Available online at www.sciencedirect.com
ScienceDirect
Neurocomputing 70 (2007) 2371–2378
NEUROCOMPUTING
www.elsevier.com/locate/neucom

Time series prediction with ensemble models applied to the CATS benchmark

Jörg D. Wichard[a,*], Maciej Ogorzałek[b]

[a]Molecular Modelling Group, Institute of Molecular Pharmacology, Robert Rössle Straße 10, D-13125 Berlin-Buch, Germany
[b]Department of Electrical Engineering, AGH University of Science and Technology, al. Mickiewicza 30, 30-059 Kraków, Poland

Available online 25 February 2007

Abstract

We describe the use of ensemble methods to build models for time series prediction. Our approach extends the classical ensemble methods for neural networks by using several different model architectures. We further suggest an iterated prediction procedure to select the final ensemble members.

Keywords: Neural networks; Ensemble methods; Time series prediction

1. Introduction

Ensemble building is a common way to improve the performance of the resulting model for classification and regression tasks, since it was noticed that an ensemble of individual predictor performs better than a single predictor in the average [6,10]. Usually an ensemble consists of models taken from one single class, e.g. neural networks [7,17,11,16], support vector machines [24] or regression trees [2]. We suggest a different strategy. We train several models from different model classes and combine them to build the final ensemble. This is done in order to introduce model diversity which is the central feature of the ensemble approach [11]. The novelty of our approach consists of building heterogeneous ensembles with several model classes combined with an iterated prediction scheme for final model selection. For the CATS Benchmark [13] we propose a combined model strategy in order to cope with the different timescales of the data set. In Section 3 we present our investigation that leads to the assumption, that the time series has two different timescales. We noticed that other research groups had the same idea while dealing with the CATS data set [19,3].

*Corresponding author. Tel.: +49 30 4681 2096.
E-mail addresses: joergwichard@web.de, wichard@fmp-berlin.de (J.D. Wichard), maciej@agh.edu.pl (M. Ogorzałek).

0925-2312/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.neucom.2005.12.136

2. Method of prediction

We build a simple polynomial model to cover the long term oscillations and combine that with an ensemble model for the small scale dynamics. In the following section we like to introduce the ensemble approach that we used to build the small scale model.

2.1. Ensembles

If we average the output of several different models $f_i(\mathbf{x})$, we call it an ensemble model

$$\hat{f}(\mathbf{x}) = \sum_{i=1}^{K} \omega_i f_i(\mathbf{x}). \quad (1)$$

We further assume that the model weights ω_i sum to one $\sum_{i=1}^{K} \omega_i = 1$. The central feature of this method is its generalization ability. The generalization error of the ensemble is in general lower than the mean of the generalization error of the single ensemble members [11]. This holds in general, independent of the model class.

It was shown by several authors, that the generalization error of an ensemble model could be improved if the single models on which averaging is done disagree and if their output is uncorrelated [10,16]. This becomes obvious, if we

investigate the expected generalization error

$$Err(\mathbf{x}) = \|\hat{f}(\mathbf{x}) - y\|^2$$

and its bias/variance decomposition given by Geman et al. [6]

$$Err(\mathbf{x}) = \sigma^2 + (Bias(\hat{f}(\mathbf{x})))^2 + Var(\hat{f}(\mathbf{x})), \tag{2}$$

where σ^2 is the variance of y given $\mathbf{x}$. The variance term could be decomposed in the following way:

$$Var(\hat{f}) = \sum_{i=1}^{K} \omega_i^2 (E[f_i^2] - E^2[f_i]) + 2\sum_{i<j} \omega_i \omega_j (E[f_i f_j] - E[f_i]E[f_j]), \tag{3}$$

where the expectation is taken with respect to the data set under investigation. The first sum in Eq. (3) marks the lower bound of the ensemble variance and is the weighted mean of the variances of the ensemble members. The second sum contains the cross-terms of the ensemble members and vanishes if the models are completely uncorrelated [10]. This shows that the reduction in the variance of the ensemble is related to the degree of independence of the single models [16]. There are several ways to introduce model diversity to the ensemble in order to decorrelate the output of the individual ensemble members. A general approach is to train various models on selected subsets of the training data [11,2] or to initiate the training algorithm with randomly chosen initial conditions [16]. A different approach was recently introduced by Bakker et al. [1], where the ensemble consists of representative models that are selected by clustering the model outputs. We introduce model diversity in such a way, that we train several model classes on different subsets of the training data, using a cross validation scheme. An overview of the different model classes that we use for ensemble building is given below. The implementation of our ensemble approach together with a detailed description can be found as a freely available toolbox in [15].

2.2. Linear and polynomial models

The d-dimensional linear model has the form

$$f(\vec{x}) = a_0 + \sum_{i=1}^{d} a_i x_i, \tag{4}$$

where a_0 is the offset. The coefficients are calculated with the standard method for ridge regression (see Hastie et al. [8] for a detailed description). The optimal ridge parameter is evaluated by performing a cross-validation on the training data. The polynomial model is given by $f(\vec{x}) = \sum_{i=1}^{P} a_i p_i(\vec{x})$, wherein the monoms have the form subplot$(3,1,2)$ and $n_i \in \mathbf{N}$ is the ith exponent. We use an iterative term selection for the monoms, wherein we add successively the terms to the polynomial model that decrease the out-of-sample error on a subset of the training data.

2.3. Nearest neighbor models

A k-nearest-neighbor model takes a weighted average over those observations z_i in the training set that are closest to the query point $\vec{x}$. This is,

$$f(\vec{x}) = \frac{1}{\sum w_i} \sum_{\vec{z}_i \in N_k(\vec{x})} w_i z_i, \tag{5}$$

where $N_k(\vec{x})$ denotes the k-element neighborhood of $\vec{x}$, defined in a given metric. Common choices are the L_1, L_2 and the L_∞ metrics. To compensate for irrelevant input dimensions, distances are computed using a weighted metric:

$$d(\vec{x},\vec{z}) = \left(\sum_{i=1}^{D} m_i (x_i - z_i)^M\right)^{1/M}, \quad 0 \leqslant m_i \leqslant 1. \tag{6}$$

The vector of metric coefficients $\vec{m}$ is adapted by a Genetic Algorithm. One vector of metric coefficients is an individual of the population. The fitness value is assigned to each individual according to its error on the training data set.

2.4. Nearest trajectory models

The nearest trajectory model is based on a strategy for time series prediction introduced by McNames [14]. It is based on the assumption that the time series stems from a dynamical system and the states can be reconstructed with a time delay embedding, which is possible for a large class of systems [20–22]. The nearest trajectory model looks for the nearest trajectory segments in the reconstructed state space instead of the nearest neighbors. The prediction is done with a local linear model of the closest trajectory points as described in [14]. The number of neighboring trajectories is chosen randomly at the start of the training algorithm to introduce model diversity.

2.5. Neural networks

We use a multilayer feed-forward neural network (MLP: multi layer perceptron) with the $\tanh(\vec{x})$ as nonlinear element. In order to increase the ensemble ambiguity, we initialize the weights with Gaussian distributed random numbers having zero mean and scaled variances, following a suggestion of LeCun et al. [12]. The number of hidden layers is chosen at random to be one or two and the numbers of neurons in also random (3–9 neurons in the first layer, 4–32 in second layer). We also use two different training procedures: a first order training algorithm based on the Rprop Algorithm [18] with the improvements given in [9]. The second order training algorithm is a Levenberg–Marquart gradient descent [12]. As regularization method we use the