NATIONAL INSTITUTE OF TECHNOLOGY, KARNATAKA

DWM ASSIGNMENT

NEURAL NETWORK CLASSIFICATION

Submitted to:

Dr. M. Venkatesan

Dept. of Computer Science and Engineering

National Institute of Technology, Karnataka

By:

Ganavi G (13CO118)

Aditya Kumar (13CO108)

Rakesh Kumar (13CO135)

OBJECTIVE

The key objectives of the assignment is to:

i) Understand the basic principles of the working of neural networks,
ii) See how data mining can be merged with neural networks to open up new potential possibilities,
iii) Do a reading on the previous works and come up with a brief literature survey,
iv) Gain an hands-on experience with the R language used for statistical computing,
v) Use RStudio to classify a forest cover dataset into different types of forest,
vi) Try out several combination of attributes and come up with a neural network model for each followed by calculation of accuracy that serves as a benchmark metric.

RELATED WORK

The simplest definition of a neural network, more properly referred to as an 'artificial' neural network (ANN), is provided by the inventor of one of the first neuro computers, Dr. Robert Hecht-Nielsen. He defines a neural network as: "a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs".

Data mining, also popularly known as knowledge discovery in databases, refers to the process of automated extraction of hidden, previously unknown and potentially useful information from large databases. Classification is one of the data mining problems receiving great attention recently in the database community. A neural network-based approach to mining classification rules from given databases has been proposed by Setiono R and Liu. The network is first trained to achieve some required accuracy rate. Redundant connections of the network are then removed by a network pruning algorithm. The activation values of the hidden nodes in the network are analysed, and classification rules are generated using the result of this analysis.

Craven and Shavlik have come up with two classes of approaches for data mining with ANNs. The first approach, often called rule extraction, involves extracting symbolic models from trained neural networks. The second approach is to directly learn simple, easy-to-understand networks.

Saito and Nakano have proposed a medical diagnosis expert system based on a multilayer ANN. They treated the network as a black box and used it only to observe the effects on the network output caused by change the inputs.

Two methods for extracting rules from ANN are described by Towell and Shavlik. The first method is the subset algorithm, which searches for subsets of connections to a node whose summed weight exceeds the bias of that node. The major problem with subset algorithms is that the cost of finding all subsets increases as the size of the ANNs increases. The second method, the MofN algorithm, is an improvement of the subset method that is designed to explicitly search for M-of-N rules from knowledge based ANNs. Instead of considering an ANN connection, groups of connections are checked for their contribution to the activation of a node, which is done by clustering the ANN connections.

Anbananthen and Sainarayanan have proposed a data mining approach that makes use of pruned artificial neural network tree (ANNT). ANNT pruning approach consists of three phases: training, pruning and rule extraction. It improved the generalization ability of the network and the number of rules extracted is reduced.

The combination of data mining method and neural network model can greatly improve the efficiency of data mining techniques, and has been widely used. Given the current state of the art, neural network deserves a place in the tool boxes of data mining specialists.

ALGORITHM

Neural network classification algorithms are inspired by the human brain. Neural network are a collection of neural nodes (artificial neurons). It has electronic networks of neurons node as on the neural structure of the brain. Each neuron nodes takes several binary inputs, x1,x2,...x1,x2,..., and produces a output(Use summation function ∑wjxj). They process records one at a time, and learn by comparing their classification of the record (i.e., largely arbitrary) with the known actual classification of the record. The errors from the initial classification of the first record is fed back into the network, and used to modify the networks algorithm for further iterations. We are using sigmoid neurons since it is similar to perceptrons, but modified so that small changes in their weights and bias cause only a small change in their output.

A neural network consists of:

- A set of nodes (neurons) or units connected by links
- A set of weights associated with links
- A set of thresholds or levels of activation

Feedforward Neural Networks:

- Each unit is connected only to that of the next layer.
- The processing proceeds smoothly from the input unit to output.
- There is no feedback (directed acyclic graph or DAG).
- They have no internal states.

What is a Perceptron?

Each neuron nodes takes several binary inputs, x1,x2,...x1,x2,..., and produces a single output (Use summation function ∑wjxj). Perceptrons are basic concept of artificial neural network for decision making. If we want neural networks to be used as one of the learning algorithms used within machine learning, we need a small change in some weight (or bias) in the network, the reason being, we use sigmoid neurons.

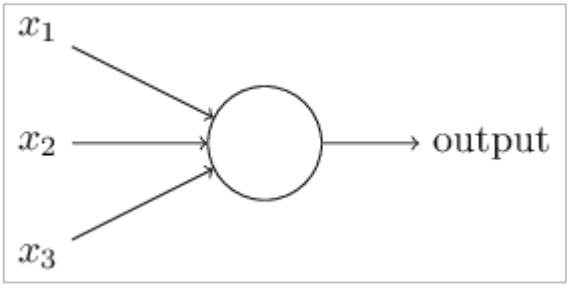


Fig. a: Perceptron

A perceptron can represent all boolean primitive functions AND, OR, NAND, NOR.

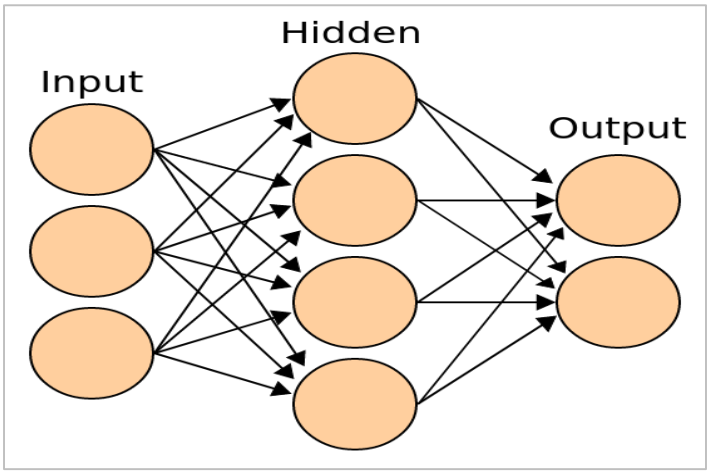
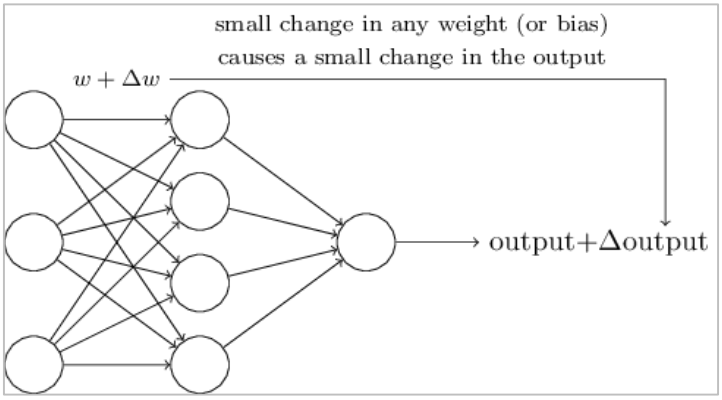


Fig. b: Complex network of perceptrons

Sigmoid Neurons

We'd like the network to learn weights and biases so that the output from the network performs the necessary classification. To see how learning might work, suppose we make a small change in some weight (or bias) in the network, what we'd like is for this small change in weight to cause only a small corresponding change in the output from the network.



It's σ(w·x+b), where σ is called the sigmoid function.

$$\sigma(z) \equiv \frac{1}{1+e^{-z}}.$$

More explicit output of a sigmoid neuron with inputs x1, x2,...x3, x4,..., weights w1,w2,...w3,w4,..., and bias b is

$$\frac{1}{1+\exp(-\sum_j w_j x_j - b)}.$$

The smoothness of σ means that small changes Δwj in the weights and Δbin the bias will produce a small change Δoutput.

$$\Delta \text{output} \approx \sum_j \frac{\partial \,\text{output}}{\partial w_j}\Delta w_j + \frac{\partial \,\text{output}}{\partial b}\Delta b,$$

where, ∂output/∂wj and ∂output/∂b denote partial derivatives of the output.

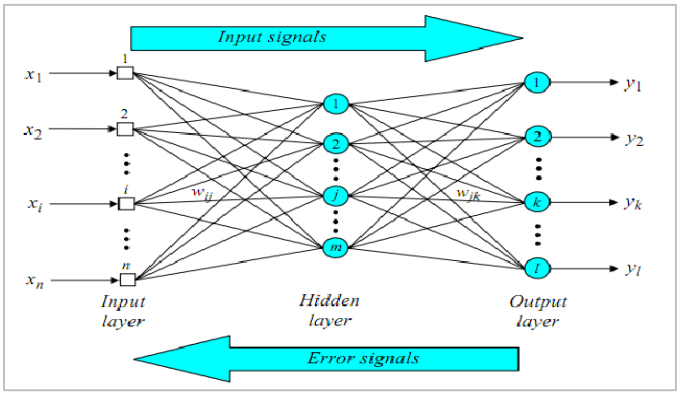
Neuron Error

The rule commonly used to adjust the weights of a neuron is the delta rule or Widrow-Hoff rule. Let x = (x1, ...,xn) provided the input to the neuron. If t and y are, respectively, the desired output and the output neural, the error δ is given by δ=t−y.

The delta rule states that the change in the general weight Δwi is: Δwi=ηδxi where η ∈ [0, 1] is learning rate. The learning rate determines the learning speed of the neuron.

Back propagation principle

The use of the sigmoid function gives the extra information necessary for the network to implement the back-propagation training algorithm. Back-propagation works by finding the squared error (the error function) of the entire network, and then calculating the error term for each of the output and hidden units by using the output from the previous neuron layer.



The training process normally uses some variant of the Delta Rule, which requires the inputs, the output, and the actual output and the processing element which starts with the calculated difference between the actual outputs and the desired outputs. Using this error, connection weights are increased in proportion to the error times. The most complex part of this algorithm is determining which input contributed the most to an incorrect output and how the input must be modified to correct the error. (An inactive node would not contribute to the error and would have no need to change its weights.) To solve this problem, training inputs are applied to the input layer of the network, and desired outputs are compared at the output layer. During the learning process, a forward sweep is made through the network, and the output of each element is computed by layer. The difference between the output of the final layer and the desired output is back-propagated to the previous layer(s), usually modified by the derivative of the transfer function. The connection weights are normally adjusted using the Delta Rule. This process proceeds for the previous layer(s) until the input layer is reached.

Back Propagation Algorithm Steps

- STEP 1
  Normalize the input and output with respect to maximum value.
- STEP 2
  Let l be number of input, m be number of neurons in hidden layer, [V] be weights connecting input neurons to hidden layer and [W] weights connecting hidden layer to output layer. Initialize weight to random (-1 to 1). Let $\lambda = 1$ and threshold = 0.00001.
- STEP 3
  Calculate the output of Input Layer.
- STEP 4
  Calculate the input to Hidden Layer.
- STEP 5
  Calculate the output of Hidden Layer.
- STEP 6
  Calculate the input to Output Layer.
- STEP 7
  Calculate the output of Output Layer.
- STEP 8
  Calculate the error.
- STEP 9
  Find ΔW.
- STEP 10
  Find ΔV.
- STEP 11
  Calculate $[V]_{t+1} = [V]_t + [\Delta V]_{t+1}$ and $[W]_{t+1} = [W]_t + [\Delta W]_{t+1}$.

Repeat steps 3 to 11 till error is less than the threshold value.

DATASET

1. Title

   Forest Covertype data

2. Sources

   Jock A. Blackard (jblackard 'at' fs.fed.us)

   USFS - Forest Inventory & Analysis

   Rocky Mountain Research Station

   Colorado State University

3. Description

   Natural resource managers responsible for developing ecosystem management strategies require basic descriptive information including inventory data for forested lands to support their decision-making processes. However, managers generally do not have this type of data for inholdings or neighbouring lands that are outside their immediate jurisdiction. One method of obtaining this information is through the use of predictive models. The study area included four wilderness areas found in the Roosevelt National Forest of northern Colorado. A total of twelve cartographic measures were utilized as independent variables in the predictive models, while seven major forest cover types were used as dependent variables. Several subsets of these variables were examined to determine the best overall predictive model.

4. Number of observations

   581,012 with no attributes missing

5. Number of attributes

   12 measures, but 54 columns of data (10 quantitative variables, 4 binary wilderness areas and 40 binary soil type variables)

6. Attribute Information

   The 12 different measures are:

   i) Elevation
   ii) Aspect
   iii) Slope
   iv) Horizontal distance to water
   v) Vertical distance to water
   vi) Horizontal distance to roadways
   vii) Hill shade at 9 am
   viii) Hill shade at noon
   ix) Hill shade at 3pm
   x) Horizontal distance to wildfire ignition points
   xi) Soil type
   xii) Cover type

7. Output attributes

There are seven different classes of output or forest cover type:

i) Spruce/Fir
ii) Lodgepole Pine
iii) Ponderosa Pine
iv) Cottonwood/Willow
v) Aspen
vi) Douglas-fir
vii) Krummholz

8. Distribution of class

Number of records of Spruce-Fir:	211840
Number of records of Lodgepole Pine:	283301
Number of records of Ponderosa Pine:	35754
Number of records of Cottonwood/Willow:	2747
Number of records of Aspen:	9493
Number of records of Douglas-fir:	17367
Number of records of Krummholz:	20510

ALGORITHMIC WORKOUT

Code:

//Read the dataset from .data file. ‘,’ serves as the delimiter

```
dataset <- read.table(file="covtype.data", sep=",", fill=FALSE, strip.white = TRUE)
```

//Converting integer to categorical binary format

```
ideal <- class.ind(dataset$V55)
```

//Applying Z-score normalization

```
datasetNormZ <- as.data.frame(scale(dataset[2:10]))
datasetF = cbind(datasetNormZ,dataset[11:54])
```

//Dividing the dataset into training set and test set with 7:3 ratio

```
datasetrain <- sample(1:581012,406708)
datasetest <- setdiff(1:581012,datasetrain)
trainX = datasetF[datasetrain,]
trainY = ideal[datasetrain,]
testX = datasetF[datasetest,]
testY = datasetF[datasetest,]
```

//Creating the neural network with one hidden layer of size 16 over an iteration of 200

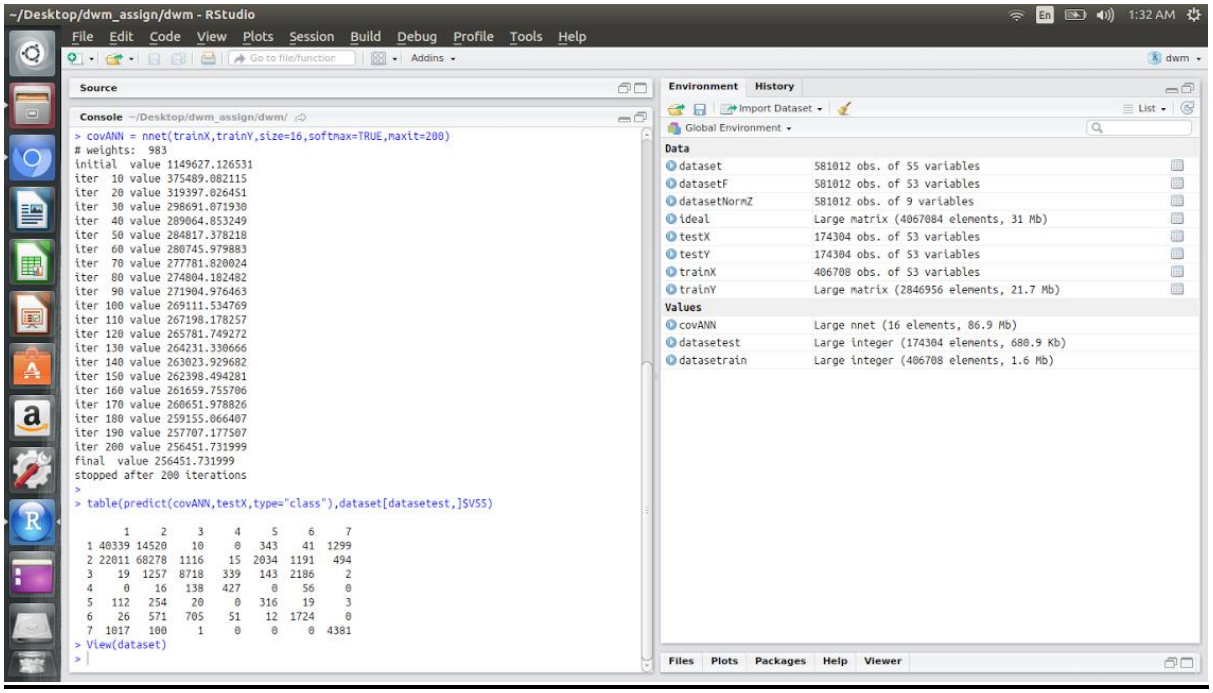
```
covANN = nnet(trainX,trainY,size=16,softmax=TRUE,maxit=200)
```

//Plotting a table between predicted results and the actual results

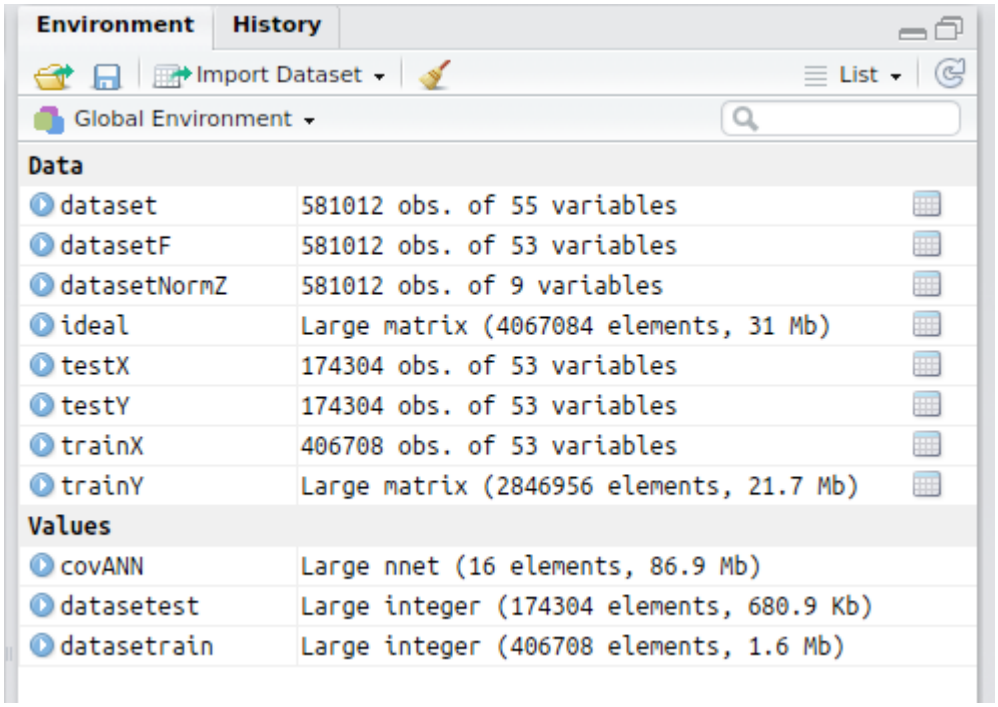
```
table(predict(covANN,testX,type="class"),dataset[datasetest,]$V55)
```

IMPLEMENTATION

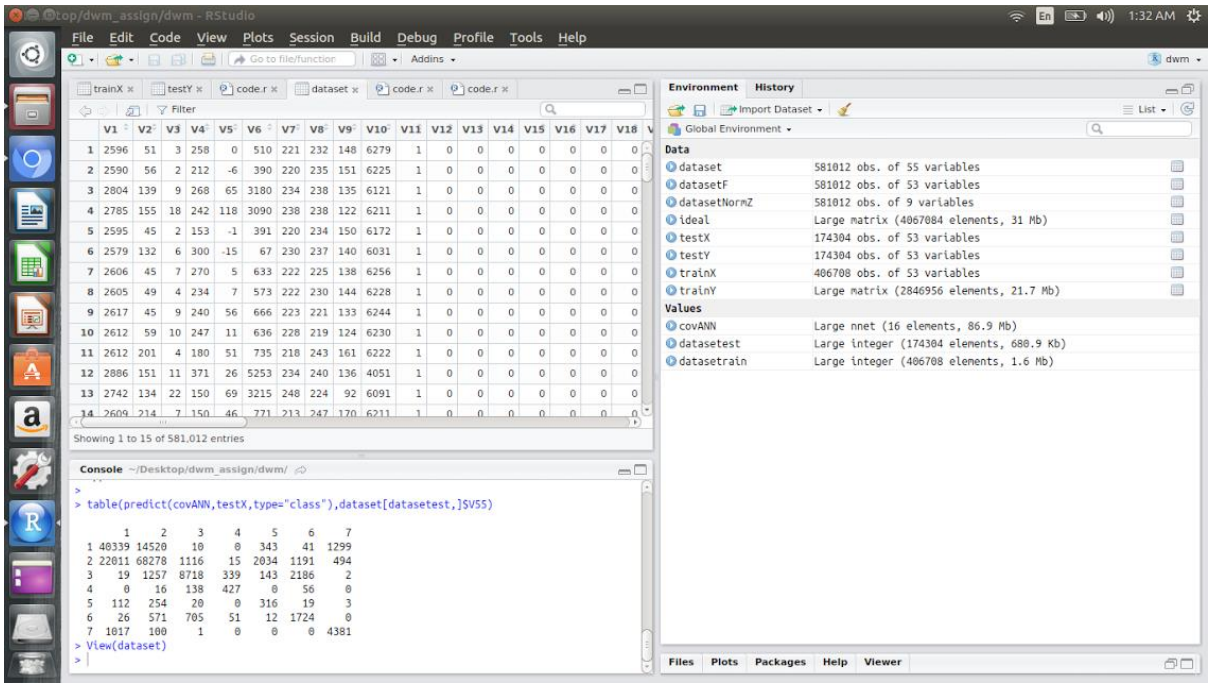
Command Window



Workspace



Viewing the dataset (and other variables)



CONCLUSION

The following image shows the training of the dataset:

```
# weights:  983
initial  value 1149627.126531
iter  10 value 375489.082115
iter  20 value 319397.026451
iter  30 value 298691.071930
iter  40 value 289064.853249
iter  50 value 284817.378218
iter  60 value 280745.979883
iter  70 value 277781.820024
iter  80 value 274804.182482
iter  90 value 271904.976463
iter 100 value 269111.534769
iter 110 value 267198.178257
iter 120 value 265781.749272
iter 130 value 264231.330666
iter 140 value 263023.929682
iter 150 value 262398.494281
iter 160 value 261659.755706
iter 170 value 260651.978826
iter 180 value 259155.066407
iter 190 value 257707.177507
iter 200 value 256451.731999
final  value 256451.731999
stopped after 200 iterations
```

Plotting predicted values against the actual values:

```
> table(predict(covANN,testX,type="class"),dataset[datasetest,]$V55)

       1     2     3     4     5     6     7
  1 40339 14520    10     0   343    41  1299
  2 22011 68278  1116    15  2034  1191   494
  3    19  1257  8718   339   143  2186     2
  4     0    16   138   427     0    56     0
  5   112   254    20     0   316    19     3
  6    26   571   705    51    12  1724     0
  7  1017   100     1     0     0     0  4381
>
```

The above table is interpreted as:

- Forest cover type 1 matches at 40339 instances of the actual results in the test set.
- Forest cover type 2 matches at 68278 instances of the actual results in the test set.
- Forest cover type 3 matches at 8718 instances of the actual results in the test set.
- Forest cover type 4 matches at 427 instances of the actual results in the test set.
- Forest cover type 5 matches at 316 instances of the actual results in the test set.
- Forest cover type 6 matches at 1724 instances of the actual results in the test set.
- Forest cover type 7 matches at 4381 instances of the actual results in the test set.

Ideally, for an accuracy of 100% we should have had a table with only the left-right diagonal elements filled and zero in the other cells. However, accuracy depends on many more factors like the amount of available data, data augmentation and distribution of data.

In the table above, we can see that there are 22011 predicted instances of type 2 forest cover matching with that of forest cover 1. This is an incorrect prediction.

$$Accuracy = \frac{\text{Number of correctly matched instances}}{\text{Total number of instances in the test set}}$$

This formula applies to our data since our dataset doesn't have missing values.

Number of correctly matched instances

= 40339 + 68278 + 8718 + 427 +316 + 1724 + 4381

= 124183

Total number of instances

= Size of the test set

= 174304

Accuracy = (124183 / 174304)*100

= *71.25% (rounded to two decimals)*

We have achieved an accuracy of 71.25% which is close to what is mentioned in the dataset description (70%).

Please note that all the attributes except for the instance ID and forest cover type have been taken into consideration for the prediction.