{F}}}}{2\sigma^2})\end{aligned}$$

By multiplying both Z and $\exp\langle -\log(\sigma\sqrt{2\pi}) - \frac{(y-f)^2}{2\sigma^2}\rangle_{\underline{\mathbf{F}}}$ with $\exp(-\frac{\langle f\rangle^2_{\underline{\mathbf{F}}}}{2\sigma^2})$ in the expression of $\bar{p}(y)$ (4.9) we get

$$\bar{p}(y) = \left[\int dy \frac{1}{\sigma\sqrt{2\pi}}\exp(-\frac{(y-\langle f\rangle_{\underline{\mathbf{F}}})^2}{2\sigma^2})\right]^{-1}\frac{1}{\sigma\sqrt{2\pi}}\exp(-\frac{(y-\langle f\rangle_{\underline{\mathbf{F}}})^2}{2\sigma^2}).$$

The first factor

$$\left[\int dy(\sigma\frac{1}{\sqrt{2\pi}}\exp[-\frac{1}{2\sigma^2}(y-\langle f\rangle_{\underline{\mathbf{F}}})^2]\right]^{-1}$$

is one, since it is the inverse of the integral over a density function, so we get

$$\bar{p}(y) = \frac{1}{\sigma\sqrt{2\pi}}\exp(-\frac{(y-\langle f\rangle_{\underline{\mathbf{F}}})^2}{2\sigma^2}).$$

The density function associated with the average predictor has the same form as the group of density functions for the predictors f, so the average predictor is $\bar{f} = \langle f\rangle_{\underline{\mathbf{F}}}$. By using the decomposition in (4.7) we get (omitting irrelevant constants and factors)

$$\langle (f-y)^2\rangle_{\underline{\mathbf{F}}} = (\bar{f}-y)^2 + \langle (f-\bar{f})^2\rangle_{\underline{\mathbf{F}}}$$

as noted in [27] and chapter 3

A similar result applies to all members of the exponential family (see appendix E). It is shown for the natural form of the one parameter exponential family (E.4):

$$\hat{p}(y) = p(y;\eta) = \exp[\eta T(y) + d_0(\eta) + S(y)]$$

The density $\hat{p}(y)$ is normalized:

$$\int dy \hat{p}(y) = \exp(d_0(\eta)) \int dy \exp[\eta T(y) + S(y)] = 1,$$

or

$$\int dy \exp[\eta T(y) + S(y)] = \exp(-d_0(\eta)). \tag{4.10}$$

Note that the mean operator $\langle \cdot \rangle$ now is with regard to the probability/density of the parameter η. The average estimator is given by

$$\begin{aligned} \bar{p}(y) &= \frac{1}{Z} \exp[\langle \eta T(y) + d_0(\eta) + S(y) \rangle_{\underline{\eta}}] \\ &= \frac{1}{Z} \exp[\langle \eta \rangle_{\underline{\eta}} T(y) + \langle d_0(\eta) \rangle_{\underline{\eta}} + S(y)]. \end{aligned}$$

The normalization constant Z is

$$\begin{aligned} Z &= \int dy \exp[\langle \eta \rangle_{\underline{\eta}} T(y) + \langle d_0(\eta) \rangle_{\underline{\eta}} + S(y)] \\ &= \exp \langle d_0(\eta) \rangle_{\underline{\eta}} \int dy \exp[\langle \eta \rangle_{\underline{\eta}} T(y) + S(y)] \\ &= \exp \langle d_0(\eta) \rangle_{\underline{\eta}} \exp(-d_0(\langle \eta \rangle_{\underline{\eta}})), \end{aligned}$$

where the last equality comes from (4.10). The average estimator becomes

$$\bar{p}(y) = \exp[\langle \eta \rangle_{\underline{\eta}} T(y) + d_0(\langle \eta \rangle_{\underline{\eta}}) + S(y)]. \tag{4.11}$$

So the average estimator is also a member of the one parameter exponential family with a parameter equal to the mean of the parameter of the estimators:

$$\bar{\eta} = \langle \eta \rangle_{\underline{\eta}}$$

For the Normal distribution the η parameter equals the mean of the distribution (except for a constant). The connection between η and the mean is not always linear. The function describing the connection is the *canonical link* function $\eta = c(\mu)$. If one wants the predictors to correspond to the mean, the average predictor is not always the mean of the predictors, but is given by

$$\bar{f} = c^{-1}(\langle c(f) \rangle_{\underline{\mathbf{F}}}).$$

In table 4.1 the average predictor for the most commonly used distributions are given.

DENSITY	AVERAGE PREDICTOR
Normal	$\bar{f} = \langle f \rangle_{\underline{\mathbf{F}}}$
Poisson	$\bar{f} = \exp \langle \log(f) \rangle_{\underline{\mathbf{F}}}$
Binomial	$\bar{f} = [\exp \langle \log(f^{-1} - 1) \rangle_{\underline{\mathbf{F}}} - 1]^{-1}$
Gamma	$\bar{f} = \langle f^{-1} \rangle_{\underline{\mathbf{F}}}^{-1}$
Inverse Gauss	$\bar{f} = \langle f^{-2} \rangle_{\underline{\mathbf{F}}}^{-2}$

Table 4.1: Average predictor for LOP

The logarithmic variance is given in (4.5). The logarithmic variance for the natural form of the one parameter exponential family is

$$\text{var}(\hat{p}) = \langle d_0(\langle \eta(f) \rangle_{\underline{\mathbf{F}}}) - d_0(\eta(f)) \rangle_{\underline{\mathbf{F}}}. \tag{4.12}$$

Proof:

$$\begin{aligned}
\text{var}(\hat{p}) =& \langle E_{KL}(\bar{p}, \hat{p}) \rangle_{\underline{\mathbf{F}}} \\
=& \langle \int dy \bar{p}(y) \log \frac{\hat{p}(y)}{\bar{p}(y)} \rangle_{\underline{\mathbf{F}}} \\
=& \int dy \bar{p}(y) \langle \langle \eta(f) \rangle_{\underline{\mathbf{F}}} T(y) + d_0(\langle \eta(f) \rangle_{\underline{\mathbf{F}}}) + S(y) \rangle_{\underline{\mathbf{F}}} \\
& - \int dy \bar{p}(y) \langle \eta(f) T(y) + d_0(\eta(f)) + S(y) \rangle_{\underline{\mathbf{F}}} \\
=& \int dy \bar{p}(y) \langle d_0(\langle \eta(f) \rangle_{\underline{\mathbf{F}}}) - \langle d_0(\eta(f)) \rangle_{\underline{\mathbf{F}}} \rangle_{\underline{\mathbf{F}}} \\
=& \langle d_0(\langle \eta(f) \rangle_{\underline{\mathbf{F}}}) - d_0(\eta(f)) \rangle_{\underline{\mathbf{F}}}
\end{aligned}$$

In table 4.2 an overview of some variance expression is given.

4.4.1 Logarithmic ambiguity

In section 3.2 the ambiguity for MSE is described, and it is shown that the ambiguity decomposition for the MSE is mathematically equivalent with the bias/variance decomposition. In [40] an ambiguity expression for the KL error is presented, this is also mathematically equivalent to the bias/variance decomposition of the KL error.

DENSITY	VARIANCE
Normal	$\frac{1}{\sigma^2}\langle[f-\bar{f}]^2\rangle_{\underline{\mathbf{F}}}$
Poisson	$\langle f-\bar{f}\rangle_{\underline{\mathbf{F}}}$
Binomial	$n\langle \log\frac{1-\bar{f}}{1-f}\rangle_{\underline{\mathbf{F}}}$
Gamma	$\nu\langle \log\frac{f}{\bar{f}}\rangle_{\underline{\mathbf{F}}}$
Inverse Gauss	$\chi\langle\bar{f}-f\rangle_{\underline{\mathbf{F}}}$

Table 4.2: Variance for LOP

The ambiguity expression for the KL error will be called for *logarithmic ambiguity*. The logarithmic ambiguity expression will be discussed in the context of predictors outputting class probabilities. As mentioned in section 4.4 there are other contexts, e.g. where the predictor output is the mean in a density function. The discussion below is readily generalized to that perspective.

An ensemble consists of M ensemble members f_i. The output of f_i is $\{f_i^1,\ldots,f_i^N\}$, where f_i^c is the estimate of the probability of class c. Each ensemble member has associated a weight α_i. The weights obey $\sum_i \alpha_i = 1$ and $\alpha_i \geq 0$. The weights can be viewed as the probabilities of the corresponding predictors, and $\sum_i^N \alpha_i$ can be regarded as the mean operator for the stochastic variable $\underline{\mathbf{F}}$. The combined LOP predictor for a single class c becomes

$$F^c = \frac{1}{Z}\exp\langle \log f^c\rangle_{\underline{\mathbf{F}}},$$

where Z is a normalization factor given by

$$Z = \sum_j^N \exp\langle \log f^{c_j}\rangle_{\underline{\mathbf{F}}}$$

Now we have a decomposition of the error of the combined LOP predictor on target $\vec{y}$:

$$E_{KL}(\vec{y},F) = \langle E_{KL}(\vec{y},f)\rangle_{\underline{\mathbf{F}}} - \langle E_{KL}(F,f)\rangle_{\underline{\mathbf{F}}}$$

The first term on the right side of the equality sign is the average error of the ensemble members, while the second term is the logarithmic ambiguity, which is independent of the target function. In chapter 11 there is an example of the use of logarithmic ambiguity.

4.5 Zhu [96] and Heskes [39]

In [96] it is claimed that the decomposition in (4.2) is a generalization of the decomposition in [39]. As noted in section 4.3 the decomposition in (4.2) is not a bias/variance decomposition, but it can be reinterpreted to become one. Even in that case it is not a generalization of the decomposition from [39] in (4.6). There are some similarities. The decomposition in 4.3 involves the information loss function that compares density functions. The information function is parameterized by γ and for two values of γ (zero and one) the information loss function becomes the Kullback-Leibler divergence measure. The decompositions are still not equivalent.

In [96] (see section 4.3) the information loss for $\gamma \in \{0, 1\}$ is given by

$$D_1(p', q') = \int p' \log \frac{p'}{q'}.$$

$$D_0(p', q') = \int q' \log \frac{q'}{p'}.$$

The information loss has associated a mean operator $\langle \cdot \rangle_z$ that is with regard to p'. In [39] (see section 4.4) the Kullback-Leibler divergence is given by

$$K(q, p) = \int q \log \frac{q}{p}.$$

The Kullback-Leibler divergence has associated a mean operator $\langle \cdot \rangle_{\underline{\mathbf{p}}}$ that is with regard to p. Both integrals are over the input and output space.

The following must be equivalent

- $p' \sim p$.
- $\langle \cdot \rangle_{\underline{\mathbf{p}}} \sim \langle \cdot \rangle_z$,

But then $D_1(q', p')$ is not equivalent with $K(q, p)$ since the parameters are interchanged and the functions are asymmetric[4].

The functions $D_0(q', p')$ and $K(q, p)$ are equivalent, but the two definitions of the average densities is not. The average density in section 4.4 is $\bar{p} = \frac{1}{Z} \exp \langle \log p \rangle_{\underline{\mathbf{p}}}$, while the average density in section 4.3 is $\hat{p}'_z = \exp \langle \log p' \rangle_z$, but these are not the same because of the normalization factor $Z = \int \bar{p} \neq 1$.

[4]Note that D_1 has a natural decomposition with a γ-mean $\langle p' \rangle_z$ that is a density. However the decomposition is practically useless, since q' appears in the denominator in the expression for D_1

4.6 Others

Friedman [25] and Bauer & Kohavi [5]

In Friedman [25] and Bauer & Kohavi [5] two bias/variance decomposition for classification probabilities are given. Let y be a class label from a finite set of classes Y. Let $p(y|\vec{x})$ be the target probability and $f(y|\vec{x})$ be the predictor. We will present the decomposition at one input point and therefore omit $\vec{x}$ from the notation. Let $\langle \cdot \rangle$ be the mean operator with regard to a distribution over the predictors. Let $\bar{f}(y) = \langle f(y) \rangle$ be the average predictor. The decomposition in [25] is

$$\langle \sum_{y \in Y} (p(y) - f(y))^2 \rangle = \sum_{y \in Y} (p(y) - \bar{f}(y))^2 + \langle \sum_{y \in Y} (f(y) - \bar{f}(y))^2 \rangle, \quad (4.13)$$

where $\sum_{y \in Y} (p(y) - \bar{f}(y))^2$ is the bias and $\langle \sum_{y \in Y} (f(y) - \bar{f}(y))^2 \rangle$ is the variance. This is the same decomposition as for regression MSE, the only difference is that probabilities are compared.

The decomposition in [5] is based on the error function

$$E(p, f) = (1 - \sum_{y \in Y} p(y) f(y)).$$

The error function is sensible since it always is positive or zero. It attains it's minimal value if $p = f$. If the target probabilities are class examples, so $p(y) \in \{0, 1\}$, then the error becomes $1 - f(y_{correct})$, where $y_{correct}$ is the correct class. The decomposition is given by

$$(1 - \sum_{y \in Y} p(y) f(y)) = \frac{1}{2} \sum_{y \in Y} [p(y) - f(y)]^2 + \frac{1}{2} [1 - \sum_{y \in Y} f(y)^2] + \frac{1}{2} \sigma^2, \quad (4.14)$$

where $\sigma^2 = [1 - \sum_{y \in Y} p(y)^2]$ is the intrinsic noise (irreducible error), the term $\sum_{y \in Y} [p(y) - f(y)]^2$ is the bias, and $[1 - \sum_{y \in Y} f(y)^2]$ is the variance.

The bias/variance decomposition in (4.13) is traditional in the sense that the error in question is the average over a set of predictors and the decomposition involves an average predictor. The decomposition in (4.14) is not a bias/variance decomposition in the usual sense. The variance term $[1 - \sum_{y \in Y} f(y)^2]$ does not involve the variation around an average predictor. It is better interpreted as a measure of "certainty". If the predictor is certain (but not necessary correct) about the classification, i.e. if $f(y) = 1$ for some class y, then the variance is zero, while a completely uncertain predictor, i.e. if $f(y) = \frac{1}{|Y|}$, has the largest variance. The bias in (4.14) is equivalent with the MSE error function in (4.13). Taking the average over a set of predictors and combining the two decompositions yields a decomposition comprised of four terms: bias, variance, the certainty of the target (intrinsic noise), and the certainty of the predictor. ■

Friedman [25]

The paper by Friedman [25] has been quoted several times for the following statement "For classification it is possible for fixed bias, that an increase in variance to yields a decrease in error". This seems to contradict common sense about bias/variance decomposition. This is not so, because while the statement above is correct, it would be more meaningful to rephrase it to "For classification it is possible for fixed **MSE** bias, that an increase in **MSE** variance yields an decrease in **classification** error". The MSE bias and variance are defined in (4.13), while the classification error is defined as the average probability of misclassification. Let the problem in question be a two-class problem, so a predictor misclassifies if the predicted probability of the correct class is less than 50 %.

The situation where increasing variance yields decreasing classification error arises if the predictors on average make the wrong prediction, i.e. the average predictor predicts the correct class with less than 50 % probability. The average classification error will be large but not necessary 100 % misclassification. Some of the predictors could predict the class with over 50 % probability. Fix the bias (and thereby the average predictor) and increase the variance, now more predictors will predict the class with more than 50 % and the classification error will decrease.

There is nothing spectacular about the statement in [25], because the errors that are used are of different types. ■

Breiman [10]

In Breiman [10] two bias/variance decomposition are given. The usual MSE decomposition for regression and a radical different decomposition for classification. Normally an error function and corresponding decomposition involves comparison of metric values, for regression the comparison is directly on the values, for classification the probabilities can be compared as in (4.13). The decomposition in [10] uses non-metric class labels. So we have a classifier $c(\vec{x})$ that outputs a class label. We use the symbol c to emphasize that we are dealing with pure classification. The classifier is a member of a set of classifiers with associated distribution denoted by the stochastic variable $\underline{\mathbf{C}}$. The classes $y \in Y$ have associated a distribution denoted by the stochastic variable $\underline{\mathbf{Y}}$. The average misclassification error for a classifier is defined as

$$PE(c) = \langle c(\vec{x}) \neq y \rangle_{\underline{\mathbf{X}},\underline{\mathbf{Y}}}$$

The Bayes optimal classifier is

$$c^*(\vec{x}) = \underset{y}{\operatorname{argmax}}\ P(y|\vec{x}),$$

where $P(y|\vec{x})$ is the true probability of the class y at point $\vec{x}$. Instead of an average predictor the aggregated classifier is defined as

$$\bar{c}(\vec{x}) = \underset{y}{\text{argmax}} \ \langle c(\vec{x}) = y \rangle_{\underline{\mathbf{C}}}$$

This is aggregating by voting. A classifier is *unbiased* at point $\vec{x}$ if the aggregated classifier equals the Bayes optimal classifier, i.e. the classifiers more often predicts the same class as the Bayes optimal classifier on average than any other class. Let U be the set of input point for which the classifiers are unbiased, and let B be the complementary set. Bias is defined as

$$bias(c) = PE_B(c^*) - \langle PE_B(c) \rangle_{\underline{\mathbf{C}}},$$

where PE_B denotes the average generalization error, where the average over input points is taken over the input points in the set B. The variance is defined as

$$variance(c) = PE_U(c^*) - \langle PE_U(c) \rangle_{\underline{\mathbf{C}}},$$

where PE_U is analog to PE_B. The average generalization error can be decomposed as

$$PE(c) = PE(c^*) + bias(c) + variance(c)$$

This decomposition has a number of desirable properties

- Bias and variance are always positive or zero.
- The variance of the aggregated classifier is zero.
- The bias of the Bayes optimal classifier is zero.
- The decomposition holds for all classifiers.

But the decomposition lacks one important property, namely target independent variance. ■

Many of the bias/variance decompositions in the literature assume implicitly or explicitly that the distribution of targets and predictors, estimators, or classifiers are independent. Often this is a necessary requirement for the decomposition to hold. In [90] the problem of dependency is addressed. Correction terms are presented for e.g. the MSE bias/variance decomposition.

In chapter 5 we present a general bias/variance decomposition with desirable properties, and show for which family of error functions that decomposition hold.

Chapter 5

General bias/variance decomposition

Erst wenn die Wolken schlafen gehen,
kann man uns am Himmel sehen.
— RAMMSTEIN, Sehnsucht, Engel.

A great deal of research has gone into finding an expression for a bias/variance decomposition that applies to all error functions or as many error functions as possible (see [10, 27, 90, 25, 44, 39, 96] or chapter 4). The properties sought are not always the same. One of the more common is that the decomposition must be proper, meaning that the error can be expressed by the sums of bias and variance (maybe plus some intrinsic noise term from the target function). Another common property is that bias and variance must be positive or zero, and the variance is zero if the different predictors are functional identical.

A general decomposition will be presented below, developed by the author of this dissertation and Tom Heskes. A paper based on the results have been accepted for presentation at the 15th International Conference on Pattern Recognition (ICPR2000) [35].

5.1 Introduction

In chapter 3 the bias/variance decomposition for the MSE was given. The MSE is by far the most commonly used error function, since it has many nice properties. The use of the MSE for regression problems makes the implicit assumption that there is Gaussian noise on the target function. This is a reasonable assumption, without specific knowledge about the noise, since the

Gaussian (or Normal) distribution is the "standard" distribution, but with more specific knowledge we can often do better. In this section we present a family of error functions with almost all of the nice properties of the MSE, e.g. a bias/variance decomposition with target independent variance. The family of error functions corresponds exactly to the error functions derived from the exponential family of distributions. The assumption of Gaussian noise can therefore be replaced by any kind of noise assumption with distributions from the exponential family.

An important concept is the average predictor (see definition 12). The set of predictors, on which the average predictor is defined, has a distribution associated with it. The mean operator $\langle \cdot \rangle_{\underline{\mathbf{F}}}$ is with regard to that distribution of predictors. The distribution can be either continuous or discrete. A continuous distribution could e.g. come from a stochastic learning method. The predictors in an ensemble with normalized ensemble coefficients can be viewed as a discrete distribution. The most common definition of the average predictor is the linear average predictor (LAP) defined as $\bar{f} = \langle f \rangle_{\underline{\mathbf{F}}}$.

5.2 Requirements for bias/variance decomposition and error functions

We will state the natural requirements for any error function and a strict set of requirements for the bias/variance decomposition. The MSE and corresponding decomposition is used as an example. The MSE is given by

$$E_{\text{MSE}}(t, f) = \frac{1}{2}(t - f)^2$$

The shape of $E_{\text{MSE}}(t, f)$ as a function of the predictor f is a parable with a global minimum at t. Furthermore the value at t is zero. Any other error function should also be minimal at t and have no other local or global extremes. Note that it is easy to achieve that the value at t is zero, as the error function $E'(t, f) = E(t, f) - E(t, t)$ automatically ensures that. The requirements are stated mathematically as

1. $\underset{f}{\text{argmin}}\ E(t, f) = t$
2. $E(t, t) = 0$

The first requirement is not the whole story. It only specifies that the global minimum must be at t. There could be other extremes, so we really want that $\frac{\partial E(t,f)}{\partial f} = 0$ only at one point, namely at t. Furthermore, $\frac{\partial^2 E(t,f)}{\partial f \partial f}$ should be positive. We will not always mention all the specific properties needed to ensure the requirements. Often we shall assume the relation in (5.1) holds

both ways.

$$\hat{z} = \underset{z}{\operatorname{argmin}}\ g(y,z) \Leftrightarrow \frac{\partial g(y,z)}{\partial z} = 0\Big|_{\hat{z}=z} \tag{5.1}$$

The relation in (5.1) holds for the MSE. We show $\hat{f} = t$ for MSE in details

$$\begin{aligned} \frac{\partial E_{\text{MSE}}(t,f)}{\partial f} = 0\Big|_{\hat{f}=f} &\Leftrightarrow \\ f - t = 0\Big|_{\hat{f}=f} &\Leftrightarrow \\ \hat{f} = t, & \end{aligned}$$

and

$$\frac{\partial^2 E_{\text{MSE}}(t,f)}{\partial f \partial f} = 1.$$

The bias/variance decomposition of the generalization error for MSE (3.2) is on the form

$$\begin{aligned} \langle E_{\text{MSE}}(t,f)\rangle_{\underline{\mathbf{F}}} &= E_{\text{MSE}}(t,\bar{f}) + \langle E_{\text{MSE}}(\bar{f},f)\rangle_{\underline{\mathbf{F}}} \\ &= \text{bias}(t,\bar{f}) + \text{var}(f), \end{aligned}$$

where the average predictor $\bar{f}$ is $\langle f\rangle_{\underline{\mathbf{F}}}$.

The bias/variance decomposition for the MSE has some very desirable properties. The bias depends only on the predictors through the average predictor. The variance does not depend on the target. Furthermore, the average predictor minimizes the variance:

$$\bar{f} = \underset{t}{\operatorname{argmin}}\ \langle E(t,f)\rangle_{\underline{\mathbf{F}}}. \tag{5.2}$$

This will later become the general definition of the average predictor, so we will expand upon it. We will assume that the implication in (5.3) holds both ways and omit any other details.

$$\bar{z} = \underset{y}{\operatorname{argmin}}\ \langle g(y,z)\rangle_{\underline{\mathbf{Z}}} \Leftrightarrow \frac{\partial \langle g(y,z)\rangle_{\underline{\mathbf{Z}}}}{\partial y} = 0\Big|_{\hat{z}=y} \tag{5.3}$$

The definition in (5.3) is equivalent to the definition of systematic mean (see definition 15 in appendix C) from [44]. Note the difference between (5.1) and (5.3): the value $\hat{z}$ from (5.1) is the minimal value of z for a fixed y, while the value $\bar{z}$ from (5.3) is the minimal value of y on average with regard to z. If the function $g(y,z)$ obeys $\hat{z} = y$, then $\bar{z}$ is a weighted (by the function g) minimum of z.

The definition in (5.2) yields $\bar{f} = \langle f \rangle_{\underline{\mathbf{F}}}$ for the MSE. We show that in detail by using (5.3):

$$\begin{aligned}
\frac{\partial \langle E_{\text{MSE}}(t,f) \rangle_{\underline{\mathbf{F}}}}{\partial t} = 0 \Big|_{\bar{f}=t} &\Leftrightarrow \\
\langle \frac{\partial E_{\text{MSE}}(t,f)}{\partial t} \rangle_{\underline{\mathbf{F}}} = 0 \Big|_{\bar{f}=t} &\Leftrightarrow \\
\langle (f-t) \rangle_{\underline{\mathbf{F}}} = 0 \Big|_{\bar{f}=t} &\Leftrightarrow \\
\langle f \rangle_{\underline{\mathbf{F}}} = t \Big|_{\bar{f}=t} &\Leftrightarrow \\
\bar{f} = \langle f \rangle_{\underline{\mathbf{F}}} &
\end{aligned}$$

A general bias/variance decomposition with the same desirable properties as the MSE bias/variance decomposition has the form

$$\langle E(t,f) \rangle_{\underline{\mathbf{F}}} = E(t,\bar{f}) + \langle E(\bar{f},f) \rangle_{\underline{\mathbf{F}}}, \tag{5.4}$$

where the average predictor is given by (5.2).

We end this section by summarizing the requirements in figure 5.1.

R1: $\text{argmin}_f E(t,f) = t$.

R2: $E(t,t) = 0$.

R3: The bias/variance decomposition is

$$\langle E(t,f) \rangle_{\underline{\mathbf{F}}} = E(t,\bar{f}) + \langle E(\bar{f},f) \rangle_{\underline{\mathbf{F}}}, \tag{5.5}$$

where

$$\bar{f} = \underset{t}{\text{argmin}} \, \langle E(t,f) \rangle_{\underline{\mathbf{F}}} \tag{5.6}$$

is the average predictor.

Figure 5.1: Requirements

5.3 Deviance error functions

The *deviance* is a measure of how similar two statistical models are. If one thinks of the example set as one of the models and the predictor as the other

model, the deviance naturally can be thought of as an error function. In order to do so error functions must be linked to density functions.

Before the definition of the deviance error functions is given, we show how the MSE can be interpreted as the negative log likelihood under the assumption of Gaussian (Normal) noise. The Normal distribution with unit standard deviation is given by

$$p(z|\mu) = \frac{1}{\sqrt{2\pi}} \exp[-\frac{1}{2}(z-\mu)^2]$$

The negative log likelihood is

$$-\log p(z|\mu) = \log\sqrt{2\pi} + \frac{1}{2}(z-\mu)^2$$

By interpreting the outcome z as the target and the parameter μ as the predictor the expression above only differs from the MSE by an additive constant. The desired result is found by subtracting $-\log p(t|t)$:

$$E_{\text{MSE}}(t,f) = -\log p(t|f) + \log p(t|t)$$

This can be viewed as the deviance [53].

In the general definition of deviance two statistical models, represented by their densities, is used to describe the same set of data: $\{z_1,\ldots,z_N\}$. The deviance is used to compare the statistical models. The measure is the logarithm of the quotient of the two densities:

$$D(z_i,\tilde{\theta}_i,\theta_i) = \log\frac{p_1(z_i|\tilde{\theta}_i)}{p_2(z_i|\theta_i)},$$

here presented for only one data point. A natural upper bound on the likelihood is when the density p_1 is parameterized to have maximum likelihood at exactly the data points. Let the density p_1 have that property, ie. $\text{argmax}_z \log p_1(z,\tilde{\theta}_i) = z_i$. The density p_2 will naturally represent the model which accuracy we wish to measure. The parameters $\{\tilde{\theta}_1,\ldots,\tilde{\theta}_n\}$ are given by the definition of the full model, while the parameters $\{\theta_1,\ldots,\theta_n\}$ are the parameters we wish to predict. This yields a natural definition of an error function derived from a density

$$E(t_i, f(\vec{x}_i)) = D(t_i,\tilde{\theta}_i, f(\vec{x}_i)),$$

where the predictor $f(\vec{x}_i)$ is the estimate of parameter θ_i. This defines a family of error functions

Definition 13 (The Deviance Error Function)

Let p_1 and p_2 be two densities, each with one parameter and scalar outcome. Let f be a predictor and let $S = \{(t_1, \vec{x}_1), \ldots, (t_n, \vec{x}_n)\}$ be an example set. The deviance error function for a single example is given by

$$E(t_i, f(\vec{x}_i)) = -\log p_1(t_i|f(\vec{x}_i)) + \log p_2(t_i|\tilde{\theta}_i), \tag{5.7}$$

where the parameters $\tilde{\theta}_i$ obey

$$\underset{t}{argmax}\ \log p(t, \tilde{\theta}_i) = t_i.$$

■

Often the densities p_1 and p_2 will come from the same class of distributions. This is the case in section 5.4

5.4 Deviance error functions for the exponential family of distributions

Let the density p correspond to a member of the one-parameter exponential family of distributions (see appendix E):

$$p(z|\theta) = \exp[c(\theta)T(z) + d(\theta) + S(z)],$$

where c is the canonical link and T is the sufficient statistics. The deviance error function (see definition 13) is

$$E(t, f) = [c(\tilde{\theta}) - c(f)]T(t) + d(\tilde{\theta}) - d(f)$$

The error function requirement R1 (see figure 5.1) is generally not obeyed, but by reparameterizing the density p the requirement can be ensured. From 5.1 and 5.7 it follows that

$$\forall t: \frac{\partial[-\log p(t|f) + \log p(t|\tilde{\theta}_i)]}{\partial f} = -\frac{\partial \log p(t|f)}{\partial f} = 0\Big|_{f=t}$$

This yields for an of the one-parameter member exponential family

$$\forall t: c'(f)T(t) + d'(f) = 0\Big|_{f=t}$$

so the functions c, T, and d must be related or constrained by

$$\forall y: c'(y)T(y) + d'(y) = 0 \tag{5.8}$$

Note that the constraint in (5.8) means that the parameter with maximum log likelihood is equal to the outcome, so for the density $p(z|\theta)$ it should

apply that $\hat{\theta} = z$. This is generally not obeyed, but it is possible to reparameterize the density to another parameter ϕ, that ensures $\hat{\phi} = z$. Let the relation between the parameter θ and ϕ be given by $\theta = g(\phi)$. The density $p(z|g(\phi))$ is the same as $p(z|\theta)$, only the value of the parameter is not the same.

For $\hat{\phi} = z$ to hold $g(y)$ must equal $h^{-1}(T(y))$, where $h(y) = -d'(y)/c'(y)$. Using (5.1) yields

$$\begin{aligned}
\frac{\partial \log p(z|g(\phi))}{\partial \phi} &= 0\Big|_{\hat{\phi}=\phi} \Leftrightarrow \\
c'(g(\phi))g'(\phi)T(z) + d'(g(\phi))g'(\phi) &= 0\Big|_{\hat{\phi}=\phi} \Leftrightarrow \\
-\frac{d'(g(\phi))}{c'(g(\phi))} &= T(z)\Big|_{\hat{\phi}=\phi} \Leftrightarrow \\
h(g(\phi)) &= T(z)\Big|_{\hat{\phi}=\phi} \Leftrightarrow \\
g(\hat{\phi}) &= h^{-1}(T(z)).
\end{aligned}$$

Implying that $g(y) = h^{-1}(T(y))$ yields $\hat{\phi} = z$.

As an example it is shown how to reparameterize the Normal distribution and the Gamma distribution.

Example 1: Normal distribution The Normal distribution is given by

$$p(z;\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp[-\frac{(z-\mu)^2}{2\sigma^2}]$$

Assuming that the standard deviation is known, it is an one-parameter density. The free parameter is the mean μ. The defining functions are $c(\mu) = \mu/\sigma^2$, $T(z) = z$, and $d(\mu) = \mu^2/(2\sigma^2)$, yielding that $h(\mu) = -d'(\mu)/c'(\mu) = \mu$. The reparameterization function $g(\phi) = h^{-1}(T(\phi))$ yields $\phi = \mu$. This implies the well-known result that $\hat{\mu} = z$, because $\hat{\phi} = z$, so for the Normal distribution the predictor f is an estimate of the mean parameter. ■

Example 2: Gamma distribution The Gamma distribution is given by

$$p(x;\nu,\lambda) = \begin{cases} \frac{\lambda^\nu}{\Gamma(\nu)} x^{\nu-1} e^{-\lambda x} & x \geq 0 \\ 0 & \text{else} \end{cases}$$

Let the parameter ν be known and the parameter λ be the free parameter. Setting $\lambda = \theta$ yields

$$p(z|\theta) = \exp[-\theta z + \nu \log(\theta) + (\nu - 1)\log(z) - \log(\Gamma(\nu)),$$

where $c(\theta) = -\theta$, $T(z) = z$, and $d(\theta) = \nu \log(\theta)$. This yields $h(\theta) = \nu/\theta$, which implies $\theta = g(\phi) = \nu/\phi$. The mean of the Gamma distribution is given by $\nu/\lambda = \nu/(\nu/\phi) = \phi$, so the parameter ϕ is the mean parameter. As for the Normal distribution the predictor f is an estimate of the mean parameter. Renaming the functions $c(g(\phi))$ and $d(g(\phi))$ to $c(\phi)$ and $d(\phi)$ yields $c(\phi) = -\nu/\phi$ and $d(\phi) = \nu \log(\nu/\phi)$. The reparameterized density is

$$p(z|\phi) = \exp[-\frac{\nu}{\phi}z + \nu \log(\frac{\nu}{\phi}) + (\nu - 1)\log(z) - \log(\Gamma(\nu)),$$

which obeys the constraint (5.8). ■

With the reparameterizing, the parameters $\tilde{\phi}_i$ in the full model become easy to find. They are equal to the target t_i. The deviance error function becomes

$$E(t, f) = [c(t) - c(f)]T(t) + d(t) - d(f). \tag{5.9}$$

We will later show that only error functions with the form in (5.9) are decomposable as in (5.5). Observe that the function d is determined completely from the functions c and T by the constraint (5.8), so the canonical link function c and the sufficient statistic function T can be viewed as the fundamental defining functions.

As noted above the Normal distribution does not need reparameterization and has $c(f) = f, T(t) = t$, and $d(f) = -\frac{1}{2}f^2$ for the standard deviation equal to one. This yields $E_{\text{NORMAL}}(t, f) = E_{\text{MSE}}(t, f)$ as expected.

The error function in (5.9) obeys R1-R3 in figure 5.1. That R1 is obeyed comes from the constraint (5.8). Requirement R2 is obeyed because of the definition of deviance. For the error functions in (5.9) we have the following corollaries

C1: $\bar{f} = \underset{t}{\text{argmin}} \langle E(t, f)\rangle_{\underline{\mathbf{F}}} = c^{-1}(\langle c(f)\rangle_{\underline{\mathbf{F}}})$

Proof:

Using (5.3) we have

$$\begin{aligned}
\bar{f} = \underset{t}{\text{argmin}} \langle E(t, f)\rangle_{\underline{\mathbf{F}}} \Leftrightarrow \\
\frac{\partial \langle E(t, f)\rangle_{\underline{\mathbf{F}}}}{\partial t} = 0\Big|_{t=\bar{f}} \Leftrightarrow \\
\langle c'(t)T(t) + c(t)T'(t) - c(f)T'(t) + d'(t)\rangle_{\underline{\mathbf{F}}} = 0\Big|_{t=\bar{f}} \Leftrightarrow \\
\langle c(t)T'(t) - c(f)T'(t)\rangle_{\underline{\mathbf{F}}} = 0\Big|_{t=\bar{f}} \Leftrightarrow \\
\langle c(t)T'(t)\rangle_{\underline{\mathbf{F}}} = \langle c(f)T'(t)\rangle_{\underline{\mathbf{F}}}\Big|_{t=\bar{f}} \Leftrightarrow \\
c(t) = \langle c(f)\rangle_{\underline{\mathbf{F}}}\Big|_{t=\bar{f}} \Leftrightarrow \\
\bar{f} = c^{-1}(\langle c(f)\rangle_{\underline{\mathbf{F}}}),
\end{aligned}$$

where the constraint (5.8) has been used.

C2: $\bar{t} = \underset{f}{\operatorname{argmin}} \langle E(t,f)\rangle_{\underline{\mathbf{T}}} = T^{-1}(\langle T(t)\rangle_{\underline{\mathbf{T}}})$

The term $\bar{t}$ will be called the *average target.*

Proof:

Using (5.3) we have

$$\begin{aligned}
\bar{t} = \underset{f}{\operatorname{argmin}} \langle E(t,f)\rangle_{\underline{\mathbf{T}}} \Leftrightarrow \\
\frac{\partial \langle E(t,f)\rangle_{\underline{\mathbf{T}}}}{\partial f} = 0\Big|_{f=\bar{t}} \Leftrightarrow \\
\langle -c'(f)T(t) - d'(f)\rangle_{\underline{\mathbf{T}}} = 0\Big|_{f=\bar{t}} \Leftrightarrow \\
\langle -c'(f)T(t) + c'(f)T(f)\rangle_{\underline{\mathbf{T}}} = 0\Big|_{f=\bar{t}} \Leftrightarrow \\
T(f) = \langle T(t)\rangle_{\underline{\mathbf{T}}}\Big|_{f=\bar{t}} \Leftrightarrow \\
\bar{t} = T^{-1}(\langle T(t)\rangle_{\underline{\mathbf{T}}}),
\end{aligned}$$

where the constraint (5.8) has been used.

The average predictor is defined by corollary C1. The bias/variance decomposition (R3) is proven by inspection.

$$\begin{aligned}
&E(t,\bar{f}) + \langle E(\bar{f},f)\rangle_{\underline{\mathbf{F}}} = \\
&[c(t) - c(\bar{f})]T(t) + d(t) - d(\bar{f}) + \langle [c(\bar{f}) - c(f)]T(\bar{f}) + d(\bar{f}) - d(f)\rangle_{\underline{\mathbf{F}}} = \\
&[c(t) - c(\bar{f})]T(t) + d(t) + [c(\bar{f}) - \langle c(f)\rangle_{\underline{\mathbf{F}}}]T(\bar{f}) - \langle d(f)\rangle_{\underline{\mathbf{F}}} = \\
&[c(t) - \langle c(f)\rangle_{\underline{\mathbf{F}}}]T(t) + d(t) - \langle d(f)\rangle_{\underline{\mathbf{F}}} = \\
&\langle E(t,f)\rangle_{\underline{\mathbf{F}}},
\end{aligned}$$

where $c(\bar{f}) = \langle c(t)\rangle_{\underline{\mathbf{F}}}$ has been used multiple times.

The variance is by definition independent of the target, and given by

$$\mathrm{var}(f) = \langle E(\bar{f},f)\rangle_{\underline{\mathbf{F}}} = \langle d(\bar{f}) - d(f)\rangle_{\underline{\mathbf{F}}} \tag{5.10}$$

The decomposition in (5.5) is defined for a single target. If we also average over the distributions of targets the bias term can be split into an intrinsic noise term and a bias term. For the generalization error averaged over both the predictors and the targets we get

$$\langle E(t,f)\rangle_{\underline{\mathbf{F}},\underline{\mathbf{T}}} = \langle E(t,\bar{t})\rangle_{\underline{\mathbf{T}}} + E(\bar{t},\bar{f}) + \langle E(\bar{f},f)\rangle_{\underline{\mathbf{F}}}, \tag{5.11}$$

where $\langle E(t,\bar{t})\rangle_{\underline{\mathbf{T}}}$ is the intrinsic noise (irreducible error) and $E(\bar{t},\bar{f})$ is the bias. The systematic part of the error, the bias, is minimal at the average target defined by corollary C2.

The corollary C2 deserves more remarks. First, note the symmetry between corollary C1 and C2. The average predictor and the target predictor are of the same form for the canonical link and sufficient statistic respectively. This suggests interchangeability that will be explored in section 5.8. The corollary C2 holds for any mean operator over $\underline{\mathbf{T}}$, even if the distribution of targets is not equal to the distribution from which the error function is derived. If the distribution of targets is equal to the distribution from which the error function is derived, it yields that $\bar{t} = \phi$. It is shown by using the general relation from appendix E (E.5) for the reparameterized density $p(z|\phi)$.

$$\begin{aligned}
\langle \frac{\partial \log p(z|\phi)}{\partial \phi} \rangle_{\underline{\mathbf{T}}} &= 0 \Leftrightarrow \\
\langle c'(\phi)T(z) + d'(\phi) \rangle_{\underline{\mathbf{T}}} &= 0 \Leftrightarrow \\
\langle T(z) \rangle_{\underline{\mathbf{T}}} &= \langle -\frac{d'(\phi)}{c'(\phi)} \rangle_{\underline{\mathbf{T}}} \Leftrightarrow \\
\langle T(z) \rangle_{\underline{\mathbf{T}}} &= \langle T(\phi) \rangle_{\underline{\mathbf{T}}} \Leftrightarrow \\
T^{-1}(\langle T(z) \rangle_{\underline{\mathbf{T}}}) &= \phi,
\end{aligned}$$

where $T(\phi) = -d'(\phi)/c'(\phi)$ has been used. Note that the above always holds, not just for a particular value of ϕ, but for all ϕ. By using (5.3) we find $\bar{t} = T^{-1}(\langle T(t) \rangle_{\underline{\mathbf{T}}}) = \phi$. Even though the predictor f is associated with the parameter, we cannot conclude the $f = \bar{t}$, because f is an estimation of ϕ, and the above holds for the "real" unknown parameter.

We conclude this section by summarizing the most important results.

- The deviance error functions with the form

  $$E(t, f) = [c(t) - c(f)]T(t) + d(t) - d(f)$$

  obey the requirements in figure 5.1, if the error function obeys the constraint $c'(f)T(t) + d'(f) = 0$.

- Any one-parameter exponential density can be reparameterized so the corresponding deviance error function obeys the constraint (5.8). The new parameter ϕ must obey

  $$\theta = g(\phi) = h^{-1}(T(\phi)),$$

  where θ is the old parameter and $h(y) = -d'(y)/c'(y)$.

- The variance is given by $\mathrm{var}(f) = \langle d(\bar{f}) - d(f) \rangle_{\underline{\mathbf{F}}}$, which is independent of the target.

- The average predictor is given by $\bar{f} = c^{-1}\langle c(f) \rangle_{\underline{\mathbf{F}}}$.

- The average target is given by $\bar{t} = T^{-1}\langle T(t)\rangle_{\underline{\mathbf{T}}}$.
  If the distribution of $\underline{\mathbf{T}}$ is the same as the distribution from which the error function is derived then $\bar{t} = \phi$.
- The defining functions are the canonical link function and the sufficient statistic function.
- The bias/variance decomposition.
  Decomposition of the average generalization error with regard to the predictors only is

  $$\langle E(t,f)\rangle_{\underline{\mathbf{F}}} = E(t,\bar{f}) + \langle E(\bar{f},f)\rangle_{\underline{\mathbf{F}}}.$$

  Decomposition of the average generalization error with regard to both predictors and targets is

  $$\langle E(t,f)\rangle_{\underline{\mathbf{F}},\underline{\mathbf{T}}} = \langle E(t,\bar{t})\rangle_{\underline{\mathbf{T}}} + E(\bar{t},\bar{f}) + \langle E(\bar{f},f)\rangle_{\underline{\mathbf{F}}}.$$

5.5 Completeness of the family of deviance error functions derived from the one-parameter exponential family of distribution

In section 5.4 it was shown that error functions of the form

$$E(t,f) = [c(t) - c(f)]T(t) + d(t) - d(f)$$

with constraint $c'(f)T(t) + d'(f) = 0$ (5.8) obeys the requirements in figure 5.1. Also it was shown that if the density from which the error function was derived did not obey the constraint (5.8), it could be reparameterized without changing the density class. Furthermore, it was noted that the defining functions are the canonical link and the sufficient statistic. These occur as a product in the error function. It seems like a fundamental property of the error function. The proof or verification of the requirements suggests that exactly a product is necessary for the bias/variance decomposition to hold. This is indeed the case, which is proven below.

We will show that $\frac{\partial^2 E(t,f)}{\partial t \partial f} = a_1(f)a_2(t)$. This suffices since it implies that $E(t,f)$ is of the form

$$A_1(f)A_2(t) + k_1(f) + k_2(t),$$

where A_1 and A_2 are the anti-derivative of a_1, and a_2 respectively. The functions k_1 and k_2 are arbitrary functions. If the error function is interpreted as a deviance error function we identify A_1 and A_2 as the canonical link function and the sufficient statistic respectively. As noted in section 5.4

they are the defining functions, so k_1 and k_2 is determined from A_1 and A_2, so is sufficient to show $\frac{\partial^2 E(t,f)}{\partial t \partial f} = a_1(f)a_2(t)$ for any functions a_1 and a_2.

We will show $\frac{\partial^2 E(t,f)}{\partial t \partial f} = a_1(f)a_2(t)$ for a particular value of f, namely for the average predictor $\bar{f}$. The proof will not depend on the form of the distribution, so the relation $\frac{\partial^2 E(t,f)}{\partial t \partial f} = a_1(f)a_2(t)$ will still hold for any value of f by choosing the distribution of predictor to be the appropriate Dirac's delta function or Kronecker delta function.

The crux of the proof is that a slight change in the distribution over f yields a change in the average predictor, that affects $\frac{\partial^2 E(t,\bar{f})}{\partial t \partial f}$ multiplicatively and independently of a fixed t. Mathematically this means that $\frac{\partial^2 E(t,\bar{f})}{\partial t \partial f}$ must be of the form $ka(f)$, where the constant k can depend on t.

The derivative with regard to t of the bias/variance decomposition (5.5) is

$$\frac{\partial \langle E(t,f) \rangle_{\underline{\underline{\mathbf{F}}}}}{\partial t} = \frac{\partial E(t,\bar{f})}{\partial t},$$

for all distributions $\underline{\underline{\mathbf{F}}}$. Let $\underline{\underline{\Pi}}$ denote a particular distribution with density $\pi(f)$ and average predictor $\bar{f}$. Let $\delta(f)$ be a function that integrates to zero over the domain of f. Then $\pi(f|\epsilon) = \pi(f) + \epsilon\delta(f)$ is a density with average predictor $\bar{f}(\epsilon)$, yielding

$$\frac{\partial \langle E(t,f) \rangle_{\underline{\underline{\Pi}}(\epsilon)}}{\partial t} = \frac{\partial \langle E(t,f) \rangle_{\underline{\underline{\Pi}}}}{\partial t} + \epsilon \frac{\partial \int df \delta(f) E(t,f)}{\partial t},$$

or equivalently

$$\frac{\partial E(t,\bar{f}(\epsilon))}{\partial t} = \frac{\partial E(t,\bar{f})}{\partial t} + \epsilon \frac{\partial \int df \delta(f) E(t,f)}{\partial t}.$$

Differentiation with regard to ϵ yields

$$\frac{\partial^2 E(t,\bar{f}(\epsilon))}{\partial t \partial f} \frac{\partial \bar{f}(\epsilon)}{\partial \epsilon} = \frac{\partial \int df \delta(f) E(t,f)}{\partial t}.$$

Note that $\frac{\partial^2 E(t,\bar{f}(\epsilon))}{\partial t \partial f}$ is short-hand for $\left.\frac{\partial^2 E(t,f)}{\partial t \partial f}\right|_{f=\bar{f}(\epsilon)}$. In the limit $\epsilon \to 0$, $\frac{\partial^2 E(t,\bar{f}(\epsilon))}{\partial t \partial f}$ becomes $\frac{\partial^2 E(t,\bar{f})}{\partial t \partial f}$. The right hand side does not depend on $\bar{f}$, while $\frac{\partial \bar{f}(\epsilon)}{\partial \epsilon}$ does not depend on t. Taking the limit of ϵ and rearranging yields

$$\frac{\partial^2 E(t,\bar{f})}{\partial t \partial f} = \left[\frac{\partial \bar{f}(\epsilon)}{\partial \epsilon}\right]^{-1} \frac{\partial \int df \delta(f) E(t,f)}{\partial t},$$

this is exactly on the form $a_1(\bar{f})a_2(t)$. Note that the value of $\frac{\partial \bar{f}(\epsilon)}{\partial \epsilon}$ is still well-defined in the limit, because $\bar{f}(\epsilon)$ is continuous in ϵ. Another way of

seeing the result is to choose two values of ϵ: ϵ_1 and ϵ_2. Then the quotient of $\frac{\partial^2 E(t,\bar{f}(\epsilon_i))}{\partial t \partial f}$ becomes

$$\frac{\partial^2 E(t,\bar{f}(\epsilon_1))}{\partial t \partial f} \Big/ \frac{\partial^2 E(t,\bar{f}(\epsilon_2))}{\partial t \partial f} = \frac{\partial \bar{f}}{\partial \epsilon_2} \Big/ \frac{\partial \bar{f}}{\partial \epsilon_1},$$

which is independent of t.

5.6 Examples of deviance error functions

We consider two special cases: linear sufficient statistic and linear canonical link.

The common univariate distributions in the exponential family have linear sufficient statistics, but generally non-linear canonical links. This also applies to the generalized linear models (see appendix D). In table 5.1 is an overview of the c, T, and d functions, respectively the canonical link, sufficient statistic, and normalization term from the density.

DISTRIBUTION	CAN. LINK	SUF. STAT.	NORM. TERM
Normal(μ, σ)	$\sigma^{-2}\mu$	t	$-2\sigma^{-2}\mu^2$
Poisson(μ)	$\log \mu$	t	$-\mu$
Binomial(μ, k)	$k \log \frac{\mu}{1-\mu}$	t	$k \log(1-\mu)$
Gamma(ν, μ)	$-\nu/\mu$	t	$-\nu \log \mu$
Inv. Gauss(μ, θ)	$-1/(\theta 2\mu^2)$	t	$1/(\theta\mu)$

Table 5.1: The defining functions and the normalization term for five members of the exponential family of distributions

Some of the densities in table 5.1 have more than one parameter, e.g. the gamma density. We have chosen the mean parameter μ to be the unknown and all other parameters to be known constant parameters. When the densities later are reparameterize, the mean parameter is a good candidate for the parameter that obeys the constraint 5.8.

For densities with linear sufficient statistics the constraint (5.8) becomes $d'(y) = -c'(y)y$, which is equivalent with $d(y) = -c(y)y + C(y)$, where C is

the anti-derivative of c. All of the densities in table 5.1 obeys the constraint in the given form, so no reparameterization is needed.

Table 5.2 gives an overview of some of the error functions with linear sufficient statistics. All constant factors are omitted. Since the canonical link function is non-linear for all errors except the Normal error, the average predictors are generally non-linear means.

DISTRIBUTION	ERROR FUNCTION	DOMAIN
Normal(μ, σ)	$\frac{1}{2}(f-t)^2$	$]-\infty;\infty[$
Poisson(μ)	$[f-t]+t\log\frac{t}{f}$	$[0;\infty[$
Binomial(μ, k)	$t\log\frac{t}{f}+(1-t)\log\frac{1-t}{1-f}$	$[0;1]$
Gamma(ν, μ)	$(\frac{t}{f}-1)+\log\frac{f}{t}$	$[0;\infty[$
Inv. Gauss(μ, θ)	$(f-t)^2/(f^2t)$	$]0;\infty[$

Table 5.2: Error functions with linear sufficient statistics.

The densities in table 5.1 have linear sufficient statistics and generally non-linear canonical links, while the densities in table 5.3 have linear canonical links and generally non-linear sufficient statistics. In case of more than one parameter we choose the free parameter to be the mean μ and all other parameters to be constant. Note that the last distribution "unknown" only has been added to the table because it matches the Inverse Gauss distribution. It has not been possible to find the distribution in the literature, nor a complete expression for the density. The constraint (5.8) becomes $d(y) = T^{\int}(y)$, where $T^{\int}$ is the anti-derivative of T.

The average predictor is given by $\bar{f} = c^{-1}\langle c(f)\rangle_{\underline{\mathbf{F}}}$, so the corresponding error functions have linear average predictors. In table 5.4 is an overview of error functions with linear average predictor corresponding to the densities in table 5.3. Constant factors are omitted. Most of the densities need reparameterization. The relation between the new parameter ϕ and the old parameter θ is given by $\theta = (-d')^{-1}(T(\phi))$, where $(-d')^{-1}(y)$ is the inverse of $-d'(y)$.

The Gamma$(\mu,1)$ and the Beta$(k\mu, k(1-\mu))$ error functions are approximations because the corresponding density contains the gamma function as a

DISTRIBUTION	C. L.	SUF. STAT.	NORM. TERM
Normal(μ, σ)	μ	$\sigma^{-2}t$	$-2\sigma^{-2}\mu^2$
Gamma(μ,1)	μ	$\log t$	$-\log\Gamma(\mu)$
Beta($k\mu, k(1-\mu)$)	μ	$k\log\frac{t}{1-t}$	$-\log\Gamma(k\mu)\Gamma(k[1-\mu])$
Inv. Gamma(μ, ν)	μ	$-\nu/t$	$(\nu-1)\log\mu$
Unknown(θ)	θ	$-1/(2t^2)$	$-1/(2\theta)$

Table 5.3: The defining function and the normalization term for five members of the exponential family of distributions with linear canonical link.

factor. The Gamma function can be approximated by (see appendix G)

$$\Gamma(y) \approx \sqrt{2\pi}\Big[\frac{y-\frac{1}{2}}{e}\Big]^{y-\frac{1}{2}} \tag{5.12}$$

Let us use the Gamma(μ,1) density as an example.

$$p_{\Gamma(\mu,1)}(z|\mu) = \exp[(\mu-1)\log t - \log\Gamma(\mu) - t]$$

The sufficient statistic function is $\log t$, while the normalization term is $\log\Gamma(\mu)$. To reparameterize the density we must find $\frac{\partial\log\Gamma(y)}{\partial y}$, the *diGamma* function. The approximation (5.12) yields $\frac{\partial\log\Gamma(y)}{\partial y} \approx \log(y-\frac{1}{2})$. The relation between the old and new parameter becomes $\phi \approx \mu - \frac{1}{2}$. ■

Note that the error functions in table 5.4 match the error functions in table 5.2 in pairs, only the predictor and the target are interchanged. This will be explained in section 5.8. Also note that the Normal error function and the counterpart both are the mean square error. The Beta error function is not very useful, since it is undefined for target equal to one or zero.

5.7 Connection to other bias/variance decompositions

The general bias/variance decomposition in this chapter is connected in various ways to two other general bias/variance decompositions. In section 5.7.1 we show that the error functions that can be decomposed as deviance error

DISTRIBUTION	ERROR FUNCTION	DOMAIN
Normal(μ, σ)	$\frac{1}{2}(f-t)^2$	$]-\infty;\infty[$
Gamma$(\mu, 1)$	$[t-f]+f\log\frac{f}{t}$	$[0;\infty[$
beta$(k\mu, k(1-\mu))$	$f\log\frac{f}{t}+(1-f)\log\frac{1-f}{1-t}$	$[0;1]$
Inverted Gamma(μ, ν)	$(\frac{f}{t}-1)+\log\frac{t}{f}$	$]0;\infty[$
Unknown(μ, θ)	$(f-t)^2/(t^2 f)$	$]0;\infty[$

Table 5.4: Error functions with linear canonical links.

functions derived from the one-parameter exponential family of distributions (see section 5.4) are exactly the error functions for which the bias/variance-*effect* decomposition in [44] (see section 4.2) reduces to the bias/variance decomposition also defined in [44]. In section 5.7.2 we show that deviance error functions derived from the one-parameter exponential family of distributions can be reformulated in terms of the Kullback-Leibler error function from [39] (see section 4.4).

5.7.1 Connection to James 1996 [44]

In [44] (James & Hastie) (see section 4.2) a bias/variance-*effect* decomposition is presented that holds for any error function. In section 4.2 we chose to regard it as a general bias/variance decomposition. This was not the intention of James & Hastie, since they defined bias and variance differently from bias-*effect* and variance-*effect*, which were introduced because the bias and variance do not generally form a proper decomposition of the average generalization error. The deviance error functions derived from the exponential family of distributions are exactly the error functions for which the bias and variance form a proper decomposition. To see that, we begin with the definitions of bias and variance in [44]. Variance is defined as

$$\mathrm{var}(f) = \langle E(f, Sf)\rangle_{\underline{\mathbf{F}},\underline{\mathbf{T}}} = \langle E(f, Sf)\rangle_{\underline{\mathbf{F}}}$$

and bias is defined as

$$\mathrm{bias}(St, Sf) = \langle E(St, Sf)\rangle_{\underline{\mathbf{F}},\underline{\mathbf{T}}} = E(St, Sf)$$

The symbol S denote the systematic mean operator (see definition 15 in appendix C). The definition is slightly unclear in [44], but it is reasonable to believe that Sf is equivalent with the average predictor $\bar{f}$ and St

is equivalent with the average target $\bar{t}$. In that case the bias and variance terms are identical to the bias and variance terms in (5.11), which is the bias/variance decomposition for the deviance error functions derived from the exponential family. From section 5.5 we know that these error functions are the only error functions with that decomposition. Furthermore, the bias/variance-*effect* decomposition reduces to the bias/variance decomposition in (5.11) for the deviance error functions derived from the exponential family. This property can be seen by showing equivalence for each term in the decomposition.

Recall the form of the deviance error functions derived from the one-parameter exponential family of distributions

$$E(t, f) = [c(t) - c(f)]T(t) + d(t) - d(f),$$

and the bias/variance decomposition from (5.11)

$$\langle E(t, f)\rangle_{\underline{\mathbf{F}},\underline{\mathbf{T}}} = \langle E(t, \bar{t})\rangle_{\underline{\mathbf{T}}} + E(\bar{t}, \bar{f}) + \langle E(\bar{f}, f)\rangle_{\underline{\mathbf{F}}},$$

- The bias-effect is equivalent to the bias for the deviance error functions. The bias-effect is given by $BE(t, Sf) = \langle E[t, Sf] - E[t, St]\rangle_{\underline{\mathbf{T}}}$:

  $$\begin{aligned}
  &BE(t, \bar{f}) = \langle E[t, \bar{f}] - E[t, \bar{t}]\rangle_{\underline{\mathbf{T}}} = \\
  &\langle [c(t) - c(\bar{f})]T(t) + d(t) - d(\bar{f}) - [c(t) - c(\bar{t})]T(t) - d(t) + d(\bar{t})\rangle_{\underline{\mathbf{T}}} = \\
  &[c(\bar{t}) - c(\bar{f})]\langle T(t)\rangle_{\underline{\mathbf{T}}} - d(\bar{f}) + d(\bar{t}) = \\
  &[c(\bar{t}) - c(\bar{f})]T(\bar{t}) - d(\bar{f}) + d(\bar{t}) = \\
  &E(\bar{t}, \bar{f}) = \text{bias}(\bar{t}, \bar{f}),
  \end{aligned}$$

  where corollary C2 has been used.

- The variance-effect is equivalent to the variance for the deviance error functions.
  The variance-effect is given by $VE(t, f) = \langle E[t, f] - E[t, Sf]\rangle_{\underline{\mathbf{F}},\underline{\mathbf{T}}}$:

  $$\begin{aligned}
  &VE(t, f) = \langle E[t, f] - E[t, \bar{f}]\rangle_{\underline{\mathbf{F}},\underline{\mathbf{T}}} = \\
  &\langle [c(t) - c(f)]T(t) + d(t) - d(f) - [c(t) - c(\bar{f})]T(t) - d(t) + d(\bar{f})\rangle_{\underline{\mathbf{F}},\underline{\mathbf{T}}} = \\
  &\langle [c(\bar{f}) - c(f)]\langle T(t)\rangle_{\underline{\mathbf{T}}} - d(f) + d(\bar{f})\rangle_{\underline{\mathbf{F}}} = \\
  &\langle [c(\bar{f}) - c(f)]T(\bar{t}) - d(f) + d(\bar{f})\rangle_{\underline{\mathbf{F}}} = \\
  &\langle [c(\bar{f}) - c(f)]T(\bar{t})\rangle_{\underline{\mathbf{F}}} + \langle d(\bar{f}) - d(f)\rangle_{\underline{\mathbf{F}}} = \\
  &0 + \langle d(\bar{f}) - d(f)\rangle_{\underline{\mathbf{F}}} = \\
  &E(\bar{f}, f) = \text{var}(f),
  \end{aligned}$$

  where corollary C1 and C2 has been used.

- The intrinsic noise is equivalent to the intrinsic noise for the deviance error functions

$$\text{var}(t) = \langle E[t, St] \rangle_{\underline{\mathbf{F}},\underline{\mathbf{T}}} = \langle E(t, \bar{t}) \rangle_{\underline{\mathbf{T}}}.$$

We conclude that the family of deviance error functions derived from the one-parameter exponential family of distributions are the error functions for which the bias-effect/variance-effect decomposition reduces to the bias/variance decomposition defined by the bias and variance terms from [44]. Since the family of deviance error functions derived from the one-parameter exponential family of distributions are the only error functions for which the bias/variance decomposition in (5.11) holds, we have that the family of error functions for which the bias-effect/variance-effect decomposition reduces to the bias/variance decomposition defined by the bias and variance terms from [44] and the family of error functions that can be decomposed as in (5.11) are identical.

5.7.2 Connection to Heskes 1998 [39]

Comparing the results in section 4.4 from [39] due to Tom Heskes and the results in 5.6 it is easy to see that there is a connection between the Kullback-Leibler error functions and the deviance error function from 5.4. We show that the deviance error function derived from the one-parameter exponential family of distribution can be reformulated in term of the Kullback-Leibler error.

The deviance error function is given by

$$E(t, f) = \log \frac{p(t|t)}{p(t|f)}$$

Let p be a density from the one-parameter exponential family of distribution. The density is reparameterized according to the method in section 5.4. The density then obeys

$$\forall z : \hat{\phi} = \underset{\phi}{\text{argmax}}\ p(z|\phi) = z$$

The density has the form

$$p(z|\phi) = \exp[c(\phi)T(z) + d(\phi) + S(z)].$$

Because of the reparameterization the functions c, T, and d obey the constraint

$$d'(y) = -c'(y)T(y).$$

We will show that the deviance error function can be reformulated as

$$E'(t', f') = \int_t p(t|t') \log \frac{p(t|t')}{p(t|f')} \tag{5.13}$$

We think of t' and f' as instances of respectively the target and the predictors viewed as stochastic variables $\underline{\mathbf{T}}$ and $\underline{\mathbf{F}}$. Furthermore, we think of $p(t|t')$ as the true density of the targets conditioned on t'. This is not mathematically necessary, but makes it easier to comprehend. The error function E' can be written as $E'(t', f') = \langle \log \frac{p(t|t')}{p(t|f')} \rangle_{\underline{\mathbf{T}}}$ or in terms of the functions c, T, and d

$$E'(t', f') = [c(t') - c(f')]\langle T(t) \rangle_{\underline{\mathbf{T}}} + d(t') - d(f').$$

By using the general relation from appendix E (E.5) on the density $p(z|\phi)$ we find $\langle T(t) \rangle_{\underline{\mathbf{T}}} = T(t')$.

Proof:

$$\begin{aligned} \langle \frac{\partial \log p(t|t')}{\partial t'} \rangle_{\underline{\mathbf{T}}} = 0 \Leftrightarrow \\ \langle c'(t')T(t) + d'(t') \rangle_{\underline{\mathbf{T}}} = 0 \Leftrightarrow \\ \langle T(t) \rangle_{\underline{\mathbf{T}}} = -\frac{d'(t')}{c'(t')} \Leftrightarrow \\ \langle T(t) \rangle_{\underline{\mathbf{T}}} = T(t'), \end{aligned}$$

where the constraint on d has been used. Now the error function can be rewritten as

$$E'(t', f') = [c(t') - c(f')]T(t') + d(t') - d(f'),$$

which is exactly the deviance error $E(t', f')$.

The error function in (5.13) is also a Kullback-Leibler error function comparing the two densities $p(t|t')$ and $p(t|f')$, yielding $KL(p(t|t'), p(t|f')) = E'(t', f') = E(t', f')$. What is left to show is the connection between the average predictor $\bar{f}$ and the average estimator $\bar{p}$. Note that it was shown in section 4.4 that the average estimator is in the same class as the set of estimators, if the estimators all are members of the same class of one-parameter exponential densities. So the average estimator is in the same class as $p(t|f')$. Let the parameter of the average estimator be $\bar{\theta}$ then $\bar{p} = p(t|\bar{\theta})$. The definition of the average estimator is

$$\bar{p} = \underset{p}{\operatorname{argmin}} \langle KL(p, q) \rangle_{\underline{\mathbf{Q}}},$$

where the mean is over the estimators q. For the densities $p(t|t')$ and $p(t|f')$ the average estimator is

$$p(t|\bar{\theta}) = \underset{p(t|\theta)}{\operatorname{argmin}} \langle KL(p(t|\theta), p(t|f)) \rangle_{\underline{\mathbf{F}}},$$

where the running parameter to argmin is θ. Reformulated for the deviance error function we have

$$\bar{\theta} = \underset{\theta}{\operatorname{argmin}} \langle E(\theta, f) \rangle_{\underline{\mathbf{F}}}$$

This is exactly the definition of the average predictor $\bar{f}$.

We conclude that for reparameterized densities from the one-parameter exponential family of distributions the error function derived by the deviance as in section 5.4 and by the Kullback-Leibler divergence as in section 4.4 are identical. Furthermore, the parameter in the average estimator and in the average predictor are identical. Implying that the bias/variance decompositions are mathematically identical.

5.8 Conjugated families of posterior densities

The densities we have investigated from the one-parameter exponential families of distributions are denoted $p(z|\theta)$, where the outcome z is associated with the target t and the parameter θ is associated with the predictor f. In a Bayesian setting, the parameter θ can be viewed as the outcome of a stochastic variable $\underline{\Theta}$. The density of $\underline{\Theta}$ is denoted $\pi_1(\theta|z)$ and is connected to p by

$$\pi_1(\theta|z) \propto \pi_2(\theta) p(z|\theta).$$

This is a simplified version of Bayes' formula. The density $\pi_2(\theta)$ is the *prior* density, which expresses some fundamental (prior) knowledge about θ, before the outcome z is know. The density $\pi_1(\theta|z)$ is the *posterior* density. If π_1 and π_2 are from the same class of densities, then that class of densities is called *conjugated* (see [6] section 2.4).

We will use conjugated densities for three things

- In the case where there are a number of targets for a given input point, minimizing the error function does not generally yield an uniform mean of the targets as one might expect. We show that minimizing the error function is the optimal prediction.
- The examples of error functions in section 5.6 are identical in pairs, except the predictor and target are interchanged. We show that these pairs are connected by Bayes' formula. The transformation will be called *Transpose*.
- An alternative definition of the deviance error function exists, where the predictor is associated with the outcome, and the target is associated with the parameter. We show that the two definitions are connected by Bayes' formula.

First we define and discuss Bayes' formula and conjugated densities in greater detail.

In the Bayesian setting the parameter(s) in a density function can be regarded as outcome of a stochastic variable themselves. The density of the parameters can be found with Bayes' Formula. The formula for a one-parameter density $p(z;\theta)$ is

$$p(z;\theta) = \exp[c(\theta)T(z) + S(z) + d(\theta)]. \tag{5.14}$$

In Bayesian setting the parameter θ can be considered an outcome of a stochastic variable $\underline{\Theta}$, so the density $p(z;\theta)$ is conditional on θ: $p(z;\theta) = p(z|\theta)$.[1] Given a prior density $\pi_2(\theta)$ the posterior density of θ conditioned on z can be found with Bayes' formula

$$\pi_1(\theta|z) = \frac{\pi_2(\theta)p(z|\theta)}{\int dt\pi_2(t)p(z|t)} \tag{5.15}$$

The normalization constant $\int dt\pi_2(t)p(z|t)$ does not dependent on θ, which yields

$$\pi_1(\theta|z) \propto \pi_2(\theta)p(z|\theta),$$

If the densities π_1 and π_2 are in the same class, they are called conjugated.

Based on the density in (5.14) a family of conjugated densities can be found. Let π_2 be given by a two-parameter member of the exponential family

$$\pi_2(\theta|a,b) = \exp[aT_2^1(\theta) + bT_2^2(\theta) + d_2(a,b)].$$

The form of the density - two parameters and linear canonical link functions - is chosen with the benefit of hindsight.

The density π_1 is given by $\pi_1(\theta|\alpha,\beta) \propto \pi_2(\theta|a,b)p(\theta|z)$

$$\pi_1(\theta|\alpha,\beta) \propto \exp[aT_2^1(\theta) + bT_2^2(\theta) + d_2(a,b) + c(\theta)T(z) + S(z) + d(\theta)]$$

We are looking for a conjugated posterior, so π_1 must be on the form

$$\pi_1(\theta|\alpha,\beta) = \exp[\alpha T_2^1(\theta) + \beta T_2^2(\theta) + d_1(\alpha,\beta)].$$

One solution is to set $T_2^1 = c$ and $T_2^2 = d$:

$$\pi_1(\theta|\alpha,\beta) \propto \exp[(a + T(z))T_2^1(\theta) + (b+1)T_2^2(\theta)] \exp[d_2(a,b) + S(z)]$$

The factor $\exp[d_2(a,b) + S(z)]$ has no influence, and is replaced with the normalization factor $\exp d_1(\alpha,\beta)$ for the normalized density. The density π_1 and π_2 are in the same class if we associate α with $a+T(z)$ and $\beta = b+1$. So

[1] We do not normally distinguish between $p(z;\theta)$ and $p(z|\theta)$. Here we see that the difference is only in the perspective.

any one-parameter density on the form $p(z|\theta) = \exp[c(\theta)T(z) + d(\theta) + S(z)]$ has a corresponding conjugated two-parameter posterior density on the form

$$\pi(\theta|\alpha(z), \beta) = \exp[\alpha(z)c(\theta) + \beta d(\theta) + d_1(\alpha(z), \beta)].$$

Let the density $p(z|\phi)$ obey the constraint in (5.8) and let z' be a specific outcome. The prior density $\pi(\phi|\alpha(z'), \beta)$ can now be interpreted as the density of the parameter ϕ given some specific parameters $\alpha(z')$ and β. What can be said about the posterior density? We know that since the density p obeys the constraint, the maximum likelihood of the parameter ϕ is $\hat{\phi} = z'$. Furthermore, before z' is known, it cannot be assumed that one value of ϕ is more likely than another. The prior density must express the lack of knowledge. In that case we would expect the posterior log likelihood to be maximal for $\phi = z'$. The maximal posterior density can be found from:

$$\begin{aligned}
\frac{\partial \log p(\phi|\alpha(z'), \beta)}{\partial \phi} &= 0 \Leftrightarrow \\
\frac{\partial \alpha(z')c(\phi) + \beta d(\phi) + d(\alpha(z'), \beta)}{\partial \phi} &= 0 \Leftrightarrow \\
\alpha(z')c'(\phi) + \beta d'(\phi) &= 0 \Leftrightarrow \\
\alpha(z')c'(\phi) &= \beta c'(\phi)T(\phi) \Leftrightarrow \\
\alpha(z') &= \beta T(\phi).
\end{aligned}$$

Remember that $\alpha = a + T(z')$ and $\beta = b + 1$ so only for $a = b = 0$ can the maximal log likelihood of ϕ be equal to z'. Setting $a = b = 0$ yields a constant prior density. This is exactly the prior that expresses no knowledge about ϕ. The problem is that a constant prior density can not be normalized. For the Bayes' formula (5.15) the normalization terms in the prior cancels out in the quotient and is therefore of no consequence. Such a prior is called an *improper* prior. We conclude that setting the parameters a and b to zero fulfill the requirement of no prior knowledge and the maximal log likelihood of ϕ be equal to t. The posterior density becomes

$$\pi(\phi|T(z'), 1) = \exp[T(z')c(\phi) + d(\phi) + d_1(T(z'), 1)]$$

This can be reinterpreted as an one-parameter member of the exponential family of distributions

$$\pi(\phi|z') = \exp[T(z')c(\phi) + d_1(z') + d(\phi)] \tag{5.16}$$

Remember that the density π is the posterior of the density p that obeys the constraint (5.8). This density corresponds to the deviance error $E(t, f)$. The density π in the one-parameter version can also be used in our definition of deviance error function.

$$E_\pi(t, f) = [T(t) - T(f)]c(t) + d_1(t) - d_1(f) \tag{5.17}$$

Instead of reparameterizing we would like the defining functions - the old canonical link function and the old sufficient statistic function - to remain unchanged in the deviance error function derived from π. The error function must still obey the constraint (5.8), so $d_1'(y) = -T'(y)c(y)$. This can be ensured by setting

$$d_1(y) = -d(y) - T(y)c(y)$$

It is easy to verify that d_1 now obeys the constraint. The deviance error function derived from the density π will be called the *transposed* error function of the error function $E(t, f)$.

$$\begin{aligned} E_T(t, f) &= [T(t) - T(f)]c(t) + d_1(t) - d_1(f) \\ &= [T(t) - T(f)]c(t) - d(t) - T(t)c(t) + d(f) + T(f)c(f) \\ &= [c(f) - c(t)]T(f) + d(f) - d(t) \end{aligned} \tag{5.18}$$

To go from the error function $E(t, f)$ to the transposed error function $E_T(t, f)$ the predictor and the target are interchanged. In section 5.6 there are examples of pairs of error functions that differed only in the interchanging of target and predictor. The connection between the pairs is now established as the connection between the density, from which the error function was derived, and the posterior density with only the very natural assumption that there was no prior knowledge about the distribution of the parameter in the original density.

In the definition of the deviance error function we associated the target t with the outcome and the predictor f with the parameter. The minimization of the error function for the predictor is easily interpreted as the maximization of the log likelihood of the parameter. An alternative definition of the deviance error function has the role of the predictor and the target interchanged

$$E_C(t, f) = \log \frac{p(f|f)}{p(f|t)}. \tag{5.19}$$

With the benefit of hindsight $E_C(t, f)$ will be called for the *conjugated* deviance error function.

For a one-parameter density from the exponential family of distribution that obeys the constraint (5.8), the conjugated deviance error function is given by

$$E_C(t, f) = [c_C(f) - c_C(t)]T_C(f) + d_C(f) - d_C(t). \tag{5.20}$$

Having $d_C'(y) = -c_C'(y)T_C(y)$ (the constraint) ensures that error function $E_C(t, f)$ obeys the requirements in figure 5.1, and therefore have a "nice" bias/variance decomposition. The direct statistical interpretation of the

conjugated deviance error function is difficult because the predictor is associated with the outcome and the target is associated with the parameter. It can be interpreted as the deviance error function derived from the posterior density π of a density p. In that case the interpretation makes more sense. The outcome, with which the predictor is associated, is really the (stochastic) parameter of the density p. Optimizing the predictor by minimizing the error function $E_C(t, f)$ can be interpreted as maximizing the log likelihood of the parameter for the density p. The connection between the posterior density and the density p is given by (5.16). Let the density p be on the form

$$p(z|\theta) = c(\theta)T(z) + d(\theta) + S(z)$$

Comparing $E_\pi(t, f)$ (5.17) and $E_C(t, f)$ (5.20) yields that $c(y) = T_C(y)$ and $T(y) = c_C(y)$. As with the transposed error function we would like the link between the posterior density and the density p to be the defining function c and T, so in order for the density p to obey the constraint (5.8) we set $d(y) = -d_C(y) - c(y)T(y)$. The conjugated deviance error function expressed in terms of the c, T and d becomes

$$\begin{aligned} E_C(t, f) &= [c_C(f) - c_C(t)]T_C(f) + d_C(f) - d_C(t) \\ &= [T(f) - T(t)]c(f) - d(f) - c(f)T(f) + d(t) + c(t)T(t) \\ &= [c(t) - c(f)]T(t) + d(t) - d(f) \\ &= E(t, f). \end{aligned}$$

The conjugated deviance error function $E_C(t, f)$ is identical to the normal deviance error function $E(t, f)$.

The posterior density can be used for yet another purpose. We will look at the situation were there are n targets ($n > 1$) for a single input point.[2]

The error function for the n targets is given by the sum

$$\begin{aligned} E(\vec{t}, f) &= \frac{1}{n}\sum_1^n E(t_i, f) \\ &= \frac{1}{n}[\sum_i^n c(t_i)T(t_i) - c(f)\sum_i^n T(t_i) + \sum_i^n d(t_i) - nd(f)] \end{aligned}$$

The optimal predictor f_{opt} is the predictor that minimizes the error function

[2] An almost similar situation is were the input point for n targets are very close together so the predictor is effectively constant. The analysis also holds for this situation.

$E(\vec{t}, f)$

$$\begin{aligned}
\frac{\partial E(\vec{t}, f)}{\partial f} = 0\Big|_{f=f_{opt}} &\Leftrightarrow \\
\frac{\partial \frac{1}{n}[\sum_i^n c(t_i)T(t_i) - c(f)\sum_i^n T(t_i) + \sum_i^n d(t_i) - nd(f)]}{\partial f} = 0\Big|_{f=f_{opt}} &\Leftrightarrow \\
-\frac{1}{n}c'(f)\sum_i^n T(t_i) - d'(f) = 0\Big|_{f=f_{opt}} &\Leftrightarrow \\
c'(f)T(f)\frac{1}{n} = c'(f)\sum_i^n T(t_i)\Big|_{f=f_{opt}} &\Leftrightarrow \\
f_{opt} = T^{-1}(\frac{1}{n}\sum_i^n T(t_i)) &
\end{aligned}$$

The result is slightly surprising. The optimal predictor is generally not linear in the targets, as one is used to from the MSE error. For the MSE the sufficient statistic T is the identity function and we get the expected result $f_{opt} = \frac{1}{n}\sum_i^n t_i$.

The optimality of f_{opt} can be verified by using the posterior density for the n targets. The likelihood for each target is $p(t_i|\phi)$. Assuming independently sampling of the targets, the combined likelihood is given by

$$p(\vec{t}|\phi) = \prod_i^n p(t_i|\phi)$$

Letting p be an one-parameter exponential density and assuming a improper prior (constant prior), it is easy to extend the posterior in (5.16) to the n targets:

$$\pi(\phi|\vec{t}, n) = \exp[\sum_i^n T(t_i)c(\phi) + d_1(\vec{t}, n) + nd(\phi)] \tag{5.21}$$

For a single target t the maximum log likelihood of the parameter ϕ for the posterior density is at t, reflecting that the predictor should estimate the target, so the maximum log likelihood of ϕ for multiple targets reflects what the predictor should estimate. The maximal log likelihood of the parameter

for the posterior density is

$$\begin{aligned}
\frac{\partial \log \pi(\phi|\vec{t}, n)}{\partial \phi} = 0\Big|_{\phi=\hat{\phi}} \Leftrightarrow \\
\sum_i^n T(t_i)c'(\phi) + nd'(\phi) = 0\Big|_{\phi=\hat{\phi}} \Leftrightarrow \\
\sum_i^n T(t_i)c'(\phi) - nc'(\phi)T(\phi) = 0\Big|_{\phi=\hat{\phi}} \Leftrightarrow \\
\hat{\phi} = T^{-1}(\frac{1}{n}\sum_i^n T(t_i)).
\end{aligned}$$

The optimal predictor found by minimizing the error function is identical to the maximum posterior log likelihood of the parameter.

We conclude this section by summarizing the results

- The transposed error function $E_T(t, f)$ is related to the deviance error function $E(t, f)$ through the conjugated posterior given by (5.16). The posterior error function obeys the requirements in figure 5.1. It is given by
$$E_T(t, f) = [c(f) - c(t)]T(f) + d(f) - d(t),$$
where the functions c, T, and d are from the deviance error function $E(t, f)$.

- The conjugated deviance error function is an alternative definition to the deviance error function.
The predictor is associated with the outcome of a density π, and the target is associated with the parameter. The definition is
$$E_c(t, f) = \log \frac{\pi(f|f)}{\pi(t|f)}$$
If the density π is a member of the one-parameter exponential family of distribution and obeys the constraint (5.8), then the conjugated deviance error function obeys the requirements in figure 5.1.

  If the density π is the conjugated posterior of a density p as in (5.16), the conjugated deviance error function is identical to the normal deviance error function derived from p.

- The optimal predictor for a set of targets with identical input points is
$$f_{opt} = T^{-1}(\frac{1}{n}\sum_i^n T(t_i))$$
Minimizing the error of the examples $\sum_i^n E(t_i, f)$ with regard to the predictor yields the same value.

5.9 Ambiguity for deviance error functions

The bias/variance decomposition for the deviance error functions can be used to define an ambiguity decomposition (see section 3.3). Ambiguity is defined for an ensemble of predictors in connection with an ensemble method. This means we have an finite ensemble of ensemble members $\{f_1, \ldots, f_M\}$. Each ensemble member f_i has associated a weight α_i. Normally the weights sum to one and are positive. In that case the weights can to viewed as probabilities and defines a mean operator over the ensemble members

$$\langle g(f) \rangle_{\underline{\mathbf{F}}} = \sum_i^M \alpha_i g(f_i).$$

The deviance error functions derived from the one-parameter exponential family of distributions have the form

$$E(f,t) = [c(t) - c(f)]T(t) + d(t) - d(f)$$

The combined predictor for the ensemble is defined similar to the average predictor (Corollary C1):

$$F = c^{-1}(\langle c(f) \rangle_{\underline{\mathbf{F}}}) = c^{-1}(\sum_i^M \alpha_i c(f)).$$

The ambiguity decomposition for F is

$$E(F,t) = \langle E(f,t) \rangle_{\underline{\mathbf{F}}} - \langle E(F,f) \rangle_{\underline{\mathbf{F}}},$$

where $\langle E(f,t) \rangle_{\underline{\mathbf{F}}}$ is the mean error of the ensemble members, and $\langle E(F,f) \rangle_{\underline{\mathbf{F}}}$ is the deviance ambiguity. The ambiguity is independent of the target function, thus it can be estimated by unlabeled data.

Chapter 6

Machine learning methods

There are many different machine learning methods. Some of them are kernel-based methods, radial basis methods, nearest neighbor methods [17], support vector methods [18], tree predictor methods such as C4.5 [65] and CART (Classification And Regression Trees) [13], splines (see section 6.4 in [48]) such as MARS (Multivariate Adaptive Splines) [24]. Two references discussing many of the mentioned methods are [7] and [27].

In this section only the commonly used machine learning method, neural network with back-propagation [75], will be discussed.

6.1 Neural Network

An (artificial) neural network is a predictor that is inspired by the first models of the human brain (see introduction in [37]). There are many types of neural networks. The standard feed-forward neural network will be presented. In figure 6.1 is a picture of a neural network with one hidden layer. Each layer consists of a number of nodes connected by weights. The first layer is the *input* layer, while the last layer is the *output* layer. The number of weights in layer i is denoted n_i, so in a fully connected network there are $n_i \times n_{i+1}$ weights between layer i and $i+1$. The weights between layer i and $i+1$ are denoted w^i_{kj}, where the subscripts k and j indicates the weight connecting node number j in layer i to node number k in layer $i+1$. The *state* of node j in layer i is denoted by l^i_j. For all layers, except the input layer, the state of the nodes is given by

$$l^{i+1}_j = \sum_{k=1}^{n} w^i_{jk} g^i_k(l^i_k) - w^i_0,$$

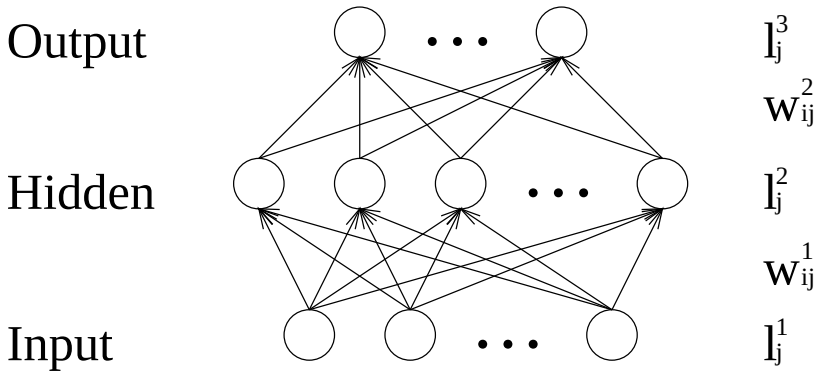


Figure 6.1: Structure of a neural network.

where w_0^i is the threshold weight and g_k^i is the activation function. Normally, all the activation functions are identical, so g is used for all activation functions. One can view the threshold weight as coming from a node l_0^i that is "hard-wired" so $g(l_0^i) = -1$. In that way the state of node l_j^{i+1} can be simplified to

$$l_j^{i+1} = \sum_{k=0}^{n} w_{jk}^i g(l_k^i). \tag{6.1}$$

There are different kinds of activation functions. Among the commonly used are the Sigmoid function

$$g(a) = \frac{1}{1 + e^{-a}}$$

and the hyperbolic tangent function

$$g(a) = tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}.$$

For a discrete neural network the sign function can be used as activation function. In appendix B it is shown that a particular choice of activation function can be of theoretical and practical importance.

A neural network nn is a function $\vec{y} = nn(\vec{x})$, where $\vec{x}$ is the input and $\vec{y}$ is the output vector. The neural network function nn is evaluated by setting the nodes in the input layer equal to the scalars in the input vector $\vec{x}$, then the states of the layers are calculated layer for layer by the formula in (6.1). This is called feed-forward propagation. The output of the neural network can be read from the nodes in the output layer. Note that by definition there is no activation function for the output layer, but often the states of the output nodes are post-processed by some function. Sometimes the *post-processing function*[1] depends on the states of all the nodes, e.g. the

[1]The post-processing function is in some situations called transfer function, output activation function, or squashing function.

SOFTMAX post-processing function (2.1) (see chapter 11 for an application of the SOFTMAX post-processing function). Let the output of the neural network (the state of the nodes in the output layer) be $\vec{y}$ then the post-processed output is for the post-processing function pp given by $\vec{f} = pp(\vec{y})$. In this dissertation it will be clear from the context if the output in question is the post-processed output or the output of the neural network.

A neural network can be trained using the gradient descent optimization methods. This is presented in section 6.2.

6.2 Back propagation

If the predictor f is a neural network, the error function E can be minimized by the gradient descent optimization method (see appendix I). The parameters that are to be optimized are the weights of the neural network. Let the predictor f depend on weights $\vec{w} = \{w_1, \ldots, w_n\}$, which is signified by writing $f(x; \vec{w})$. The notation $E(\vec{w}) = E(f(\vec{x}; \vec{w}), t)$ is used, because the free parameters are the weights $\vec{w}$. The first order Taylor expansion of $E(\vec{w})$ is

$$E(\vec{w} + d\vec{w}) \approx E(\vec{w}) + \frac{\partial E(\vec{w})}{\partial \vec{w}} \cdot d\vec{w}$$

The vector $\frac{\partial E(\vec{w})}{\partial \vec{w}} = \{\frac{\partial E(\vec{w})}{\partial w_1}, \ldots, \frac{\partial E(\vec{w})}{\partial w_n}\}$ is the gradient. Since we want $E(\vec{w} + d\vec{w}) < E(\vec{w})$ we must have $\frac{\partial E(\vec{w})}{\partial \vec{w}} \cdot d\vec{w} < 0$. This is guaranteed under the approximation if $d\vec{w} = -\frac{\partial E(\vec{w})}{\partial \vec{w}}$. The gradient descent method has the following update rule

$$\vec{w}' = \vec{w} - r\frac{\partial E(\vec{w})}{\partial \vec{w}},$$

where r is called the learning rate. The learning rate is a user defined parameter that controls the rate of descent. Too high a learning rate, and the method cannot find the global minimum because it "overshoots". Too low a learning rate, and the method stops in a local minimum, or learning takes "forever". The right learning parameter is notoriously difficult to find.

For a neural network, the gradient descent method is called back propagation, because the error is propagated from the output layer and back towards the input layer. We assume that the post-processing function is the identity function. The number of layers is N. The number of nodes in the last layer is n_N and thereby also the size of the output of the neural network. We assume that the error for one input is given by

$$E(f(\vec{w}), t) = \sum_{h=0}^{n_N} E(l_h^N, t),$$

which is the straightforward linear generalization of the error for a predictor with scalar output. Let $E'(l_h^N)$ denote $\left.\frac{\partial E(x)}{\partial x}\right|_{x=l_h^N}$ and let g' be the derivative of the activation function. The gradient is recursive in an auxiliary parameter δ_j^i called the *back-propagated error*:

$$\delta_j^i = \begin{cases} E'(l_j^N) & i = N \\ g'(l_j^{i+1}) \sum_h^{n_{i+1}} \delta_h^{i+1} w_{hj}^i & i < N \end{cases}$$

The gradient for the weight w_{jk}^i is given by

$$\frac{\partial E(\vec{w})}{\partial w_{jk}^i} = \delta_j^{i+1} g(l_k^i) \tag{6.2}$$

Note that the gradient can be calculated efficiently in time proportional to the number of weights plus the number of nodes. There are two main ways of updating the weights. The deterministic update rule, where the gradient is accumulated for all training examples, and then the weights are updated. The other update rule is the stochastic update rule, where the weights are updated for each training example. The stochastic update rule is generally an order of magnitude faster than the deterministic, but does not guarantee convergence. An intermediate update rule is batch update where the gradient is accumulated for a number of training examples, usually much smaller than the number of training examples, before update.

The back propagation method is not regarded as the best learning method for neural networks (see [55, 7]), but it is by far the most commonly used method. To speed up convergence the back-propagation method is usually used with the addition of *momentum*. The momentum method works this way: Let Δw_{jk}^i be the amount the weight w_{jk}^i was changed in the last update. Find the gradient $\frac{\partial E(\vec{w})}{\partial w_{jk}^i}$ and set Δw_{jk}^i to

$$\alpha \Delta w_{jk}^i - \frac{\partial E(\vec{w})}{\partial w_{jk}^i} \rightarrow \Delta w_{jk}^i, \tag{6.3}$$

where $\alpha \in [0; 1[$ is the *momentum rate*. Update the weight according to

$$w_{jk}^i + \Delta w_{jk}^i \rightarrow w_{jk}^i.$$

Momentum dampens oscillating gradients and speeds up learning for small and constant sign gradients. For a more detailed description of momentum see [7] p. 267-268.

Chapter 7

Meta machine learning methods

Meta machine learning (MML) methods are the designation of all methods that combines a set of predictors (or experts as they are called in connection with mixtures of experts) to form a combined predictor.

Normally the MML methods are divided into two groups: The ensemble methods and the mixtures of experts (ME) methods. The ensemble methods can again be divided into two groups: The *parallel ensemble* methods (e.g. Bagging), and the *boosting ensemble* methods (e.g. AdaBoost). The lines between the groups are blurred. In figure 7.1 a sketch of the different methods is given. The sketch is somewhat simplified, but captures some of the characteristics normally associated with the different methods. The main components are pre-processing of input, training, and post-processing of output. Post-processing also constitutes the combination rule of the predictors.

The figure in 7.1 will be discussed in greater detail in section 7.1 for ensemble methods and in section 7.2 for ME methods. Six representatives will be presented and literature will be discussed.

Two important references are the book: "Combining Artificial Neural Nets" [79], that covers ensemble methods as well as ME methods, and the special issue of Connection Science: "Combining Artificial Neural Nets: Ensemble Approach" [80].

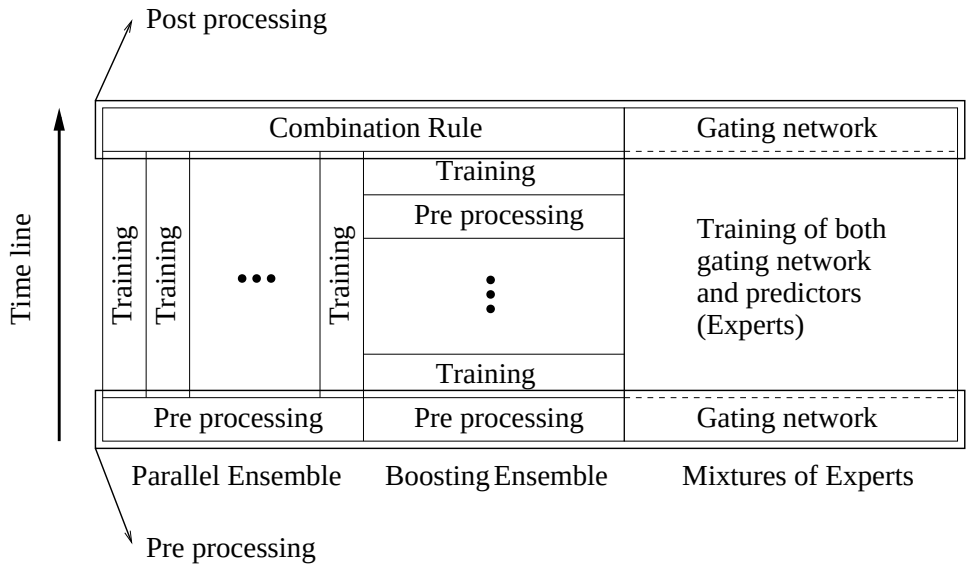


Figure 7.1: A simplified sketch of meta machine methods

7.1 Ensemble methods

The main point for ensemble methods is: "Two heads are better than one". The translation to machine learning language is that a group (ensemble) of predictors potentially gives better generalization than the individual predictors (ensemble members). The theoretical justification for regression with the MSE lies in the ambiguity decomposition (see section 3.2 or [51]). The ambiguity decomposition yields that the combined predictors always has a lower generalization error than the average of the ensemble members. The corresponding theoretical justification for classification can be found in [76], that states that a weak learner can be transformed into a strong learner. A weak learner produces predictors that with high probability, misclassify less than half the time (for a two class problem). A strong learner produces predictors that with high probability, misclassify with an arbitrary low probability. The crux of the constructive proof is an application of boosting, which will be discussed later. The result is that a weak learner that produces predictors that maximally misclassifies $a < \frac{1}{2}$ of the times, can - by combining three predictors - produce a (combined) predictor with maximal misclassification rate of $3a^2 - 2a^3 < a$. This can be repeated until a (combined) predictor is produced, that misclassifies with arbitrary low probability.

The two fundamental articles [51, 76] have been extended several times, e.g. the result in chapter 3 is an extension of [51] to cover many error functions, while [76] has been extended in e.g. [2] to include regression.

The two articles [51, 76] represent the two types of ensemble methods - parallel ensemble methods and boosting ensemble methods. In [51] an ensemble of predictors are combined by weighting to produce the combined predictor. It is assumed that the predictors differ to some degree due to mechanisms such as stochastic training, different architecture, or different training sets. The last of the methods covers different examples of pre-processing of the training set. The training set can be partitioned arbitrarily or by some criteria. The Bagging parallel ensemble method employs resampling with replacement (see section 7.1.2). In [51] it is shown how to find near optimal weights for the combination rule. This is an example of post-processing (see section 7.1.1). Without any further assumptions on the ensemble members, it is possible to train them in parallel, making it a parallel ensemble method. This is illustrated in figure 7.1 by having training concurrent on the time line.

In contrast, the boosting method in [76] requires sequential training of the ensemble members because the training set of one ensemble member depends on the training of the previously trained predictors. This is illustrated in figure 7.1 by having pairs of pre-processing and training sequential on the time line.

We use the practical version of the constructive proof in [76] as an example of boosting. The problem domain is two class classification. First a predictor (classifier) is trained on the entire training set. It is assumed that the predictor misclassifies less than $a < \frac{1}{2}$ of the time. Construct a new training set with all the examples on which the first predictor misclassifies and an equal number of correctly classified examples. A second predictor is trained on that training set, so it becomes an "expert" on the examples on which the first predictor failed. There is no gain in that, because if we present a new input to both predictors and they disagree, we cannot with any confidence say what class the input belongs to. Therefore a third predictor is trained on all the examples where the first two predictors disagree. This is the "tie-breaker" expert. The three predictors are combined using majority vote, and it can be shown,[1] that the misclassification rate of the combined predictor is less than $3a^2 - 2a^3$, which is lower than the misclassification rate of the first predictor. The training set has been *boosted* between training sessions. This has given name to the Boosting ensemble methods.

The majority vote mechanism is a very simple form of combination rule. In other boosting methods, e.g. AdaBoost (see section 7.1.5), the combination rule is more complicated and uses information about the ensemble members found during training.

[1] The proof is not for the practical version presented here, but for the PAC learning model (see e.g. [89]), which is beyond the scope of this dissertation. The practical version has been shown empirically to lower misclassification rates in most cases.

Below in sections 7.1.2, 7.1.3, and 7.1.4 three parallel ensemble methods are presented. The most popular boosting ensemble method, AdaBoost, is present in the regression version in section 7.1.5. Before the presentation of the ensemble methods, combination rules are discussed in section 7.1.1.

For further references covering interesting aspects of ensemble methods see [73, 60, 82, 76, 21]. In [73] a de-correlation term is added to the error function for the combined predictor to achieve diverse ensemble members. Another method of diversifying the ensemble members, by using a genetic algorithm, is given in [60]. A "dogma" for ensemble methods is that overfitting can be useful. A proof is given in [82]. A practical version of the constructive proof in [76] is presented and tested in [21].

7.1.1 Combination rules

A very important aspect of ensemble methods is how the ensemble members are combined. Some combination rules are very simple i.e. uniform weighting for regression or majority voting for classification. See [50, 49, 4] for other simple combination methods. More complex combination rules have been presented (see e.g. [82, 69, 35, 1]). AdaBoost (see section 7.1.5) uses a more complicated combination rule, where information found during training is incorporated.

Let us look at the general case. We have an example set T divided into a training set T' and the rest T''. We restrict the situation to be one-input/one-output problems, so $T = \{\vec{x}, \vec{y}\}$. An ensemble of predictors $\vec{f}$ is trained on T'. Let us assume that we have a combination function $F(x) = C(\vec{f}(x), x; \vec{w})$ parameterized by $\vec{w}$. The direct dependency on input x is unusual for a combination rule, but it has been used (one example can be found in [84]). The parameters $\vec{w}$ could be the weights of a regression ensemble. Furthermore, assume that we have a fitness function g for the combination function. That would typically be an estimate of the generalization error of the combined predictor, and that could be the error on the example set T''. In general, the fitness function can depended on T, $F(\vec{x})$, $\vec{f}(\vec{x})$, and $\vec{w}$. The general problem is to optimize $g(T, F(\vec{x}), \vec{f}(\vec{x}), \vec{w})$ with regard to $\vec{w}$. This is a particular instance of *stacking* [88]. Stacking has been used on Bagging [92] and on kernel functions [81, 69] to find optimal coefficients.

There is not far from the combination function $C(\vec{f}(x), x; \vec{w})$ to a gating network for mixtures of experts (see section 7.2). The difference between a complex combination function, that also depends on the input and a gating network is that a combination function is found after the ensemble members have been trained, while the gating network is trained concurrently with

the experts (corresponding to the ensemble members). Also see section 7.3 for further discussion about differences and similarities between ensemble methods and ME methods.

7.1.2 Bagging

In algorithm 1, the standard Bagging algorithm from [9, 10] is presented.

Algorithm 1 (Bagging)

1. *Choose machine learning method L and ensemble size M.*
2. *Generate M training sets T_i from the original set T by resampling with replacement. The size of the T_i's equals the size of T.*
3. *$f_i = L(T_i)$ for all i*
4. *Construct ensemble predictor $F(\vec{x}) = \frac{1}{M}\sum_{i=1}^{M} f_i(\vec{x})$.*

Bagging can be used to combine any kind of predictors. In [9] it is stated that the benefit of Bagging lies in variance reductions (for a description of bias and variance see [10, 27, 44] or chapter 3). This is achieved by generating an ensemble of different predictors, where the variance is "averaged out" in the combined predictor. The pre-processing is resampling with replacement of the training set. For each ensemble member about 37 % of the examples are not used for training (see appendix H). These have been used for other purposes in [15, 69]. In [15] the "unused" examples are used to estimate the generalization error of the combined predictor. In [69] the "unused" examples are used to estimate the weights of the ensemble members, which in the standard version are uniform.

The creator of Bagging, Leo Breiman, has extended Bagging in [12] by using the unused examples to reduce bias as well as variance.

Stacking has been used to estimate the generalization error of Bagging [92] and to improve learning [91].

Four versions of Bagging have been compared in [38]. These versions are Bumping, where the weight of the best ensemble member is set to one, Bagging in the standard version, and two versions where the weights are optimized to, respectively, maximize ambiguity and minimize the generalization error as in [82]. The last is called Balancing. Bumping is the worst performing method and Balancing is the best performing method.

7.1.3 Simple

Even though Bagging is a fairly simple ensemble method, it is possible to imagine an even more simple method: Train a group of predictors separately on the same training set and combine them by the linear average predictor (LAP) with uniform weights. We call this method for *Simple*. There is no pre-processing and the simplest possible combination rule is used.

If the machine leaning method used is deterministic on the training set, Simple will do no better than a single ensemble member. If the machine learning method is nondeterministic on the training set, the only benefit for Simple lies in the variation among the ensemble members, therefore we consider Simple as a natural "zero" for MML methods, i.e. all other methods should do better than Simple to have any merit.

7.1.4 Logarithmic opinion pool ensemble

A logarithmic opinion pool (LOP) ensemble is a classification ensemble that uses the logarithmic opinion pool (see section 4.4) as a combination function. The outputs are estimates of the probability of the classes. The LOP ensemble method is very similar to the Simple ensemble method (see section 7.1.3), only the combination rule is different.

The ensemble consists of M predictors f_i that each outputs a vector with class probabilities $\{f_i^{c_1}, \ldots, f_i^{c_N}\}$. Each ensemble member has associated a weight α_i. The weights obey $\sum_i^M \alpha_i = 1$ and $\alpha_i \geq 0$, so they can be regarded as the probabilities of the ensemble members. The weights define a mean operator:

$$\langle g(f) \rangle_{\underline{\mathbf{F}}} = \sum_i^M \alpha_i g(f_i).$$

The target $\vec{y}$ is a vector with the target class probabilities $\{y^{c_1}, \ldots, y^{c_N}\}$. Often the target is a class example, so $y^{c_j} \in \{0, 1\}$. The combined predictor F is also a class probability vector $\{F^{c_1}, \ldots, F^{c_N}\}$. The combination rule for a class c is the logarithmic opinion pool:

$$F^c = \frac{1}{Z} \exp \langle \log(f_i^c(\vec{x})) \rangle_{\underline{\mathbf{F}}}, \tag{7.1}$$

where Z is a normalization factor satisfying

$$Z = \sum_j^N \exp \langle \log(f_i^{c_j}(\vec{x})) \rangle_{\underline{\mathbf{F}}}.$$

This combination rule is non-linear and asymmetric as opposed to the linear average predictor.

The error function is the Kullback-Leibler (KL) error function. The error on target $\vec{y}$ and combined predictor F is given by

$$E(\vec{y}, F) = \sum_j y^{c_j} \log\left(\frac{y^{c_j}}{F^{c_j}}\right). \tag{7.2}$$

The error is zero if F^c is equal to y^c for all c. If the target probabilities are restricted to one and zero, the error function (7.2) reduces to $E(\vec{y}, F) = -\log(F^k)$, where y^k is one. This would be the case if the error function is used on a training set consisting of class examples.

The error in (7.2) can be decomposed into two terms:

$$\begin{aligned} E(\vec{y}, F) &= \sum_{i=1}^{M} \alpha_i E(\vec{y}, f_i) - \sum_{i=1}^{M} \alpha_i E(F, f_i) \\ &= \langle E(\vec{y}, f_i) \rangle_{\underline{\mathbf{F}}} - A(f), \end{aligned} \tag{7.3}$$

where $A(f)$ is the ambiguity and $\langle \cdot \rangle$ is the weighted ensemble mean.

In chapter 11 the LOP ensemble and an extension, the cross-validation LOP ensemble, is tested on an important real life problem: the prediction of the secondary structure of proteins. This has been published in [36].

The decomposition in (7.3) is due to Tom Heskes (see [39, 40] and sections 4.4 and 5.7.2). The view in [40] is different from the one taken here. In [40] the densities or probabilities are estimators (see definition 3), while the the LOP ensemble uses predictors that output class probabilities.

If the predictors are neural networks (see section 6.1) then a suitable post-processing function is the SOFTMAX function (2.1). This automatically ensures the necessary constraint on the post-processed output.

In appendix B training of a LOP ensemble with neural network predictors is discussed, and the Hessian matrix is investigated. An alternative activation function is proposed in appendix B that ought to eliminate some sources of poor learning.

In appendix J a surprising connection between a LOP ensemble and a physical canonical ensemble is discussed.

7.1.5 AdaBoost

The regressor version of AdaBoost presented here in algorithm 2 is from [20], which is a modification of *AdaBoost.R* in [95].

Algorithm 2 (AdaBoost)

1. *Choose machine learning method* L
2. *Let* T *be the training set with size* N, *Let* $\vec{P} = (p_1, ..., p_N)$, $p_i = \frac{1}{N}$. *Let* $k = 1$.
3. *Generate training set* T_k *by sampling from* T *with replacement, where the probability for sampling training example* i *is* p_i. *The size of* T_k *equals the size of* T.
4. $f_k = L(T_k)$.
5. *Let* $(l_1, \ldots, l_N)$ *be the loss vector, where*
$$l_i = \frac{|y_i - f_k(\vec{x_i})|^2}{D^2} \text{ and } D = \sup_i |y_i - f_k(\vec{x_i})|$$
6. *Calculate*
$$\beta_k = \frac{1 - \sum_{i=1}^{N} p_i l_i}{\sum_{i=1}^{N} p_i l_i}.$$
7. *Let*
$$p_m \leftarrow \frac{p_m \beta_k^{l_m}}{\sum_{i=1}^{N} p_i \beta_k^{l_i}} \text{ for all } m.$$
8. $k \leftarrow k + 1$. *If a stopping criterion is not satisfied, go to 3.*
9. *Construct ensemble predictor:*
$$F(\vec{x}) = \inf_y (y : \sum_{k: f_k(\vec{x}) \leq y} log(\beta_k) \geq \frac{1}{2} \sum_k log(\beta_k)) \tag{7.4}$$

The AdaBoost method consists of a number of boosting sessions. In each session a new training set is generated by sampling with replacement from the original training set, and a predictor is trained using this training set. The difference compared to Bagging, is that the probability of a training example being sampled is not uniform, but depends on the training error of previous predictors.

In step 5 the *loss* for each example is found. The loss is scaled so a correct learned example yields a loss of zero, while the example that by the absolute measure is most wrong yields a loss of one. Beside the square loss function in step 5 two other loss function is suggested in [20]. The linear loss function

is given by

$$l_i = \frac{|y_i - f_k(\vec{x_i})|}{D},$$

and the exponential loss function is given by

$$l_i = 1 - \exp\Big[- \frac{|y_i - f_k(\vec{x_i})|}{D} \Big].$$

Empirical testing has not showed significant differences in performance for the different loss functions, so the square loss is always used. In step 6 the β_k value is calculated. A high value indicates that the examples with high weights have been learned to great accuracy, and a low value indicates that the examples with high weights have not been learned to great accuracy. It is assumed that $\sum_{i=1}^{N} p_i l_i < 0.5$,[2] so the value of β_k is restricted to $[1; \infty[$. The loss vector and the β_k are used to find the resampling weights or probabilities $\vec{P}$. A high loss yields a high resampling weight, so the predictor in the next boosting session is trained mainly on examples where the current predictor did poorly.

The combination rule for AdaBoost uses information found during training. The output of the combined predictor is the median of the ensemble members weighted by the logarithm of the β_k values.

AdaBoost was designed to decrease the training error, but has been reported to also decrease the generalization error. In literature AdaBoost is considered to be one of the best performing ensemble methods.

7.1.6 Other boosting ensemble methods

A number of boosting methods have been developed. They share a fundamental trait with AdaBoost, namely that examples on which the current predictor does poorly are propagated (boosted) to the training set of the next predictor. In [10] the claim is that this trait is the reason for the success of AdaBoost. A heuristic boosting ensemble method, called Arching-x4, was developed to support the claim. In AdaBoost the connection between the resampling probability and the "error" of an example (the loss) is exponential (the probability is proportional to β^{loss}), while in Arching-x4 the connection is polynomial (the probability is proportional to $error^4$). Arching-x4 is reported in [10, 59] to achieve errors comparable to AdaBoost.

Other boosting ensemble methods and discussions hereof can be found in [22, 26, 23].

[2] The requirement $\sum_{i=1}^{N} p_i l_i < 0.5$ can be used as a stopping criterion in step 8

7.2 Mixtures of experts

In contrast to the ensemble methods, the view in mixtures of experts (ME) methods is not to combine a group of global predictors, but to decompose the problem into a set of subproblems that can be solved by local experts. Therefore the focus is more on the decomposition than on the experts. The experts can be very simple predictors [43] or generalized linear models (see appendix D or [53] for definition of generalized linear models and [93, 45] for examples of ME methods using generalized linear models), but also complex neural networks have been used as experts (see [42] and DynCo in section 7.2.2). The gating network (that corresponds to the combination function) is responsible for the decomposition of the problem. The decomposition will often be a decomposition of input space, so the gating network must depend on input. In that case the gating network works as both pre- and post-processing. The weights of the examples for a given expert are determined by the gating network. This corresponds to pre-processing of the training set. The gating network is also the combination function. This corresponds to post-processing. In figure 7.1 the boundary between training, pre- and post-processing are broken lines to illustrate that these "stages" is not separated in time.

Mixtures of experts methods, also called *modular network* methods, are often used in specialized versions to solve specific problems. e.g. in [28] the vowel-speaker problem was investigated using a specialized modular network called the Meta-pi network. Another specialized modular network was used on handwritten letter recognition in [67]. In [58] medical prognosis of survival of AIDS patients was done using a specialized modular network, while time series prediction was investigated in [87].

Below we present two ME methods. First an extension of 'Hierarchical Mixtures of Experts' [45] is presented in section 7.2.1. This version can be found in [94]. We call the method *XuME* after the author.

The method DynCo found in section 7.2.2 is similar to the method in [43], but was reinvented by the author of this dissertation. It will be given special attention. The relation of DynCo to other ME methods in the literature will be discussed in section 7.2.3.

7.2.1 XuME

The output of the combined function is a density estimation $P(\vec{y}|\vec{x})$, which is defined as

$$P(\vec{y}|\vec{x}) = \sum_j g_j(\vec{x}, \nu_j) P(\vec{y}|\vec{x}, \theta_j).$$

The parameters ν_j are defined later. The probabilities $P(\vec{y}|\vec{x},\theta_j)$ are defined as

$$P(\vec{y}|\vec{x},\theta_j) = \frac{1}{(2\pi)^{\frac{n}{2}}|\Gamma_j|^{\frac{1}{2}}} e^{-\frac{1}{2}(\vec{y}-\hat{\vec{y}}_j)^T\Gamma_j^{-1}(\vec{y}-\hat{\vec{y}}_j)},$$

where Γ_j is a covariance matrix. The output of the experts is the estimated mean: $\hat{\vec{y}}_j$. The experts are linear predictors: $\hat{\vec{y}}_j = w_j^T[\vec{x},1]$, where w_j are the weights and $\vec{x}$ is the input, so θ_j is $\{\Gamma_j, w_j\}$ and θ is $\bigcup_i \theta_j$. The notation $[\vec{x},1]$ represents the vector $(x_1,\ldots,x_n,1)^T$ if $\vec{x} = (x_1,\ldots,x_n)^T$.

The g_j's are the outputs of the gating network defined as

$$g_j(\vec{x},\nu) = \frac{\alpha_j P(\vec{x}|\nu_j)}{\sum_i \alpha_i P(\vec{x}|\nu_i)}, \sum_j \alpha_j = 1, \alpha_j \geq 0,$$

$$P(\vec{x}|\nu_j) = \frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma_j|^{\frac{1}{2}}} e^{-\frac{1}{2}(\vec{x}-m_j)^T\Sigma_j^{-1}(\vec{x}-m_j)},$$

where Σ_j is a covariance matrix. The outputs of the gating network are the coefficients by which the experts are weighted. The outputs can also be considered the probability of choosing an expert, so $g_j(\vec{x},\nu) = P(j|\vec{x})$. The $P(\vec{x}|\nu_j)$ depends on the parameters $\{\Sigma_j, m_j\}$, so ν_j is $\{\Sigma_j, m_j\}$ and ν is $\bigcup_i \nu_i$.

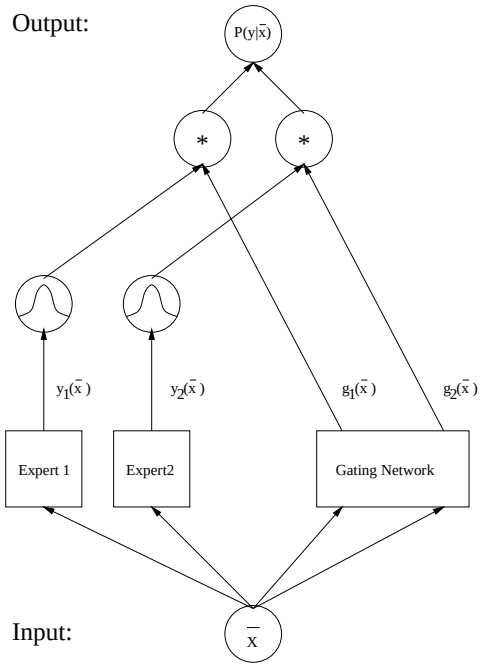


Figure 7.2: Mixtures of Experts.

An example of a two-expert ME is illustrated in figure 7.2. The input is distributed to the experts and the gating network. The experts are simple linear predictors, they output the mean of Gaussian distributions. The gating network outputs a probability for each expert. The density corresponding to each expert is weighted by this probability and added together to form the combined estimator.

The Expectation-Maximization (EM) algorithm (see [45] and [46] for proof of convergence) is used to maximize a log likelihood function, which in the case of XuME is

$$l(\theta,\nu) = log(\prod_j P(\vec{y}_j|\vec{x}_j)) = \sum_j log(P(\vec{y}_j|\vec{x}_j)).$$

The theoretical background is well described in the literature, so we will proceed to the update rules after introducing the expectation variables $h_j(\vec{y}|\vec{x}) = P(j|\vec{x},\vec{y})$:

$$h_j(\vec{y}_t|\vec{x}_t) = \frac{\alpha_j P(\vec{x}_t|\nu_j)P(\vec{y}_t|\vec{x}_t,\theta_j)}{\sum_i \alpha_i P(\vec{x}_t|\nu_i)P(\vec{y}_t|\vec{x}_t,\theta_i)}.$$

The update rules for the parameters α_j are

$$\alpha_j^{new} = \frac{1}{N}\sum_t h_j(\vec{y}_t|\vec{x}_t).$$

The update rules for the parameters $\nu_j = \{\vec{m}_j, \Sigma_j\}$ are

$$\vec{m}_j^{new} = \frac{\sum_t h_j(\vec{y}_t|\vec{x}_t)\vec{x}_t}{\sum_t h_j(\vec{y}_t|\vec{x}_t)}.$$

$$\Sigma_j^{new} = \frac{\sum_t h_j(\vec{y}_t|\vec{x}_t)(\vec{x}_t - \vec{m}_j^{new})(\vec{x}_t - \vec{m}_j^{new})^T}{\sum_t h_j(\vec{y}_t|\vec{x}_t)}.$$

It is more complicated to find $\theta_j^{new} = \{\Gamma_j^{new}, w_j^{new}\}$, but it can be solved by a set of linear equation systems: $\vec{d}_k = A\vec{b}_k$, where $\vec{b}_k$ is the unknown k'th column of w_j^{new}. The index k takes values between one and the size of the output vector. The vector $\vec{d}_k$ and the matrix A are constructed as follows

$$\vec{d}_k = \sum_t h_j(\vec{y}_t|\vec{x}_t)\vec{y}_t^{(k)}[\vec{x}_t, 1]$$

$$A = \sum_t h_j(\vec{y}_t|\vec{x}_t)[\vec{x}_t, 1][\vec{x}_t, 1]^T.$$

The update rules for Γ_j^{new} are

$$\Gamma_j^{new} = \frac{\sum_t h_j(\vec{y}_t|\vec{x}_t)(\vec{y}_t - (w^{new})^T[\vec{x}_t, 1])(\vec{y}_t - (w^{new})^T[\vec{x}_t, 1])^T}{\sum_t h_j(\vec{y}_t|\vec{x}_t)}.$$

This version of ME has a long history. We have traced the origin to [42] where a heuristic ME method was presented. Both the gating network and the predictors were neural networks trained with back propagation. The error function for the gating network was *ad hoc*. In the much cited article [43] the ideas were taken further, and the emphasis was on strong combination, while the predictors were simple. This version was tested in [57] on a vowel classification problem. In [47] the Hierarchical mixtures of experts is presented, where the gating network has a tree structure. The learning EM-algorithm is introduced in [45]. In [68] pruning and growing are added to the presented here version from [94]. It has been used in [16] on speaker identification.

7.2.2 DynCo

DynCo is a variant on the ME methods presented in [43].

Definition 14 (DynCo Combined Predictor)

A DynCo combined predictor F consists of a group of M regressors f_i, called experts, *and another group of M regressors z_i, called* coefficient predictors. *The combined predictor is given by the linear average predictor (LAP)*

$$F(\vec{x}) = \sum_{i}^{M} c_i(\vec{x}) f_i(\vec{x}) \tag{7.5}$$

The coefficients are given by the SOFTMAX function

$$c_i(\vec{x}) = \frac{e^{z_i(\vec{x})}}{\sum_{l}^{M} e^{z_l(\vec{x})}}. \tag{7.6}$$

■

The SOFTMAX function is used to automatically ensure that the coefficients obey $0 \leq c_i(\vec{x}) \leq 1$ and $\sum_i^M c_i(\vec{x}) = 1$.

In figure 7.3 is an illustration of a DynCo combined predictor with three experts. Note the similarities with figure 7.2. The gating network block in figure 7.2 corresponds to the part of figure 7.3 where the coefficient predictors z_i are combined by the SOFTMAX function to form the coefficients c_i.

The DynCo training algorithm can use any machine learning method that is iterative and use gradient descent. The error function for the experts is the MSE function on the combined predictor:

$$E_f(t, f_i) = \frac{1}{2}(t - F(f_i))^2, \tag{7.7}$$

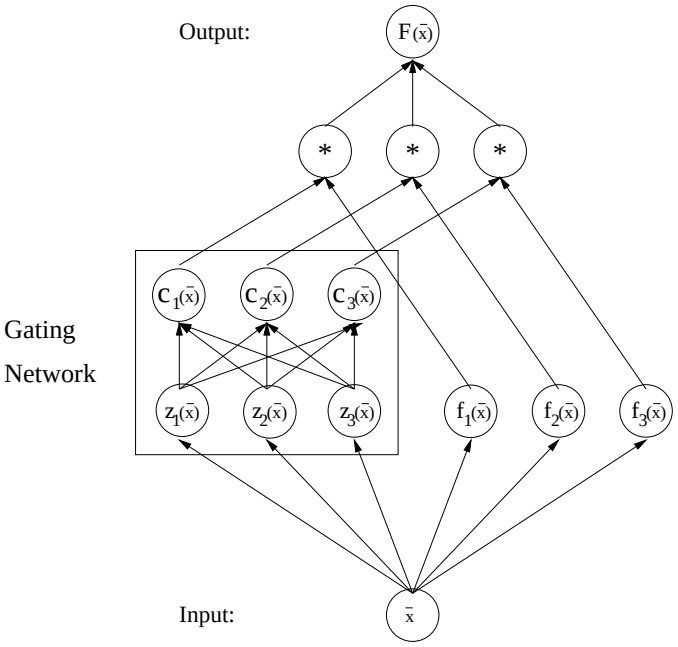


Figure 7.3: DynCo architecture

while the error function for the coefficient predictors is the MSE function on the combined predictor plus a penalty term. The penalty term in the error function for the coefficients predictors has been added to prevent the DynCo training method from emphasizing on only a few experts by setting the coefficients of all other experts to zero (see section 7.3.1).

$$E_z(t, z_i) = \frac{1}{2}(t - F(c_i))^2 + \gamma \frac{1}{2} \sum_{i=1}^{M} (c_i - \frac{1}{M})^2. \tag{7.8}$$

Algorithm 3 (DynCo Training Algorithm)

1. *Choose an iterative machine learning method L and ensemble size M.*
2. *Generate M ensemble members f_i and M coefficient predictors z_i.*
3. *$f_i^{new} = L(T, f_i^{old})$ for all i using the error function E_f.*
4. *$z_i^{new} = L(T, z_i^{old})$ for all i using the error function E_z.*
5. *If a stop criterion is not satisfied, go to 3.*

6. *Construct the combined predictor*

$$F(\vec{x}) = \sum_{i=1}^{M} c_i(\vec{x}) f_i(\vec{x}),$$

where c_i is defined in (7.6).

Back-propagation can be used to train a DynCo combined predictor, if the experts and coefficient predictors are neural networks. This is assumed in the following.

7.2.3 The family of gradient descent ME methods

DynCo is a variation on the ME method presented in [41, 43, 47]. There are a number of similarities: a group of experts are combined using weighting with non-constant coefficients (the gating network). In all methods the experts are trained using gradient descent (see appendix I). The differences are to be found in the error function and the architecture. In [41] the error function for each expert is the MSE function. This encourages competition between the experts (see section 9.1). The error function for the gating network is an *ad hoc* function that does not guarantee normalized coefficients. The methods in [43, 47] and DynCo use the SOFTMAX function to obtain auto-normalization of the coefficients. In Jacobs *et al.* [43] three different error functions are discussed. The two first are MSE functions for respectively the combined predictor and the experts. The first error function ($(y - \sum_j c_j f_j)^2$) is the error function used in DynCo. It encourages cooperation between the experts, since the experts can cooperate to form an accurate combined predictor without being accurate themselves. The second error function ($\sum_j c_j (y - f_j)^2$) is the one preferred by Jacobs *et al.* It encourages competition, since it is advantageous to use only the most accurate expert. Our experiments show that the first error function is the better (see section 9.1). Jacobs *et al.* also suggest a third error function $-log(\sum_j c_j e^{-\frac{1}{2}(y-f_j)^2})$, which is derived from a likelihood measure. The error function in [47] is also a log likelihood measure. The experts are still neural networks in [47], but the gating networks are simple affine predictors before they are combined using the SOFTMAX function. The architecture in [47] is hierarchical with gating networks forming a binary tree.

7.3 Ensemble and mixtures of expert methods

In section 7.1 some ensemble methods were presented and in section 7.2 some mixtures of experts methods were presented. They are often considered as

two separate groups of machine learning methods, but even though they have been presented as two groups, with different characteristics, they must be considered inseparable facets of the same group of methods. That is why they together are denote meta machine learning methods. Admitted it is difficult to see the ME characteristics in Simple (section 7.1.3), and it also difficult to find the ensemble members in Hierarchical mixtures of experts [45], but they are extremes. The overlap between the two groups can be seen in e.g. the boosting ensemble methods. They can to some extend be seen as decomposing the problem. The next predictor in a boosting session becomes an expert on a subset of the problem, often the examples for which the previous predictor did poorly on.

The combination rule and the gating network share common traits. The combination rule often is very simple, and the gating network is often complex and often contains the most of the expressive power of the combined predictor, but in principle they have the same possible functionality. There is nothing wrong in the gating network choosing to weigh all examples and all experts with uniform weight (but highly unlikely). The combination rule for an ensemble method can be very complex, e.g. the combination rule for AdaBoost uses information from training. Estimation of coefficients in parallel ensemble methods can be complex. The combination function can depend on the input [84]. We have chosen to define the difference between ensemble methods and ME methods as whether the gating network/combination rule is found during training, but this differentiation is to some degree arbitrary and not really well defined.

Some methods are in both groups, e.g. in [1] where boosting and mixture of experts are combined. The predictors are trained with boosting sessions, where the selection of examples for the next boosting session is based on a confidence measure defined on the already trained predictors. This makes it a boosting ensemble method. The confidence measure is also used to combine the predictors. This makes it a mixture of experts method.

Any subdividing of the group of meta machine learning methods can not be strict, and will depend on the view of the "subdivider". Another counter-example of a strict grouping is the DynCo method, which is discussed below.

7.3.1 DynCo and the γ parameter

The DynCo method can continuously be set from a "pure" ME method to a "pure" ensemble method via the γ parameter. If γ is set to zero, the coefficient predictors decompose the input space as an ME method. If γ goes toward infinity, the coefficient predictors are forced towards their common mean, which is uniform weighting as in many parallel ensemble methods.

A natural question is if there is an optimal γ parameter? If there was, we would know the optimal blend of ensemble method and ME method. In order to determine this, test runs on six different training sets have been done, where the value of γ was varied from 0.001 to 20. The example sets Building, Brain I, Brain II, Abalone, Thyroid, and Spiral are described in section 9.2. In figures 7.4–7.9 graphs of the error as function of γ for the six example sets can be found.

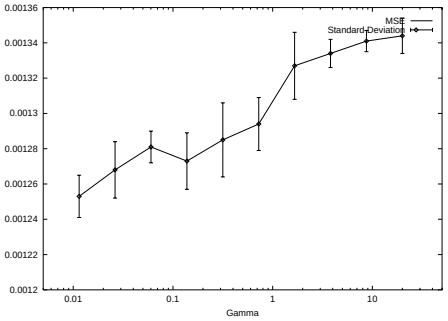


Figure 7.4: Building

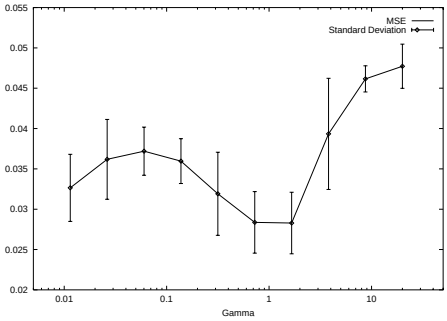


Figure 7.5: Brain I

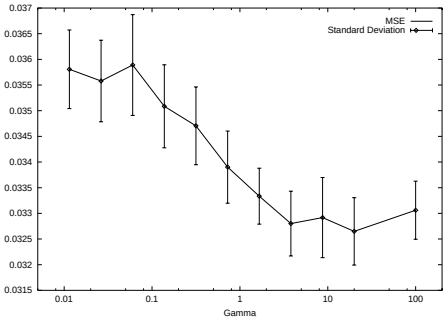


Figure 7.6: Brain II

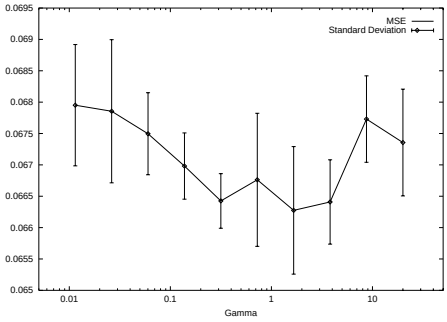


Figure 7.7: Abalone

As can be seen in the figures the optimal value of γ is problem dependent. The problems can be divided into three groups. The first group (Building, Thyroid, and Spiral) contains problems, which require a small value of γ ($\ll 1$), whereas the second group (Brain II) requires a large value of γ ($\gg 1$). The third group (Abalone, and Brain I) has intermediate optimal value of γ (≈ 1). The optimal value of γ can be interpreted in the following way: If the optimal value is small, the corresponding problem benefits strongly from decomposition, in contrast, if the optimal value is large there is no gain in applying decomposition.

We conclude that the optimal value of γ can give valuable information about the training set. This also indicates that for a given class of predictors, with

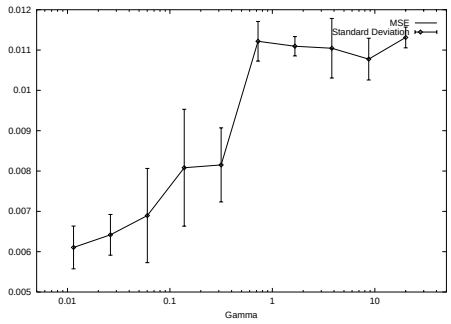


Figure 7.8: Thyroid

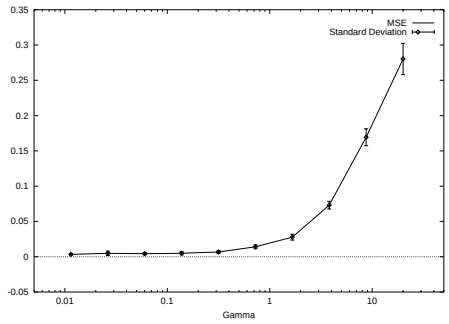


Figure 7.9: Spiral

the same "complexity", some problems benefit from decomposition, while others do not.

Part II

Experiments

Chapter 8

Empirical comparison of the deviance error functions

In chapter 5 we presented a general bias/variance decomposition for a large group of error functions. The error functions are derived from densities in the one-parameter exponential family of distributions through the deviance. No assumption was made about the distribution of the noise on the targets, but intuitively it would make sense to use an error function that is derived from the noise distribution in order to improve learning. On the other hand, it is not impossible that a particular error function, e.g. the popular MSE, generally is better for a wide range of noise distributions.

In this section we empirically test the MSE function, Poisson error function, and Gamma error function on six artificial functions with four different noise distributions, one of them being no noise. We will investigate if there is a connection between the type of error function and the type of noise, and see if a particular error function generally is better. The empirical result indicates that the noise distribution and the corresponding error function are correlated as expected. The tests show that the Poisson error function generally outperforms the two other error functions.

The test is not conclusive, but only indicative because of the empirical nature of the test. The number of error functions and target functions are limited. The learning method could have influence on the result, and it is not clear what noise on the target function should mean. We will address the noise in section 8.1 and the learning method in section 8.2. The result of the test will be presented and discussed in section 8.3.

8.1 Noise on target functions

Noise on the target function can come from a number of sources: the target function can be stochastic, or the measuring process can be inaccurate. The normal model of Gaussian noise is as an additive term to a deterministic target function

$$t(\vec{x}) = t^*(\vec{x}) + \epsilon(\vec{x}),$$

where t is the stochastic target from which the examples are sampled, t^* is the "real" target function and ϵ is the noise term with zero mean and variance $\sigma(\vec{x})$. This model is tailor made for Gaussian noise, since the Gaussian distribution is invariant under transformation of the mean, so if $\epsilon(\vec{x})$ is distributed as $N(0, \sigma(\vec{x}))$ then $t(\vec{x})$ is distributed as $N(t^*(\vec{x}), \sigma(\vec{x}))$. This is generally not possible, e.g. the Gamma and Poisson distribution are not invariant under transformation of the mean. Both distributions can not have negative mean. Furthermore, if $\epsilon(\vec{x})$ is Poisson distributed then $t^*(\vec{x}) + \epsilon(\vec{x})$ is not Poisson distributed.

In order to amend this problem we choose that the "real" target t^* is the mean parameter in the distribution and we assume that the variance is known:

$$p(t(\vec{x}) = z) = \pi(z|t^*(\vec{x}), \sigma),$$

where π is the density corresponding to the noise distribution.

8.2 Gradient descent for deviance error functions

In section 6.2 the back propagation algorithm is described for the MSE. Back propagation is a gradient descent method for neural networks. If the error function is changed the only effect on the update rule is the term connected to the error function, i.e. it will only affect the back-propagated error δ_j^N. In the following scalar output of the neural network is assumed, but it is not assumed that the post-processing function is the identity function. Let the output of the neural network be given by nn and the post-processing function be given by $f = pp(nn)$ yielding the back-propagation error:

$$\delta_1^N = \frac{\partial E(f,t)}{\partial f} \frac{\partial pp(nn)}{\partial nn}.$$

This parameter negated is an expression for how much the output of the neural network should be changed. We denote that change Δnn and call it the *intended shift* of the neural network:

$$\Delta nn = -\delta_1^N \tag{8.1}$$

A post-processing function can be a simplifying tool in connection with the deviance error functions, e.g. the Poisson or Gamma error functions that only accept positive predictors. The output of a neural network can take any value on the real line, but by setting the post-processing function to be the exponential function, the post-processed output is forced to be positive.

Let us use the Poisson error function as an example, and let the post-processing be the exponential function $f = e^{nn}$. The post-processed output is the actual output of the predictor. The error function is given by

$$E(t, f) = [t - f] + t \log \frac{f}{t},$$

and is only suppose to take positive parameters t and f. The intended shift of the neural network Δnn is given by (8.1)

$$\Delta nn = -\frac{\partial [t - f] + t \log \frac{f}{t}}{\partial f} \cdot \frac{\partial e^{nn}}{\partial nn} = [1 - \frac{t}{f}] e^{nn}$$

Remember that $f = e^{nn}$ so the intended shift in output of the **neural network** is given by $[\frac{t}{f} - 1]f = t - f$, which is the signed difference between the target and the output of the **post-processed** output. This difference can take on any value on the real line, but so can the output of the neural network. Note that the intended shift Δnn is reduced to the gradient for the MSE error function. The post-processing functions and intended shifts for the error functions in section 5.6 are given in appendix F. In the next section three error functions are compared using the post-processing functions given in appendix F.

8.3 The empirical test of deviance error functions

In order to test the hypothesis of a connection between the types of noise on the target function and the types of error functions, three error functions are trained on six target functions with four types of noise. The error functions are the Normal error function (MSE), the Gamma error function, and the Poisson error function (see section 5.4). The six target functions are SinC, Gabor, Multi, Friedman1, Friedman2, and Friedman3 (see section 10.1 for a description). Four types of noise are tested on the six target functions, namely Normal distributed, Gamma distribution, Poisson distributed noise, and no noise. The Normal and Gamma distributed noise are both in two versions with different deviation. One where the deviation is 5 % of the output domain, and one where the deviation is 25 % of the output domain. The Poisson distribution does not have a separate variance parameter, so there is only one version. The data is distorted by noise as described in section 8.1.

The errors of the different error functions are not directly comparable. Furthermore, the post-processing output domain of the error functions are different, making direct comparison meaningless. Therefore the output of the post-processing function (from now on called the output and denoted by f) and the output of the target function are rescaled to the interval $[0;1]$, and the absolute error ($|f-t|$) is found on the noise-free target function. In this way the absolute error is the same as the average difference between the target and the output in per cent. During training, the target functions are rescaled to naturally fit the output domain of the error function. For the MSE the target functions are rescaled to the domain $[-1;1]$. For both the Gamma and Poisson error function the target functions are rescaled to the domain $[1;100]$. In table 8.1 is an overview of the output domain (**OUT. DOM.**), rescaling domain (**RES. DOM.**), post-processing function (**P.P. FUNC.**), and intended shifts (Δnn) for the three error functions.

ERROR	**OUT. DOM.**	**RES. DOM.**	**P.P. FUNC.**	Δnn
MSE	$]-\infty;\infty[$	$[-1;1]$	$f = nn$	$t-f$
Gamma	$[0;\infty[$	$[1;100]$	$f = e^{nn}$	$\frac{t}{f}-1$
Poisson	$[0;\infty[$	$[1;100]$	$f = e^{nn}$	$t-f$

Table 8.1: Error function specifications

For each combination of noise and error function 30 neural networks were trained. For each training run a new training set was generated. The size of the training sets was as default 2500, but only 1000 for the SinC target function. The neural networks had one hidden layer. The size of the input and output layer were determined by the number of inputs and outputs of the targets function. The hidden layer had as default 20 nodes. The only exception was for the SinC target function, which is so simple, that only 10 hidden nodes were used. Training was done using back propagation as described in section 8.2. The learning rate was set to 0.01 for the Gamma and Poisson error function and 0.5 for the MSE. Momentum (see section 6.2) was used with a momentum rate of 0.9. All neural networks were trained for 5000 epochs. The weights were updated after a batch of 100 training examples were seen.

In tables 8.2 and 8.3 is an overview of the test errors from the training runs. As previously mentioned the test errors were found on the noise-free target function, even for training runs where the training set had noise. Technically the noise-free test error was found by sampling the noise-free target function a number of times equal to the size of the training set and

calculating the error on that set. It can be a little difficult to see what the test error on the noise-free target function would mean on natural data. We know the noise-free target functions, because artificial target functions were used, but for natural data the noise-free target function is not even well defined. If the noise originates from measuring process, then the noise-free target function corresponds to the unknown "truth" we wish to find, and is therefore well defined. But if the target function is inherently stochastic, e.g. in example data that involves atomic decay, there is no deterministic "truth". We have defined noise in section 8.1 as a distribution with a well defined mean parameter. Furthermore, the deviance error functions in this test have minimum in what corresponds to the mean parameter (see section 5.4), so the noise-free target function corresponds to the mean parameter as a function of input, and is therefore well defined.

Each number in tables 8.2 and 8.3 represents 30 training runs. The average of the test error is the first number, while the digits in the parentheses is the error on the last digits of the test error, i.e 0.0114(48) means 0.0114 ± 0.0048.

TARGET	NOISE	MSE	POISSON	GAMMA
SinC	None	0.0200(47)	0.0048(13)	0.0114(48)
SinC	Normal (5)	0.0095(36)	0.0087(24)	0.0136(37)
SinC	Normal (25)	0.0125(27)	0.0146(35)	0.0308(33)
SinC	Gamma (5)	0.0342(49)	0.0187(30)	0.0240(98)
SinC	Gamma (25)	0.1486(85)	0.0357(31)	0.0137(43)
SinC	Poisson	0.0230(25)	0.0096(17)	0.0241(83)
Gabor	None	0.01405(52)	0.00371(49)	0.0085(19)
Gabor	Normal (5)	0.0095(10)	0.0108(32)	0.0180(51)
Gabor	Normal (25)	0.0196(25)	0.0174(31)	0.037(10)
Gabor	Gamma (5)	0.0169(26)	0.0089(22)	0.0120(30)
Gabor	Gamma (25)	0.0858(95)	0.0163(16)	0.0153(22)
Gabor	Poisson	0.0236(22)	0.00361(80)	0.0057(15)
Multi	None	0.00372(74)	0.00774(88)	0.0121(12)
Multi	Normal (5)	0.0221(27)	0.0217(24)	0.0232(24)
Multi	Normal (25)	0.0433(26)	0.0440(28)	0.0573(79)
Multi	Gamma (5)	0.0220(26)	0.0147(18)	0.0158(24)
Multi	Gamma (25)	0.0814(88)	0.0242(17)	0.0249(15)
Multi	Poisson	0.0263(36)	0.0201(16)	0.0205(29)

Table 8.2: Test results for noise type versus error function type I

The test error is between 0.20 % and 15 % and the deviation is between 3.3 % and 42 % of the test error. The average test error is close to 2.4 % and the typical deviation is around 10 % of the test error.

TARGET	NOISE	MSE	POISSON	GAMMA
Friedman1	None	0.0063(13)	0.0145(15)	0.0247(36)
Friedman1	Normal (5)	0.0311(17)	0.0311(26)	0.0354(33)
Friedman1	Normal (25)	0.0566(24)	0.0617(34)	0.0656(46)
Friedman1	Gamma (5)	0.0300(29)	0.0207(12)	0.0273(37)
Friedman1	Gamma (25)	0.0811(49)	0.0318(11)	0.0343(11)
Friedman1	Poisson	0.0356(17)	0.0329(31)	0.0366(22)
Friedman2	None	0.0133(36)	0.00203(31)	0.00467(74)
Friedman2	Normal (5)	0.01036(45)	0.00255(37)	0.00091(11)
Friedman2	Normal (25)	0.00738(93)	0.00592(46)	0.0123(12)
Friedman2	Gamma (5)	0.0121(31)	0.00456(60)	0.0136(20)
Friedman2	Gamma (25)	0.0206(43)	0.0160(14)	0.0341(51)
Friedman2	Poisson	0.01519(80)	0.0056(11)	0.0103(23)
Friedman3	None	0.0081(19)	0.0117(14)	0.0234(23)
Friedman3	Normal (5)	0.0290(83)	0.0255(27)	0.0235(24)
Friedman3	Normal (25)	0.0397(44)	0.0440(50)	0.0565(46)
Friedman3	Gamma (5)	0.0168(21)	0.0152(24)	0.0159(17)
Friedman3	Gamma (25)	0.0433(49)	0.0210(18)	0.0214(32)
Friedman3	Poisson	0.031(13)	0.0268(19)	0.0248(32)

Table 8.3: Test results for noise type versus error function type II

Due to the number of test errors in tables 8.2 and 8.3, it is not easy to draw conclusions. To get a clearer picture we introduce a quality measure of the error functions: The percentage that a given error function would produce the lowest test error for a given noise and target function averaged over the target functions (see appendix K). In practice this percentage is found by using the results in tables 8.2 and 8.3, and assuming that the test errors for a given error function, target function, and noise type are Normal distributed with the mean and standard deviation given in the table.

The quality measure for error functions versus noise type is given in table 8.4. In the table the highest probability (quality measure) for each error function is written in boldface, excluding the probabilities for no noise.

We see that the MSE is best on training sets with no noise and with Gaussian noise, while the Poisson error function is best on Poisson and Gamma noise, and Gamma error function is best on Gamma and Poisson noise. This strongly support the hypothesis that an error function derived from a distribution is best suited for training set with noise from that distribution. The Poisson error function is almost as good on Gamma noise as on Poisson noise, while the Gamma error function is almost as good on Poisson noise as on Gamma noise. This is not very surprising, since Gamma and Poisson distributed noise share some characteristics, e.g. lower-bound at zero, and

NOISE	MSE	POISSON	GAMMA
None	0.498	0.502	0.000
Normal (5) and Normal (25)	**0.489**	0.430	0.081
Poisson	0.033	**0.807**	0.160
Gamma (5) and Gamma (25)	0.006	0.761	**0.233**

Table 8.4: Noise type versus error function type

an infinite positive tail. The most surprising is that the Poisson error function is the best performing error function for three of of the four types of noise (including no noise), and the probability that the Poisson error function yields the lowest error with Normal distributed noise is almost as large as the the probability for the MSE function. This questions the role of MSE as the commonly used error function, since not even for Normal distributed noise is the MSE unchallenged in performance. It should be noted that the test is empirical and by nature limited. The conclusions can therefore only be indicative. Furthermore, the Poisson error function is lower-bounded, while the MSE is defined on the entire real line. Still, if the training examples have a natural lower-bound, there should be no reason to use the MSE function over the Poisson error function.

8.4 Further analysis of the tests

The results in tables 8.2 and 8.3 can be used for further analysis. In table 8.5 the average test error over noise types and error function types for the target functions are given. This can be used as an empirical measurement of the difficulty of the target functions. The number in the parentheses is the average standard deviation expressed as the inaccuracy on the last digits of the average test error. Even though there are some differences between the difficulty of the targets function, the spread between the most difficult target function (Friedman1) and the least difficult target function (Friedman2) is less than a factor 3.3. Since this difficulty measure is very crude, we cannot with confidence say that there is substantial differences between the target functions. This is an advantage, since it indicates a uniform background for the test and conclusions in section 8.3.

Note that the difficulty measure for SinC is not reliable, since the predictors used for the SinC target function are less complicated (half the number of

TARGET FUNCTION	AVERAGE TEST ERROR
SinC	0.0254(44)
Gabor	0.0181(25)
Multi	0.0269(27)
Friedman1	0.0365(24)
Friedman2	0.0111(13)
Friedman3	0.0265(32)

Table 8.5: The average test error for the target functions

hidden nodes) and the number of training examples is less (1000 instead of 2500).

In table 8.6 the average means for the different noise types are given. There is no problem in connection with the SinC target function, since the results for SinC are equally distributed between the results for the different types of noise.

NOISE TYPE	AVERAGE TEST ERROR
None	0.0108(15)
Normal (5)	0.0186(22)
Normal (25)	0.0348(33)
Gamma (5)	0.0180(25)
Gamma (25)	0.0416(34)
Poisson	0.0208(23)

Table 8.6: The average test error for the noise types

The spread between the smallest and the biggest average test error is about a factor four, which is larger than the corresponding difference in table 8.5, making the difference more significant.

As expected the average of the test errors for training without any noise is smallest. The average of the test errors with noise are all larger and the average of the test errors increases with increasing deviation. The average of the test errors for Normal (5) noise and Gamma (5) noise are equal, and the average of the test errors for Normal (25) noise and Gamma (25) noise are close to each other. This indicates that the deciding factor is not the type of noise, but the deviation. The average of the test errors for Poisson noise is between the average of the test errors for small deviation (5 %) and large deviation (25 %).

Chapter 9

DynCo compared with four other methods

The results in this section are the main results from the article "Combining Predictors: Comparison of Five Meta Machine Learning methods", that has been published in "Information Science, an International Journal" [33]. Some of these results have been published in the proceedings of the third International Conference on Computational Intelligence and Neuroscience (ICCIN'98), which was part of the fourth International Conference on Information Sciences (JCIS'98) [32], and in the progress report for the author's part A exam [31]. The results were presented orally at ICCIN'98.

In section 9.1 the optimal error function for DynCo (see section 7.2.2) is investigated. In section 9.2 three ensemble methods and two mixtures of experts (ME) methods are compared.

9.1 Cooperation or competition?

Remember that the combined predictor for DynCo is given by the LAP $F = \sum_j c_j f_j$. As mentioned in section 7.2.3 Jacobs *et al.* [43] prefer the error function

$$E(F,t) = \sum_j c_j (y - f_j)^2$$

to

$$E(F,t) = (y - \sum_j c_j f_j)^2.$$

Note that in the ambiguity decomposition of the MSE function (3.5) the first term is $\sum_j c_j (y - f_j)^2$ (apart from a constant). By introducing an ambiguity

factor (AF) in the second term of (9.1), we can test which of the two error functions is the better.

$$E(F,t) = \frac{1}{2}\sum_{i=1}^{M} c_i(y - f_i)^2 - AF\frac{1}{2}\sum_{i=1}^{M} c_i(f_i - F)^2 \qquad (9.1)$$

The value of AF was varied in test runs between 0 and 1.8 in steps of size 0.2 for 12 problems.[1] For each problem and value of AF ten test runs were performed, and the mean and standard deviation were found. Even though different values of AF were used during training, AF was set to one for the test set error in order to make the comparison equal and fair.

In figures 9.1–9.12 the graphs of error as function of AF are given for the 12 problems.

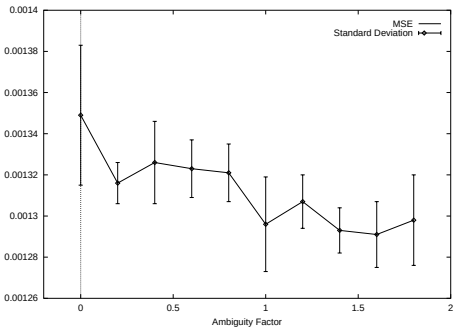


Figure 9.1: Building

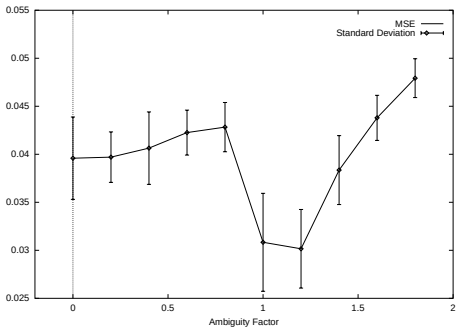


Figure 9.2: Brain I

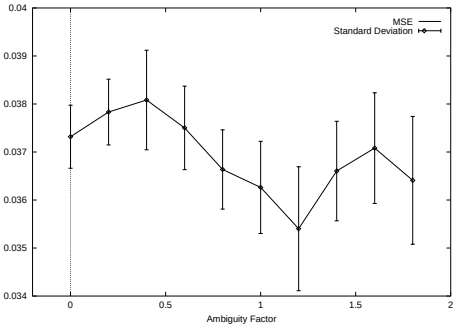


Figure 9.3: Brain II

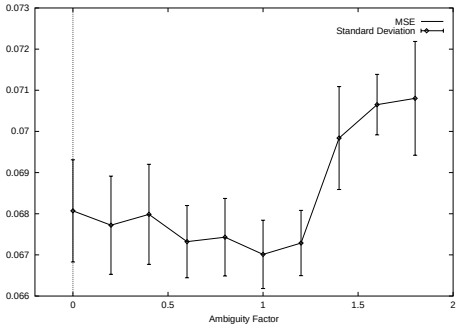


Figure 9.4: Abalone

None of the 12 problems indicates that zero should be the best value of AF. To give an over-all picture the following is calculated: the probability for each value of AF that a test run with that value would yield the lowest

[1] The problems are described in section 9.2.

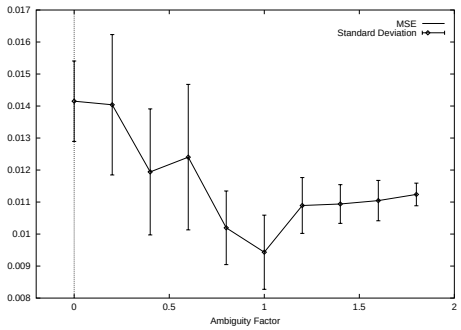


Figure 9.5: Thyroid

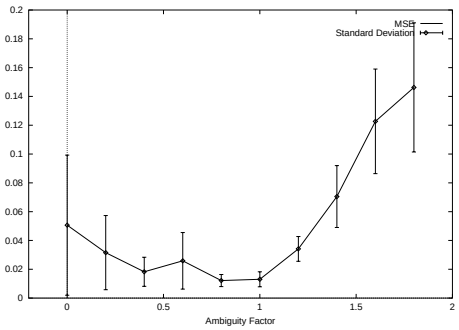


Figure 9.6: Spiral

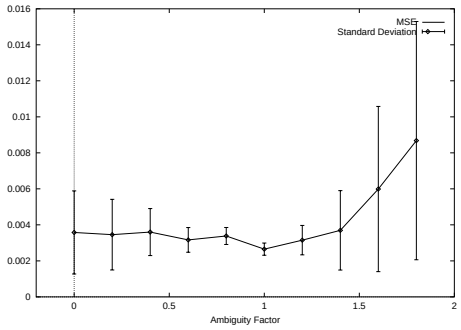


Figure 9.7: SinC

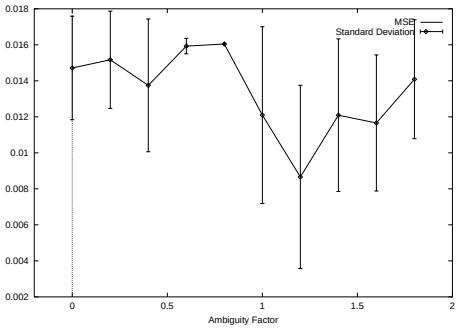


Figure 9.8: Gabor

error, where the problems are weighted uniformly (see appendix K). The resulting probabilities are plotted in figure 9.13.

As can be seen the percentages are peaked around AF equal to one. There is 25.3 % probability that training with $AF = 1$ yields the lowest error, while it for $AF = 0$ is only 7.1 % or about 3.6 times less. We have therefore chosen always to use $AF = 1$, which corresponds to the error function:

$$E(F,t) = \frac{1}{2}(y - \sum_j c_j f_j)^2 = \frac{1}{2}(y - F)^2.$$

9.2 Empirical tests of meta machine learning methods

We have performed two groups of tests: One where we have implemented five of the MML methods from chapter 7 and compared their test set errors. This group of tests is called relative-tests. The other group of tests is called absolute-tests, because we compare DynCo with results published

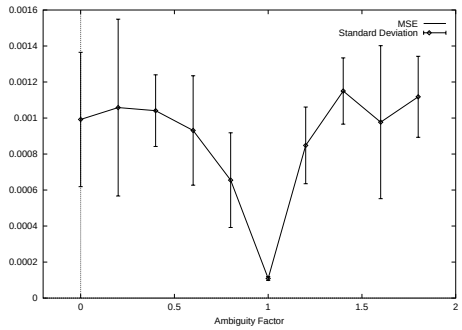


Figure 9.9: Multi

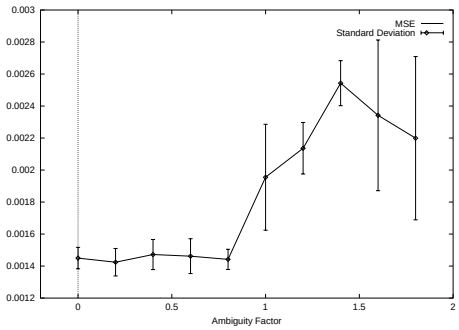


Figure 9.10: Friedman 1

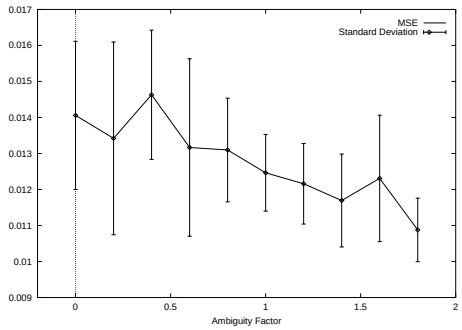


Figure 9.11: Friedman 2

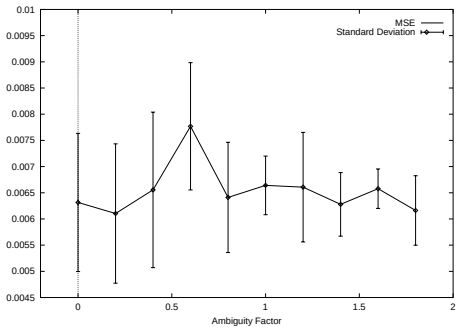


Figure 9.12: Friedman 3

for AdaBoost, Bagging (both from [20]), and for an extended version of XuME, that employs pruning/growing [68], which we call XuME++. The implementation of DynCo, Bagging, AdaBoost, and Simple employs neural networks as ensemble members/experts. The predictors used in [20] for AdaBoost and Bagging are regression trees. The experts for XuME and XuME++ are given by the definitions of the methods.

For both groups of tests the quality measure is the MSE on a test set. This only makes sense if the combined predictors from the implemented methods are regressors. AdaBoost, Bagging, Simple, and DynCo already have a regression version, while XuME is a density estimation method. To transform XuME into a regressor, one could take the mean over the output density, but to use the advantage of density estimation, we have chosen to use the output with the highest density:

$$\underset{\vec{y}}{argmax}(P(\vec{y}|\vec{x})) \approx \underset{j\in(1,\ldots,k):\hat{\vec{y}}_j}{argmax} (g_j(\vec{x},\nu_j)P(\hat{\vec{y}}_j|\vec{x},\theta_j)).$$

The error of the approximation is small if the peaks of high density are not too close and there is not too much overlap between the experts.

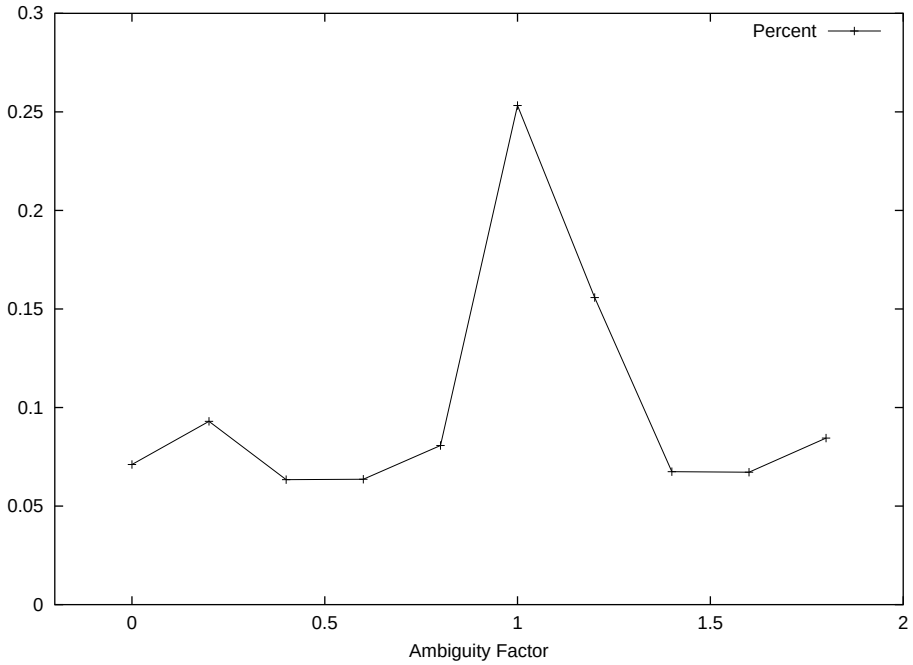


Figure 9.13: Ambiguity Factor

A test consisted of ten training runs on an example set. Exactly the same partitioning of the example set into training set (75 %), and a test set (25 %) was used every time and for every MML method. If a method uses early stopping (see definition 10) a third of the training set was used as validation set. The mean MSE and the standard deviation have been calculated for the ten test runs.

To make the comparison as fair as possible, we have tried to use the best possible parameters for each test. An extensive work has been done to find e.g. the best number of ensemble members/experts (see figures 9.14–9.19) or the best values of hand-set parameters such as learning rate and momentum. DynCo has the unique parameter γ, for which the empirically found optimal values were used (see section 7.3.1). Some parameters were not the same for the different methods e.g the number of hidden units in the experts/ensemble members. AdaBoost, Simple, and Bagging require global predictors in contrast to DynCo, so we chose as a default that the ensemble members in AdaBoost, Simple, and Bagging had one hidden layer with 25 nodes, while the DynCo predictors had one hidden layer with only four to eight nodes. There is one exception: the architecture for AdaBoost, Bagging and Simple ensemble members had two hidden layers with each ten nodes for the Spiral problem, since it is difficult for neural network to learn. The coefficient predictors for the DynCo method had one hidden layer with eight

nodes, except for the Brain I problem where 30 nodes was used.

The experts in DynCo were trained for 5000 epochs with early stopping, i.e. if the validation error had not decreased in 500 epochs, training was terminated, and the error on the test set when the validation error was lowest was reported. The ensemble members in AdaBoost, Simple, and Bagging were trained for 5000 epochs without early stopping, since it can be advantageous for the predictors to overfit (see e.g [82]). XuME was trained for 200 epochs, which is enough due to the fast convergence of the EM algorithm. XuME also uses early stopping. Training was stopped if the validation error had not decreased after eight epochs.

9.2.1 Relative tests

The relative-tests were done on six example set. Table 9.1 gives an overview of the sets.

NAME	SIZE	TARGET FUNCTION
Building	4208	PROBEN1 [64]
Brain I	784	PET Center [56]
Brain II	1000	PET Center [56]
Abalone	4177	UCI [54]
Thyroid	7200	PROBEN1 [64] & UCI [54]
Spiral	1800	Generated. See figure 10.1

Table 9.1: Example Sets for Relative-tests

In figures 9.14–9.19 the graphs of the error as function of the ensemble size/number of experts are given for the Bagging, Bagging, DynCo and Simple. The graphs for the XuME+ method have not been included, since the nature of the XuME+ algorithm makes them incomparable, e.g. the experts in XuME+ are very simple in contrast to neural networks in the other methods, so a larger number of experts is usually needed. The graphs in figures 9.14–9.19 has been used to find the optimal number of experts/ensemble members, but the structure of the graphs also deserves some remarks.

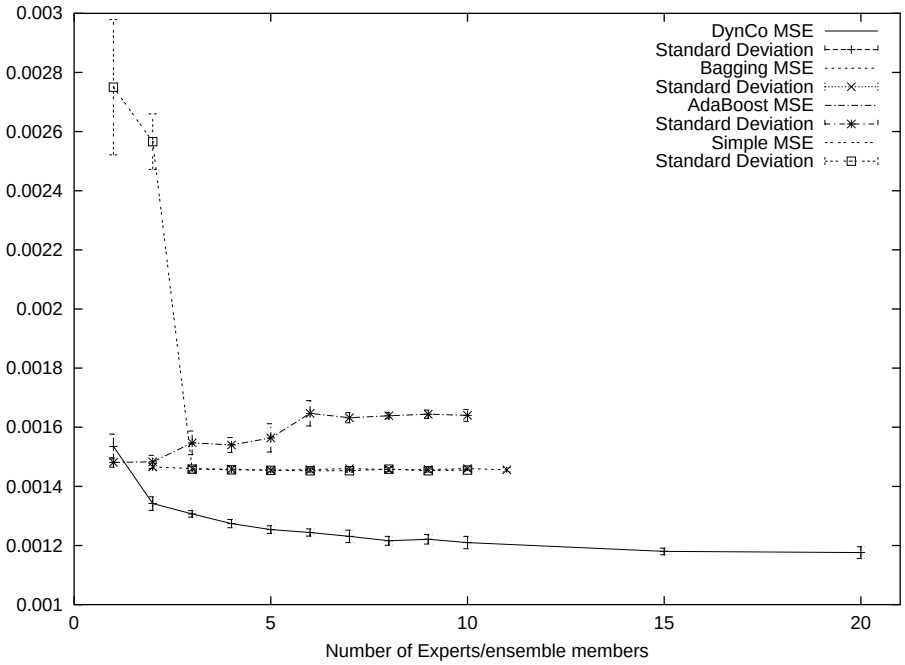


Figure 9.14: Building

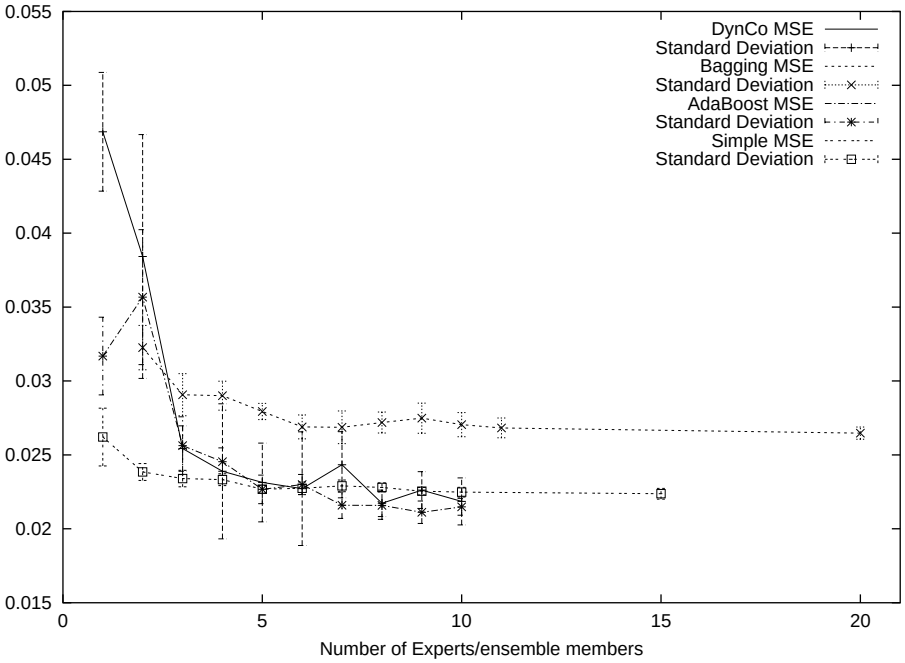


Figure 9.15: Brain I

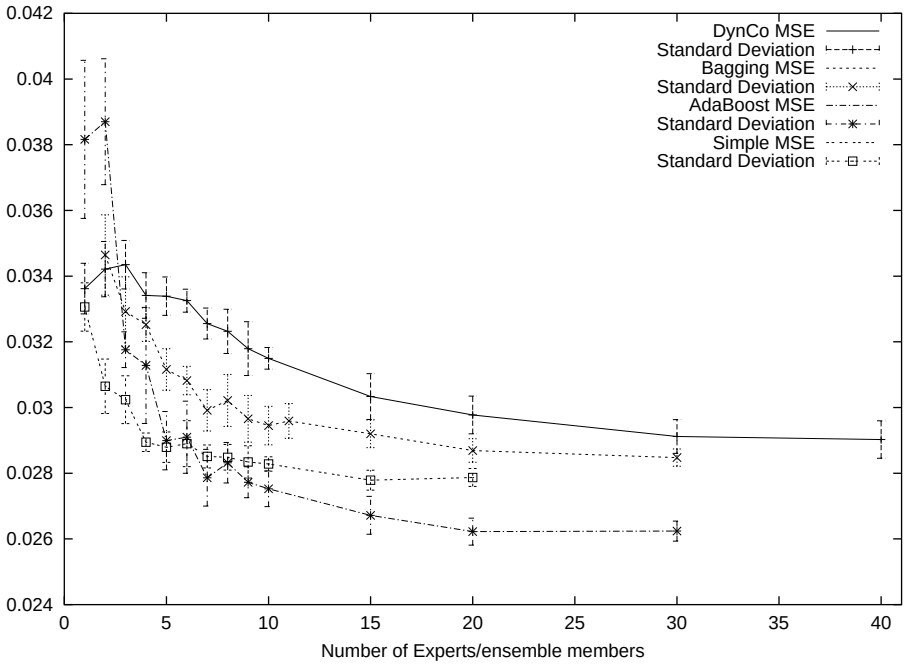


Figure 9.16: Brain II

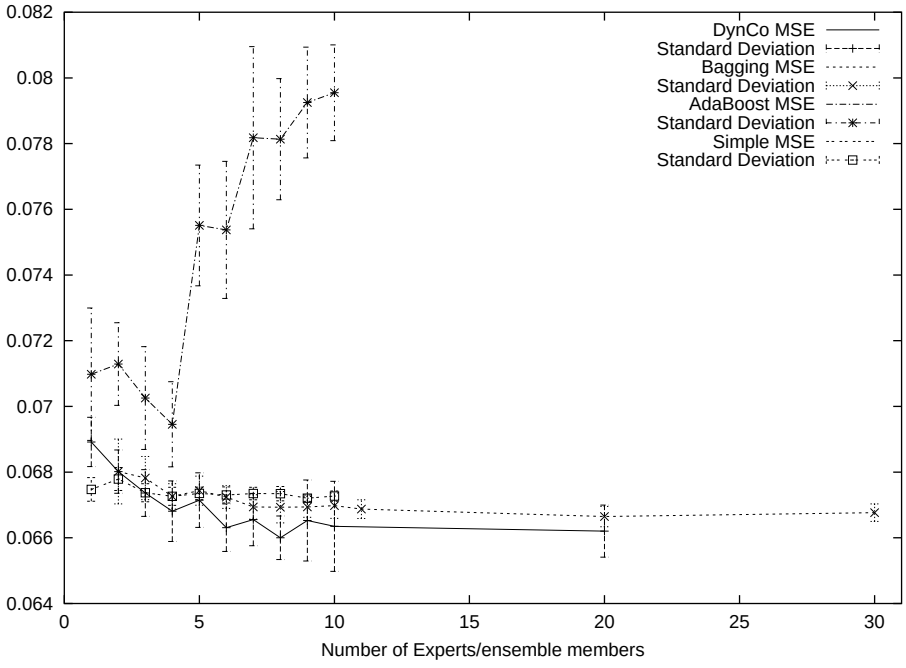


Figure 9.17: Abalone

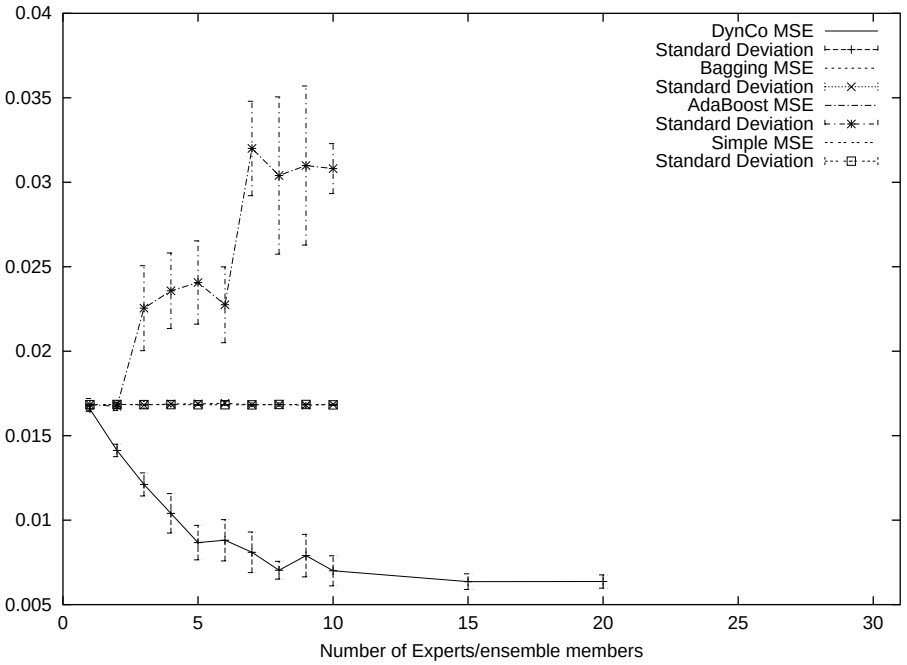


Figure 9.18: Thyroid

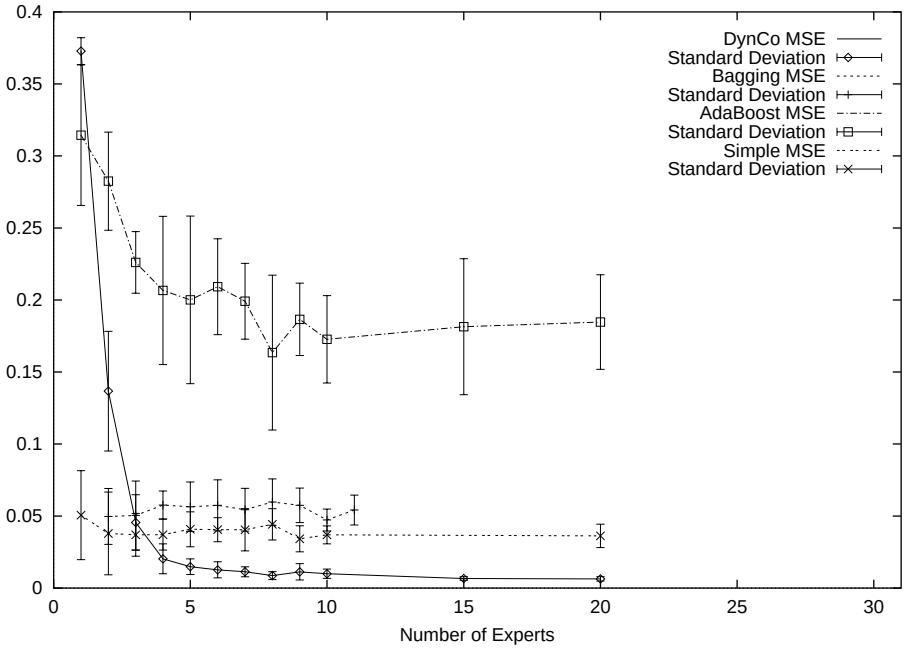


Figure 9.19: Spiral

Some of the figures seem to have the same structure, e.g. figures 9.15 and 9.16, which corresponds to the example sets Brain I and Brain II. All the graphs in figures 9.15 and 9.16 follow the "classical" pattern of decreasing error for increasing number of experts/ensemble members and the decrease in error becomes smaller and smaller. For both figures, AdaBoost achieves the lowest error among the methods depicted.

It is not surprising that figures 9.15 and 9.16 resemble each other, since both examples sets originate from the same problem. More surprising is the similarities in figures 9.14 and 9.18, since the example sets Building and Thyroid are unrelated. Also, the structure of the figures are interesting and "non-classic". It seems as if the ensemble methods have a lower bound for the error, which is equal to the error of a single ensemble member. We call that for the "lower error bound for ensemble methods". The ME method DynCo does not suffer from the lower bound, and it significantly outperforms the ensemble methods. Especially remarkable is the structure of the graph for AdaBoost in figure 9.18. The error of AdaBoost is on the lower error bound for one and two ensemble members, but then the error increases until it is the double of the lower error bound. This kind of behavior has been reported in literature [66, 59] and indicates noise and/or outliers.

The structure of the two pairs of figures are correlated with the optimal value of γ (see section 7.3.1). For the figures 9.15 and 9.16 the optimal values of γ are one or greater, while the optimal values for figures 9.14 and 9.18 are close to zero. This is in agreement with the interpretation of the optimal value of γ. A low optimal value indicates that the example set benefits from decomposition of the problem, so ME methods should perform well. A high optimal value on the other hand indicates, that global combination is better, so ensemble methods should perform well.

Note that even though DynCo is not the best performing methods on Brain I and Brian II, it is not much worse than the ensemble methods. This is due to the fact that DynCo resembles an ensemble method for high values of γ.

The two figures 9.17 and 9.19 have structures in between the discussed figures. Generally, the behavior of the individual graphs are classical with the exception of AdaBoost in figure 9.17. But the decrease in error for the ensemble methods compared with a single predictor is small, especially compared with the decrease for the DynCo method. This indicates a lower error bound for the ensemble methods. The behavior of AdaBoost in figure 9.17 indicates noise or outliers.

The graphs in figures 9.14–9.19 together with the graphs for XuME+ have been used to decide the optimal number of experts/ensemble members. Since the error in most cases do not increase, the optimal number of experts/ensemble members is the number for which the error in a statistical sense

has stopped to decrease. The other parameters have been found in a similar way, e.g. the optimal values of the γ parameters have been found from the graphs in figures 7.4–7.9. After the optimal value of the parameters have been estimated, the different methods have been run on the examples sets again. In table 9.2 there is an overview of the MSE for the test runs in per cent. The errors on the two last digits are given in the parentheses.

SET	DynCo	Bagging	AdaBoost	XuME	Simple
Building	**0.1180(11)**	0.1457(13)	0.1468(16)	1.205(21)	0.1458(9)
Brain I	2.28(22)	2.646(41)	**2.133(71)**	3.4(1.9)	2.238(18)
Brain II	2.911(40)	2.847(27)	2.685(43)	**1.238(37)**	2.801(27)
Abalone	**6.599(81)**	6.689(22)	6.94(10)	7.33(37)	6.729(27)
Thyroid	**0.502(44)**	1.6881(59)	1.674(25)	2.21(0)	1.6844(22)
Spiral	**0.31(13)**	5.0(10)	17.0(35)	1.12(50)	4.64(92)

Table 9.2: Test Results for the Relative-tests (Per cent)

The DynCo method achieves the lowest error for four of the six example sets. If the problems are weighted uniformly the probability that DynCo gives the best result is 66.3 %. The second best is XuME with 21.9 % or three times less than DynCo. See appendix K for the calculation of the probabilities.

Note that DynCo performance is correlated with the optimal value of γ (see section 7.3.1). DynCo performs best in the group of problems corresponding to a low optimal value of γ (Building, Thyroid, and Spiral), while DynCo performs worst for the group corresponding to a high optimal value of γ (Brain II). The last group corresponds to an intermediate optimal value of γ (Brain I and Abalone). Here DynCo gives the lowest test set error for Abalone, though not by much, and DynCo is not the best for Brain I.

9.2.2 Absolute tests

Tables 9.3 and 9.4 give an overview of the example sets used in absolute-tests.

The **SIZE** fields have the format: "train set size"+"test set size". The example sets in table 9.3 are from [68], while the example sets in table

NAME	SIZE	ORIGIN
Building2	2104+1052	PROBEN1 [64]
SinC	100[a]	$sin(x)/x + N(0, \frac{1}{4})$
Gabor	64+192	$\frac{2}{\pi}\exp[-0.5(x^2+y^2)\cos(2\pi(x+y))]$
Multi	820+204	$0.79 + 1.27x_1x_2 + 1.56x_1x_4 + 3.42x_2x_5 + 2.06x_3x_4x_5$

[a]The test error is found by testing against the true SinC function without noise.

Table 9.3: Example Sets for Absolute-tests I

NAME	SIZE	ORIGIN
Friedman1[a]	240+5000	$10\sin(\pi x_1x_2) + 20(x_3 - \frac{1}{2})^2 + 10x_4 + 5x_5 + N(0,1)$
Friedman2	240+5000	$\sqrt{x_1^2 + (x_2x_3 - (\frac{1}{x_2x_4}))^2} + N(0,a)^b$
Friedman3	240+5000	$\tan^{-1}\frac{x_2x_3 - 1/(x_2x_4)}{x_1} + N(0,a)^b$
Housing	481+25	UCI [54]

[a]There are ten inputs: $\{x_1, \ldots, x_{10}\}$, only the first five have influence on the output.
[b]The Gaussian noise $N(0,a)$ is a third of signal power of the function, i.e. a^2 is a third of the variation of the function, calculated with uniform density over the input domain.

Table 9.4: Example Sets for Absolute-tests II

9.4 are from [20]. Note that six of the sets are generated by sampling the input domain uniformly and computing the function in **ORIGIN**. The input range for SinC is $[-10; 6]$. For both Multi and Gabor the domains for all inputs are $[-1:1]$. Friedman1 has input domain $[0:1]$ for all inputs. For Friedman2 and Friedman3 the input domains are $x_1 \in [0:100], x_2 \in [10\pi : 140\pi], x_3 \in [0:11]$, and $x_4 \in]1:11]$. The outputs have been scaled so the output domain is $[-1:1]$ for all functions. This changes the test set errors in tables 9.5 and 9.6, compared with the results in the literature. The test set errors have been scaled accordingly. The numbers in the parentheses are the deviation on the last digits.

The errors in both [68] and [20] are reported without deviation, so statistical comparison is impossible. We adopt the terminology that two errors are

SET	DYNCO	XUME+
Buil2	**0.1222(10)**	0.42
SinC	**0.450(90)**	**0.435**
Gabor	**0.450(79)**	1.48
Multi	**0.0038(4)**	0.00529

Table 9.5: Test results for Absolute-tests I (Per cent)

SET	DYNCO	BAGGING	ADABOOST
Friedman1	**0.1476(61)**	0.272	0.217
Friedman2	**0.675(61)**	0.7385	**0.7155**
Friedman3	**0.492(27)**	0.993	0.642
Housing2	1.31(13)	1.225	**1.057**

Table 9.6: Test results for Absolute-tests II (Per cent)

comparable, if the published result is in the range defined by the deviation of the error of DynCo, otherwise the error is higher or lower. We see in tables 9.5 and 9.6 that the error of DynCo is lower than the published results for five of the eight example sets, it is comparable with the published results for two example sets. Only for the Housing2 example set is the error of DynCo higher than one of the published errors, namely the error of AdaBoost.

We conclude this section with a resume.

- It was shown empirically that the cooperative error function of DynCo is superior to the competitive error function preferred in [43].
- DynCo was shown empirical to give lowest error 66.3 % of the time of the five implemented MML methods, or 3.3 times more often than if the five methods were equally good.[2]
- DynCo has been compared with published results for XuME with

[2]Five equally good methods would each have a probability of 20 % of yielding the lowest error.

growing/pruning, Bagging, and AdaBoost. It was shown that DynCo gives the lowest error about 75 % of the time. This is about three times more often than if DynCo were compared with equally good methods.

- It is believed that the strength of the DynCo method is that it can use strong predictors (e.g. neural networks) and uses strong combination (dynamic coefficients). In contrast the ensemble methods can use strong predictors, but have simple combination (constant coefficients), and XuME uses strong combination, but have linear predictors.
- The ensemble methods show no improvement over single predictors on certain example sets. They cannot pass the "lower error bound for ensemble methods". This must be due to the simple combination rule, because DynCo achieves significantly better results, and the difference between DynCo and ensemble methods lies in the combination rule.
- The optimal value of the γ parameter gives information about how suitable decomposition is for a given example set.

Chapter 10

Comparison of four meta machine learning methods

The results in this section are to be published in the article [34]. The article has been presented at the fourth International Conference on Computational Intelligence and Neuroscience (ICCIN2000), which is part of the fifth International Joint Conference on Information Sciences (JCIS2000).

In chapter 9 five MML methods were compared on a group of natural example sets. The conclusion was that the two ME methods DynCo and XuME performed best. The ensemble methods Bagging, AdaBoost, and Simple performed about equally well. This is slightly surprising, since Bagging, and AdaBoost are renowned methods, while Simple is the simplest ensemble method conceivable. The common belief in the machine learning community is that AdaBoost is the overall best performing ensemble method because of the adaptive resampling method used (see [20, 72, 66, 95, 10]). Bagging should perform well for noisy example sets, especially with only a few examples, because of the variance reducing ability (see chapter 3). DynCo and AdaBoost should be able to take advantage of a large number of training examples. DynCo can use the extra information to decompose the input space and fine tune the individual experts, while AdaBoost through the adaptive resampling can use extra information. AdaBoost is known for disregarding a large number of examples (see [10]), so extra examples could be advantageous. AdaBoost should not do as well on the noisy example sets, because AdaBoost is known to have problems with outliers [66, 59, 10]. Simple should be a natural "zero" among the ensemble methods. If a method should be of any use, it should be better than Simple.

In a series of tests described below, the three ensemble methods are compared with each other to check the beliefs above. In the tests, seven different target functions are used to generate different size example sets and different

amounts of noise.

Below is the questions that will be answered in the following

1. How do the methods behave as a function of the size of the training set?
2. How do the methods behave when different amounts of noise are added to the training set?
3. Is AdaBoost generally the best performing method?
4. Does Bagging perform best on small and/or noisy example sets?
5. How does DynCo perform, which was the best method in chapter 9?
6. Are the other methods generally better than the Simple method?

10.1 The tests

The target function used to generate the example sets are listed in table 10.1

NAME	TARGET FUNCTION
SinC	$\frac{sin(x)}{x}$
Gabor	$\frac{2}{\pi}\exp[-2(x^2+y^2)]\cos[2\pi(x+y)]$
Multi	$0.79 + 1.27x_1x_2 + 1.56x_1x_4 + 3.42x_2x_5 + 2.06x_3x_4x_5$
Friedman1 [a]	$10\sin(\pi x_1x_2) + 20(x_3 - \frac{1}{2})^2 + 10x_4 + 5x_5$
Friedman2	$\sqrt{x_1^2 + (x_2x_3 - (\frac{1}{x_2x_4}))^2}$
Friedman3	$\tan^{-1}\frac{x_2x_3 - 1/(x_2x_4)}{x_1}$
Spiral	The Intertwined Spiral. See figure 10.1.

[a] There are ten inputs: $\{x_1, \ldots, x_{10}\}$, only the first five have influence on the output

Table 10.1: Target Functions

The target functions SinC, Gabor, and Multi are from [20], while the target functions Friedman1, Friedman2, and Friedman3 are from [68]. The Spiral target function is the well-known intertwined spiral target function (figure

10.1). The target functions were used to generate example sets by sampling the input domain uniformly, calculating the function, and maybe applying noise. For each target function three test rows were done. The first test row was without noise, and the size of the training set size was varied from 10 to 1000 in 10 exponential increasing steps. This is the *noise-free* test row. In the second test row, Gaussian noise with standard deviation 0.1 was added. The size of the training set size was varied from 10 to 4642 in 13 exponential increasing steps. This is the *moderate-noise* test row. The third test had Gaussian noise with standard deviation 0.5 added. The training set size was varied from 10 to 5000 in 13 exponential increasing steps. This is the *high-noise* test row. See table 10.2 for an overview of the test rows. The reason for choosing to train with larger training sets in the noisy test rows is that there might be effects that first would occur at low errors, as it must be assumed that more training examples are need to achieve low error for the noisy test rows.

All predictors (ensemble members or experts) were neural networks trained using back propagation (BP) with momentum. The default architecture of the neural networks for the ensemble method was one hidden layer with 25 hidden units, while the default architecture of the experts for DynCo was one hidden layer with five hidden units. The coefficient predictors always has one hidden layer with eight hidden nodes. The architectures used during training on the spiral example set were more complex, because it is more difficult for neural network predictors to learn than the other example sets. For the ensemble methods the architecture were two hidden layers each with twenty hidden units. The architecture of the DynCo predictors were one hidden layer with ten hidden units. The DynCo method does not need as complex experts, since the problem is decomposed by the coefficient predictors. The γ (see section 7.3.1) parameter of DynCo was always set to zero in order to make DynCo a "pure" ME method.

The ensemble members for AdaBoost and Bagging were trained for 5000 epochs on the entire training set. The predictors in Simple and DynCo were trained on 75 % of the training set. The last 25 % of the training examples were used for early stopping of training of the combined predictor.

For each amount of noise and training set size the training was repeated 20 times. For each training run a new training set was generated. The error measure was the MSE. The test error on the target function, i.e. the noise free function was found. The mean and standard deviation of the 20 training runs were calculated.

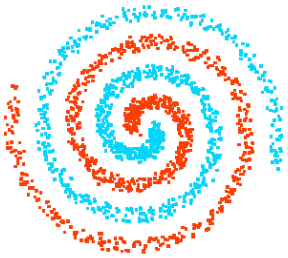


Figure 10.1: Spiral

NAME	NOISE	SIZE INTERVAL
Noise-free	0.0	10-1000
Moderate-noise	0.1	10-4642
High-noise	0.5	10-5000

Table 10.2: Test Rows

10.2 The results

In the order of 15000 training runs were done, so the results had to be processed in order to get an overview. This was done in two ways: The average test error as a function of the training set size (see figure 10.2–10.7) and the probability of a method giving the lowest error as a function of training set size (see figures 10.8–10.11).

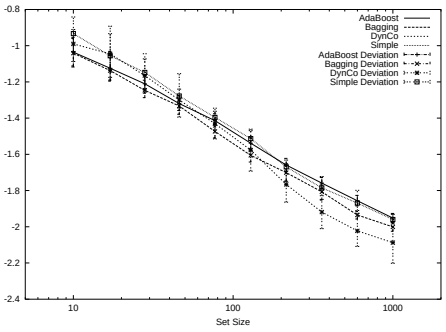


Figure 10.2: Noise-free.

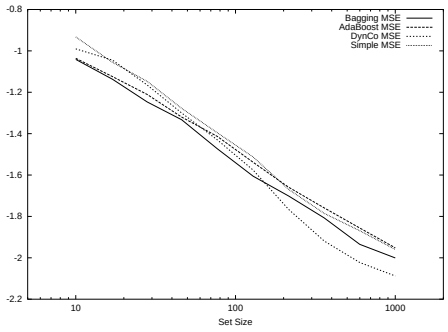


Figure 10.3: Noise-free.

The axis in figure 10.2–10.7 are scaled logarithmically, so a straight line indicates a polynomial connection between the Y-value and the X-value.[1] In this case Y-value and the X-value are respectively the test error and the training set size. The graphs are "sums" of the methods error on the seven target functions. The "sum" was found this way: First, the logarithm was taken on all test errors. For each target function the errors were scaled linearly, so the average error for all four methods for the smallest and biggest training set were respectively -1.0 and -2.0. This was done in order to avoid that test runs with large difference between lowest and highest error dominated the "sum". Note that the ordering of the four methods is preserved. The graphs are qualitative and not quantitative due to the rescaling.

[1] A straight line with slope a indicates the connecting $\log y = a \log x + b \Leftrightarrow y = e^b x^a$.

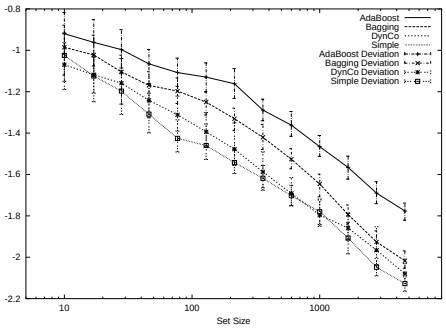


Figure 10.4: Moderate-noise.

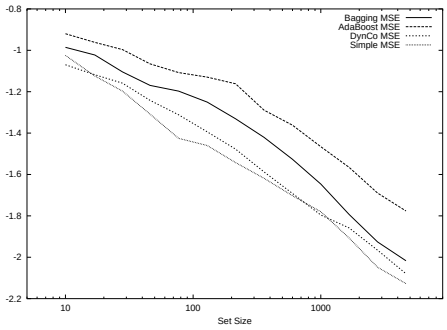


Figure 10.5: Moderate-noise.

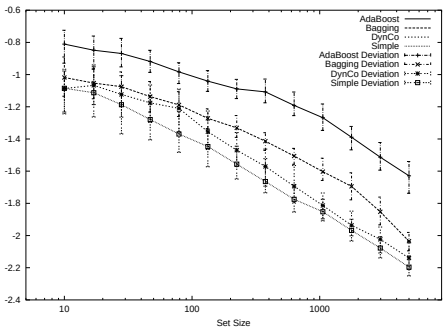


Figure 10.6: High-noise.

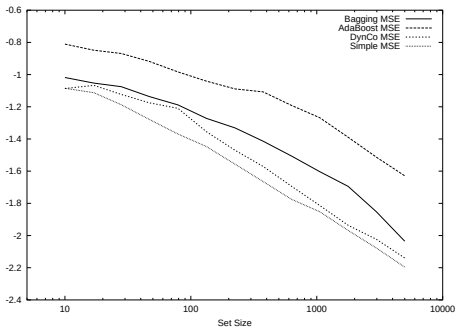


Figure 10.7: High-noise.

The scaled errors for all target functions are averaged over the seven target functions to give the final graphs. The graphs are presented in two versions, one with error bars, and one without. The graphs pairs are in figures (10.2,10.3), (10.4,10.5), and (10.6,10.7) for the noise-free, moderate-noise, and high-noise test rows respectively. The general picture is a straight line with a negative slope, most clearly for the noise-free test row (figures 10.2 and 10.3). This indicates that the graphs follow a power law. In table 10.3 the average slope is given for the log-log graph before rescaling of the four methods for each target function and noise level. The target function can be divided into three groups. The first group consists only of the SinC target function, where the slope is large even for high-noise. The second group consists of the Gabor, Friedman1, Friedman2, Friedman3, and Multi target function, which have comparable slops for the different noise levels: Around -1.0 for noise-free, -0.80 for moderate-noise and -0.65 for high-noise. The last group consists of the Spiral target function with a slope close to -0.44. The slops, especially for the noisy test rows, seem to reflect the difficulty of learning the problem, indicating that SinC is the most easy and Spiral is the most difficult. The slope decreases in magnitude as the noise level rises, which signifies that for a noisy training set, more examples are needed in

NAME	NO-NOISE	MODERATE	HIGH
SinC	-1.11 ± 0.21	-1.05 ± 0.13	-0.90 ± 0.14
Gabor	-0.97 ± 0.13	-0.79 ± 0.14	-0.67 ± 0.17
Friedman1	-1.22 ± 0.13	-0.82 ± 0.14	-0.63 ± 0.14
Friedman2	-1.18 ± 0.25	-0.91 ± 0.15	-0.66 ± 0.13
Friedman3	-0.80 ± 0.12	-0.82 ± 0.19	-0.64 ± 0.18
Multi	-1.04 ± 0.10	-0.79 ± 0.14	-0.65 ± 0.18
Spiral	-0.435 ± 0.050	-0.468 ± 0.049	-0.408 ± 0.049

Table 10.3: Slope of error for each target function and noise level

order to get the same decrease in test error, than for a noise-free training set. The errors on the slopes are loose overestimates of the possible maximal and minimal slope.

If we use the error bars as a standard unit, we see that the graphs of the methods become more clearly separated as the noise is increased. It can also be seen that the methods performs about equal for small training set.

To get a more quantitative measure of the methods, the average probability over the seven target functions that a given method will yield the lowest test error is found. The calculations are based upon the mean test errors and the standard deviations (see appendix K). The probabilities for three test rows can be found in figures 10.8, 10.10, and 10.11. For the noise-free test row (figure 10.8) we see that the methods start about equal in performance. AdaBoost and Simple quickly dwindle away, while Bagging and DynCo begin to outperform the others. First Bagging is best, but at around a couple of hundred training examples DynCo passes Bagging and rises to 80 % or three times more than for equally good methods.[2] This indicates that for large noise free training sets the DynCo method performs best. If only the three ensemble methods are compared, we get the graphs in figure 10.9. Bagging is the best performing method about 55 % to 60 % of the time for larger

[2]Equally performing methods would give the lowest error $\frac{1}{M}$ of the time for M methods. In this case M is four, so with a probability of 80 % DynCo is $0.8M = 3.2$ times better than it would have been if all methods performed equally.

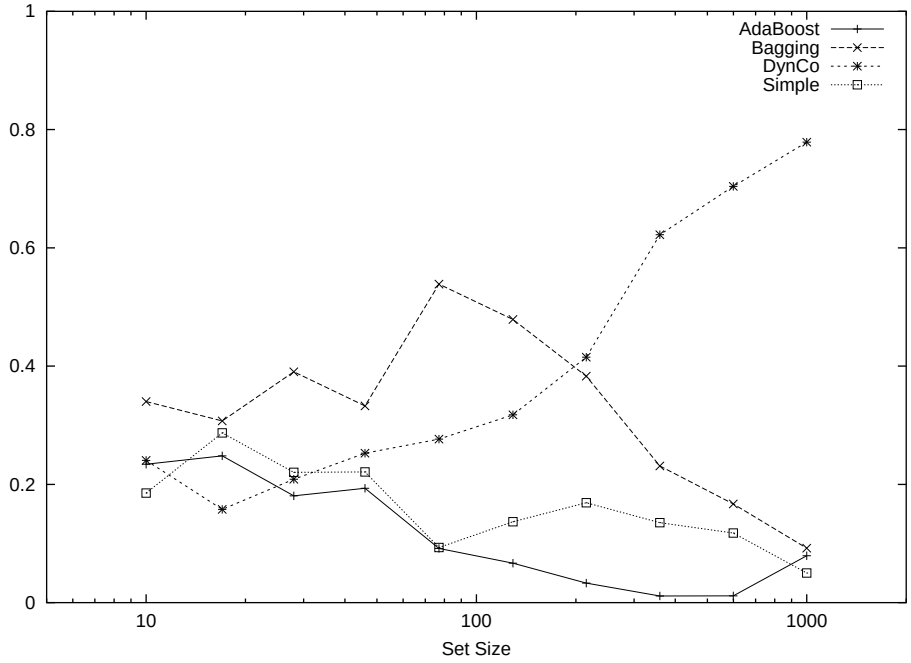


Figure 10.8: Noise-free. All.

training sets, which is more than 1.5 times the result for equal performing methods. AdaBoost and Simple perform approximately equal.

The figures 10.10 and 10.11 are very alike and will be treated as one. Again the methods perform almost equally for small training sets, with the exception of AdaBoost that does not seem to react well towards noise. Bagging dwindles away and Simple quickly reaches between 50 and 60 % or about 2-2.5 times more than for equally good methods. DynCo is constant around 30-40 % or a little more than for equally good methods.

We are now ready to give some empirical answers to the questions asked in the beginning of this chapter. It must be stressed that tests and thereby the answers are limited in a number of ways: The data is artificial and of low difficulty. There is no missing data or outliers. The noise is Gaussian with zero mean and uniform variance over the input domain. Still, some of the indications from the test, are surprising.

1. What is the behavior of the methods as function of size of the training set?

   Not surprisingly the test set errors decrease when the size of the training sets is increased. The errors seem to approximately follow a power law (see figures 10.3,10.5, and 10.7 and table 10.3). Generally the

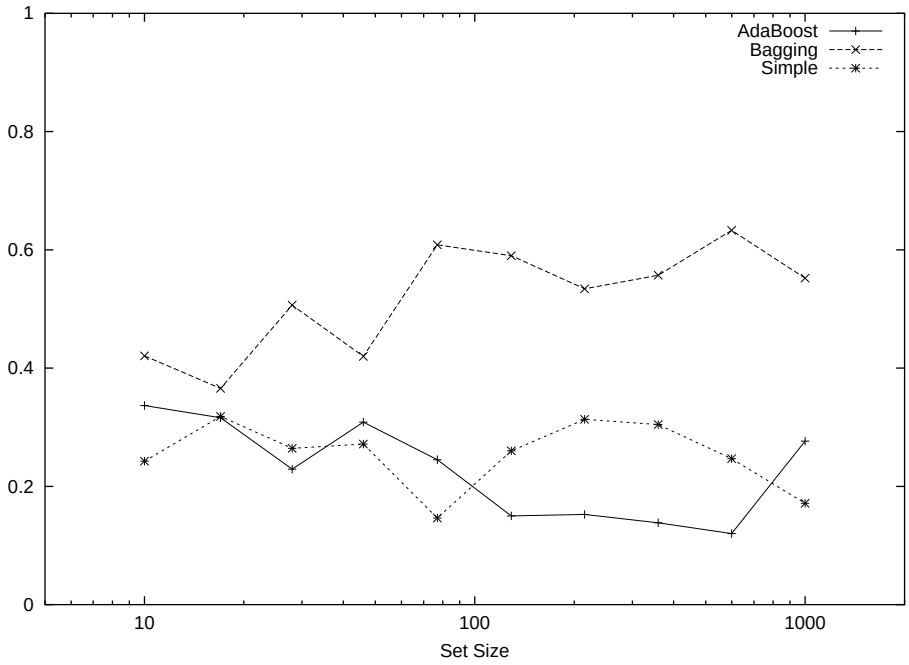


Figure 10.9: Noise-free. Ensembles.

four methods perform about equally for small training sets. For larger training sets one or two methods dominate.

2. What is the behavior of the methods as function of the amount of noise?

   More noise means that more examples are needed for a given reduction in test error. When no noise is added the training set size must be doubled for the error to be halved for most of the target functions. The size of the training set must be increased by almost a factor three when high noise is added. More surprisingly it turns out that Bagging is not as good a noise reducer as Simple and DynCo. AdaBoost seems to suffer greatly under noise.

3. AdaBoost is generally the worst performing method.

   AdaBoost performs about as well as Simple in the noise-free test row and worse than both DynCo and Bagging. For both the noisy test rows AdaBoost is the worst performing method. The probability that AdaBoost gives the lowest test error is mostly under 5 %, which is more than five times worse than for equally performing methods. It is well-known that AdaBoost is sensitive to outliers, and it is indicated that AdaBoost also is sensitive to noise, even with zero mean.

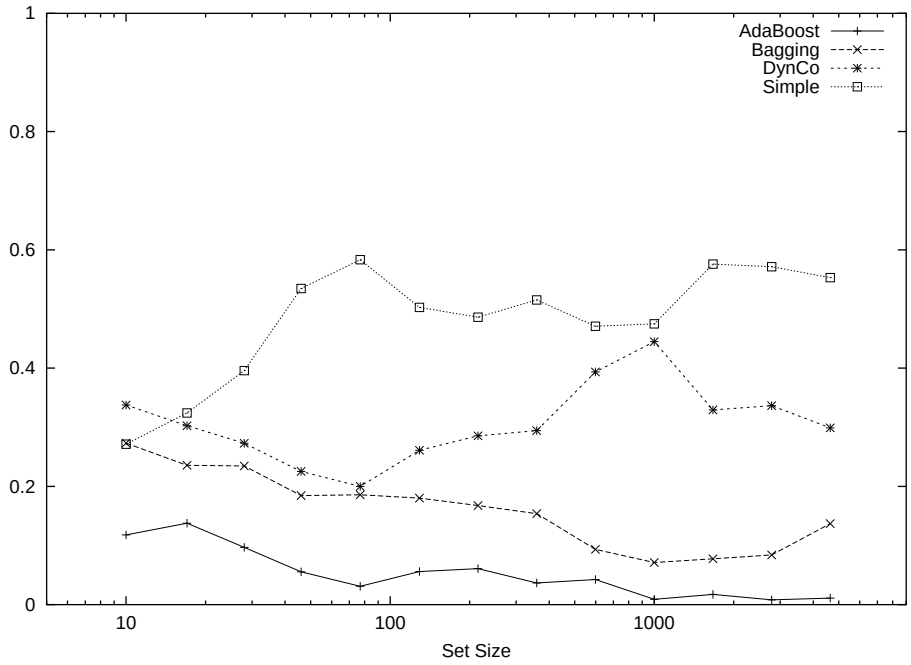


Figure 10.10: Noise 0.10.

4. Bagging performs best on small training sets *without* noise.

   Surprisingly Bagging is the best performing method only on noise-free training sets of size between 10 and 200 (see figure 10.8). It is the best performing ensemble method on the entire noise-free test row (see figure 10.9). Since Simple is so much better on the noisy test rows, the role of noise reducer for the Bagging method is questioned.

5. DynCo performs best on large noise-free training sets.

   DynCo seems to be able to use the extra information on large noise-free training set to achieve the lowest test error, as suggested above.

6. Simple performs best on large noisy training sets.

   Simple clearly outperforms all other methods on the noisy test rows.

That Simple performs so well is surprising. In section 10.3 there is an analysis that offers an explanation.

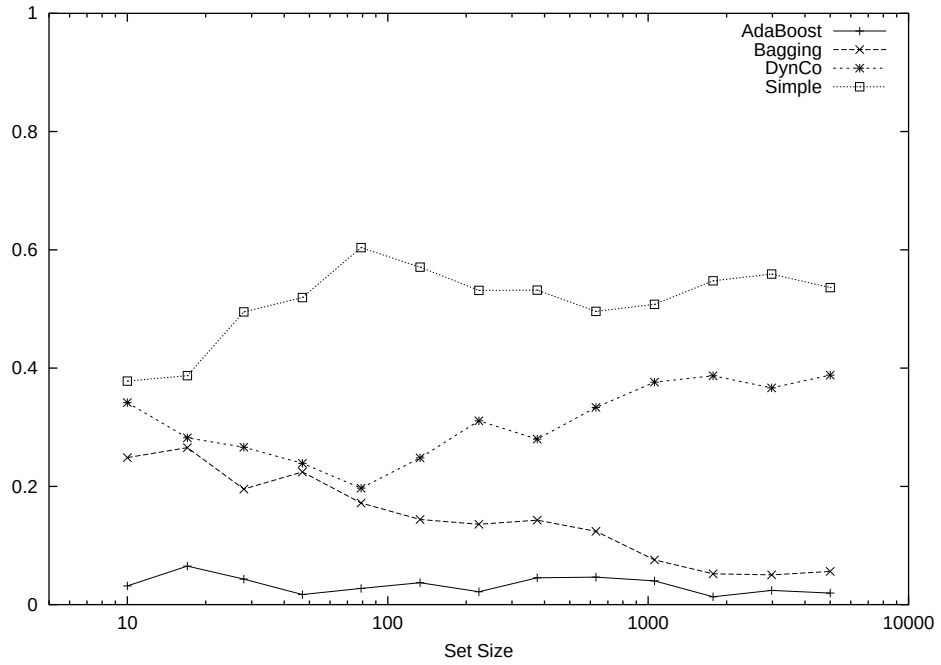


Figure 10.11: Noise 0.50.

10.3 Why Simple performs best on large noisy training sets

As noted in section 10.2 the Simple method gives the lowest error on the noise free target function when there are many training examples and noise. To better understand the situation let us look at a small part of the input space denoted A. We assume it is small enough for both the target function and the predictor to be effectively constant, while there still are some examples in A. The target function is the constant t over A, and the output of the predictor is the constant f over A. Let us say there are N examples from the original training set in A. These examples are not equal over A because of the noise. They are denoted $t_i^* = t + dt_i$. The noise has zero mean, so we assume that the dt_i's sum to zero. The difference between the three ensemble methods Bagging, Simple, and AdaBoost lie in the probability that an example is in the training set. Let w_i indicate the number of times example i is in the training set. Simple has $w_i = 1$, while Bagging can have an example in the training set any number of times, since the training set is generated by resampling with replacement. The probability that an example is in the training set k times is denoted $P(w_i = k)$. AdaBoost assigns resampling probabilities to each example as a part of the training method, so it is difficult to find the distribution of $P(w_i = k)$. Generally,

the examples with the highest previous error get the highest resampling probability. This will often give outliers a high probability.

The error function that we would like to minimize is the MSE on the target function

$$\bar{E}(t,f) = \frac{1}{N}\sum_{i}^{N}(t-f)^2$$

But the error function that actually is minimized is

$$\bar{E}(t^*,f) = \frac{1}{N}\sum_{i}^{N} w_i(t_i^*-f)^2.$$

This can be rewritten as

$$\begin{aligned}\bar{E}(t^*,f) &= \frac{1}{N}\sum_{i}^{N} w_i(t+dt_i-f)^2 \\ &= \frac{1}{N}\sum_{i}^{N} w_i[(t-f)^2+dt_i^2-2(t-f)dt_i]\end{aligned}$$

We ignore the intrinsic noise term $\frac{1}{N}\sum_i^N w_i dt_i^2$ that has no influence on training. For simple $w_i = 1$ so $\bar{E}(t^*,f)$ becomes

$$\bar{E}(t^*,f) = \frac{1}{N}\sum_{i}^{N}(t-f)^2,$$

since $\sum_i^N w_i 2(t-f)dt_i = 2(t-f)\sum_i^N dt_i = 0$. This error function has minimum in $f = t$ as we would like it to have. This is not so for Bagging where the probability for $w_i = k$ is (see appendix H)

$$P(w_i = k) \approx e^{-1}\frac{1}{k!}.$$

Let us simplify things and assume that there are two examples in A and the noise is given by $dt_1 = 1$ and $dt_2 = -1$, then the error function to minimize becomes

$$\bar{E}(t^*,f) = \frac{1}{2}(w_1+w_2)(t-f)^2 - (w_1-w_2)(t-f).$$

Let B denote $\frac{1}{2}(w_1+w_2)$ and let D denote $w_1 - w_2$. The mean over all possible training sets of B and D is one and zero respectively. But for a given training set B and D will generally not be one and zero. The deviation of B is $\frac{1}{\sqrt{2}}$, while the deviation of D is $\sqrt{2}$. Typically B is $1 \pm \frac{1}{\sqrt{2}}$ and D is $\pm\sqrt{2}$. The error function has minimum in $f = t + \frac{D}{2B}$. In the scenario above

$\frac{D}{2B}$ would typically be in the order of a half times the standard deviation of the noise, which is substantial. There is a number of things that makes this an overestimate. The estimates are based on the implicit assumption that the input space A is independent of the rest of input space. This is not so, the predictor will be influenced by nearby training examples, in all likelihood towards a better prediction. Also, the estimations were done on a single ensemble member and for $N = 2$. An entire ensemble will smooth the error. This is related to the variance reduction discussed in section 3.1. A more elaborate analysis with M ensemble members weighted uniformly and with N an arbitrary number of examples in A yields that the standard deviation of the offset $\frac{D}{2B}$ scales as $\frac{SD}{\sqrt{NM}}$, where SD is the deviation of the noise.

Still Bagging performs worse than Simple for noisy training sets with sufficiently many examples as the analysis above indicates.

It is more difficult to analyze the resampling probability of AdaBoost in the same setting as above. As a rule of thumb, AdaBoost will place high probability on outliers. This should have an even stronger effect on the minimum of the error function than for Bagging. This is in agreement with the empirical results from section 10.2, that unanimously declared AdaBoost the worst performing MML method, but this is not in agreement with the results in the majority of the literature on AdaBoost compared with other MML methods (see [20, 72, 66, 95, 10]). This will be discussed in section 10.4.

DynCo performs second best on the noisy training set. An analysis of the behavior of DynCo is difficult, because DynCo is not an ensemble method. If it is assumed that in a given area of the input space only one predictor has a coefficient substantially above zero, which is not unreasonable since the γ parameter always is set to zero, then DynCo effectively behaves as a single predictor. So the analysis above holds for DynCo, which explains that DynCo performs better than Bagging and AdaBoost. That DynCo performs worse than Simple can be explained by the variance reduction achieved by Simple.

10.4 Why AdaBoost performs poorly

In chapter 9, AdaBoost was compared with two other ensemble methods and two ME methods on natural data. AdaBoost did no better than the other ensemble methods and worse than the ME methods. In this chapter, AdaBoost was compared with two ensemble methods and one ME method on artificial data and performed worse than any other method. This is in contrast to the results in the literature (see [20, 72, 66, 95, 10, 5]) where

AdaBoost nearly always performs best. It is difficult to give a clear answer to this discrepancy. A possible explanation is that the base learner in the literature nearly always is a tree predictor (see [26, 23, 20, 10, 11, 66, 95, 72]) such as C4.5 [65], CART [13], or MARS [24], while the base learners in chapter 9 and chapter 10 are neural networks. It is a common belief that a tree predictor, at least with a limited depth, is not a very good stand-alone predictor, while a neural network can perform very well as a stand-alone predictor. The advantage of tree predictors are the learning speed. This might be the reason for the popularity of tree predictors in literature, since experiments with ensemble methods often demand a large number of training runs on many ensemble members. It is possible that AdaBoost and Bagging are well suited methods for improving "weak" learners like tree predictors, while the improvement on "strong" predictors are less than other methods.

Note that the explanation above is highly speculative, but the empirical results and the analysis do question the common belief that AdaBoost and related boosting methods are the methods of choice under all circumstances.

There are empirical studies in the literature, that supports the conclusions in this chapter. In [59] Opitz & Maclin present an empirical study of ensemble methods. The methods in question are AdaBoost, Arching-x4, Bagging, and Simple. Both tree predictors and neural networks were used as ensemble members. The conclusions for neural networks as ensemble members are very similar to the conclusions above: the Simple ensemble method performs well in comparison with the other methods, giving better or just as good results in many cases. AdaBoost and Arching-x4 are sensitive to noise. From the results in [59] no definite conclusion can be made whether tree predictors or neural networks are best as ensemble members. The overall conclusion is that Bagging and Simple (parallel ensemble methods) are the most stable in that they almost always yield an error reduction, while AdaBoost and Arching-x4 (boosting ensemble methods) can yield greater reduction in some cases, but can also increase the error on noisy training sets.

Chapter 11

Cross-validation logarithmic opinion pool ensemble and prediction of the secondary structure of protein.

A paper [36] written by the author of this dissertation and Anders Krogh presenting the meta machine learning method 'Cross-validation LOP ensemble' and the results on prediction of the secondary structure of proteins has been published in the proceedings of the conference 'Artificial Neural Networks in Medicine and Biology' (ANNIMAB-1). The paper has been presented orally at ANNIMAB-1.

Ensemble methods have been used before to predict the secondary structure of protein: see [74, 71] for examples in protein secondary structure, [3] for an overview of applications in molecular biology. The novelty in this chapter is the application of the LOP combination rule, the KL error function, and the cross-validation technique in that connection.

In section 11.1 we present the extension of the LOP ensemble method (see section 7.1.4). In section 11.2 the protein secondary structure problem is presented. The standard LOP ensemble and the cross-validation LOP ensemble are tested on the protein secondary structure problem and compared with single predictors and linear average predictor ensemble method in section 11.3.

11.1 The cross-validation ensemble method

The definition of the LOP ensemble (see section 7.1.4) is

The ensemble consists of M predictors f_i that each outputs a vector with class probabilities $\{f_i^{c_1}, \ldots, f_i^{c_N}\}$. Each ensemble member has associated a weight α_i. The weights are positive and sum to one, so they can be regarded as the probabilities of the ensemble members. If so the weights defines the mean operator $\langle \cdot \rangle_{\underline{\mathbf{F}}}$ as $\langle g(f) \rangle_{\underline{\mathbf{F}}} = \sum_i^M \alpha_i g(f_i)$. The target $\vec{y}$ is a vector with the target class probabilities $\{\vec{y}^{c_1}, \ldots, y^{c_N}\}$. If the targets are class examples the class probabilities are restricted to $y^{c_j} \in \{0, 1\}$. The combined predictor F is also a class probability vector $\{F^{c_1}, \ldots, F^{c_N}\}$. The combination rule for a class c is

$$F^c = \frac{1}{Z} \exp \langle \log(f_i^c(\vec{x})) \rangle,$$

where Z is a normalization factor satisfying:

$$Z = \sum_j^N \exp \langle \log(f_i^{c_j}(\vec{x})) \rangle.$$

This combination rule is non-linear and asymmetric as opposed to the linear average predictor.

The error on target $\vec{y}$ and combined predictor F is given by the Kullback-Leibler error function

$$E(\vec{y}, F) = \sum_j y^{c_j} \log \left(\frac{y^{c_j}}{F^{c_j}} \right). \tag{11.1}$$

The error is zero if F^c is equal to y^c for all c. For all appearances of this error function, the mean over the training set is implicitly taken. It can be decomposed into two terms with the LOP:

$$\begin{aligned} E(\vec{y}, F) &= \sum_{i=1}^M \alpha_i E(\vec{y}, f_i) - \sum_{i=1}^M \alpha_i E(F, f_i) \\ &= \langle E(\vec{y}, f_i) \rangle - A(f), \end{aligned} \tag{11.2}$$

where $A(f)$ is the ambiguity and $\langle \cdot \rangle$ is the weighted ensemble mean. This decomposition is from [39, 40], also see section 4.4 and 5.7.2.

The ambiguity term in (11.2) is independent of the target probability, which means that the ambiguity can be estimated using unlabeled data, or if the input distribution is known without data. Assuming we have estimates of the generalization error of the ensemble members and an estimate of the ambiguity, the estimated generalization error comes directly from (7.3). This

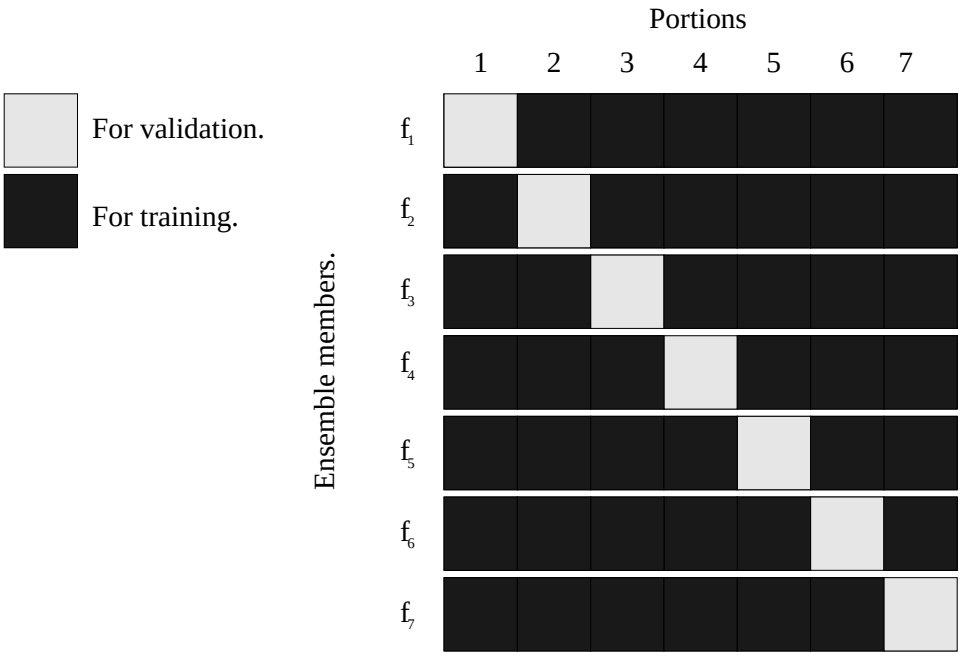


Figure 11.1: The cross-validation technique for the LOP ensemble method

can be achieved in the *cross-validation LOP ensemble*, where the training set is divided into M equally sized portions. Ensemble member f_i is trained on all the portions except for portion i. This is illustrated in figure 11.1.

Each of the seven bars represents the entire training set. The error on portion i is independent of training of ensemble member f_i and can be used to estimate the generalization error of the ensemble members. With the estimated ambiguity, this gives us a technique for obtaining an unbiased estimate of the ensemble error and still use all the training data. This is the *cross-validation technique*. Note that the term 'cross-validation' also is used for other techniques.

11.2 The protein secondary structure problem

Proteins [19] are sequences of amino acids, of which there are 20 different, so at the primary level of analysis they can be viewed as long sequences of letters from a 20 letter alphabet;[1] this is called the primary structure. A protein folds into a complex three-dimensional structure. This so-called tertiary structure determines the function of the protein, and is therefore of great interest in molecular biology. Because of the difficulties in determining

[1] The amino acids are denoted E,Y,L,M,Q,D,R,H,I,A,N,W,F,T,G,S,V,K,C, and P.

the structure experimentally, there is much focus on computational methods for predicting the structure from the sequence of amino acids. In principle this should be possible for most (but not all) proteins, because they fold spontaneously. However, the problem has proven to be very difficult, and therefore there has been quite an effort to predict the local structure as a first step.

There are two large classes of local structures that are stabilized by hydrogen bonds. The α-helix is a helical conformation of the amino acid chain with an average of 3.6 amino acids per turn and typical lengths between 4 and 20 amino acids. The β-sheet is a sheet-like structure, in which the protein folds back on itself and form hydrogen bonds between two or more strands of amino acids. These strands typically consist of 2–10 amino acids. When the structure is known, each amino acid of the sequence can be classified in one of three classes: part of an α-helix, a β-strand, or something else, which is usually called coil. This is the secondary structure. In the secondary structure prediction problem, the task is to predict these classes.

The local structure of an amino acid depends on the surrounding amino acids. For prediction it is therefore necessary to use a window of amino acids in the sequence surrounding the one to be predicted. For most methods it has proven optimal to use a window of 13-15 amino acids. The 20 different amino acids are usually coded as an indicator vector of size 20 with a one at the index of an amino acid and zeros for the rest. In order to cope with the ends of a protein "null amino acids" must be coded. This can be done by having all zeros, or extending the vector to size 21, where the last scalar represents the null amino acid. This sparse encoding ensures that there are no artificial correlations between the amino acids.

The secondary structure is known for less than 1000 non-homologous proteins, that is, proteins that have significantly different amino acid sequences, whereas the sequence is known for a very large number of proteins.

11.3 Empirical tests

The LOP ensemble is suitable for the protein problem for several reasons. First the protein problem is a classification problem, and the LOP ensemble method is tailor-made for classification, and secondly there is a huge amount of unlabeled data, i.e. proteins of unknown structure, which can be used to estimate the ambiguity.

The ensemble members were chosen to be neural networks. The output of

a neural network was post-processed by the SOFTMAX function defined as

$$f^j = \frac{\exp(g^j)}{\sum_j \exp(g^j)},$$

where g^j is the linear output (the weighted sum of the hidden units) of output unit j. This ensured that the ensemble members obeyed $\sum_j f_i^j = 1, f_i^j \geq 0$. The ensemble coefficients were uniform. Learning was done with back-propagation and momentum. A window of 13 amino acids was used, together with a sparse coding of amino acid into a vector of size 20, so each network had 260 inputs and three outputs. Each network had a hidden layer with 50 nodes. All examples in the training set were used once and only once during each epoch. Weights were updated after a certain number of randomly chosen examples had been presented (batch-update). During training the batch size was increased every tenth epoch by a number of examples equal to the square root of the training set size. The learning rate was decreased inversely proportional to the batch size. Training was stopped after 500 epochs, or if the validation error was increased by more than 10 % over the best value.

The data set consisted of 650 non-homologous protein sequences with a total of about 130,000 amino acids [52]. Four-fold cross-validation was used for each test, which means that the training set was divided into four equally sized sets. Training was done on three sets with 97,000 amino acids in total, while testing was done on the remaining set of 33,000 amino acids. The test set was rotated for each cross-validation run. The average of the four test runs was calculated.

In the cross-validation ensemble the validation error was the estimate of the generalization error calculated from (7.3) by using a set of 70,000 unlabeled amino acids to estimate the ambiguity. Seven ensemble members were used, which means that each one was trained on about 83,000 amino acids and validated on 14,000. Apart from the cross-validation ensemble, we also tested what we will term a simple ensemble, in which all the ensemble members were trained on the same training set of 83,000 amino acids, and the validation error was calculated from an independent set of labeled data containing 14,000 amino acids. Note that there still was the four-fold cross-validation as an 'outer loop' for both ensemble methods.

For every tenth epoch the ensemble validation error, the ensemble test error, and the average test error of the individual ensemble members were calculated. The graphs in figure 11.2 show these values for the cross-validation ensemble for a single test run.

It is clearly seen that an ensemble was better than the average of the individual members, as proven by the ambiguity decomposition.

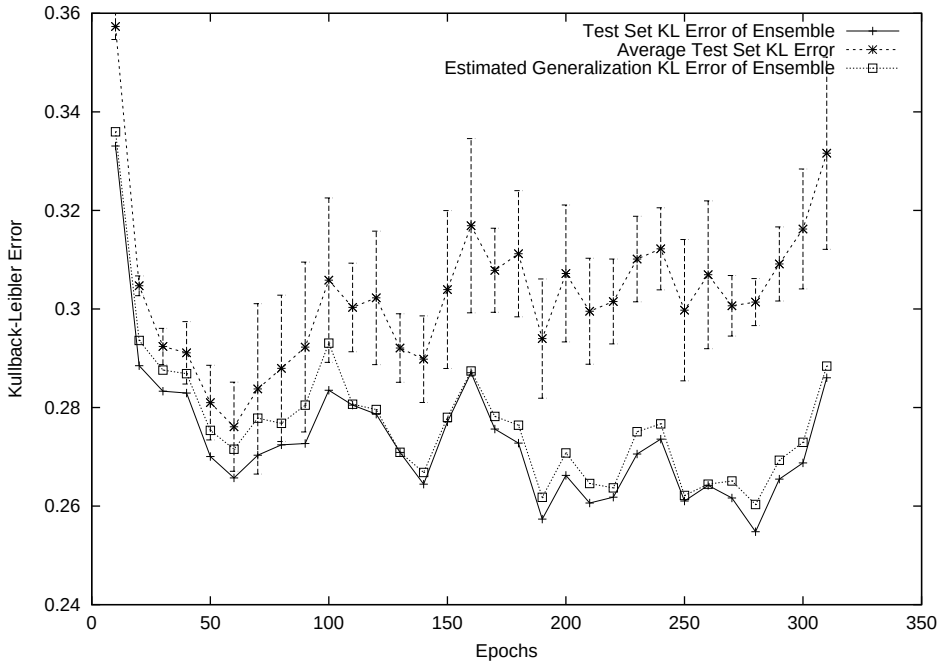


Figure 11.2: Cross-validation LOP ensemble

The cross-validation ensemble reached a lower generalization error (represented by the test error) than the simple ensemble. This can be explained by the fact that the difference in training sets makes the ensemble members differ more than if they were trained on the same data, and this increased the ambiguity, which in turn lowers the generalization error.

In a practical application, one would select the ensemble at the training epoch with the lowest validation error. The various errors are shown for the ensemble selected in this manner. Using the validation error to select the ensemble training makes training dependent on the validation error, and therefore the estimate of the generalization error becomes biased. We will call the time of lowest validation error the *stopping time*.

In table 11.1 the Kullback-Leibler error and misclassification rate at stopping time for the test runs is given. In these runs the validation error fluctuated by about ± 3 % around the test error. However, the oscillations of the validation error followed the oscillations of the test set error, as can be seen in figure 11.2, so the validation error could still be used to find the lowest test error. The validation error was not always at its minimum when the test set error was lowest, so a measure of the usability of validation error for stopping is the average difference between the lowest test set error and the test set error at stopping time. For both types of ensembles this difference was as low as 0.0001 or close to 0.05 %. So the estimated generalization

Combination rule	LOP	LOP	LAP
Validation error	Cross-validation	Simple	Cross-validation
Train error function	KL	KL	MSE
Test error	0.2546	0.2585	0.6293
Average of indiv.	0.287 ± 0.010	0.2802 ± 0.0065	1.46 ± 0.12
Misclassification rate	0.3331	0.3385	0.3542
Ambiguity	0.0327	0.0217	0.8320

Table 11.1: The Kullback-Leibler errors at stopping time.

error can very accurately be used to find the right stopping time.

The cross-validation ensemble reached a test error that is 1.5 % lower than for the simple ensemble, and a misclassification rate that was 1.6 % lower. The explanation lies in a larger ambiguity for the cross-validation ensemble, since the average test error of the ensemble members were comparable. The ambiguity of cross-validation ensemble was 1.5 times the ambiguity for the simple ensemble.

As noted in section 11.1 the error of the combined predictor was always better than the average of the error of the ensemble members. Still, one of the ensemble members could be better than the combined predictor. For the cross-validation ensemble method the test error was 0.2546, while the average of the error of the ensemble members was 0.2873. The difference (the ambiguity) was 0.0327. A gain of 12.8 %, which is substantial. The standard deviation on the test error of the ensemble members was 0.010, so the ambiguity was more than three times larger. It is very unlikely that any ensemble member had a lower generalization error than the ensemble error. For the simple ensemble method the ambiguity was smaller: 0.0217 or a gain of 8.4 %. The standard deviation among the ensemble members was 0.0065, so the ambiguity was more than three times the deviation.

The lowest average error for the ensemble members does not necessary happen when the ensemble error is lowest. Typically the lowest average generalization error of the ensemble members will be reached before the lowest generalization error for the ensemble, so the ensemble can actually gain from

overfitting in the individual ensemble members. This effect can be seen in figure 11.2. Also the optimal architecture for a simple predictor is often smaller than for ensemble members. A number of single predictors with different size hidden layer have been trained. The number of nodes in the hidden layer were varied from 3 to 400. The training must be done with a separate validation set, since there was no ambiguity for a single predictor. The best result was achieved with 10 hidden nodes giving an average generalization error of 0.2598, and a misclassification rate of 0.3413, which was respectively 2.0 % and 2.5 % more than the cross-validation LOP ensemble.

A *standard* cross-validation ensemble using the MSE error function and LAP combination rule was trained on the same data as the cross-validation LOP ensemble. The validation error was calculated using (7.3) even though the outputs do not necessary sum to one. The validation error have lost it's meaning as an error, e.g. it could be negative, but it was still valid as an early stopping indicator. The generalization error for the standard ensemble was much higher measured with the KL error function, namely 0.6293 or about 2.5 times more than the cross-validation LOP ensemble, but this is not a fair comparison, since the LOP ensemble was trained to minimize the KL error. Another measure is the misclassification rate: for the standard ensemble the misclassification rate was 0.3542, which is 6.3 % more than the misclassification rate of the cross-validation LOP ensemble.

Surprisingly the benefit was not in the combination rule. A test run, where the LOP was replaced with the LAP yields a generalization of 0.2543, which was essentially the same as for the LOP combination rule. The misclassification rate for the LAP was 0.3363, which 1.0 % more than the LOP.

Below we summarizes the results in this section.

- Estimation of generalization error with the cross-validation technique.

  The generalization error of an ensemble of predictors using a logarithmic opinion pool (LOP) can be estimated using cross-validation on the training set and an estimate of the ambiguity from an independent unlabeled set of data. When testing on prediction of protein secondary structure problem, it was shown that this estimate follows the oscillations of the error measured on an independent test set. The estimated error can be used to stop training when it is at a minimum.

- Cross-validation LOP ensemble performed best.

  The cross-validation LOP ensemble method is superior to single predictors, simple LOP ensemble, and standard ensemble methods using mean square error function on the protein problem. The benefit is not as much in the combination rule, as in the use of the Kullback-Leibler error function and the cross-validation technique.

11.4 Training of ensembles on parallel super computers

The training of ensembles on the secondary structure protein example set is very computational intensive, demanding in the order of 10^{13} weight updates for a training session. Also the amount of data is in the order of 100 megabytes (100×10^6 bytes). This is very demanding of the computers on which it is done. Fortunately, we were allowed to use the parallel super computing system at UNI-C, University of Aarhus, Denmark. Homepage: `http://supercomputing.uni-c.dk/aarhus`

The newest system at UNI-C is 'Karlsen' consists of 64 CPUs (Mips R12000, 300 MHz, 8 MB cache). The system has 64 GB memory (64×10^9 bytes) and approx. 1 TB (10^{12} bytes) of disk storage. Even on that system the training time for a single training session was in the order of days.

In order to use the parallel system a program was developed, based on PVM3, which stands for "Parallel Virtual Machine" version 3. Homepage: `http://www.epm.ornl.gov/pvm/pvm_home.html`. The ensemble members were trained in parallel gaining a time factor of about the ensemble size.

It is also possible to distribute the training on a local area network of workstations, but due to the large amounts of data that have to be copied over the network, we loose about a time factor of two on workstation as powerful as the individual CPUs on Karlsen. The data traffic cannot be avoided, since some of the computations are none-local, e.g. calculation of the validation error of the ensemble.

Chapter 12

Conclusion and recommendation

Was tust du?
Was fühlst du?
Was bist du?
— RAMMSTEIN, Sehnsucht, Tier.

In this dissertation I have presented advances in two areas of machine learning, namely in the theoretical area of bias/variance decompositions, and in the more empirical area of developing, testing, and analyzing meta machine learning methods. The work has resulted in five papers. Two have been published [32, 33], two have been accepted for presentation [34, 36], and one is in submission [35]. After the dissertation has been handed in, the articles [34, 36] have been presented and published. The article [35] has been accepted for presentation.

The previously open problem on exactly which error functions have a natural bias/variance decomposition with target independent bias/variance is closed. The error functions is derived through the deviance from the one-parameter exponential family of distributions. This work was done in cooperation with Tom Heskes.

The common use of the mean square error with the corresponding assumption of Gaussian noise on the target function can now be extended to any error function derived from the one-parameter exponential family of distribution with assumption of noise from that distribution, without loosing any of the essential properties including a bias/variance decomposition with target independent variance.

The DynCo mixtures of experts method was reinvented and shown empirically to perform well, indicating that a cooperative error function is superior

to a competitive error function. The DynCo method can also be used to test whether a problem benefits from decomposition or not. The Simple ensemble method was shown empirically to outperform such renowned methods as Bagging and AdaBoost in many cases. The logarithmic opinion pool ensemble method was developed and it was shown empirically to perform well, especially in connection with the cross-validation technique. This was done in cooperation with Anders Krogh.

I will now bring the results together and present the concluding unified recommendation of this dissertation.

We have seen that both the DynCo method with low γ value and the Simple ensemble are well-performing methods. Also the cross-validation LOP ensemble method performs well, due to the cross-validation technique. The Simple ensemble method and the LOP ensemble method are similar except for the combination rule.

We define a family of Simple ensemble methods, that differs only in their combination rule, and thereby generally also in the error function. We state the conjecture that this family of ensemble methods is well-performing, and benefits from the cross-validation technique. The cross-validation technique demands that the error function in question has an ambiguity decomposition with target independent ambiguity. We have shown that exactly the deviance error functions derived from the one-parameter exponential family of distributions have such an decomposition. It is also possible to use these error functions with the DynCo method. This brings us to the recommendation of the dissertation:

To learn a target function, use a deviance error function that corresponds to the noise on the problem. Test whether the problem benefits from decomposition or not (use the optimal value of γ to make this decision). If the problem benefits from decomposition, then apply the DynCo mixture of experts method with low γ value using the chosen deviance error function. If not, apply the Simple ensemble method using the cross-validation technique and the chosen deviance error function.

Appendix A

Notation and symbols

A.1 Statistical notation

- $P_{\underline{\mathbf{X}}}(x)$: The probability of outcome x of the discrete stochastic variable $\underline{\mathbf{X}}$. If the stochastic variable is given by the context then the short hand $P(x)$ is used.
- $p_{\underline{\mathbf{X}}}(x)$: The density of outcome x of the continuous stochastic variable $\underline{\mathbf{X}}$. If the stochastic variable is given by the context then the short hand $p(x)$ is used.
- $P(x|y)$ or $p(x|y)$: Respectively the conditional probability or density of $\underline{\mathbf{X}}$ on $\underline{\mathbf{Y}}$. The definitions are

$$P(x|y) = \begin{cases} \frac{P(x,y)}{P(y)} & P(y) > 0 \\ 0 & P(y) = 0 \end{cases},$$

and

$$p(x|y) = \begin{cases} \frac{p(x,y)}{p(y)} & p(y) > 0 \\ 0 & p(y) = 0 \end{cases}.$$

- $\langle g(\underline{\mathbf{X}})\rangle_{\underline{\mathbf{X}}}$: The mean of the function g with respect to stochastic variable $\underline{\mathbf{X}}$. If the stochastic variable in question is given by the context, the short hand $\langle g(\underline{\mathbf{X}})\rangle$ will be used. If $\underline{\mathbf{X}}$ is discrete then

$$\langle g(\underline{\mathbf{X}})\rangle_{\underline{\mathbf{X}}} = \sum_{x:P_{\underline{\mathbf{X}}}(x) \text{ i.d.}} P_{\underline{\mathbf{X}}}(x)g(x),$$

where "i.d" stands for "is defined". If $\underline{\mathbf{X}}$ is continuous then

$$\langle g(\underline{\mathbf{X}})\rangle_{\underline{\mathbf{X}}} = \int_{\underline{\mathbf{X}}} dx p_{\underline{\mathbf{X}}}(x)g(x).$$

A.2 Commonly used machine learning symbols

- f: A predictor. Often the predictor depends on a set of parameters $\vec{w}$, in that case it will be written $f(\vec{x}, \vec{w})$.
- F: The combined predictor F depends on a group of predictors.
- $\bar{f}$: The average predictor $\bar{f}$ is defined on a mean operator over a group of predictors.
- $\hat{f}$: The optimal predictor $\hat{f}$ is defined as

$$\hat{f} = \operatorname*{argmin}_f E(f, t).$$

- nn: A feed-forward neural network.
- T: A training set is a set of input-output pairs $(\vec{x}_i, \vec{y}_i)$.
- L: A machine learning method L is a function that takes a training set T as input and generates a predictor as output; $f = L(T)$.
- MML: A meta machine learning method. Input is a machine learning method L and a training T. Output is a combined predictor $F = MML(L, T)$.
- MSE: The mean square error is defined as

$$E(f, t) = \frac{1}{2}(f - t)^2.$$

- KL: The Kullback-Leibler error is defined as

$$E(t, f) = \sum_c t^c \log \frac{t^c}{f^c}$$

  where c is a class label.
- LAP: The linear average predictor. A predictor combination function defined as

$$F = \sum_i \alpha_i f_i.$$

- LOP: The logarithmic opinion pool. A estimator combination function

$$F = \frac{1}{Z} \exp[\sum_i \alpha_i \log f_i]$$

- ME: Mixtures of experts. See section 7.2.

Appendix B

Training of LOP ensemble while achieving well-conditioned Hessian matrix

The logarithmic opinion pool (LOP) ensemble method (see section 7.1.4) is a probability estimation ensemble method for classification. It is developed by the author of this dissertation based on [40]. The LOP ensemble method has been tested on the secondary structure of proteins problem (see chapter 11).

In this appendix we present the information necessary to train a LOP ensemble with neural network predictors with back-propagation (see section 6.1). Furthermore, we investigate the Hessian Matrix, that contains second order derivative information, in order to achieve better learning. Based on this investigation we propose an activation function that removes some of the reasons for an ill-conditioned Hessian matrix. An ill-conditioned Hessian matrix yields a low convergence rate for gradient descent (see [55] p. 17). The investigation of the Hessian matrix on ensemble members in a LOP ensemble is similar to an investigation of the Hessian matrix for standard neural networks in [85].

B.1 Definitions and notation

Our problem area is classification, so the outputs are probability estimations, which means that the sum of the outputs is one. The inputs are not

limited. The learning mechanisms are neural networks and ensembles of neural networks. The following notation is used for the networks and the ensembles:

- $f_i^c(\vec{x})$: The probability output for class c of ensemble member f_i with input $\vec{x}$.
  Each ensemble member has a corresponding weight α_i. The weights are positive and sum to one.
- $F^c(\vec{x})$: The combined output of the ensemble members for class c with input $\vec{x}$.
  The combined output is defined with the logarithmic opinion pool (LOP)
  $$F^c(\vec{x}) = \frac{1}{Z} \exp[\sum_i \alpha_i \log f_i^c(\vec{x})],$$
  where Z is a normalization factor defined as
  $$Z = \sum_c \exp[\sum_i \alpha_i \log f_i^c(\vec{x})].$$
- $t_c(\vec{x})$: The target function for class c at input $\vec{x}$.
- $T = \{(\vec{y}_1, \vec{x}_1), \ldots, (\vec{y}_N, \vec{x}_N)\}$ is an example set.
  The vector $\vec{y}_p = (y_1^{(p)}, \ldots, y_{N_c}^{(p)})$ is the density vector for input-output pair $(\vec{y}_p, \vec{x}_p)$. Often only one $y_c^{(p)}$ takes the value one.

In a LOP ensemble the ensemble members are trained separately. We will therefore consider training of a single ensemble member, so we omit the ensemble index i. We assume that all ensemble members are three layer neural networks. So we have an output layer, a hidden layer, and an input layer. The numbers of nodes in each layer are respectively N_c, N_h, and N_i. An extra node is added to the hidden layer and the input layer, with a forced activation value of -1. This is equivalent to having threshold for the nodes in the hidden layer and the output layer. The real number of nodes in the input layer and the hidden layer are then $N_h + 1$ and $N_i + 1$. The activation function used is $s = \tanh$. The state of output node c is:

$$a_c(\vec{x}) = \sum_h^{N_h+1} w_{hc} s(\sum_i^{N_i+1} w_{ih} x_i) = \sum_h^{N_h+1} w_{hc} s(a_h(\vec{x})),$$

where a_h is the state value of the hidden node h. The activations values of the output nodes are combined to form the output of the network with the SOFTMAX post-processing function

$$f^c(\vec{x}) = \frac{\exp(a_c(\vec{x}))}{\sum_{c'} \exp(a_{c'}(\vec{x}))}$$

The error measurement used is the Kullback-Leibler cross-entropy (see e.g. [39]) error function:

$$E(f^c(\vec{x}^{(p)})) = \sum_p E(\vec{x}^{(p)}) = \sum_p \sum_c y_c^{(p)} \log \frac{y_c^{(p)}}{f^c(\vec{x}^{(p)})} \tag{B.1}$$

In the following we will omit the data set index p, the dependency on input $\vec{x}$ and only concentrate on one term in the error function in equation B.1

B.2 The derivatives

Many learning methods, e.g. back-propagation (see [7] and section 6.2) use information about the first derivatives of the error function with regard to the parameters of the predictor. To investigate characteristics of learning, information about the second derivatives (the elements of the Hessian matrix) are used. Below the first and second derivatives of the KL-error function (B.1) with regards to the weights of a neural network are calculated.

Before we show the derivation of the derivatives, we present a couple of useful results. The post-processed outputs of an ensemble member are class probabilities, so we define a mean operator

$$\langle h(c) \rangle_{\underline{\mathbf{C}}} = \sum_c f^c h(c),$$

The weights between a given node in the hidden layer and the output layer can be viewed as a function on the classes. Let the hidden node be h, then the mean of the weights w_{hc} is

$$\langle w_{hc} \rangle_{\underline{\mathbf{C}}} = \sum_c f^c w_{hc}.$$

For notational convenience we omit the stochastic variable $\underline{\mathbf{C}}$ in the following.

In a similar fashion we define the covariance over weights from two hidden nodes h and h' as

$$\begin{aligned} \mathrm{Cov}(w_{hc}, w_{h'c}) &= \langle [w_{hc} - \langle w_{hc} \rangle][w_{h'c} - \langle w_{h'c} \rangle] \rangle \\ &= \langle w_{hc} w_{h'c} \rangle - \langle w_{hc} \rangle \langle w_{h'c} \rangle \end{aligned} \tag{B.2}$$

An often used derivative is

$$\frac{\partial f^{c'}}{\partial a_c} = \delta_{(c',c)} f^{c'} - f^{c'} f^c, \tag{B.3}$$

Where $\delta_{(c',c)}$ is the Kronecker delta defined as $\delta_{(c',c)} = \begin{cases} 1 \text{ if } c = c', \\ 0 \text{ else.} \end{cases}$

$$\frac{\partial a_{c'}}{\partial w_{hc}} = \delta_{(c',c)} s(a_h). \tag{B.4}$$

$$\frac{\partial a_c}{\partial w_{ih}} = x_i w_{hc} s'(a_h). \tag{B.5}$$

Using the chain rule for derivation and (B.3) we get

$$\frac{\partial f^{c'}}{\partial w_{hc}} = f^{c'}(\delta_{(c,c')} - f^c)\frac{\partial a_c}{\partial w_{hc}}. \tag{B.6}$$

$$\frac{\partial f^c}{\partial w_{ih}} = f^c(\frac{\partial a_c}{\partial w_{ih}} - \sum_{c'} f^{c'} \frac{\partial a_{c'}}{\partial w_{ih}}). \tag{B.7}$$

We have $E = \sum_c y^c \log \frac{y^c}{f^c}$ so

$$\frac{\partial E}{\partial a_c} = f^c - y^c. \tag{B.8}$$

Note that this is exactly the same derivative as for the MSE with the identity function as post-processing function (no post-processing function). The Kullback-Leibler error function and the SOFTMAX post-processing function "annihilates" each other. This has been observed in connection with some of the deviance error function (see section 8.2).

Using the chain rule we can find the first derivatives:

$$\frac{\partial E}{\partial w_{hc}} = \sum_{c'} \frac{\partial E}{\partial a_{c'}} \frac{\partial a_{c'}}{\partial w_{hc}} = (f^c - y^c)\frac{\partial a_c}{\partial w_{hc}}. \tag{B.9}$$

$$\frac{\partial E}{\partial w_{ih}} = \sum_{c'} \frac{\partial E}{\partial a_{c'}} \frac{\partial a_{c'}}{\partial w_{ih}} = \sum_{c'} (f^{c'} - y^{c'})\frac{\partial a_{c'}}{\partial w_{ih}} \tag{B.10}$$

Using (B.4) and (B.5) we get

$$\frac{\partial E}{\partial w_{hc}} = (f^c - y^c) s(a_h).$$

$$\frac{\partial E}{\partial w_{ih}} = \sum_{c'} (f^{c'} - y^{c'}) x_i w_{hc} s'(a_h).$$

The second order derivatives are

$$\begin{aligned}\frac{\partial^2 E}{\partial w_{h'c'}\partial w_{hc}} &= \frac{\partial}{\partial w_{h'c'}}(f^c - y^c)\frac{\partial a_c}{\partial w_{hc}} \\ &= \frac{\partial f^c}{\partial w_{h'c'}}\frac{\partial a_c}{\partial w_{hc}} + (f^c - y^c)\frac{\partial^2 a_c}{\partial w_{h'c'}\partial w_{hc}} \\ &= f^c(\delta_{(c',c)} - f^{c'})\frac{\partial a_{c'}}{\partial w_{h'c'}}\frac{\partial a_c}{\partial w_{hc}} + (f^c - y^c)\frac{\partial^2 a_c}{\partial w_{h'c'}\partial w_{hc}}\end{aligned}$$

$$\begin{aligned}\frac{\partial^2 E}{\partial w_{ih'}\partial w_{hc}} &= \frac{\partial}{\partial w_{ih'}}(f^c - y^c)\frac{\partial a_c}{\partial w_{hc}} \\ &= \frac{\partial f^c}{\partial w_{ih'}}\frac{\partial a_c}{\partial w_{hc}} + (f^c - y^c)\frac{\partial^2 a_c}{\partial w_{ih'}\partial w_{hc}} \\ &= f^c(\frac{\partial a_c}{\partial w_{ih'}} - \sum_{c'} f^{c'}\frac{\partial a_{c'}}{\partial w_{ih'}})\frac{\partial a_c}{\partial w_{hc}} + (f^c - y^c)\frac{\partial^2 a_c}{\partial w_{ih'}\partial w_{hc}}\end{aligned}$$

$$\begin{aligned}\frac{\partial^2 E}{\partial w_{h'c}\partial w_{ih}} &= \frac{\partial}{\partial w_{h'c}}\sum_{c'}(f^{c'} - y^{c'})\frac{\partial a_{c'}}{\partial w_{ih}} \\ &= \sum_{c'}\frac{\partial f^{c'}}{\partial w_{h'c}}\frac{\partial a_{c'}}{\partial w_{ih}} + \sum_{c'}(f^{c'} - y^{c'})\frac{\partial^2 a_{c'}}{\partial w_{h'c}\partial w_{ih}} \\ &= \sum_{c'} f^{c'}(\delta_{(c,c')} - f^c)\frac{\partial a_{c'}}{\partial w_{h'c}}\frac{\partial a_c}{\partial w_{ih}} + \sum_{c'}(f^{c'} - y^{c'})\frac{\partial^2 a_{c'}}{\partial w_{h'c}\partial w_{ih}}\end{aligned}$$

$$\begin{aligned}\frac{\partial^2 E}{\partial w_{i'h'}\partial w_{ih}} &= \frac{\partial}{\partial w_{i'h'}}\sum_{c'}(f^{c'} - y^{c'})\frac{\partial a_{c'}}{\partial w_{ih}} \\ &= \sum_{c'}\frac{\partial f^{c'}}{\partial w_{i'h'}}\frac{\partial a_c}{\partial w_{ih}} + \sum_{c'}(f^{c'} - y^{c'})\frac{\partial^2 a_{c'}}{\partial w_{i'h'}\partial w_{ih}} \\ &= \sum_{c'} f^{c'}(\frac{\partial a_{c'}}{\partial w_{i'h'}} - \sum_{c} f^{c}\frac{\partial a_c}{\partial w_{i'h'}})\frac{\partial a_{c'}}{\partial w_{ih}} + \sum_{c'}(f^{c'} - y^{c'})\frac{\partial^2 a_{c'}}{\partial w_{i'h'}\partial w_{ih}}\end{aligned}$$

Generally we have

$$\frac{\partial^2 E}{\partial w_i \partial w_j} = \sum_c \frac{\partial f^c}{\partial w_i}\frac{\partial a_c}{\partial w_j} + \sum_c (f^c - y^c)\frac{\partial^2 a_c}{\partial w_i \partial w_j}, \tag{B.11}$$

where index i and j are a renaming of either index ih and hc. Using the approximation $\sum_c (f^c - y^c)\frac{\partial^2 a_c}{\partial w_i \partial w_j} = 0$, which is valid when f^c is close to y_c yields

$$\frac{\partial^2 E}{\partial w_i \partial w_j} \approx \sum_c \frac{\partial f^c}{\partial w_i}\frac{\partial a_c}{\partial w_j} \tag{B.12}$$

With this approximation the elements of the Hessian matrix can be calculated from first order derivatives.

Assuming $f^c \approx y^c$ we have

$$\frac{\partial^2 E}{\partial w_{i'h'} \partial w_{ih}} = x_{i'} x_i s'(a_{h'}) s'(a_h)(\langle w_{h'c} w_{hc}\rangle - \langle w_{hc}\rangle\langle w_{h'c}\rangle). \tag{B.13}$$

$$\frac{\partial^2 E}{\partial w_{h'c} \partial w_{ih}} = x_i s'(a_h) s(a_{h'}) f^c (w_{hc} - \langle w_{hc}\rangle). \tag{B.14}$$

$$\frac{\partial^2 E}{\partial w_{ih} \partial w_{h'c}} = x_i s'(a_{h'}) s(a_h) f^c (w_{hc} - \langle w_{hc}\rangle). \tag{B.15}$$

Note that $\frac{\partial^2 E}{\partial w_{h'c} \partial w_{ih}} = \frac{\partial^2 E}{\partial w_{ih} \partial w_{h'c}}$ as they should be. The term $w_{hc} - \langle w_{hc}\rangle$ is a measure of the size of w_{hc} compared to the average.

$$\frac{\partial^2 E}{\partial w_{h'c'} \partial w_{hc}} = s(a_{h'}) s(a_h) f^c (\delta_{(c,c')} - f^{c'}). \tag{B.16}$$

Let $w_{hc}^{c'}$ be a pseudo weight defined as

$$w_{hc}^{c'} = \begin{cases} 1 \text{ if } c = c', \\ 0 \text{ else.} \end{cases} \tag{B.17}$$

So $w_{hc}^{c'} = \delta_{(c,c')}$. The second derivatives can be reformulated in terms of covariances:

$$\frac{\partial^2 E}{\partial w_{i'h'} \partial w_{ih}} = x_{i'} x_i s'(a_{h'}) s'(a_h) \mathrm{Cov}(w_{h'c}, w_{hc}). \tag{B.18}$$

$$\frac{\partial^2 E}{\partial w_{h'c}\partial w_{ih}} = x_i s'(a_h) s(a_{h'}) \mathrm{Cov}(w^c_{hc}, w_{hc}). \tag{B.19}$$

$$\frac{\partial^2 E}{\partial w_{ih}\partial w_{h'c}} = x_i s'(a_{h'}) s(a_h) \mathrm{Cov}(w^c_{hc}, w_{hc}). \tag{B.20}$$

$$\frac{\partial^2 E}{\partial w_{h'c'}\partial w_{hc}} = s(a_{h'}) s(a_h) \mathrm{Cov}(w^c_{hc}, w^{c'}_{hc}). \tag{B.21}$$

B.3 The condition of the Hessian matrix

Above the second derivatives are expressed using covariance between the weights. The term $\langle w_{h'c} w_{hc}\rangle - \langle w_{hc}\rangle\langle w_{h'c}\rangle$ in (B.13) is the covariance of the weights from the hidden nodes h and h' with regard to the output of the net interpreted as probabilities. It is not unreasonable to expect high correlation between the weight groups w_{hc} and $w_{h'c}$, since the hidden nodes h and h' are interchangeable in the network architecture.

Note that the covariance is zero if the probability of one of the classes is one, that is if an output is one. For three of the expression for covariance (B.19–B.21), the covariance is zero if one of the class probabilities is zero. The sum of the second derivatives over the example set are the elements of the Hessian matrix. A badly conditioned Hessian implies low learning rate for back-propagation (see [55] p. 17).

A matrix can be ill-conditioned if some of the diagonal elements are much smaller than the others. As has been explained above the covariance can become zero in some situations, thereby making the Hessian matrix ill-conditioned. That occurs if the output is one or zero, but this generally happens when the target function is learned, so we assume that the covariance factors is not zero. If we assume that inputs are distributed with mean zero, and are independent, we see that the elements in (B.18–B.20) are close to zero because $\langle x_j x_i\rangle = \langle x_i\rangle = 0$ except if $i = j$ in (B.18). We can therefore assume that the off-diagonal elements are smaller than the diagonal elements. A similar analysis can be applied to the value of the elements from (B.20). It is more complicated since the input has been through a transformation, but the general picture is the same. There is one more possibility for the elements to become zero, and that is if the hidden units are saturated, so $s'(a_h) \approx 0$. Then the diagonal elements from equation B.18 become much smaller than the diagonal elements from (B.21) and this can happen

before the target function is learned. To amend this, let s be replaced with a function g that does not have any derivatives that goes asymptotically towards zero. Such a g could be $g(x) = \tanh(x) + \log(\cosh(x))$ [85]. The function $\log(\cosh(x))$ is approximately equal to $|x| - log(2)$ when $|x| > 0$, and is close to zero if $|x| \approx 0$, so $g(x)$ behaves as $\tanh(x)$ for input close to zero and as $|x| + constant$ for input not close to not close to zero. Note that $\frac{\partial \log(\cosh(x))}{\partial x} = \tanh(x)$, so $g'(x) = s'(x) + s(x)$. The new definition of a_c becomes

$$a_c = \sum_c w_{hc} g(\sum_h w_{ih} x_i) = \sum_c w_{hc}(s(a_h) + S(a_h)),$$

where $S = \log\cosh$. Note that in order to calculate the derivatives of the new function we only have to replace s with g in the derivatives above.

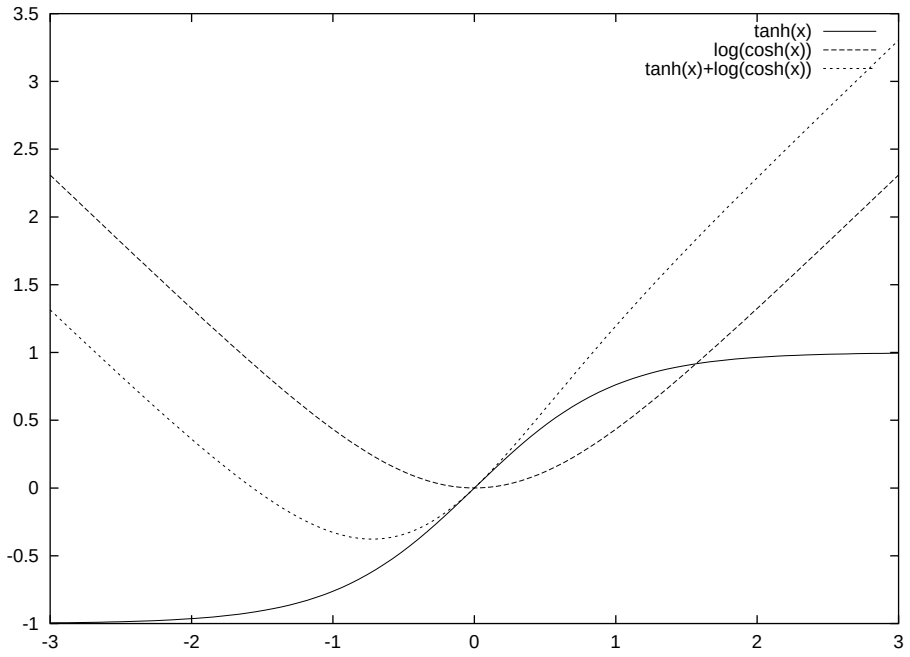


Figure B.1: $\tanh(x) + \log\cosh(x)$

In figure B.1 the $\tanh(x)$, $\log\cosh(x)$, and $\tanh(x) + \log\cosh(x)$ are plotted. We see that the alteration of the $\tanh(x)$ function is substantial. Empirical tests have indicated that the alteration is too large, therefore we suggest using the activation function $\tanh(x) + \frac{1}{10}\log\cosh(x)$.

In figure B.2 this function is plotted together with $\tanh(x)$, $\frac{1}{10}\log\cosh(x)$, and $\frac{1}{10}\log\cosh(x) + \tanh(x)$. We see that the difference between the t-wo activation functions $\tanh(x)$ and $\tanh(x) + \frac{1}{10}\log\cosh(x)$ is much more

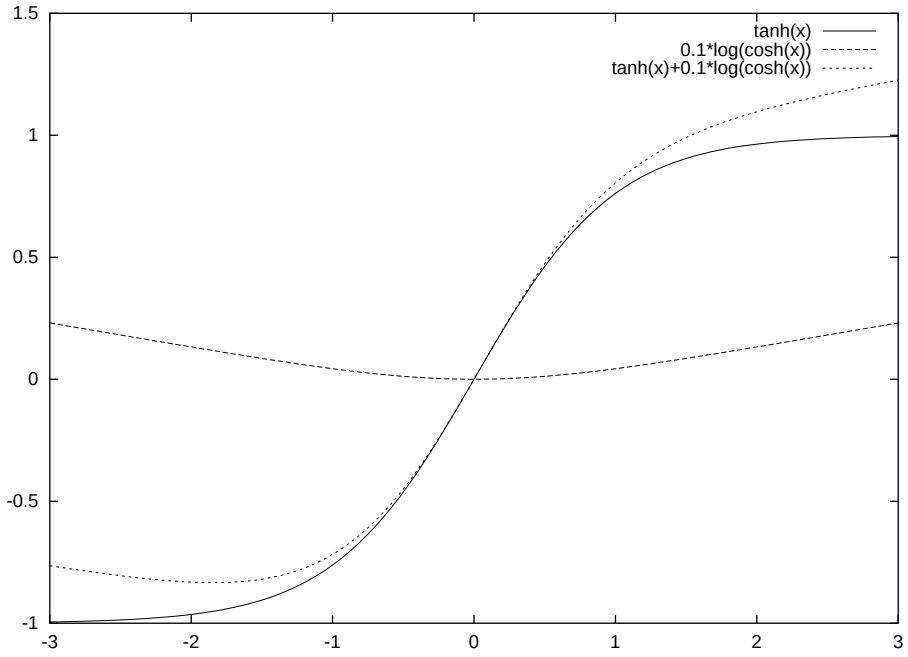


Figure B.2: $\tanh(x) + \frac{1}{10}\log\cosh(x)$

moderated. The activation function proposed was used in the test of LOP ensemble on the protein secondary structure problem (see chapter 11).

Appendix C

Bias-effect/variance-effect

In [44] a general decomposition of any error function can be found. It is assumed that both the target functions and the predictors are outcomes of stochastic variables, and the mean operator used is with regard to both stochastic variables. The predictors are outcomes of the stochastic variable $\underline{\mathbf{F}}$. The expected error is defined as $\langle E(t,f)\rangle_{\underline{\mathbf{F}},\underline{\mathbf{T}},\underline{\mathbf{X}}}$. To simplify matters only the error at a point will be discussed, one dimensional output will be assumed, and the subscript $\underline{\mathbf{F}},\underline{\mathbf{T}}$ will be omitted when obvious, so the error under investigation is $\langle E(t,f)\rangle$.

Variance is defined as $\langle E[f,Sf]\rangle$ and bias is defined as $E[Sy,Sf]$. The symbol S denotes an average operator defined as

Definition 15 (Systematic Mean)
The minimal mean is the result of the operator S working on a stochastic variable $\underline{\mathbf{G}}$ and an error function E defined as

$$Mg = \operatorname*{argmin}_z \langle E(g,z)\rangle_{\underline{\mathbf{G}}}. \tag{C.1}$$

■

The definition of Sf and St are respectively

$$Sf = \operatorname*{argmin}_t \langle E(t,f)\rangle_{\underline{\mathbf{F}}}$$

and

$$St = \operatorname{argmin}_f \langle E(t,f)\rangle_{\underline{\mathbf{T}}}.$$

Bias and variance as defined above do not constitute a general decomposition of the error, and S is generally not the mean operator. To alleviate the first problem the bias-*effect* and variance-*effect* are defined

Definition 16 (Bias Effect)

$$BE(t, Sf) = \langle E[t, Sf] - E[t, St] \rangle \tag{C.2}$$

■

Definition 17 (Variance Effect)

$$VE(t, f) = \langle E[t, f] - E[t, Sf] \rangle. \tag{C.3}$$

■

We arrived at the following general decomposition:

$$\langle E[t, f] \rangle = \text{var}(t) + BE(t, Sf) + VE(t, f), \tag{C.4}$$

where $BE(t, Sf)$ and $VE(t, f)$ are defined above and $\text{var}(t)$ is the intrinsic noise of the target function defined as $\langle E[t, St] \rangle$. It is easy to check that the decomposition is legal, since

$$\begin{aligned}
&\text{var}(t) + BE(t, f) + VE[t, f) = \\
&\langle E[t, St] \rangle + \langle E[t, Sf] \rangle - \langle E[t, St] \rangle + \langle E[t, f] \rangle - \langle E[t, Sf] \rangle = \\
&\langle E[t, f] \rangle.
\end{aligned}$$

The Bias-effect is positive by definition of St. Unfortunately nothing similar can be said about the variance-effect.

If E is the MSE then the decomposition in (C.4) reduces to the well known bias/variance decomposition for the MSE and S reduces to the mean operator. To see the latter note that

$$\frac{\partial}{\partial a} \langle (g - a)^2 \rangle = 2 \langle g - x \rangle = a - \langle g \rangle$$

which implies that $\underset{a}{\text{argmin}} \langle E(g, a) \rangle_{\underline{\mathbf{G}}} = \langle g \rangle$. Now we have:

$$\begin{aligned}
BE(t, f) &= \langle E[t, Sf] - E[t, St] \rangle \\
&= \langle [t - \langle f \rangle]^2 - [t - \langle t \rangle]^2 \rangle \\
&= [\langle t \rangle - \langle f \rangle]^2 \\
&= \text{bias}(\langle f \rangle, \langle t \rangle)
\end{aligned}$$

and similar for

$$VE(t, f) = \langle [f - \langle f \rangle]^2 \rangle = \text{var}(f).$$

Appendix D

Generalized linear models

The family of generalized linear models (GLM) is a set of distributions that encompass some of the commonly used distributions, e.g. the Normal, Poisson, Binomial, Gamma and Inverse Gauss distributions (see [53]). The GLM has been used in machine learning, e.g. [45].

Associated with any distributions is a function that maps the outcome of a stochastic variable to the density or probability of the outcome. A distribution depends on some parameters e.g. the mean and the variance. If the number of unknown parameters is k, we indicate it by a subscript k on GLM_k. The notation GLM is used for GLM_1. The definition of the family of generalized linear models is from "Generalized Linear Models" by McCullagh and Nelder [53]

Definition 18 (Generalized Linear Models)

The members of the family of Generalized Linear Models are defined by a vector of parameters $\vec{\theta} = (\theta_1, \ldots, \theta_n)$*, the parameter* ϕ*, and three functions* a*,* b *and* c*. The parameters* $\vec{\theta}$ *are the* natural parameters *and* ϕ *is called the* dispersion parameter*. All the parameters* $\vec{\theta}$ *are unknown, while the parameter* ϕ *can be either known or unknown. If* ϕ *is known then* $k = n$ *else* $k = n + 1$*. Let* $\underline{\mathbf{Y}} = (\underline{\mathbf{Y}}_1, \ldots, \underline{\mathbf{Y}}_n)$ *be the stochastic variable and let* $\vec{y} = (y_1, \ldots, y_n)$ *be the outcome. The density for a given set of parameters and functions is*

$$p(\vec{y}; \vec{\theta}, \phi) = \exp[\frac{\vec{y} \cdot \vec{\theta} - b(\vec{\theta})}{a(\phi)} + c(y, \phi)]. \tag{D.1}$$

■

The natural parameter θ_i is connected to the marginal mean of $\underline{\mathbf{Y}}_i$ by:

$$\frac{\partial b(\vec{\theta})}{\partial \theta_i} = \langle y_i \rangle_{\underline{\mathbf{Y}}} = \mu_i. \tag{D.2}$$

The variance of $\underline{\mathbf{Y}}_i$ is connected to the dispersion parameter and the natural parameter θ_i by

$$Var(y_i) = \frac{\partial^2 b(\vec{\theta})}{\partial \theta_i \partial \theta_i} a(\phi). \tag{D.3}$$

The covariance of $\underline{\mathbf{Y}}_i$ and $\underline{\mathbf{Y}}_j$ is given by

$$Cov(y_i, y_j) = \frac{\partial^2 b(\vec{\theta})}{\partial \theta_i \partial \theta_i} a(\phi). \tag{D.4}$$

It is given that there exist functions $\theta_i(\mu_i) = \theta_i$, that are the inverse of $\frac{\partial b(\vec{\theta})}{\partial \theta_i}$, so we have $\frac{\partial b(\vec{\theta}(\vec{\mu}))}{\partial \theta_i} = \mu_i$. The functions $\theta_i(\mu_i)$ are the canonical link functions.

The GLM can be reformulated in terms of the mean μ:

$$p(y; \{\mu, \phi\}) = \exp[\frac{y\theta(\mu) - b(\theta(\mu))}{a(\phi)} + c(y, \phi)], \tag{D.5}$$

Besides the constraint $\frac{\partial b(\theta(\mu))}{\partial \theta} = b'(\theta) = \mu$, the density is normalized, so we have

$$\int dy \exp[\frac{y\theta(\mu) - b(\theta(\mu))}{a(\phi)} + c(y, \phi)] = 1$$

The name 'Generalized linear model' refers to that the mean parameter is defined as a linear sum of *covariates* $\{x_1, \ldots, x_p\}$. We have

$$\mu = \sum_i^p \beta_i x_i$$

The covariates can be viewed as input to an estimator (see definition 3 in chapter 2), while $\{\beta_1, \ldots, \beta_p\}$ are the parameters of the estimator.

If ϕ is known, the family GLM_k is a subset of the k-parameter exponential family (see appendix E). In table D.1 the connection between the k-parameter exponential family density in natural form (E.2) and GLM_k density from (D.1) is given.

The GLM_k is a real subset of the exponential family. If the sufficient statistic function T is $T(y) = y^2$ there is no corresponding density for y in the GLM_k. This is the case for the normal distribution if the variance is unknown.

DENSITY IN (D.1)	DENSITY IN (E.2)
$a(\phi)^{-1}\theta_i$	η_i
y_i	$T_i(\vec{y}) = y_i$
$-a(\phi)^{-1}b(\vec{\theta})$	$d_0(\vec{\eta})$
$c(\vec{y}, \phi)$	$S(\vec{y})$

Table D.1: The connection between the family of generalized linear models with known dispersion parameter and the exponential family.

The density in (D.1) is not necessarily a member of the exponential family if the dispersion parameter ϕ is unknown, but if the function $c(\vec{y}, \phi)$ is constrained, the density is a subset of the exponential family. Assume that the c function is given by

$$c(\vec{y}, \phi) = c_1(\phi)c_2(\vec{y}) + c_3(\phi) + c_4(\vec{y}).$$

the GLM_k family is a subset of the k-parameter exponential family. In table D.2 the corresponding function from E.2 and GLM_k with the constraint on c is given.

DENSITY IN (D.1)	DENSITY IN (E.2)
$a(\phi)^{-1}\theta_i$	η_i
$c_2(\phi)^{-1}$	η_k
y_i	$T_i(\vec{y}) = y_i$
$c_2(\vec{y})$	$T_k(\vec{y})$
$a(\phi)^{-1}b(\vec{\theta}) + c_3(\phi)$	$d_0(\vec{\eta})$
$c_4(\vec{y})$	$S(\vec{y})$

Table D.2: The connection between the family of generalized linear models with unknown dispersion parameter and the exponential family.

As mentioned the normal distribution is a member of GLM:

$$\begin{aligned} N(y;\mu,\sigma) &= \frac{1}{\sigma\sqrt{2\pi}}\exp[-\frac{(y-\mu)^2}{2\sigma^2}] \\ &= \exp[\frac{y\mu-\frac{1}{2}\mu^2}{\sigma^2} - (\frac{y^2}{2\sigma^2} + \log(\sigma\sqrt{2\pi}))] \\ &= p(y;\mu,\sigma). \end{aligned} \tag{D.6}$$

The functions a, b, and c are in this case $a(\sigma) = \sigma^2$, $b(\mu) = \frac{1}{2}\mu^2$, and $c(y,\sigma) = -(\frac{y^2}{2\sigma^2} + \log(\sigma\sqrt{2\pi}))$. We get $\langle y\rangle = b'(\mu) = \mu$ and $Var(y) = b''(\mu)a(\sigma) = \sigma^2$ as we should.

Appendix E

Exponential family of distributions

The exponential family of distributions is a large set of distribution, that contains most of the commonly used distributions, among them the Gaussian (Normal), Poisson, Gamma and Binomial distributions. The family also includes most generalized linear models (see appendix D).

We present the general definition in detail. The definition is from [6].

Definition 19 (k-parameter exponential family)

Let $\vec{\omega} = (\omega_1, \ldots, \omega_k)$ be a k dimensional vector of parameters, let $\vec{z}$ be an arbitrary dimensional vector - the outcome of the stochastic variable $\underline{\mathbf{Z}}$, and let c_i, T_i, d, S be functions. A member of the k-parameter exponential family is defined by the density or probability:

$$p(\vec{z};\vec{\omega}) = \exp[\sum_{i=1}^{k} c_i(\vec{\omega})T_i(\vec{z}) + d(\vec{\omega}) + S(\vec{z})] \tag{E.1}$$

The function d is the normalization term given by

$$\exp[-d(\vec{\omega})] = \int_A \exp[\sum_{i=1}^{k} c_i(\vec{\omega})T_i(\vec{z}) + S(\vec{z})],$$

where A is the domain of $\vec{z}$. If the distribution is discrete the integral is replaced by a summation.

The functions c_i are the canonical link *functions. The functions T_i are the* sufficient statistic *functions* ■

If the model is full the density or probability can be rewritten in the *natural form*:

$$p(\vec{z};\vec{\eta}) = \exp[\sum_{i=1}^{k} \eta_i T_i(\vec{z}) + d(\vec{\eta}) + S(\vec{z})] \tag{E.2}$$

We will exclusively use full models, so when we say the exponential family we mean only full models.

We will almost always use members of one parameter exponential family with scalar outcome. It is given by

$$p(z;\omega) = \exp[c(\omega)T(z) + d(\omega) + S(z)]. \tag{E.3}$$

The natural form is

$$p(z;\eta) = \exp[\eta T(z) + d_0(\eta) + S(z)], \tag{E.4}$$

where $d_0(\eta) = d(c^{-1}(\eta))$ if the canonical link c is one-one, else d_0 is found by normalization. A very useful equation that holds for any density or probability function is

$$\langle \frac{\partial \log p(z|\omega)}{\partial \omega} \rangle = 0. \tag{E.5}$$

The only requirement is that integration and differentiation can be interchanged.

Proof:

$$\begin{aligned}
\int dz p(z;\omega) &= 1 \Leftrightarrow \\
\int dz \frac{\partial p(z;\omega)}{\partial \omega} &= 0 \Leftrightarrow \\
\int dz p(z;\omega) \frac{\frac{\partial p(z;\omega)}{\partial \omega}}{p(z;\omega)} &= 0 \Leftrightarrow \\
\langle \frac{\frac{\partial p(z;\omega)}{\partial \omega}}{p(z;\omega)} \rangle &= 0 \Leftrightarrow \\
\langle \frac{\partial \log p(z;\omega)}{\partial \omega} \rangle &= 0
\end{aligned}$$

By using E.5 it can be shown that

$$-\frac{\partial d_0(\eta)}{\partial \eta} = \langle T(z) \rangle_{\underline{\mathbf{Z}}}.$$

and

$$-\frac{\partial^2 d_0(\eta)}{\partial \eta \partial \eta} = \mathrm{var}(T(z))_{\underline{\mathbf{Z}}}.$$

Below is a list of some of the common distributions in the exponential family.

E.1 Gaussian distribution $N(\mu, \sigma^2)$

The Gaussian distribution is the most well-known distribution, therefore also called the Normal distribution. The domain of the outcome of the distribution is the entire real line ($\Re$).

$$p(z|\mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp[-\frac{(z-\mu)^2}{2\sigma^2}],$$

where $\mu \in \Re$ is the mean parameter and $\sigma > 0$ is the standard deviation parameter. The variance is σ^2. A Gaussian distribution is denoted $N(\mu, \sigma^2)$.

E.2 Poisson distribution $P(\lambda)$

The Poisson distribution is defined for positive discrete outcome. There is only one parameter λ, which is both variance and mean parameter. The density function is given by

$$P(x; \lambda) = \begin{cases} e^{-\lambda}\frac{\lambda^x}{x!} & x \in \mathcal{Z}_0 \\ 0 & \text{else} \end{cases}$$

A Poisson distribution is denoted $P(\lambda)$.

E.3 Gamma distribution $\Gamma(\nu, \lambda)$

The domain of the outcome of the gamma distribution is the positive real line. The density is given by

$$p(x; \nu, \lambda) = \begin{cases} \frac{\lambda^\nu}{\Gamma(\nu)} x^{\nu-1} e^{-\lambda x} & x \geq 0 \\ 0 & \text{else} \end{cases}$$

Both parameters are greater than zero. The mean is defined by ν/λ and the variance is defined by ν/λ^2. A Gamma distribution is denoted $\Gamma(\nu, \lambda)$.

E.4 Binomial distribution $B(n, \pi)$

The domain of the outcome of the Binomial distribution is $\{0, \ldots, n\}$. So the domain is discrete and both upper and lower bounded. The density of the Binomial distribution is given by

$$P(x; n, \pi) = \begin{cases} \binom{n}{nx} \pi^{nx} (1-\pi)^{n(1-x)} & nx \in \{0, \ldots, n\} \\ 0 & \text{else} \end{cases}$$

The mean is $n\pi$ and the variance is $n\pi(1-\pi)$. A Binomial distribution is denoted $B(n,\pi)$.

E.5 Inverse Gauss distribution

The domain of the outcome of the inverse Gauss distribution is the positive real line. The density is given by

$$p(z|\chi,\psi) = \begin{cases} \exp[\frac{1}{2}\log\frac{\chi}{\pi} + 2\sqrt{\chi\psi} - \chi z - \frac{\psi}{z} - \frac{1}{2}\log z] & z > 0 \\ 0 & \text{else} \end{cases}$$

The mean is $\sqrt{\chi}/\sqrt{\psi}$ and the variance is $\sqrt{\chi}/\sqrt{\psi^3}$.

E.6 Beta distribution $\beta(r,s)$

The domain of the outcome of the Beta distribution is $]0;1[$, so the domain is continuous and both upper and lower bound. The density is given by

$$p(z|r,s) = \begin{cases} \exp[(r-1)\log z(s-1)\log(1-z) + \log\frac{\Gamma(r+s)}{\Gamma(r)\Gamma(s)}] & 0 < z < 1 \\ 0 & \text{else} \end{cases}$$

Both parameters r and s are positive. The mean is $r/(r+s)$ and the variance is $rs/[(r+s)^2(r+s+1)]$.

E.7 Inverted Gamma distribution

The domain of the outcome of the inverted Gamma distribution is the positive positive real line. The density is given by

$$p(z|a,n) = \begin{cases} \exp[-n\log z - \frac{a}{z} + (n-1)\log(a) - \log\Gamma(n-1)] & z > 0 \\ 0 & \text{else} \end{cases}$$

Both the parameters a and n are positive.

Appendix F

Gradient descent for deviance error functions

In section 5.4 a family of error function is presented, that it is based on the deviance of an one-parameter density from the exponential family of distributions. The general form is

$$E(f,t) = [c(t) - c(f)]T(t) + d(t) - d(f)],$$

where d obeys $d'(x) = -c'(x)T(x)$. In table F.1 is an overview of deviance error functions derived from some of the commonly used densities (see appendix E). All error functions have non-linear average predictor, except the Normal error function, because the corresponding densities have non-linear sufficient statistics. As noted in section 5.8 all error functions have a transposed error function where the canonical links and sufficient statistics are interchanged. Since the error functions in table F.1 have linear canonical links, the transposed error functions of the error functions in table F.1 have linear average predictors. The results in this appendix hold for the transposed error function, only the sufficient statistics and the canonical links are interchanged.

The gradient of the error functions with regard to the predictor f must be found in order to use the gradient descent optimization method (see appendix I) on the error functions. The general result is

$$\frac{\partial E(f,t)}{\partial f} = [T(f) - T(t)]c'(f)$$

The corresponding gradient for the transposed error functions is

$$\frac{\partial E_T(f,t)}{\partial f} = [c(f) - c(t)]T'(f)$$

DISTRIBUTION	ERROR FUNCTION	DOMAIN
Normal(μ, σ)	$\frac{1}{2}(f-t)^2$	$]-\infty;\infty[$
Poisson(μ)	$[f-t]+t\log\frac{t}{f}$	$[0;\infty[$
Binomial(μ, k)	$t\log\frac{t}{f}+(1-t)\log\frac{1-t}{1-f}$	$[0;1]$
Gamma(ν, μ)	$(\frac{t}{f}-1)+\log\frac{f}{t}$	$[0;\infty[$
Inv. Gauss(μ, θ)	$(f-t)^2/(f^2t)$	$]0;\infty[$

Table F.1: Error functions with linear sufficient statistics.

From appendix I we have that the intended shift (Δf) of the predictor is proportional to the negative gradient, so we have $\Delta f \propto -\frac{\partial E(f,t)}{\partial f}$. The values of Δf for the error functions in table F.1 and their transposed counterparts are particularly simple and given by

$$\Delta f = [t-f]c'(f)$$

$$\Delta f_T = [t-f]T'(f)$$

The gradient for the error functions in table F.1 can be found in table F.2.

DISTRIBUTION	INTENDED SHIFT (Δf)
Normal(μ, σ)	$t-f$
Poisson(μ)	$\frac{t}{f}-1$
Binomial(μ, k)	$\frac{t}{f}-\frac{1-t}{1-f}$
Gamma(ν, μ)	$[\frac{t}{f}-1]\frac{1}{f}$
Inv. Gauss(μ, θ)	$[\frac{t}{f}-1]\frac{1}{f^2}$

Table F.2: Gradient of error functions with non-linear average predictor.

If the predictor is a neural network the gradient descent method becomes back propagation (see section 6.2). The domain of the output of a neural network is the entire real line, while the domain of the output of the

predictors for some error functions are limited (see table F.1), e.g. for the Poisson error function the predictor can only take positive values. In order to amend this problem a post-processing function is introduced. Let the output of the neural network be nn and the post-processing function be pp we have $f = pp(nn)$. The intended shift of the neural network (Δnn) is then connected to the intended shift of the predictor by

$$\Delta nn = \Delta f \cdot \frac{\partial f}{\partial nn} = \Delta f \cdot pp'(nn) \tag{F.1}$$

By choosing a suitable post-processing function the problem with the domains of f and nn can be avoided and the expression for the intended shift of the neural network can be simplified. In table F.3 suitable post-processing functions for the error functions in F.1 are given together with the derivative of the post-processing function expressed both in terms of f and nn.

DISTRIBUTION	$pp(nn)$	$pp'(nn)$ **I**	$pp'(pp^{-1}(f))$ **II**
Normal(μ, σ)	nn	1	1
Poisson(μ)	e^{nn}	e^{nn}	f
Binomial(μ, k)	$\frac{1}{e^{-nn}+1}$	$\frac{1}{e^{-nn}+e^{nn}+2}$	$f(1-f)$
Gamma(ν, μ)	e^{nn}	e^{nn}	f
Inv. Gauss(μ, θ)	e^{nn}	e^{nn}	f

Table F.3: Gradient of error functions with non-linear average predictor.

The derivative of the post-processing function for the error function in table F.1 can all be expressed succinctly in terms of the output of the predictor f.

With the post-processing functions in F.3 the intended shift of the neural network is greatly simplified (see table F.4). Let us use the Binomial error function as an example. The post-processing function is given by $f = \frac{1}{e^{-nn}+1}$, and the derivative of f with regard to nn is $\frac{1}{e^{-nn}+e^{nn}+2}$. Using that $1 - f = \frac{1}{e^{nn}+1}$ and straightforward manipulation yields that the derivative of f with regard to nn expressed in terms of f is $f(1-f)$. From (F.1)

we find that the intended shift of the neural network is

$$\begin{aligned}\Delta nn &= \Delta f pp'(nn) \\ &= [\frac{t}{f} - \frac{1-t}{1-f}]f(1-f) \\ &= t - tf - f + tf \\ &= t - f.\end{aligned}$$

With the chosen post-processing function the intended shift for the neural network becomes the same as for the MSE, which is remarkable.

DISTRIBUTION	Δf	Δnn
Normal(μ, σ)	$t - f$	$t - f$
Poisson(μ)	$\frac{t}{f} - 1$	$t - f$
Binomial(μ, k)	$\frac{t}{f} - \frac{1-t}{1-f}$	$t - f$
Gamma(ν, μ)	$[\frac{t}{f} - 1]\frac{1}{f}$	$\frac{t}{f} - 1$
Inv. Gauss(μ, θ)	$[\frac{t}{f} - 1]\frac{1}{f^2}$	$[\frac{t}{f} - 1]\frac{1}{f}$

Table F.4: Gradient of error functions with non-linear average predictor.

In table F.4 we see that while the expressions for Δf are different, the expressions for Δnn are all on the form $[t - f]\frac{1}{f^n}$, where n is zero for the Normal, Poisson, and Binomial error function, while n is one for the Gamma error function, and two for the Inverse Gauss error function.

Appendix G

An alternative approximation of the Faculty and Gamma functions

Stirling's formula is an approximation of the faculty $n!$. In the infinite limit the approximation is very good, but for n close to zero it becomes very poor. In the following we will treat $n!$ as a continuous function (so we are really examine $\Gamma(n+1)$). A well-known bound on $n!$ is

$$\sqrt{2\pi n}\left(\frac{n}{e}\right)^n \leq n! \leq \sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{\frac{1}{8n}} \tag{G.1}$$

An alternative approximation of $n!$ is

$$n! \approx \sqrt{2\pi}\left(\frac{n+\frac{1}{2}}{e}\right)^{n+\frac{1}{2}} \tag{G.2}$$

For various reasons we will investigate the logarithm of the three approximations. Let the logarithm of the lower bound in (G.1) be denoted $A(n)$, the logarithm of the upper bound be denoted $B(n)$, and the logarithm of the approximation in (G.2) be denoted $C(n)$. In figure G.1 the three approximations and the continuous version of the faculty is plotted.

We will show that $A(n) \leq C(n) \leq B(n)$ by investigating $C(n) - A(n)$ and $B(n) - A(n)$:

$$C(n) - A(n) = (n+\frac{1}{2})\log\frac{n+\frac{1}{2}}{n} - \frac{1}{2}. \tag{G.3}$$

$$B(n) - C(n) = \frac{1}{2} + \frac{1}{8n} - (n+\frac{1}{2})\log\frac{n+\frac{1}{2}}{n}. \tag{G.4}$$

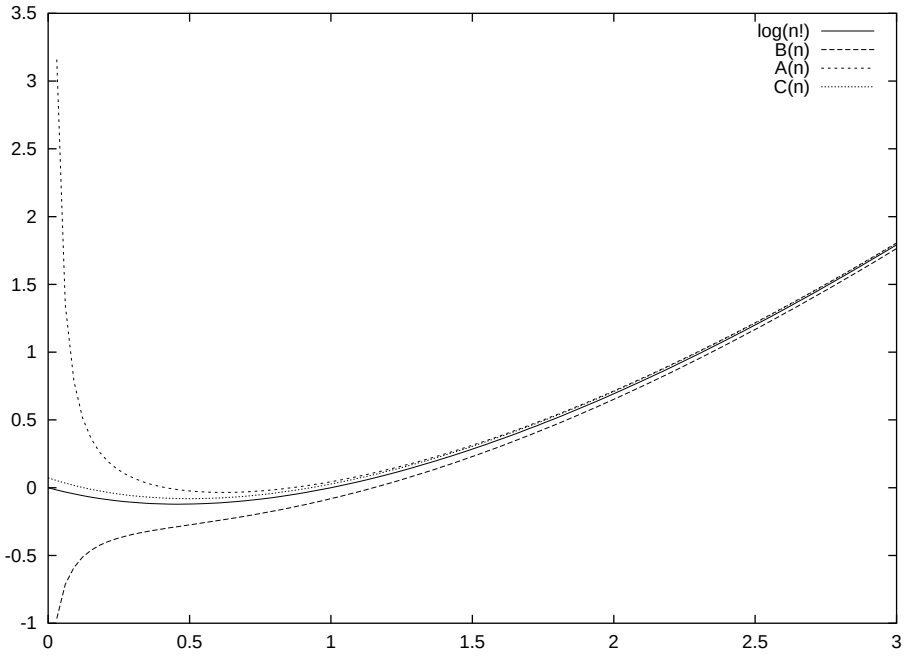


Figure G.1: Comparison of the continuous version of $log(n!)$, the upper Stirling log bound $A(n)$, lower log bound $B(n)$, and the alternative approximation of $log(n!) : C(n)$

To show $A(n) - C(n) \geq 0$ (G.3) it must be shown that

$$(n + \frac{1}{2}) \log \frac{n + \frac{1}{2}}{n} \geq \frac{1}{2}.$$

Using $\log(1 + x) \approx x$ yields that $(n + \frac{1}{2}) \log \frac{n+\frac{1}{2}}{n} \to \frac{1}{2}$ for $n \to \infty$. Furthermore, the first derivative of $(n+\frac{1}{2}) \log \frac{n+\frac{1}{2}}{n}$ is negative so $(n+\frac{1}{2}) \log \frac{n+\frac{1}{2}}{n} \geq \frac{1}{2}$.

To show $B(n) - C(n) \geq 0$ (G.4) it must be shown that

$$\frac{1}{2} + \frac{1}{8n} \geq (n + \frac{1}{2}) \log \frac{n + \frac{1}{2}}{n}.$$

The first derivative of $B(n) - C(n)$ is always negative, meaning that the difference is decreasing. For $n = 1$ the value of $B(n) - C(n)$ is greater than zero, so it must be checked that $B(n) - C(n) \geq 0$ for $n \to \infty$. Using $\log(1 + x) \approx x - \frac{x^2}{2}$ yields $(n + \frac{1}{2}) \log \frac{n+\frac{1}{2}}{n} \approx \frac{1}{2} + \frac{1}{8n} - \frac{1}{16n^2}$. Since $\frac{1}{2} + \frac{1}{8n} \geq \frac{1}{2} + \frac{1}{8n} - \frac{1}{16n^2}$ the difference is positive also in the limit $n \to \infty$.

We conclude that $A(n) \leq C(n) \leq B(n)$ and thereby

$$\sqrt{2\pi n}\left(\frac{n}{e}\right)^n \leq \sqrt{2\pi}\left(\frac{n + \frac{1}{2}}{e}\right)^{n+\frac{1}{2}} \leq \sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{\frac{1}{8n}} \qquad \text{(G.5)}$$

For $n \to 0$ the approximations in (G.2) is much better. The real value of $n!$ is one. The two bounds diverge, while the approximation in (G.2) is approximately 1.075. The relative error decreases for greater values of n.

There are even better approximation of $n!$ than $\sqrt{2\pi}(\frac{n+\frac{1}{2}}{e})^{(n+\frac{1}{2})}$:

$$n! \approx \sqrt{2\pi} n^{n+\frac{1}{2}} e^{-n+\mu(n)},$$

where

$$\mu(n) = \frac{1}{12n} - \frac{1}{360n^3} + \frac{1}{1260n^5},$$

but this approximation is very cumbersome and $\sqrt{2\pi}(\frac{n+\frac{1}{2}}{e})^{(n+\frac{1}{2})}$ is a much better approximation for $n < 0.4$

As mentioned we have also presented an approximation of the $\Gamma(x)$. It is given by

$$\Gamma(x) \approx \sqrt{2\pi}\left(\frac{x-\frac{1}{2}}{e}\right)^{x-\frac{1}{2}} \tag{G.6}$$

An important function that is connected to the $\Gamma(x)$ is the diGamma function. It is given by

$$diGamma(x) = \frac{\partial \log \Gamma(x)}{\partial x}. \tag{G.7}$$

The approximation of $\Gamma(x)$ gives an approximation for the diGamma function:

$$diGamma(x) = \frac{\partial \log \Gamma(x)}{\partial x} \approx \log(x - \frac{1}{2}).$$

Appendix H

Probability of sampling an example k times in Bagging

A Bagging training set is sampled with replacement from the original example set. Let the size of the example set be n. A particular example can be in the training set between zero and n times. The probability of sampling an example k times is binomial distributed with probability parameter $\frac{1}{n}$:

$$P(k) = \binom{n}{k} \left(\frac{1}{n}\right)^k \left(1 - \frac{1}{n}\right)^{(n-k)}$$

The probability $P(0)$ is equal to $\left(1 - \frac{1}{n}\right)^n$. For large n this is approximately $e^{-1} \approx 0.3679$. It is easy to show that for $k \neq 0$ we have

$$P(k) = \frac{1}{k} \frac{n-k}{n-1} P(k-1).$$

If k is much smaller than n, the factor $\frac{n-k}{n-1}$ is very close to one, so a very good approximation of the probability of sampling an training example k times is

$$P(k) = e^{-1} \frac{1}{k!}.$$

The first probabilities for $k = \{0, \ldots, 8\}$ are given in table H.1.

k	$P(k)$
0	0.3679
1	0.3679
2	0.1839
3	0.0613
4	0.0153
5	0.0031
6	0.0005
7	0.0001
8	0.0000

Table H.1: The probability that an example is sampled k times.

The probabilities are very accurate for $n > 1000$ and $k < 10$.

Appendix I

Gradient descent

A well known optimization method is the gradient descent method. It is used to minimize a function. Let h be the function we want to minimize. It depends some parameters $\vec{w} = \{w_1, \ldots, w_n\}$. In gradient descent method the parameters are iteratively changed in order to find the minimum of the function. So we want to know what happens if we change the weights by a small amount $d\vec{w}$. A first order Taylor expansion of the error function $E(\vec{w})$ around $\vec{w}$ gives

$$h(\vec{w} + d\vec{w}) \approx h(\vec{w}) + \frac{\partial h(\vec{w})}{\partial \vec{w}} \cdot d\vec{w}$$

The vector $\frac{\partial h(\vec{w})}{\partial \vec{w}} = \{\frac{\partial h(\vec{w})}{\partial w_1}, \ldots, \frac{\partial h(\vec{w})}{\partial w_n}\}$ is the gradient. Since we want $h(\vec{w} + d\vec{w}) < h(\vec{w})$ we must have $\frac{\partial h(\vec{w})}{\partial \vec{w}} \cdot d\vec{w} < 0$. This is guaranteed if $d\vec{w} = -\frac{\partial h(\vec{w})}{\partial \vec{w}}$. So the gradient descent method has the following update rule:

$$\vec{w}' = \vec{w} - r\frac{\partial h(\vec{w})}{\partial \vec{w}},$$

where r is called the learning rate.

Appendix J

Logarithmic opinion pool ensemble in a statistical mechanics setting

A number of similarities exists between a logarithmic opinion pool (LOP) ensemble (see section 7.1.4) and a *Canonical Ensembles*, an area inside statistical mechanics. (see [70, 61] for an explanation of canonical ensembles). Very shortly put a canonical ensembles is an abstract group of the same physical system, where each "mental copy" is characterized by a state r with a corresponding energy E_r.

In a canonical ensemble the probability of one of the systems being in state r is

$$P_r = \frac{1}{Z} e^{-\beta E_r},$$

where E_r is the energy of the system, β is the reciprocal of the temperature, and $Z = \sum_s e^{-\beta E_s}$ is a normalization constant. This is very similar to the LOP combination rule in (7.1)

$$F^c = \frac{1}{Z} \exp \sum_i \alpha_i \log(f_i^c),$$

where $Z = \sum_c \exp \sum_i \alpha_i \log(f_i^c)$. Setting $F^c = P_r$ gives

$$-\beta E_r = \sum_i \alpha_i \log(f_i^c) \tag{J.1}$$

The reciprocal temperature β has no analog. Setting $\beta = 1$ yields

$$E_r = \sum_i \alpha_i \log \frac{1}{f_i^c}.$$

The state r of a system will be viewed as the class c. Note that $E_c \geq 0$ since $f_i^c \leq 1$. Let us assume that the energy E_r is the sum of some weakly interacting particles with energy E_r^γ yielding

$$E_r = \sum_\gamma E_r^\gamma = \sum_i \alpha_i \log \frac{1}{f_i^c}.$$

To stretch the analogy further γ is set to correspond to i yielding

$$E_c^i = \alpha_i \log \frac{1}{f_i^c}, \tag{J.2}$$

which is the energy of one particle in one system in the canonical ensemble. Equation J.2 can be rewritten as

$$f_i^c = \exp[-\alpha_i^{-1} E_c^i].$$

In a canonical ensemble there are some values of interest. Among them are the mean energy $\bar{E}$, the entropy S, and the free energy F, but before we look into them, we examine the indirectly important term $\log Z$. By definition

$$\log Z = \log \frac{\exp[\sum_i \alpha_i \log f_i^c]}{F^c}, \forall c : F^c > 0.$$

To express the "for all" constraint the terms $\log \exp[\sum_i \alpha_i \log f_i^c]/F^c$ can be summed (over index c) weighted by weights that sum to one, and are zero whenever F^c is zero. The only functions that always obey this are the F^c's, so we have:

$$\log Z = \sum_c F^c \log Z = \sum_c F^c \log \frac{\exp[\sum_i \alpha_i \log f_i^c]}{F^c} = -A,$$

where A is the ambiguity. So $A = -\log Z$. Furthermore, the Helmholtz free energy is

$$H = -\beta^{-1} \log Z = A.$$

The mean energy is defined as

$$\bar{E} = \sum_r P_r E_r.$$

Using the analog between a canonical ensemble and a LOP ensemble we get

$$\bar{E} = -\sum_c F^c \sum_i \alpha_i \log f_i^c = A - \sum_c F^c \log F^c.$$

The entropy is defined as

$$S \equiv k(\log Z + \beta \bar{E}).$$

Setting $k = 1$ we get

$$S = -A + A - \sum_c F^c \log F^c = -\sum_c F^c \log F^c.$$

Finally, we have the equality $H = \bar{E} - TS$. With $T = 1$ we have

$$A = \bar{E} - S.$$

So the ambiguity of a LOP ensemble can be interpreted as the difference between the mean energy and the entropy in the corresponding canonical ensemble. Note that all three terms A, $\bar{E}$, and S are positive or zero, so we have $\bar{E} \geq S$. In thermodynamics it is well-known that the entropy of an isolated system is irreducible. It is not the same for a LOP ensemble, since we have "control" over the ensemble members and combined predictor through our choice of meta machine learning method.

The decomposition of the ambiguity can be used. It provides us with extra tools for increasing ambiguity, since we can either increase the mean energy and/or minimize the entropy.

J.1 Temperature in LOP ensemble

In (J.1) the reciprocal temperature β was set to one. It is tempting to investigate whether it is possible to introduce a temperature in a classification ensemble. We would then have the combination rule

$$F^c = \frac{1}{Z} \exp[\beta \sum_i \alpha_i \log f_i^c], \tag{J.3}$$

where $Z = \sum_c \exp[\beta \sum_i \alpha_i \log f_i^c]$. Let us first investigate what such a reciprocal temperature does: Assume two classes, and set $a = \sum_i \alpha_i \log f_i^1$ and $b = \sum_i \alpha_i \log f_i^2$, we have

$$F^1 = \frac{e^{-\beta a}}{e^{-\beta a} + e^{-\beta b}} = \frac{1}{1 + e^{-\beta(b-a)}}.$$

If $\beta \to 0$ (corresponding to infinite high temperature) then $F^1 \to \frac{1}{2}$, if $\beta \to \infty$ and $b > a$ then $F^1 \to 1$, if $\beta \to \infty$ and $b < a$ then $F^1 \to 0$. So a high temperature makes the combined predictor more "uncertain", while a low temperature forces the combined predictor to decide between the classes. Unfortunately, the decomposition of the error in section 7.1.4 does not hold for $\beta \neq 1$. To see this note that the decomposition is based on to equalities

$$K(t, F^c) = \langle K(t, f_i^c) \rangle_{\underline{\mathbf{F}}} + \log Z,$$

and

$$\log Z = -\langle K(F^c, f_i^c)\rangle_{\underline{\mathbf{F}}}.$$

Let us look at the latter equality first. With the combination rule in equation J.3, we have

$$\log Z = \sum_i \alpha_i \sum_c F^c \log \frac{(f_i^c)^\beta}{F^c} \equiv -\langle K(F^c, f_i^c)\rangle_{\underline{\mathbf{F}}},$$

so

$$K(f, f_i^c) = \sum_c F^c \log \frac{F^c}{(f_i^c)^\beta},$$

this yields

$$\begin{aligned} K(f, F^c) &= \sum_i \alpha_i \sum_c f_i \log \frac{f_i}{(f_i^c)^{\beta^2}} + \beta \log Z \\ &= \langle K(f, (f_i^c)^\beta)\rangle + \beta \log Z, \end{aligned}$$

which is not the same as $\langle K(f, f_i^c)\rangle_{\underline{\mathbf{F}}} + \log Z$ unless $\beta = 1$.

Appendix K

Comparing large test runs

In many empirical test runs a great deal of data is collected (see e.g. tables 8.2, 8.3, and 9.2). It is difficult to get a quick overview, and the main point might be lost in details. We present a method for simplifying the results, that is applicable in many cases. It will discuss as a comparison of multiple machine learning methods on multiple example sets, but is easily generalized to other cases.

The principle is to find how often a given method would yield the lowest test error averaged over the example sets.

We have M machine learning methods $\{L_1, \ldots, L_M\}$ and N example sets $\{T_1, \ldots, T_N\}$. All methods are trained on all examples sets a number of times. The test error for each training run is used to calculate the mean and deviation of the test error of each method on each example set. We have a matrix of size $M \times N$ of pairs of mean and deviation (e_{ij}, σ_{ij}), from the stochastic variables $\underline{\mathbf{X}}_{ij}$, where i is the method index and j is the example set method. Without any other data recorded the distribution of the test runs cannot be found, so it is assumed that the test error is Gamma distributed for a training run. We have $M \times N$ Gamma densities $p(e; e_{ij}, \sigma_{ij}) = p(e)_{ij}$ and $M \times N$ distributions $D(e)_{ij}$. At a specific error e the the distribution $D(e)_{ij}$ (the probability mass) is how likely it is that method i on set j has yielded an error less than or equal to e, i.e. $P(\underline{\mathbf{X}}_{ij} \leq e) = D(e)_{ij}$. The probability $P(\underline{\mathbf{X}}_{ij} > e$ is $1 - D(e)_{ij}$. The test runs are independent, so the stochastic variables are also independent. Therefore the probability that a group of methods yields a result larger than e is given by the product. Let a group be all methods except the i'th method for a given example set j. We have

$$P(\underline{\mathbf{X}}_{1j} > e \ldots \underline{\mathbf{X}}_{i-1j} > e \bigwedge \ldots \underline{\mathbf{X}}_{i+1j} > e \ldots \underline{\mathbf{X}}_{Mj} > e) = \prod_{k \neq i}^{M} (1 - D(e)_{kj}).$$

By averaging the probability above over all possible errors with regard to the method i on set j the probability that method i yields the lowest error is found:

$$P(lowest)_{ij} = \langle \prod_{k \neq i}^{M} (1 - D(e)_{kj}) \rangle_{\underline{\mathbf{X}}_{ij}}.$$

Since no example set is assumed more significant than the other and they are assumed to be independent, the overall probability that method i yields the lowest error is

$$P(lowest)_i = \frac{1}{N} \sum_{j}^{N} P(lowest)_{ij}$$

Instead of the analytic approach above one can use an iterative approximation: Sample an error from each of the distribution $D(e)_{ij}$ and find the method that yields the lowest error. Do this a large number of times while keeping track of how many times a method yields the lowest error. Let the numbers be K_{ij} then $P(lowest)_{ij}$ is approximated by

$$P(lowest)_{ij} \approx \frac{K_{ij}}{\sum_k^M K_{kj}}.$$

Index

Bibliography

[1] Avnimelech, R., and Intrator, N. Boosted mixture of experts: An ensemble learning scheme. *Neural Computation 11*, 2 (1999), 483–497.

[2] Avnimelech, R., and Intrator, N. Boosting regression estimators. *Neural Computation 11*, 2 (1999), 499–520.

[3] Baldi, P., and Brunak, S. *Bioinformatics - The Machine Learning Approach.* MIT Press, Cambridge MA, 1998.

[4] Battiti, R. Democracy in neural nets: Voting schemes for classification. *Neural Networks 7*, 4 (1994), 691–707.

[5] Bauer, E., and Kohavi, R. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning* (1999).

[6] Bickel, P. J., and Doksum, K. A. *MATHEMATICAL STATISTICS: Basic Ideas and Selected Topics.* Holden-Day, Inc., 500 Sansome Street, San Fransisco, Ca., USA., 1977.

[7] Bishop, C. M. *Neural Networks for Pattern Recognition.* Clarendon Press, Oxford, 1995.

[8] Bishop, C. M. Real-time control of a tokamak plasma using neural networks. In *Advances in Neural Information Processing Systems* (1995), G. Tesauro, D. Touretzky, and T. Leen, Eds., vol. 7, The MIT Press, pp. 1007–1014.

[9] Breiman, L. Bagging Predictors. *Machine Learning 24* (1996), 123–140.

[10] Breiman, L. Bias, Variance, and Arcing Classifiers. Tech. Rep. 460, Statistics Department, University of California, Berkeley, CA 94720, Apr. 1996. Anonymous FTP: ftp://ftp.stat.berkeley.edu/pub/users/breiman/old/arcall.ps.Z.

[11] Breiman, L. Arcing the edge. Tech. rep., Statistics Department, University of California, Berkley CA. 94720, 1998. Anonymous FTP: ftp://ftp.stat.berkeley.edu/pub/users/breiman/arcing-the-edge.ps.Z.

[12] Breiman, L. Using adaptive bagging to debias regressions. Tech. rep., Statistics Department, University of California at Berkeley, Feb. 1999. Anonymous ftp: ftp://ftp.stat.berkeley.edu/pub/users/breiman/adaptbag99.ps.Z.

[13] Breiman, L., Friedman, J. H., and ans C J Stone, R. A. O. *Classification and Regression Trees.* Wadsworth International Group, 1984.

[14] Bridle, J. Probabilistic Interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. *Neurocomputing: Algorithms, Architectures and Applications* (1990).

[15] Carney, J. G., and Cunningham, P. Tuning diversity in bagged neural network ensembles. Tech. rep., Department of Computer Science, University of Dublin, Trinity College, Ireland, Aug. 1999. Anonymous FTP: ftp://ftp.cs.tcd.ie/pub/tech-reports/reports.99/TCD-CS-1999-44.ps.

[16] Chen, K., and Chi, H. A method of combining probabilistic classifiers through soft competition on different features sets. *Neurocomputing - An International Journal* (1998).

[17] Cleary, J. G., and Trig, L. E. K*: An instance-based learner using an entropic distance measure. Tech. rep., Department of Computer Science, University of Waikato, New Zealand, 1997.

[18] Cortes, C., and Vapnik, V. Support-vector networks. *Machine Learning 20* (1995), 273.

[19] Creighton, T. E. *PROTEINS. Structures and Molecular Properties.* W. H. Freeman and Company, New York, 1992.

[20] Drucker, H. Improving Regressors using Boosting Techniques. In *Machine Learning: Proceedings of the Fourteenth International Conference* (1997), j. Douglas H. Fisher, Ed.

[21] Drucker, H., Cortes, C., Jackel, L. D., LeCun, Y., and Vapnik, V. Boosting and other ensemble methods. *Neural Computation 6*, 6 (1994), 1289–1301.

[22] Duffy, N., and Helmbold, D. A geometric approach to leveraging weak learners. *EuroColt 99* (1999).

[23] Friedman, J., Hastie, T., and Tibshirani, R. Additive logistic regression: a statistical view of boosting. Tech. rep., Department of

Statistics, Sequoia Hall, Stanford University, Stanford California 94305, 1998. Link: http://www-stat.stanford.edu/~jhf/ftp/boost.ps.

[24] FRIEDMAN, J. H. Multivariate adaptive regression splines. *Annals of Statistics*, 1 (1991).

[25] FRIEDMAN, J. H. On bias, variance, 0/1-loss and the curse-of-dimensionality. Tech. rep., Department of Statistics and Stanford Linear Accelerator Center, Stanford University, 1996. Link: http://www-stat.stanford.edu/~jhf/ftp/curse.ps.Z.

[26] FRIEDMAN, J. H. Greedy function approximation: A gradient boosting machine. Tech. rep., Department of Statistics and Stanford Linear Accelerator Center Stanford University Stanford, CA 94305, Feb. 1999. Link: http://www-stat.stanford.edu/~jhf/ftp/trebst.ps.

[27] GEMAN, S., BIENENSTOCK, E., AND DOURSAT, R. Neural networks and the bias/variance dilemma. *Neural Computation 4*, 1 (1992), 1–58.

[28] HAMPSHIRE, J. B., AND WAIBEL, A. The meta-Pi network - building distributed knowledge representations for robust multisource pattern-recognition. *Pattern Analyses and Machine Intelligence 14*, 7 (1992), 751–769.

[29] HANSEN, J. V. Optimal brain construction. Tech. rep., DAIMI, 1995. http://www.daimi.au.dk/~vogdrup/obc.ps.

[30] HANSEN, J. V. Studies of optimal strategies for war & peace using neural networks, genetic programming, and genetic algorithms. Tech. rep., DAIMI, 1996. http://www.daimi.au.dk/~vogdrup/war.ps.

[31] HANSEN, J. V. Progress report: Ensemble methods in connection with neural networks. Tech. rep., Department of Computer Science, University of Aarhus, Dec. 1997. http://www.daimi.au.dk/~vogdrup/progressreport.ps.

[32] HANSEN, J. V. Combining predictors: Some old methods and a new method. In *JCIS '98 Proceedings* (1998), G. Georgiou, Ed., Association For Intelligent Machinery, Inc., pp. 12–16.

[33] HANSEN, J. V. Combining predictors: Comparison of five meta machine learning methods. *Information Science, an International Journal* (1999).

[34] HANSEN, J. V. The superiority of simplicity. comparison of four meta machine learning methods. In *JCIS 2000 Proceedings* (Feb. 2000), P. P. Wang, Ed., Association For Intelligent Machinery, Inc., pp. 899–903. Link: http://www.daimi.au.dk/~vogdrup/cin00.ps.

[35] Hansen, J. V., and Heskes, T. Accepted for presentation at the 15th international conference on pattern recognition: General bias/variance decomposition with target independent variance of error functions derived from the exponential family of distributions. Tech. rep., Department of Computer Science, University of Aarhus, Denmark, Sept. 2000. Link: http://www.daimi.au.dk/~vogdrup/biasvar.ps.

[36] Hansen, J. V., and Krogh, A. A general method for combining predictors tested on protein secondary structure prediction. In *Proceedings of Artificial Neural Networks in Medicine and Biology* (Göteborg, Sweden, May 2000), H. Malmgren, M. Borga, and L. Niklasson, Eds., Springer-Verlag, London, pp. 259–264. Link: http://www.daimi.au.dk/~vogdrup/annimabprocrc.ps.

[37] Hertz, J., Krogh, A., and Palmer, R. G. *Introduction to the Theory of Neural Computation.* Addison-Wesley Publishing Company, 350 Bridge Parkway, Redwood City, CA 94065, 1991.

[38] Heskes, T. Balancing between bagging and bumping. In *Advances in Neural Information Processing Systems* (1997), M. C. Mozer, M. I. Jordan, and T. Petsche, Eds., vol. 9, The MIT Press, p. 466.

[39] Heskes, T. Bias/variance decompositions for likelihood-based estimators. *Neural Computation 10*, 6 (1998), 1425–1433.

[40] Heskes, T. Selecting weighting factors in logarithmic opinion pools. In *Advances in Neural Information Processing Systems* (1998), M. I. Jordan, M. J. Kearns, and S. A. Solla, Eds., vol. 10, The MIT Press.

[41] Jacobs, R. A., Jordan, M. I., and Barto, A. G. Task decomposition through competition in a modular connectionist architecture: The what and where vision tasks. Tech. rep., Department of Computer & Information Science, University of Massachusetts, Amhert, Mar. 1990. Anonymous FTP: ftp://archive.cis.ohio-state.edu/pub/neuroprose/jacobs.modular.ps.Z.

[42] Jacobs, R. A., Jordan, M. I., and Barto, A. G. Task decomposition through competition in a modular connectionist architecture - the What and Where vision tasks. *Cognitive Science 15*, 2 (1991), 219–250.

[43] Jacobs, R. A., Jordan, M. I., Nowlan, S. J., and Hinton, G. E. Adaptive mixtures of local experts. *Neural Computation 3*, 1 (1991), 79–87.

[44] James, G., and Hastie, T. Generalizations of the bias/variance decomposition for prediction error. Tech. rep., Dept. of Statistics, Stanford University, Feb. 1996. Link: http://www-stat.stanford.edu/~gareth/ftp/papers/bv.ps.

[45] JORDAN, M., AND JACOBS, R. Hierarchical mixtures of experts and the EM algorithm. *Neural Computation 6*, 2 (1994), 181–214.

[46] JORDAN, M., AND XU, L. Convergence results for the EM approach to mixtures of experts architectures. *Neural Networks 8*, 9 (1995), 1409–1431.

[47] JORDAN, M. I., AND JACOBS, R. A. Hierarchies of adaptive experts. In *Advances in Neural Information Processing Systems* (1992), J. E. Moody, S. J. Hanson, and R. P. Lippmann, Eds., vol. 4, Morgan Kaufmann Publishers, Inc., pp. 985–992.

[48] KINCAID, D., AND CHENEY, W. *Numerical Analysis*. Brooks/Cole Publishing Company, Pacific Grove, California 93950, 1991.

[49] KITTLER, J. Combining classifiers: Atheoretical framework. *Pattern Analysis and Application 1* (1998), 18–27.

[50] KITTLER, J., HATEF, M., DUIN, R. P., AND MATAS, J. On combining classifiers. *IEEE Transaction on Pattern Analysis and Machine Intelligence 20*, 3 (Mar. 1998), 226–239.

[51] KROGH, A., AND VEDELSBY, J. Neural network ensembles, cross validation, and active learning. In *Advances in Neural Information Processing Systems* (1995), G. Tesauro, D. Touretzky, and T. Leen, Eds., vol. 7, The MIT Press, pp. 231–238.

[52] LUND, O., FRIMAND, K., GORODKIN, J., BOHR, H., BOHR, J., HANSEN, J., AND BRUNAK, S. Protein distance constraints predicted by neural networks and probability density functions. *Protein Engineering 10*, 11 (1997), 1241–1248.

[53] MCCULLAGH, P., AND NELDER, J. A. *Generalized Linear Models*. Chapman and Hall, 11 New Fetter Lane, London EC4P 4EE, UK., 1983.

[54] MERZ, C., AND MURPHY, P. UCI repository of machine learning databases, 1998. Link: http://www.ics.uci.edu/~mlearn/MLRepository.html.

[55] MØLLER, M. *Efficient Training of Feed-Forward Neural Networks*. PhD thesis, Computer Science Department, Aarhus University, 1993.

[56] MUNK, O. L., AND HANSEN, S. B. Automated registration of pet brain scans using neural networks. In *Physiological Imaging of the Brain by PET* (2000), A. Gjedde, S. B. Hansen, G. M. Knudsen, and O. Paulson, Eds., Academic Press.

[57] NOWLAN, S. J., AND HINTON, G. E. Evaluation of adaptive mixtures of competing experts. In *Advances in Neural Information Processing*

Systems (1991), R. P. Lippmann, J. E. Moody, and D. S. Touretzky, Eds., vol. 3, Morgan Kaufmann Publishers, Inc., pp. 774–780.

[58] Ohno-Machado, L., and Musen, M. A. Modular neural networks for medical prognosis: Quantifying the benefits of combining neural networks for survival prediction. *Connection Science 9*, 1 (1997), 71–86.

[59] Opitz, D., and Maclin, R. Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research* (1999).

[60] Opitz, D. W., and Shavlik, J. W. Generating accurate and diverse members of a neural-network ensemble. In *Advances in Neural Information Processing Systems* (1996), D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, Eds., vol. 8, The MIT Press, pp. 535–541.

[61] Pathria, R. K. *Statistical Mechanics.* Pergamon, Elsevier Science Ltd. The Boulevard, Langford Lane, Kidlington, Oxford OX5 1GB, U.K., 1985.

[62] Peng, F., Jacobs, R. A., and Tanner, M. A. Bayesian inference in mixtures-of-experts and hierarchical mixtures-of-experts models with an application to speech recognition. *Journal of American Statistical Association* (1996).

[63] Perrone, M. P., and Cooper, L. N. When networks disagree: Ensemble method for neural networks. In *Artificial Neural Networks for Speech and Vision* (1993), Chapman Hall.

[64] Prechelt, L. PROBEN1 — A set of benchmarks and benchmarking rules for neural network training algorithms. Tech. Rep. 21/94, Fakultät für Informatik, Universität Karlsruhe, D-76128 Karlsruhe, Germany, Sept. 1994. Anonymous FTP: /pub/papers/techreports/1994/1994-21.ps.Z on ftp.ira.uka.de.

[65] Quinlan, J. R. *C4. 5: Programs for Machine Learning.* Morgan Kaufmann, San Mateo, CA, 1993.

[66] Quinlan, J. R. Bagging, boosting, and c4.5. Tech. rep., University of Sydney, Sydney, Australia 2006, Apr. 1996. Link: http://www.cse.unsw.edu.au/~quinlan/q.aaai96.ps.

[67] Rahman, A. F. R., and Fairhurst, M. C. A new hybrid approach in combining multiple experts to recognise hanwritten numerals. *Pattern Recognition Letters 18* (1997), 781–790.

[68] Ramamurti, V., and Ghosh, J. Structural adaptation in mixtures of experts, 1997. Anonymous FTP://ftp.lans.ece.utexas.edu/pub/papers/npap.ps.gz.

[69] Rao, J. S., and Tibshirani, R. The out-of-bootstrap method for model averaging and selection. Tech. rep., Cleveland Clinic, University of Toronto, May 1997. Anonymous ftp: ftp://utstat.toronto.edu/pub/tibs/outofbootstrap.ps.

[70] Reif, F. *Fundamentals of Statistical and Thermal Physics.* McGraw-Hill, 1985.

[71] Riis, S. K., and Krogh, A. Improving prediction of protein secondary structure using structured neural networks and multiple sequence alignments. *Journal of Computational Biology 3* (1996), 163–183.

[72] Robert E Schapire, Yoav Freund, P., and Wee Sun Lee. Boosting the Margin: A New Explanation for the effectiveness of voting methods. *Machine Learning: Proceedings of the Fourteenth International Conference* (1997).

[73] Rosen, B. E. Ensemble learning using decorrelated neural networks. *Connection Science, Special Issue: Combining Artificial Neural Nets: Ensemble Approaches 8*, 3 and 4 (Dec. 1996), 373–383.

[74] Rost, B., and Sander, C. Prediction of protein secondary structure at better than 70 % accuracy. *Journal of Molecular Biology 232*, 2 (Jul 20 1993), 584–599.

[75] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Learning Internal Representation by Error Propagation. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition 1* (1986).

[76] Schapire, R. E. The strength of weak learnability. *Machine Learning 5* (1990), 197.

[77] Schapire, R. E., and Singer, Y. Boostexter: A system for multiclass multi-label text categorization. Tech. rep., AT & T Labs, 180 Park Avenue, Florham Park, NJ 07932-0971 USA, Mar. 1998. Link: http://www.research.att.com/~schapire/papers/SchapireSi98b.ps.Z.

[78] Schwenk, H., and Bengio, Y. Adaptive boosting of neural networks for character recognition. Tech. rep., Département d'Informatique et Recherche Opérationelle, Université de Montréal, May 1997. Link: http://m17.limsi.fr/Individu/schwenk/Papers.A4/AdaBoostTR.ps.gz.

[79] Sharkey, A. J., Ed. *Combining Artificial Neural Nets: Ensemble and Modular Multi-Net Systems.* Springer-Verlag London Ltd, 1999.

[80] Sharkey, A. J. C., Ed. *Special Issue: Combining Artificial Neural Nets: Ensemble Approach*, vol. 8. Connection Science, Dec. 1996.

[81] SMYTH, P., AND WOLPERT, D. H. Stacked density estimation. Tech. rep., Information and computer Science Department, University of California, Irvine, Aug. 1997.

[82] SOLLICH, P., AND KROGH, A. Learning with ensembles: How overfitting can be useful. In *Advances in Neural Information Processing Systems* (1996), D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, Eds., vol. 8, The MIT Press, pp. 190–196.

[83] TESAURO, G., AND SEJNOWSKI, T. J. A "neural" network that learns to play backgammon. In *Neural Information Processing Systems* (1988), D. Z. Anderson, Ed., New York: American Institute of Physics, pp. 442–456.

[84] TRESP, V., AND TANIGUCHI, M. Combining estimators using non-constant weighting functions. In *Advances in Neural Information Processing Systems* (1995), G. Tesauro, D. Touretzky, and T. Leen, Eds., vol. 7, The MIT Press, pp. 419–426.

[85] VAN DER SMAGT, P., AND HIRZINGER, G. Why feed-forward networks are in a bad shape. In *Proceedings of the 8th International Conference on Artificial Neural Networks* (1998), M. B. L. Niklasson and T. Ziemke, Eds., Springer Verlag, pp. 159–164.

[86] WAHBA, G., LIN, X., GAO, F., XIANG, D., KLEIN, R., AND KLEIN, B. The bias-variance tradeoff and the randomized gacv. Tech. rep., Department of Statistics, University of Wisconsin, 1210 West Dayton St., Madison, WI 53706, 1998.

[87] WEIGEND, A. S., MANGEAS, M., AND SRIVASTAVA, A. N. Nonlinear gated experts for time-series - discovering regimes and avoiding overfitting. *International Journal of Neural Systems 6*, 4 (1995), 373–399.

[88] WOLPERT, D. H. Stacked generalization. *Neural Networks 5* (1992).

[89] WOLPERT, D. H., Ed. *Mathematics of Generalization.* Addison Wesley Longman, Reading, MA, 1995.

[90] WOLPERT, D. H. On Bias Plus Variance. *Neural Computation* (1997), 1211–1243.

[91] WOLPERT, D. H., AND MACREADY, W. G. Combining stacking with bagging to improve a learning algorithm. Tech. rep., Santa Fe, Sept. 1996. Anonymous ftp: ftp.santafe.edu/pub/wgm/bs.ps.

[92] WOLPERT, D. H., AND MACREADY, W. G. An efficient method to estimate bagging's generalization error. Tech. rep., Santa Fe, 1996. Anonymous ftp: ftp.santafe.edu/pub/wgm/error.ps.

[93] Xu, L., Hinton, G., and I.Jordan, M. An alternative model for mixtures of experts. In *Advances in Neural Infomation Processing Systems 7* (1994), G. Tesauro, D. S. Touretzky, and T. K. Leen, Eds., MIT Press, pp. 633–640.

[94] Xu, L., Jordan, M. I., and Hinton, G. E. An alternative model for mixtures of experts. In *Advances in Neural Information Processing Systems* (1995), G. Tesauro, D. Touretzky, and T. Leen, Eds., vol. 7, The MIT Press, pp. 633–640.

[95] Yoav Freund, and Robert E Schapire. Experiments with a New Boosting Algorithm. *Machine Learning: Proceeding on the thirteenth Conference* (1996), 148–156.

[96] Zhu, H. Error decomposition and model complexity. *Neural Computation 11* (1998).