Neural Networks

Department of Statistics & Biostatistics
Rutgers University

Nov 29, 2011
Outline

1. Model Assessment and Selection
2. Neural Networks
Typically the model will have a tuning parameter or parameters $\alpha$, which varies the complexity of the model. We wish to find the value of $\alpha$ that minimizes some error.

**Goal 1: Model Selection**

Estimating the performance of different models in order to choose the best one.

**Goal 2: Model Assessment**

Having chosen a final model, estimating its prediction error (generalization error) on new data.
The general formula of the in-sample error estimate is

\[ \hat{\text{Err}}_{\text{in}} = \bar{\text{err}} + \hat{\omega}, \]

where \( \hat{\omega} \) is an estimate of the average optimism.

When \( d \) parameters are fit under squared error loss, we have the \( C_p \) statistic as an estimate of \( \text{Err}_{\text{in}} \)

\[ C_p = \bar{\text{err}} + 2 \cdot \frac{d}{N} \hat{\sigma}^2, \]

where \( \hat{\sigma}^2 \) is an estimate of the noise variance, obtained from the mean squared error of a low-bias model.
$C_p$ and AIC

- The **Akaike information criterion** is a more generally applicable estimate of $\text{Err}_{\text{in}}$ when a log-likelihood loss function is used. It relies on

$$-2 \cdot \mathbb{E} \left[ \log P_\theta (Y^0) \right] \approx -\frac{2}{N} \cdot \mathbb{E} (\text{loglik}) + 2 \cdot \frac{d}{N}.$$  

where $\text{loglik}$ is the maximal log likelihood

$$\text{loglik} = \sum_{i=1}^{N} \log P_\theta (y_i).$$

- The **AIC** is defined as

$$\text{AIC} = -\frac{2}{N} \cdot \text{loglik} + 2 \cdot \frac{d}{N}$$

- For the Gaussian model (with noise variance $\sigma^2$ assumed known), AIC is equivalent to $C_p$, so we refer to them collectively as AIC.
Model Selection

- Given a set of models $f_\alpha(x)$ indexed by a tuning parameter $\alpha$, define

  $$\text{AIC}(\alpha) = \text{err}(\alpha) + 2 \cdot \frac{d(\alpha)}{N} \hat{\sigma}^2.$$ 

- The function $\text{AIC}(\alpha)$ provides an estimate of the test error curve, and we find the tuning parameter $\hat{\alpha}$ that minimizes it.

- The formula is only approximately true for linear models with log-likelihood loss, and does not hold in general for 0-1 loss. But AIC often still does a reasonable good job.
The generic form of BIC is

\[ AIC = -2 \cdot \text{loglik} + (\log N) \cdot d. \]

Arise from in the Bayesian approach to model selection.

For model selection purposes, there is no clear choice between AIC and BIC.

BIC is asymptotically consistent as a selection criterion.

for finite samples, BIC often chooses models that are too simple, because of its heavy penalty on complexity.
FIGURE 7.7. Boxplots show the distribution of the relative error $100 \times [\text{Err}_T(\hat{\alpha}) - \min \alpha \text{Err}_T(\alpha)] / [\max \alpha \text{Err}_T(\alpha) - \min \alpha \text{Err}_T(\alpha)]$ over the four scenarios of Figure 7.3. This is the error in using the chosen model relative to the best model. There are 100 training sets each of size 80 represented in each boxplot, with the errors computed on test sets of size 10,000.
**K-fold Cross Validation**

- Split the data into *K* roughly equal-sized parts, e.g. when *K* = 5

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train</td>
<td>Train</td>
<td>Validation</td>
<td>Train</td>
<td>Train</td>
</tr>
</tbody>
</table>

- For the *k*th part, fit the model to the other *K* − 1 parts of the data, and calculate the prediction error of the fitted model when predicting the *k*th part of the data. Denote by \( \hat{f}^{-k}(x) \) the fitted function.

- We do this for *k* = 1, 2, ..., *K* and combine the *K* estimates of prediction error.

- Let \( \kappa : \{1, \ldots, N\} \to \{1, \ldots, K\} \) be an index function that indicates the part to which the observation *i* is allocated. Then

  \[
  CV(\hat{f}) = \frac{1}{N} \sum_{i=1}^{N} L \left( y_i, \hat{f}^{-\kappa(i)}(x_i) \right).
  \]

- Typical choice of *K* is 5 or 10.
- Cross-validation only estimates effectively the average error \( Err \).
Model Selection

- Have a set of models $f(x, \alpha)$ indexed by a tuning parameter $\alpha$.
  - Best subset selection, $\alpha$ is the subset size.
  - Lasso, $\alpha$ is the penalty parameter.
  - Splines, $\alpha$ is the number of knots.
  - Kernel method, $\alpha$ is the window size.
- Denote by $\hat{f}^{-k}(x, \alpha)$ the $\alpha$th model fit with the $k$th part of the data removed. Define

$$CV(\hat{f}, \alpha) = \frac{1}{N} \sum_{i=1}^{N} L \left( y_i, \hat{f}^{-\kappa(i)}(x_i, \alpha) \right).$$

- The function $CV(\hat{f}, \alpha)$ provides an estimate of the test error curve, and we find the tuning parameter $\hat{\alpha}$ that minimizes it.
- Our final chosen model is $f(x, \hat{\alpha})$, which we then fit to all the data.
Example

“One-Standard-Error” Rule
Choose the most parsimonious model whose error is no more than one standard error above the error of the best model (e.g. $p = 9$ chosen, $p = 10$ true).
Wrong and Right Way

Consider a classification problem with a large number of predictors, as may arise, for example, in genomic or proteomic applications. A typical way for analysis might be

1. Screen the predictors: find a subset of “good” predictors that show fairly strong (univariate) correlation with the class labels.
2. Using just this subset of predictors, build a multivariate classifier.
3. Use cross-validation to estimate the unknown tuning parameters and to estimate the prediction error of the final model.
Consider a classification problem with a large number of predictors, as may arise, for example, in genomic or proteomic applications. A typical way for analysis might be

1. Screen the predictors: find a subset of “good” predictors that show fairly strong (univariate) correlation with the class labels.
2. Using just this subset of predictors, build a multivariate classifier.
3. Use cross-validation to estimate the unknown tuning parameters and to estimate the prediction error of the final model.

Wrong!!!
Wrong and Right Way

Here is the correct way

1. Divide the samples into $K$ cross-validation folds at random.
2. For each fold $k = 1, \ldots, K$
   
   (a) Find a subset of “good” predictors that show fairly strong (univariate) correlation with the class labels, using all of the samples except those in fold $k$.
   
   (b) Using just this subset of predictors, build a multivariate classifier, using all of the samples except those in fold $k$.
   
   (c) Use the classifier to predict the class labels for the samples in fold $k$.

Right way

With a multistep modeling procedure, cross-validation must be applied to the entire sequence of modeling steps.
FIGURE 7.12. Schematic of the bootstrap process. We wish to assess the statistical accuracy of a quantity $S(Z)$ computed from our dataset.

$Z = (z_1, z_2, \ldots, z_N)$

$S(Z^1)$  $S(Z^2)$  $S(Z^B)$

$Z^1$  $Z^2$  $Z^B$

Bootstrap replications
Bootstrap samples
Training sample

How can we apply the bootstrap to estimate prediction error? One approach would be to fit the model in question on a set of bootstrap samples, and then keep track of how well it predicts the original training set. If $\hat{f}^*_b(x_i)$ is the predicted value at $x_i$ from the model fitted to the $b$th bootstrap dataset, our estimate is

$$\hat{\text{Err}}_{\text{boot}} = \frac{1}{B} \sum_{b=1}^{B} \sum_{i=1}^{N} L(y_i, \hat{f}^*_b(x_i))$$

(7.54)

However, it is easy to see that $\hat{\text{Err}}_{\text{boot}}$ does not provide a good estimate in general. The reason is that the bootstrap datasets are acting as the training samples, while the original training set is acting as the test sample, and these two samples have observations in common. This overlap can make overfit predictions look unrealistically good, and is the reason that cross-validation explicitly uses non-overlapping data for the training and test samples. Consider for example a 1-nearest neighbor classifier applied to a two-class classification problem with the same number of observations in
Denote the training set by $Z = \{z_1, \ldots, z_N\}$, where $z_i = (x_i, y_i)$.

Randomly draw a sample $Z^b = \{z_1^b, \ldots, z_N^b\}$ with replacement from the training data $Z$.

Repeat the preceding step $B$ times, producing $B$ bootstrap datasets $Z^1, \ldots, Z^B$.

Fit the model in question on a set of bootstrap samples, and then keep track of how well it predicts the original training set.

Denote by $\hat{f}^b(x_i)$ the predicted value at $x_i$, from the model fitted to the $b$-th bootstrap dataset. The estimate the prediction error as

$$\hat{\text{Err}}_{\text{boot}} = \frac{1}{B} \frac{1}{N} \sum_{b=1}^{B} \sum_{i=1}^{N} L\left(y_i, \hat{f}^b(x_i)\right).$$

It favors overfit predictions!!!
Leave-One-Out Bootstrap

- Only keep track of predictions from bootstrap samples not containing that observation. The leave-one-out bootstrap estimate of prediction error is defined by

\[ \hat{\text{Err}}^{(1)} = \frac{1}{N} \sum_{i=1}^{N} \sum_{b \in C^{-i}} L \left( y_i, \hat{f}^b(x_i) \right), \]

where \( C^{-i} \) is the set of indices of the bootstrap samples \( b \) that do not contain the training point \( i \).

- The average number of distinct observations in each bootstrap sample is about \( 0.632 \cdot N \), so its bias will roughly behave like that of twofold cross-validation.

- The ".632 Estimator" is designed to alleviate the bias.

\[ \hat{\text{Err}}^{(.632)} = 0.368 \cdot \hat{\text{err}} + 0.632 \cdot \hat{\text{Err}}^{(1)}. \]

- Works well in "light fitting" situations, but can break down in overfit ones. Lead to ".632+ Estimator".
FIGURE 7.13. Boxplots show the distribution of the relative error \(\frac{\hat{\epsilon}_\alpha - \min_\alpha \epsilon(\alpha)}{\max_\alpha \epsilon(\alpha) - \min_\alpha \epsilon(\alpha)}\) over the four scenarios of Figure 7.3. This is the error in using the chosen model relative to the best model. There are 100 training sets represented in each boxplot. Figure 7.13 shows boxplots of \(\frac{\hat{\epsilon}_\alpha - \min_\alpha \epsilon(\alpha)}{\max_\alpha \epsilon(\alpha) - \min_\alpha \epsilon(\alpha)}\), the error in using the chosen model relative to the best model. There are 100 different training sets represented in each boxplot. Both measures perform well overall, perhaps the same or slightly worse than the AIC in Figure 7.7.

Our conclusion is that for these particular problems and fitting methods, minimization of either AIC, cross-validation or bootstrap yields a model fairly close to the best available. Note that for the purpose of model selection, any of the measures could be biased and it wouldn't affect things, as long as the bias did not change the relative performance of the methods.

For example, the addition of a constant to any of the measures would not change the resulting chosen model. However, for many adaptive, nonlinear techniques (like trees), estimation of the effective number of parameters is very difficult. This makes methods like AIC impractical and leaves us with cross-validation or bootstrap as the methods of choice.

A different question is: how well does each method estimate test error? On the average the AIC criterion overestimated prediction error of its chosen model.
1. Model Assessment and Selection

2. Neural Networks
Thinking of the constant "1" as an additional input feature, this bias unit captures the intercepts $\alpha_m$ and $\beta_k$ in model (11.5). The output function $g_k(T)$ allows a final transformation of the vector of outputs $T$. For regression we typically choose the identity function $g_k(T) = T_k$. Early work in $K$-class classification also used the identity function, but this was later abandoned in favor of the softmax function $g_k(T) = e^{T_k} \sum_{\ell=1}^{K} e^{T_\ell}$. (11.6) This is of course exactly the transformation used in the multilogit model (Section 4.4), and produces positive estimates that sum to one. In Section 4.2 we discuss other problems with linear activation functions, in particular potentially severe masking effects.

The units in the middle of the network, computing the derived features $Z_m$, are called hidden units because the values $Z_m$ are not directly observed. In general there can be more than one hidden layer, as illustrated in the example at the end of this chapter. We can think of the $Z_m$ as a basis expansion of the original inputs $X$; the neural network is then a standard linear model, or linear multilogit model, using these transformations as inputs. There is, however, an important enhancement over the basis-expansion techniques discussed in Chapter 5; here the parameters of the basis functions are learned from the data.
Neural Networks

- A neural network is a two-stage regression or classification model.
- For regression, typically $K = 1$ and there is only one output unit $Y_1$ at the top. Can also handle multiple quantitative responses.
- For $K$-class classification, there are $K$ units at the top, with the $k$-th unit modeling the probability of class $k$.
- The units in the middle of the network are derived features created from linear combinations of the inputs. They are called hidden units because the values $Z_m$ are not directly observed.

$$Z_m = \sigma(\alpha_{0m} + \alpha^T_m X), \quad m = 1, \ldots, M.$$  

- The targets $Y_k$ are modeled as functions of linear combinations of the $Z_m$ (with intercepts (biases))

$$T_k = \beta_{0k} + \beta^T_k Z, \quad k = 1, \ldots, K$$

$$Y_k = f_k(X) = g_k(T), \quad k = 1, \ldots, K;$$

where $Z = (Z_1, \ldots, Z_M)^T$ and $T = (T_1, \ldots, T_K)^T$. 
Neural Networks

- The activation function $\sigma(v)$ is usually chosen to be the sigmoid
  \[ \sigma(v) = \frac{1}{1 + e^{-v}}. \]

- The output function $g_k(T)$ allows a final transformation of the vector of outputs $T$. For regression we typically choose the identity function $g_k(T) = T_k$.
- For classification we typically choose the softmax function
  \[ g_k(T) = \frac{e^{T_k}}{\sum_{l=1}^{K} e^{T_l}}. \]

- A neural network can be viewed as a linear basis expansions of the original inputs.
- A neural network can be thought of as a nonlinear generalization of the linear model, both for regression and classification.
- In general there can be more than one hidden layer, and “skip-layer” connections.
- Neural networks with linear output units can approximate any continuous function $f$ uniformly on compact sets, by increasing the size of the hidden layer.
The name **neural networks** derives from the fact that they were first developed as models for the human brain. Each unit represents a neuron, and the connections represent synapses.

In early models, the neurons fired when the total signal passed to that unit exceeded a certain threshold. In the model above, this corresponds to use of a step function for $\sigma(\cdot)$ and $g_k(\cdot)$.

Later the neural network was recognized as a useful tool for nonlinear statistical modeling, and for this purpose the step function is not smooth enough for optimization. Hence the step function was replaced by a smoother threshold function,
Fitting Neural Networks

- The unknown parameters are often called **weights**. We denote the complete set of weights by \( \theta \), which consists of

  \[
  \{ \alpha_{0m}, \alpha_m; \ m = 1, \ldots, M \} \quad M(p + 1) \text{ weights},
  \]

  \[
  \{ \beta_{0k}, \beta_k; \ k = 1, \ldots, K \} \quad K(M + 1) \text{ weights}.
  \]

- We seek values of \( \theta \) to fit the data well (minimize the error function). For regression we use

  \[
  R(\theta) = \sum_{i=1}^{N} \sum_{k=1}^{K} (y_{ik} - f_k(x_i))^2.
  \]

- For classification we use either the squared error or **cross-entropy** (deviance)

  \[
  R(\theta) = - \sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} \log f_k(x_i).
  \]

- Typically we don’t want the global minimizer of \( R(\theta) \), as this is likely to be an **overfit** solution. Instead some regularization is needed, achieved directly through a **penalty term**, or indirectly by early stopping.
Implementation Issue: Starting Values

- If the weights are near zero, then the operative part of the sigmoid is roughly linear, and hence the neural network collapses into an approximately linear model. Usually starting values for weights are chosen to be random values near zero. Hence the model starts out nearly linear, and becomes nonlinear as the weights increase.

- The error function $R(\theta)$ is nonconvex, possessing many local minima. One must at least try a number of random starting values, and choose the solution giving lowest (penalized) error.

- Probably a better approach is to use the average predictions over the collection of networks as the final prediction.
Implementation Issue: Regularization

- Often neural networks have too many weights and will overfit the data at the global minimum of $R(\theta)$.
- In early developments of neural networks, either by design or by accident, an early stopping rule was used to avoid over-fitting. Since the weights start at a highly regularized (linear) solution, this has the effect of shrinking the final model toward a linear model.
- A more explicit method for regularization is weight decay. We add a penalty to the error function, and hence minimize

$$R(\theta) + \lambda J(\theta);$$

where

$$J(\theta) = \sum_{k,m} \beta_{k,m}^2 + \sum_{m,j} \alpha_{m,j}^2,$$

and $\lambda$ is a tuning parameter, typically chosen by cross-validation.
- The weight elimination penalty has the effect of shrinking smaller weights more,

$$J(\theta) = \sum_{k,m} \frac{\beta_{k,m}^2}{1 + \beta_{k,m}^2} + \sum_{m,j} \frac{\alpha_{m,j}^2}{1 + \alpha_{m,j}^2}.$$
Implementation Issue: Regularization

Neural Network - 10 Units, No Weight Decay

Training Error: 0.100
Test Error: 0.259
Bayes Error: 0.210
Implementation Issue: Regularization

Neural Network - 10 Units, Weight Decay=0.02

Training Error: 0.160
Test Error: 0.223
Bayes Error: 0.210
Implementation Issue: Regularization

No weight decay

Weight decay

Choice of the number of hidden layers is guided by background knowledge and experimentation. Each layer extracts features of the input for regression or classification. Use of multiple hidden layers allows construction of hierarchical features at different levels of resolution. An example of the effective use of multiple layers is given in Section 11.6.
Other Issues

- **Scaling of the inputs.** At the outset it is best to standardize all inputs to have mean zero and standard deviation one.

- With standardized inputs, it is typical to take random uniform weights over the range $[0.7, +0.7]$.

- With standardized inputs, typically choose $\lambda \approx 10^{-4} - 10^{-2}$ for least squares fitting, and $\lambda \approx 0.01 - 0.1$ for entropy fit.

- **Number of hidden units.** Generally speaking it is better to have too many hidden units than too few.

- It is most common to put down a reasonably large number of units and train them with regularization. The extra weights can be shrunk to zero.

- Typically the number of hidden units is somewhere in the range of 5 to 100, with the number increasing with the number of inputs and number of training cases.

- Choice of the number of hidden layers is guided by background knowledge and experimentation.