4.4 Nonlinear Semiparametric Models

Ming Lu

2022-04-02

The next two nonlinear models that we are going to study, Neural Networks (NNet) and Support Vector Machines (SVM), fall in the category of semiparametric models. What this means is that although the NNet and SVM, just as the other methods seen in this chapter, try to relate a set of input variables and weights to a set of one or more output variables, they differ from the ARMA, ARCH or GARCH methods in that their parameters are not limited in number, nor do they have to be all known or computed for the method to give an approximate solution. The reader might be acquainted with other semiparametric estimation models. For example, Taylor polynomials are instances of this class of models. Here the degree of the polynomials is not fixed, although the higher the degree and the more terms computed, the better the approximation to the target function. However, the NNet and SVM differ from polynomial approximation schemes, such as Taylor’s, in other fundamental aspect, and it is that they do not give closed form solutions. For Taylor polynomials we have a definitive formula for computing their coefficients in terms of derivatives of the function we want to approximate. But for NNet and SVM models their parameters have to be estimated from data and adjusted iteratively, much the same as it is done in maximum likelihood estimation for ARCH models. In this regard is that these are known as machine learning models, since they fit in the concept of being algorithms that improve their performance at some task through experience. The task of our interest is forecasting future values of a given time series, and the experience is constituted by the past values of such time series and other statistics. The models progressively tune their parameters by a two step process wherein the data is divided into a training set, where the learning from experience process takes place, and a testing set, where the performance of the model is evaluated with respect to some performance measure (e.g. minimizing residual sum of squares), and succeeding adjustments of the model are made. In the case of time series usually the training and the testing sets are taken as two consecutive time periods of the series of varying length. Details will be given in the following sections. The quality of being semiparametric or parametric, and the existence of closed form solutions provides two important dimensions to characterize all the forecasting models seen so far and the two models of this section. This model typology suggested by McNelis (2005) is summarized in the following table

Thus, in one extreme of McNelis model spectrum are the linear regression models, which are parametric and have easy to compute closed form solutions (e.g. the parameters of AR models are obtained by Yule-Walker equations). The downside of these models is mainly that solutions can be very imprecise due to the linearity and other constraints. On the other extreme of the spectrum are the semiparametric NNet and SVM models that give approximate solutions and are harder to estimate, but can capture the non linearity of the process and can be more precise in their forecasts.

# 4.4.1 Neural Networks

The feed-forward neural network is one of the most popular methods to approximate a multivariate nonlinear function. Their distinctive feature is the “hidden layers”, in which input variables activate some nonlinear transformation functions producing a continuous signal to output nodes. This hidden layer scheme represents a very efficient parallel distributed model of nonlinear statistical processes. In fact,feed-forward neural networks have been shown to be capable of approximate any continuous function uniformly on compact sets, for a suitable number of nodes in the hidden layer and a sufficiently large training sample from data (see Hornik et al. (1989), Ripley (1994) and references therein) .

Let us begin with an schematic presentation of a feed-forward neural network. Figure shows one such network with three input nodes and , one output node , and four hidden nodes or neurons, and , which constitute the hidden layer. Each neuron receives the input information and goes active, transmitting a signal to output, if a certain linear combination of the inputs, , is greater than some threshold value ; that is, transmits a non-zero value if

where the sum is taken over all input nodes feeding to neuron , and are the weights. The signal is produced by an activation function applied to , which is usually taken as a logistic function,

If there is no activation the signal is 0 , and when there is activation the value of the signal is . Composing a linear combination of the possible values of signals transmitted by the hidden layer, plus some value to account for the bias in the connection, with an activation function for the output node, we obtain a mathematical expression for the basic feed-forward network with connections from inputs to hidden nodes, and from these to output:

For a more complex feed-forward network one allows direct connections from input to output nodes, and thus we have a more general expression for these neural networks

where are the input values, is the output, is the activation function for each of the hidden nodes, is the threshold and are the weights along the connection for the th hidden node, and are the weights for the linear combinations of order and respectively defined, is the bias in the connection, and is the activation function for the output node. This can also be logistic, or a linear function, or a threshold function

among others. Note that the case of being linear gives Eq. (4.41) the form of an autoregressive moving average model for a given time series . Indeed, taking (i.e., as different lags of the series), and the output as the time series value to forecast, and linear in Eq. (4.41), we have

(here the moving average part is being modelled by a nonlinear function on the input lags). This shows that the feed forward neural network is a generalization of model.

The case of being a threshold function is also of interest in time series applications. If is a return to be computed, and are different lags of this return series , or some other statistics (e.g. its historical volatility), then Eq. (4.41) with threshold can be interpreted as a model to forecast the direction of returns, where implies positive returns and negative returns. Model estimation and forecasting. We are interested in fitting a feed-forward neural network to a time series of returns. Let be the available data where is the return of some financial asset and is vector of inputs (also called features) which can contain various lags of the series, or the volume, the variance or any fundamental indicator of the series. The model fitting for a neural network requires a division of the data in two subsamples, so that one subsample is used for training the model and the other subsample is used for testing. In the training step the parameters (i.e., the weights, the thresholds and connection bias) in Eq. (4.41) are chosen so that some forecasting error measure is minimized; usually it is required that the paremeters chosen minimized the mean squared error

where is the vector of required parameters and is the output of Eq. (4.41) on inputs , and the actual return. This is a non-linear estimation problem that can be solved by some iterative method. One popular method is the back propagation algorithm which works backward starting with the output and by using a gradient rule adjusts the weights iteratively (Ripley 1994, Sect. 2.1).

In the testing step the network build in the training step is used on the second subsample data to predict some values and compare the estimations with the actual sampled values. The steps are repeated several times to tune the parameters, with the possible application of cross-validation to select the best performing network with respect to forecasting accuracy. In spite of its popularity the cross-validation method can fail to find the best parameters in the presence of local minima, and may required a lot of computer time in processing neural networks. Other statistical methods for selecting networks based on maximum likelihood methods or Bayes factor are appropriate alternatives (Ripley 1995). NNet models in . We show how to fit a NNet to a time series in the appendix of this chapter.

# 4.4.2 Support Vector Machines

The Support Vector Machine (SVM) is a learning algorithm for obtaining a model from training data, developed by Vladimir Vapnik and coworkers, as an alternative to neural networks and other machine learning methods that are based on the empirical risk minimization principle. This principle refers to methods that seek to minimize the deviation from the correct solution of the training data (e.g., as measured by the mean squared error), whereas the SVM is an approximate implementation of the structural risk minimization principle, by trying to estimate a function that classifies the data by minimizing an upper bound of the generalization error. This is achieve by solving a linearly constraint quadratic programming problem, and as a consequence the SVM trained solution is unique and globally optimal, as opposed to neural networks training which require nonlinear optimization and solutions can thus get stuck at local minimum. Support vector machines for regression. In the context of forecasting time series, the SVM is more suited as a regression technique, as opposed to a classifier. Therefore we restrict our exposition of the theory of support vector machines for regression. Given a data set which is produced by some unknown function that maps each vector of real values to the real value , we want to approximate this function by regression on the vectors . The SVM considers regressions of the following form:

where the functions are the features of inputs and and are coefficients which are estimated from the data by minimizing the risk functional:

with respect to the linear -insensitive loss function

The approximation defined by Eq. (4.42) can be interpreted as a hyperplane in the -dimensional feature space defined by the functions , and the goal is to find a hyperplane that minimizes . It is shown in Vapnik (2000, Chap. 6) that such minimum is attained by functions having the following form:

where describes the inner product in the -dimensional feature space, that is, and is called the kernel function, and the coefficients are Lagrange multipliers that satisfy the equations , for , and are obtained by maximizing the following quadratic form:

subject to the constraints:

The parameters and are free and chosen by the user. The nature of this quadratic programming problem forces a certain number of the coefficients to be different from zero, and the data points associated to them are called support vectors. Now, one of the interesting facts about the SVM method is that the kernel can be computed analytically, thus saving us the trouble of computing the features functions and their inner product. Any function that satisfies Mercer’s condition can be used as a kernel function, thus reducing the computation of an inner product in high dimensional feature space to a scalar computation in input space (see Vapnik (2000, Sect. 5.6.2)). There are many choices for the kernel function, although the most commonly used are the following: - the polynomial kernel (with degree ; - the Gaussian radial basis function , with bandwidth , and - the sigmoid kernel: .

Model estimation and forecasting. The estimation of parameters in SVN model is done much the same way as for neural networks. The data is divided into two subsamples, one for training the model and the other for testing. In the training stage the parameters are estimated under the structural risk minimization principle (Eq. (4.43)), and after measuring the accuracy of the predictions in the testing stage, one repeats the process to tune the parameters for a number of iterations. At this stage cross-validation is also useful. In the next section we give some criteria for measuring the accuracy of predictions, applicable to this and all the other forecasting models.

There are at least four R packages that implements SVM models: e1071, kernlab, klaR and svmpath. The first two are the most robust and complete implementations of SVM, based on the award winning libsvm library of support vector machines classification, regression and kernel methods. The other two have less computing features and have simpler (and slower) implementations of SVM, so we do not consider their use. For an in-depth review of these four R packages for SVM and a comparison of their performance, see Karatzoglou et al. (2006). SVM models in R. We fit a SVM to a time series in the appendix of this chapter.