STATISTICS 452/652: Statistical Learning and Prediction

November 16, 2020

# Lecture 21: Neural Nets for Classification

**(Reading: Not in ISLR!)**

## 1 Goals of lecture

1. Apply flexibility of artificial neural networks to classification problem.

## 2 Neural Nets: Review

Recall the model for neural networks that we used in regression (see also Figure 1) :

1. Create hidden nodes $Z_1, \dots Z_M$ from inputs $X_1, \dots, X_p$ such that

$$Z_m = f_o(\alpha_{m0} + \alpha_{m1}X_1 + \dots + \alpha_{mp}X_p)$$

   (a) $\alpha_{0m}, \alpha_{1m}, \dots, \alpha_{pm}$, $m = 1, \dots, M$ are "weights"
      i. $p + 1$ parameters per hidden node
      ii. Unlike other linear combination components, add the constant $\alpha_{m0}$
   (b) $f_0$ is any function
      i. Called the ACTIVATION FUNCTION
      ii. *The same function for all $m$*
      iii. Often taken to be a "sigmoidal", $f_0(a) = 1/(1 + e^{-a})$
         A. Scales each hidden node's output to be potentially between 0 and 1
      iv. PPR used a different spline for each $m$.
   (c) $M$ is a tuning parameter! (as always...)

2. Create $T = \beta_0 + \beta_1 Z_1 + \dots + \beta_M Z_M$

Figure 1: Basic Neural Net Architecture

Elements of Statistical Learning (2nd Ed.) ©Hastie, Tibshirani & Friedman 2009 Chap 11

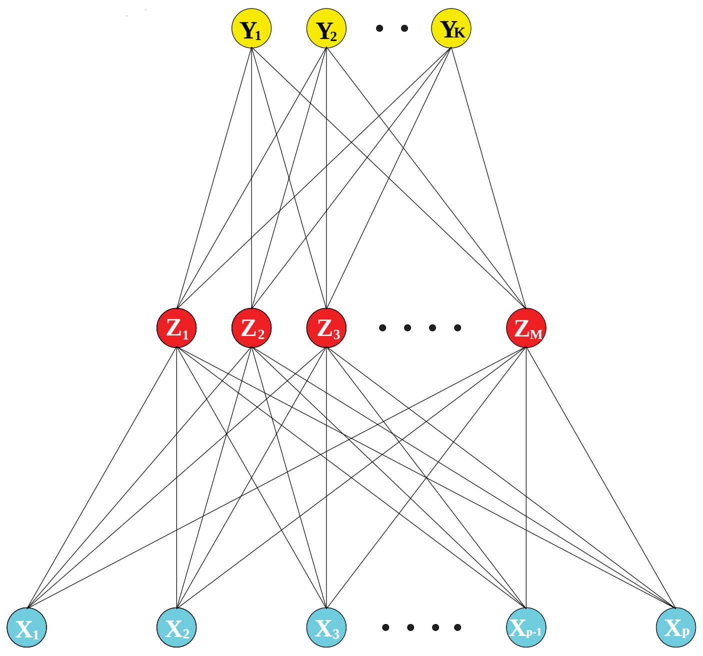


**FIGURE 11.2.** *Schematic of a single hidden layer, feed-forward neural network.*

   (a) $\beta_0, \beta_1, \ldots \beta_M$ are more weights.

3. Convert $T$ into $\hat{Y}$ by some function according to the form of $Y$

   (a) $\hat{Y} = g(T)$
   (b) If doing regression, $g$ is usually the identity function, $\hat{Y} = T$
   (c) For classification, need to predict a 0 or 1 for each observation, so need $g$ to convert $T$ into binary

4. Summarizing,

$$f(X) = g[\beta_0 + \sum_{m=1}^{M} \beta_m f_0(\alpha_{m0} + \sum_{j=1}^{p} \alpha_{mj} X_j)]$$

## 3 Classification version of NN

Goal is exploit NN structure to assign a “score” that resembles a class probability to each class for each $\mathbf{x}$ and then choose the class with the highest score.

1. Transform the problem into one of simultaneously predicting *means* for $K$ binary indicator variables rather than one mean.

   (a) Apply “one hot encoding” to $Y$ like we have done with categorical explanatories to create $Y_1, \ldots, Y_K$.
   (b) Treat 1’s and 0’s as numeric responses
   (c) Estimate the mean of each $Y_k$ as a function of $X$, just like we did before
      i. Mean *should* be between 0 and 1, like a probability
   (d) Note that probability is just the expected value of an indicator!
      i. So for each category we are modeling something like a probability when we estimate the mean of the binary indicator.

2. Need to create separate but related NN models for each

   (a) Create one set of hidden nodes, $Z_1, \ldots, Z_M$
   (b) Combine hidden nodes $K$ times with different weights $T_k = \beta_{k0} + \beta_{k1} Z_1 + \ldots + \beta_{kM} Z_M$, $k = 1, \ldots, K$
      i. Same $Z$ for all classes
      ii. Different weights $\beta_{km}$, $m = 0, 1, \ldots, M$ for each class: different combinations of the same hidden node values
      iii. Different $T_k$ for each class
      iv. Larger $T_k$ relates to a higher mean for that binary
      v. However, $\sum_{k=1}^{K} T_k \neq 1$, so $T_1, \ldots T_K$ do not form a probability distribution
   (c) Rescale $T_k$’s to add to 1

        i. $f_k(X) = g(T_k) = \exp(T_k)/\sum_{l=1}^{K}\exp(T_l)$, the SOFTMAX function.
            A. Ensures that all $f$'s are in (0,1) and that they sum to 1
            B. They look like probabilities, but they are not. They are merely relative sizes of estimated means of binary variables.
            C. Also the function that turns logits into probabilities

3. Classifier $f(x)$ is the class with the highest score among $f_k(x), k = 1, \ldots, K$

## 3.1 Notes

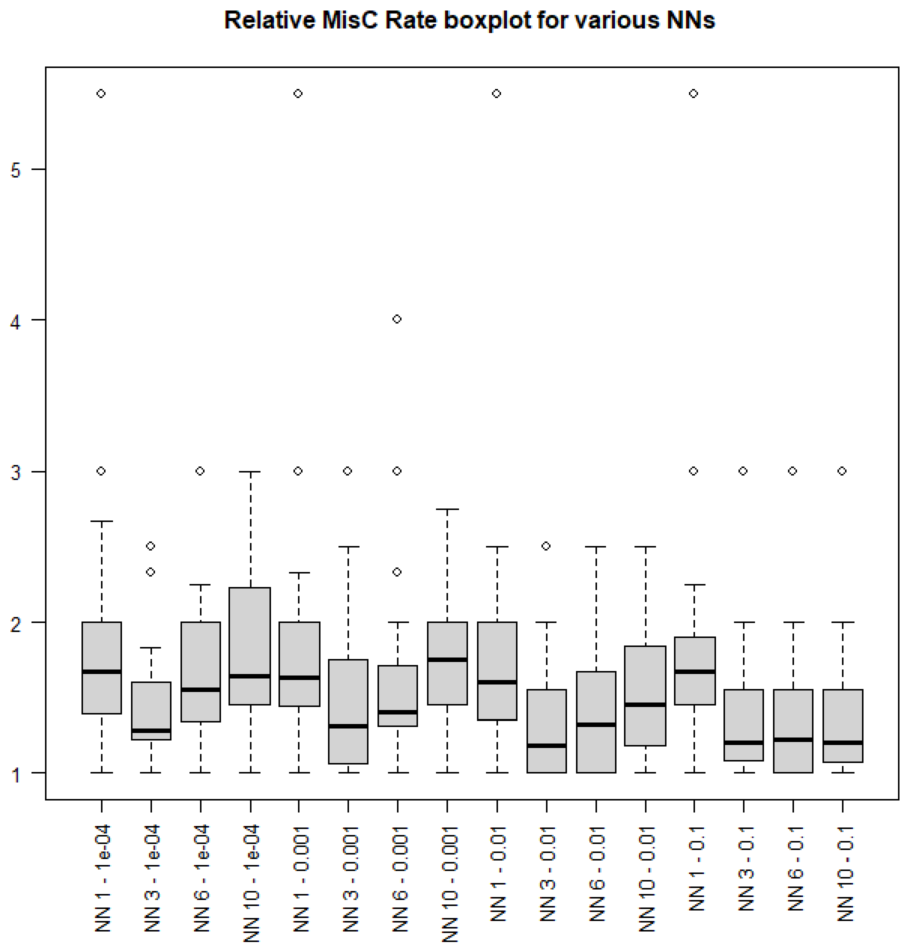
1. Since we are estimating means as before, the process of estimating weights is the same
    (a) Minimizing $RSS$!
2. Still a lot of weights being fit if $p$ and/or $K$ is at all large.
3. Same tuning parameters need to be investigated as in regression
    (a) Number of hidden nodes, $M$, that add flexibility
    (b) Weight decay or shrinkage that add regularization to prevent rapid overfitting
4. Although the `R` function `nnet()` only fits single-layer networks, it is possible to consider multi-layer networks in other software.
    (a) Can be creative in developing architectures; issues are primarily computational
        i. Extend back-propagation.
    (b) "Deep learning" is just an extremely complex NN architecture.
        i. Complications arise that need to be resolved

**Example: Neural Net for Wheat Data (L21 Neural Net Classification Wheat.R)** The same issues exist for classification versions of `nnet()` as for regression. As before, the `train()` function in `caret` can be useful, but it only runs one set of estimates at each replicate at each combination. It is probably best to tune it ourselves.

The program first runs a "default" NN with 1 node and no shrinkage. It is surprisingly not so terrible, returning a test error of 0.43, middle of the pack. I then train with `caret::train` to show how, but I don't particularly recommend this, so I will ignore it here.

Using my own code, I tune with 10-fold CV repeated twice, making 20 "fold-reps". I use 10 folds here because we have only $n = 200$ observations in the training set, which is not a large number. Larger number of folds means larger training sets and smaller validation sets, but also more computing to get all $n$ prediction errors. I replicate it twice. I also restart `nnet()` 10 times, so this means running `nnet()` 200 times for each combination of parameters. My grid is size = 1,3,6,10—reasoning that greater spacing is acceptable with larger numbers of nodes—and decay=.0001, .001, .01, .1. I got out misclassification rates for 20 fold-reps for all 16 combinations. The *relative* misclassification rates (i.e., divided by the minimum rate in each fold-rep) are plotted in Figure 2. A lot of combinations have similar

Figure 2: Basic Neural Net Architecture



error rates, there s no sharp valley requiring further exploration, and there is no obvious border issue.

The misclassification rates and very approximate confidence intervals based on the 20 fold-reps are shown below. There are numerous combinations with highly overlapping confidence intervals, so there is no single favoured combination.

```
> all.rrcv[order(RRMi),]
               RRMi  RRMi.CIl  RRMi.CIu
 [1,]   3 1e-02 1.35      1.15      1.55
 [2,]  10 1e-01 1.38      1.15      1.61
 [3,]   3 1e-01 1.38      1.15      1.61
 [4,]   6 1e-01 1.39      1.16      1.62
 [5,]   6 1e-02 1.43      1.21      1.64
 [6,]   3 1e-04 1.46      1.28      1.65
 [7,]   3 1e-03 1.48      1.22      1.73
 [8,]  10 1e-02 1.52      1.31      1.73
 [9,]   6 1e-04 1.65      1.43      1.87
[10,]   6 1e-03 1.67      1.34      2.01
[11,]  10 1e-03 1.75      1.53      1.97
[12,]  10 1e-04 1.82      1.57      2.07
[13,]   1 1e-02 1.85      1.41      2.30
[14,]   1 1e-03 1.89      1.43      2.34
[15,]   1 1e-04 1.90      1.44      2.37
[16,]   1 1e-01 1.92      1.45      2.38
```

I will try the one with lowest rate, 3 nodes and 0.01 decay. The test error for this combination is in the table below. It is the lowest rate we've seen so far!

| Method | Training | Test | Test "SE" |
|---|---|---|---|
| KNN-Opt(12) | 0.28 | 0.49 | 0.06 |
| Logistic | 0.28 | 0.40 | |
| Logist-LASSO-min | 0.28 | 0.43 | |
| LDA | 0.28 | 0.39 | |
| QDA | 0.26 | 0.43 | |
| GAM | 0.24 | 0.37 | |
| NB-normal | 0.32 | 0.39 | |
| NB-kernel | 0.23 | 0.43 | |
| Tree-min | 0.21 | 0.44 | |
| RF Default | 0 | 0.47 | |
| RF OOB-Tuned | 0 | 0.41 | |
| NeuralNet (1,0) | 0.38 | 0.43 | |
| NeuralNet(3,.01) | 0.20 | 0.35 | |

## 4 What to take away from this

1. Neural nets are very good at classification, just like they are at regression.
2. They are definitely worth considering in any problem
3. Tuning is vital, as is rescaling and restarting multiple times when the `nnet()` function is used.

## 5 Exercises

### Application

**Return to the Vehicle data used in the previous lecture. Use the same split as before.**

```
set.seed(46685326, kind="Mersenne-Twister")
perm <- sample(x=nrow(vehdata))
set1 <- vehdata[which(perm <= 3*nrow(vehdata)/4), ]
set2 <- vehdata[which(perm > 3*nrow(vehdata)/4), ]
```

We will tune neural nets on the training set and estimate error on the test set.

1. Reset the seed for the Mersenne Twister to 8236787 before tuning. Run Tom's tuning code but use 5-fold CV with 2 replicates. This way, every student should have the same CV results within `rpart`. Use the same tuning grid that I used in the example.
   (a) **Print out the matrix of misclassification rates for all parameters in all fold-reps.**
   (b) **Show the plot of relative misclassification rates.**
   (c) **Present the list of confidence intervals for the relative error rates.**
   (d) Considering all evidence, do you think any more tuning is needed? **Explain, and if you believe more is needed, what would be your next grid?**
   (e) Regardless of this answer, use the best combination of parameters from the current grid to compute test misclassification error. **Report this rate and compare it to other methods.**