Data Mining Lab 7: Neural Networks for Classification

1 Introduction

In this lab we are going to go right back to the first data set (the Titanic survival data) and do a similar analysis, but this time with a neural network.

If you need to refer to previous labs or to download the data set, they will be on the course labs website:

`http://www.maths.tcd.ie/∼louis/DataMining/`

2 Neural Networks for Classification

2.1 Getting Setup

Exercise: Download the Titanic data from the course website if you don't already have a local copy saved.
Exercise: Load the titanic data into a variable called `data` in your workspace. (Hint: see lab 2, §2.1 if you've forgotten)

```
>
```

Exercise: Load the `nnet` package, which contains the functions to build neural networks in `R`. (Hint: see lab 2, §3.1 if you've forgotten)

```
>
```

2.2 Train and Test Subsets

We are back in the nice situation of having plenty of data, so we can do the usual train and test splits in the data for model checking.
Exercise: Split the data in `data` into two variables, one called `train` and another called `test`. The split should be $\frac{1}{3}$ for the testing data and $\frac{2}{3}$ for the training data. (Hint: see lab 4, §2.3 if you've forgotten)

```
>
>
>
```

Exercise: Confirm that we did not get a freakish random split of the Survived variable. In other words, check that there are a decent number of survivors and non-survivors in both `train` and `test`. (Hint: see lab 4, §2.3 if you've forgotten)

```
>
>
```

2.3 Fitting the Neural Network

The `nnet` package has a slightly strange requirement that the target variable of the classification (ie Survived) be in a particular format. At the moment, if you look at `data$Survived` you will see it is a vector of `No`/`Yes` values, but when using `nnet()` for classification we must instead provide a two-column matrix — one column for `No` and the other for `Yes` — with a `1`/`0` as appropriate. In other words, we need to convert:

$$\begin{pmatrix} \texttt{Yes} \\ \texttt{Yes} \\ \texttt{No} \\ \texttt{Yes} \\ \texttt{No} \\ \texttt{No} \\ \vdots \end{pmatrix} \rightarrow \begin{pmatrix} 0 & 1 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 0 \\ \vdots & \vdots \end{pmatrix}$$

Fortunately there is an easy to use utility function to do this for us built-in to the package:

```
> train$Surv = class.ind(train$Survived)
> test$Surv = class.ind(test$Survived)
```

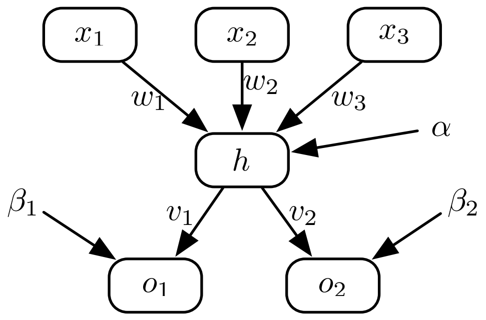
Now we are ready to fit a neural network for classification purposes:

```
> fitnn = nnet(Surv~Sex+Age+Class, train, size=1, softmax=TRUE)
> fitnn
> summary(fitnn)
```

Exercise: Draw the neural network below. Remember to label each node carefully (particularly pay attention to the fact we have categorical nodes here) and label every arc with the weights.

2.4 Calculating Probabilities

Calculating the probabilities from the neural network can seem a bit difficult, so we recap how the nnet function is doing it (which might be subtly different from lectures, but you should see the high-level similarity). Consider the following example of a classification neural network:



Then,

$$\mathbb{P}(\text{category } 1) = \frac{e^{o_1}}{e^{o_1} + e^{o_2}} \qquad \text{and} \qquad \mathbb{P}(\text{category } 2) = \frac{e^{o_2}}{e^{o_1} + e^{o_2}}$$

where

$$o_1 = v_1 h + \beta_1 \qquad , \qquad o_2 = v_2 h + \beta_2$$

and

$$h = \frac{e^{\alpha + \sum w_i x_i}}{1 + e^{\alpha + \sum w_i x_i}}$$

Exercise: Using the neural network you drew above, manually calculate the probability that an adult female passenger in first class does not survive.

2.5 Neural Network Performance

We can now look at the overall performance of the neural network by looking at a table of how predictions using the test set fare:

```
> table(data.frame(predicted=predict(fitnn, test)[,2] > 0.5,
                   actual=test$Surv[,2]>0.5))
```

How do you think it is doing? In reality the answer to this question will have an element of randomness in it due to different people's train/test splits, because once again the algorithm is not entirely stable.

2.6 Adding Decay

You have discussed in lectures how decay can help to improve the stability of neural networks. The nnet function accepts an argument called decay which implements this automatically for you (ie we can put, for example, `decay=0.02` in the `nnet()` function call). We can look at how decay affects the sum of squared errors by calculating various decay values upto 1 in a for loop.
Exercise: Fill in the missing parts of the following code and so compute the sum of squared error over a range of values between 0.0001 and 1 which is then plotted.

```
err = vector("numeric", 100)
d = seq(0.0001, 1, length.out=100)
for(i in ???) {
    fit = ???
    err[i] = sum(???)
}
plot(d, err)
```

NB: If you get obvious outliers in the y-axis direction, rerun your code until you get something resembling a curve.

2.7 The Hessian

The lectures will soon cover using the Hessian as a tool to determine how the neural network is doing. We won't use it yet, but run the following to have a look at what it looks like for the current network:

```
> fitnn = nnet(Surv~Sex+Age+Class, train, size=1, softmax=TRUE, Hess=TRUE)
> fitnn$Hess
```

The function `eigen()` in `R` can be used to find the eigenvectors and eigenvalues of a matrix. Their use will be covered in lectures:

```
> eigen(fitnn$Hess)
```

3 Comparing Neural Networks and Trees

3.1 A Disadvantage of Neural Networks

Neural networks do not, by default, pick up any interactions between variables in the data. However, you can add variable interactions manually. So, say we imagine that there is interaction between Age and Class in the Titanic data, we simply modify our `nnet()` call to:

```
> fitnn = nnet(Surv~Sex+Age+Class+Age*Class, train, size=1, softmax=TRUE)
> fitnn
> summary(fitnn)
> table(data.frame(predicted=predict(fitnn, test)[,2] > 0.5,
                   actual=test$Surv[,2]>0.5))
```

3.2 A Longer Exercise ...

Exercise: If you have time, go back to the previous labs and remember how to fit a tree to the `train` data. Do so and compare the confusion matrix to the one you obtained just now from the neural network. Which performed better without extensive tuning?