Neural Networks for Insurance Fraud Detection

Phillip Keung
Joycelin Karel
Curtis Bright

April 16, 2009

According to the Insurance Information Institute, auto insurance fraud in 2007 amounted to $4.8 billion to $6.8 billion of claims in America alone. However, looking for fraud with claims adjusters is expensive and time-consuming. The application of statistical tools for the detection of fraud can help focus the insurer's resources and reduce the number of claims adjusters required. Furthermore, sophisticated models may be able to determine connections between variables that a human being would not have considered. Although neural networks are not within the scope of this course, the team decided that a neural network would be well-suited to the problem of finding dependencies and patterns in the data set. This report discusses a 2 hidden layer feedforward neural network created to analyze fraud data.

Neural networks were inspired by discoveries in biology. Researchers hypothesized that the brain is essentially a large neural network. Connections between adjacent neurons strengthened or weakened based on the amount of relevant stimulation that the neurons received, and connected neurons fire to induce other neurons to fire. This insight provided a neural network which could be mathematically modelled, and has been used extensively in machine learning and classification.

A neural network was implemented in R using 1994 auto insurance data from an unnamed insurer as a training set. The data set was adjusted before training began: many of the timing-related columns were summarized with a single "Days between accident and claim" column. The categorical data was converted into binary variates and the timing variate was entered as an

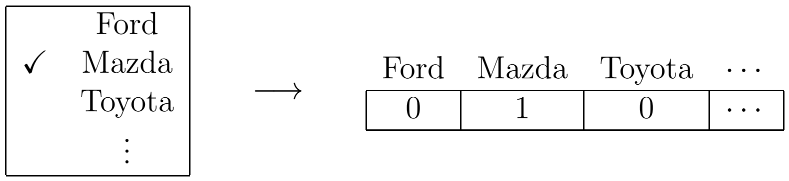

Table 1: Converting the variate ‘Make’ into binary variates

integer. (Table 1) To reduce the number of input nodes required, variates were grouped under the following labels: timing, demographics, policy, rating and socioeconomic status. Each node of the neural network took inputs, and had a weight for each input. The weights were set between -1 and 1 to reflect how certain factors can increase or decrease the probability of fraud. These weights can be interpreted as the importance of each input, as large weights affect the output significantly. The inner product of the inputs and the weights is called the activation value, denoted R.

$$R = \sum_{i=1}^{n} (\text{weight}_i) \times (\text{input}_i)$$

Each neuron outputted a number between 0 and 1 as its response to the inputs, calculated using

$$\text{Output} = \frac{1}{1 + e^{-R}}.$$

The ideal set of weights, or ‘brain’, of the network is the vector of 160 real numbers between -1 and 1 that maximizes the average precision of the network. As the brain of the network is a vector in $\mathbb{R}^{160}$, Newton’s method and other forms of deterministic optimization were deemed unfeasible. They are too sensitive to choice of initial parameters, and no good estimates were available. Instead, a genetic algorithm was used to construct a decent, if not optimal, brain. Constrained Newton-type optimization (i.e., parameters restricted to the $[-1, 1]^{160}$ hypercube) was also attempted using the genetic algorithm brain as a starting point. Unfortunately, the constrained optimization requires the calculation of the Hessian to build up a picture of the surface being optimized, and function evaluations (i.e., the neural network evaluating a set of data) were so computationally expensive that 48 hours of computing time was insufficient to find a solution within default tolerance.

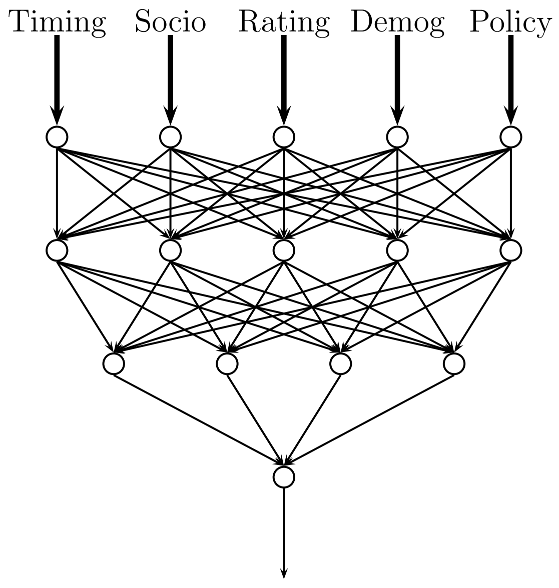

Figure 1: A schematic of the implemented network

The neural network, after 460 generations through the genetic algorithm, had an average precision of 27.7% on the training data set. This is a great improvement over the 6.7% average precision that would be expected with random ranking (note that the proportion of fraud in the training data is 6.7%.) The team members eagerly await the announcement of the effectiveness of their neural network on the test data set.

APPENDIX A: R Code

```
# load genetic algorithm library
library(genalg)

# returns a number for every observation
# this number is used to rank the observation
# against other observations in the data set
neuralNet<-function(dataMatrix,params){
  returnMe<-c()
  for(i in 1:6141){
```

```
# group the data under labels
timingData<-as.numeric(dataMatrix[i,c(3,64:71,103:106)])
socioData<-as.numeric(dataMatrix[i,c(4:20,31:39,56:59,76:83,107:110)])
ratingData<-as.numeric(dataMatrix[i,c(21,22,29,30,60:63,72:75,93:96)])
demogData<-as.numeric(dataMatrix[i,c(23:28,84:92)])
policyData<-as.numeric(dataMatrix[i,c(40:55,97:102,111:113)])
# parameters on the input nodes
ptiming<-params[1:13]
pdemog<-params[14:28]
psocio<-params[29:70]
ppolicy<-params[71:95]
prating<-params[96:111]
# parameters on first hidden layer nodes
pl11<-params[112:116]
pl12<-params[117:121]
pl13<-params[122:126]
pl14<-params[127:131]
pl15<-params[132:136]
# parameters on second hidden layer nodes
pl21<-params[137:141]
pl22<-params[142:146]
pl23<-params[147:151]
pl24<-params[152:156]
# parameters on the output node
pout<-params[157:160]
# feed data and parameters into input nodes
timing<-inputNode(timingData,ptiming)
demog<-inputNode(demogData,pdemog)
socio<-inputNode(socioData,psocio)
policy<-inputNode(policyData,ppolicy)
rating<-inputNode(ratingData,prating)
toLayer1<-c(timing,demog,socio,policy,rating)
# feed data and parameters into first hidden layer
l11<-layer1(toLayer1,pl11)
l12<-layer1(toLayer1,pl12)
l13<-layer1(toLayer1,pl13)
l14<-layer1(toLayer1,pl14)
l15<-layer1(toLayer1,pl15)
toLayer2<-c(l11,l12,l13,l14,l15)
# feed data and parameters into second hidden layer
l21<-layer2(toLayer2,pl21)
l22<-layer2(toLayer2,pl22)
l23<-layer2(toLayer2,pl23)
l24<-layer2(toLayer2,pl24)
# feed data and parameters into output node
```

```
    toOutput<-c(l21,l22,l23,l24)
    out<-output(toOutput,pout)
    returnMe<-c(returnMe,out)
  }
  return(returnMe)
}

# the nodes below simply return the inner product
inputNode<-function(datavec,paramvec){
  return(1/(1+exp(-sum(datavec*paramvec))))
}

layer1<-function(datavec,paramvec){
  return(1/(1+exp(-sum(datavec*paramvec))))
}

layer2<-function(datavec,paramvec){
  return(1/(1+exp(-sum(datavec*paramvec))))
}

output<-function(datavec,paramvec){
  return(1/(1+exp(-sum(datavec*paramvec))))
}

# the avgp function is bundled inside because
# genalg minimizes the objective function
optimizeMe<-function(optimParam){
  return(-1*avgp(fraud,neuralNet(binData,optimParam)))
}

binData<-read.csv("claims.csv")
binData<-as.matrix(binData[1:6141,])
fraud<-as.numeric(binData[,2])
# this might take a while to run...
initialGuess<-rbga(stringMin=rep(-1,160), stringMax=rep(1,160),
 suggestions=list(initialGuess),popSize=60, iters=460,
 evalFunc=optimizeMe)$population[1,]
predictions<-neuralNet(binData,initialGuess)
```