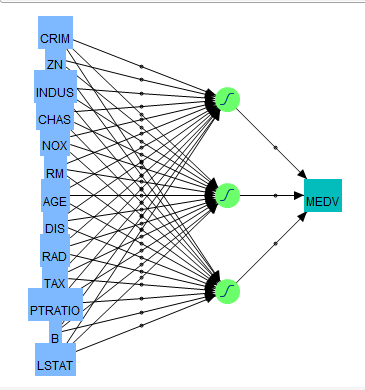
**7 - Neural Networks for Regression**

**7.1 - What is a neural network?**A ***neural network*** is a mathematical model that attempts to mimic the neurons found in the human brain. The diagram below shows the connection between the neuron in the brain and the key feature of the neural network model.



Diagram for a neural network for predicting the median home value in a census tract from the Boston Housing Data is shown below.



The neural network regression model looks like this:

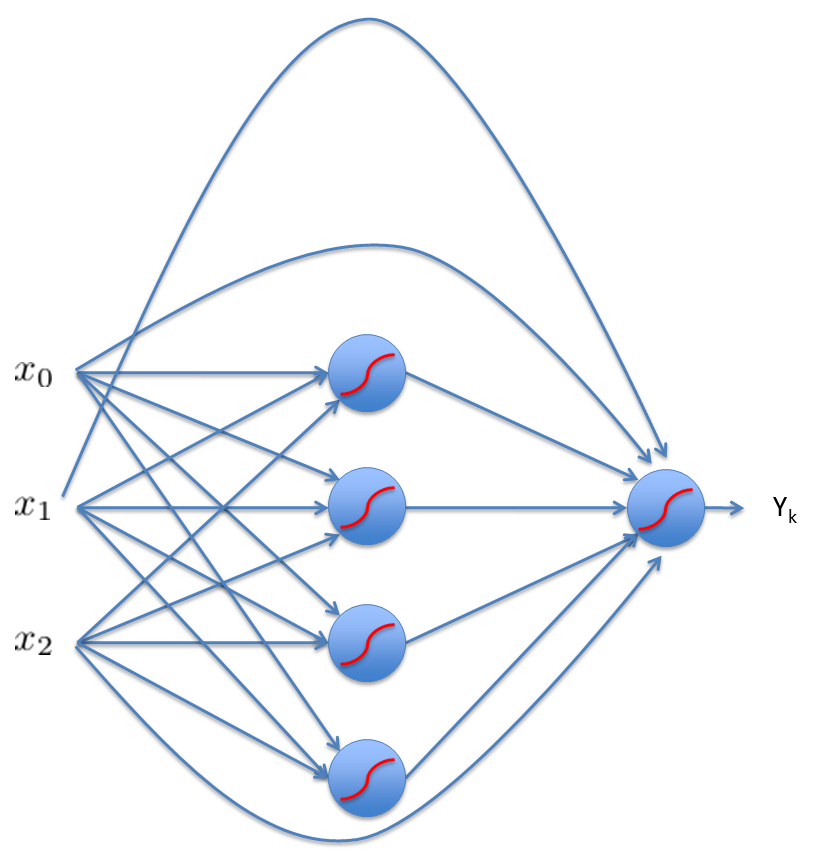
The neural network model above has one hidden layer with 3 nodes, but it is possible to have additional hidden layers and more nodes per layer.

The functions used are usually chosen from the following choices.

***Activation Functions -***



The hidden layer squash function, , that is used by JMP is the hyperbolic tangent function nnet in R uses the logistic activation function for the hidden layers. For regression problems, it is common to include a skip-layer to the neural network. Also for regression problems it is important that the final outputs be linear as we don’t want to constrain the predictions to be between 0 and 1. A simple diagram of a skip-layer neural network is shown below.



The equation for the **skip-layer neural network** for regression is shown below.

It should be clear that these models are highly parameterized and thus will tend to over fit the training data. Cross-validation is therefore critical to make sure that the predictive performance of the neural network model is adequate.

**7.2 – Neural Networks in R**  
The library nnet contains a basic neural network function, nnet().

Help File for nnet command in nnet library.

**Fit Neural Networks**

**Description:**

Fit single-hidden-layer neural network, possibly with skip-layer connections.

**Usage:**

nnet(x, ...)

## S3 method for class 'formula':

nnet(formula, data, weights, ...,

subset, na.action = na.fail, contrasts = NULL)

## Default S3 method:

nnet(x, y, weights, size, Wts, mask,

linout = FALSE, entropy = FALSE, softmax = FALSE,

censored = FALSE, skip = FALSE, rang = 0.7, decay = 0,

maxit = 100, Hess = FALSE, trace = TRUE, MaxNWts = 1000,

abstol = 1.0e-4, reltol = 1.0e-8, ...)

**Arguments:**

**formula:** A formula of the form 'class ~ x1 + x2 + ...'

**x:** matrix or data frame of 'x' values for examples.

**y:** matrix or data frame of target values for examples.

**weights:** (case) weights for each example - if missing defaults to 1.

**size:** number of units in the hidden layer. Can be zero if there are only skip-layer units.

**data:** Data frame from which variables specified in 'formula' are

preferentially to be taken.

**subset:** An index vector specifying the cases to be used in the training sample.   
 (NOTE: If given, this argument must be named.)

**na.action:** A function to specify the action to be taken if 'NA's are

found. The default action is for the procedure to fail. An

alternative is na.omit, which leads to rejection of cases

with missing values on any required variable. (NOTE: If

given, this argument must be named.)

**linout:** switch for linear output units. Default logistic output units. (must be true for regression!)

**entropy:** switch for entropy (= maximum conditional likelihood) fitting. Default by least-squares.

**skip:** switch (T or F) to add skip-layer connections from input to output.

**decay: parameter for weight decay. Default 0.**

**maxit: maximum number of iterations. Default 100.**

**MaxNWts: The maximum allowable number of weights. There is no**

**intrinsic limit in the code, but increasing 'MaxNWts' will**

**probably allow fits that are very slow and time-consuming**

**(and perhaps uninterruptable under Windows).**

**abstol: Stop if the fit criterion falls below 'abstol', indicating an**

**essentially perfect fit.**

**reltol: Stop if the optimizer is unable to reduce the fit criterion**

**by a factor of at least '1 - reltol'.**

**...: arguments passed to or from other methods.**

**Details:**

If the response in 'formula' is a factor, an appropriate classification network is constructed; this has one   
 output and entropy fit if the number of levels is two, and a number of outputs equal to the number of   
 classes and a softmax output stage for more levels. If the response is not a factor, it is passed on

unchanged to 'nnet.default'.

Optimization is done via the BFGS method of 'optim'.

**Value:**

**object of class '"nnet"' or '"nnet.formula"'. Mostly internal**

**structure, but has components**

**wts:** the best set of weights found

**value:** value of fitting criterion plus weight decay term.

**fitted.values:** the fitted values for the training data.

**residuals:** the residuals for the training data.

**Example 7.1: Boston Housing Data**

> set.seed(5555) 🡨 because search algorithm for weights has randomness to it.  
> library(nnet)  
> boston = read.table(file.choose(),header=T,sep=”,”)

> boston.trans = boston

> boston.trans$CMEDV=log(boston$CMEDV)

> boston.trans$CRIM =log(boston$CRIM)

> boston.trans$LSTAT =log(boston$LSTAT)

> boston.trans$ZN =log(boston$ZN+1)

> names(boston.trans)



> bos.nn = nnet(CMEDV~.,data=boston.trans,size=10,linout=T,skip=T,

maxit=10000,decay=.001)

> bos.nn = nnet(CMEDV~.,data=boston.trans,size=10,linout=T,skip=T,maxit=10000,decay=.001)

# weights: 164

initial value 78675.379024

iter 10 value 31079.231939

iter 20 value 5055.735379

iter 30 value 63.789628

iter 40 value 21.833956

iter 50 value 20.382927

iter 60 value 19.292936

iter 70 value 19.082762

iter 80 value 18.951263  
...

iter1900 value 3.463181

iter1910 value 3.461174

iter1920 value 3.460608

iter1930 value 3.460298

iter1940 value 3.460228

final value 3.460206

converged

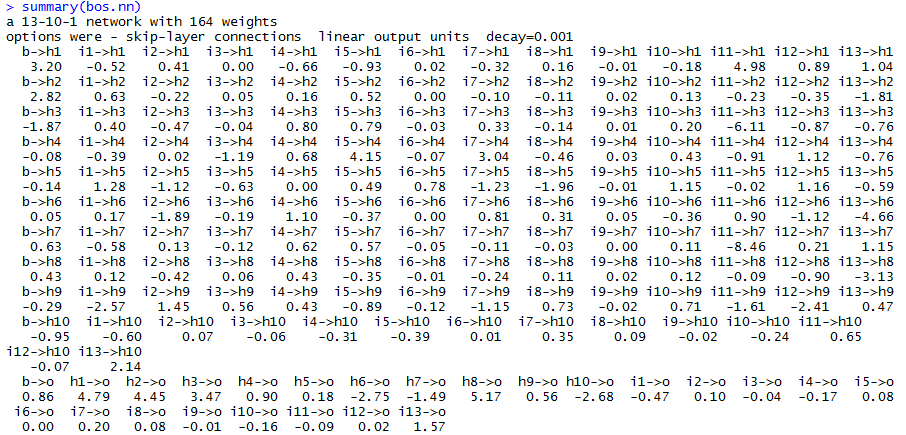
Here the following options have been chosen in fitting bos.nn:

* 10 units for the hidden layer
* linear output (necessary for regression problems using nnet) (linout = T)
* use a neural network with skip layer units (skip = T)
* set maximum number of iterations to a large number to “guarantee” convergence.
* decay = generally works “better” than the default = 0.

> summary(bos.nn)  
a 13-10-1 network with 164 weights

options were - skip-layer connections linear output units decay=1e-04

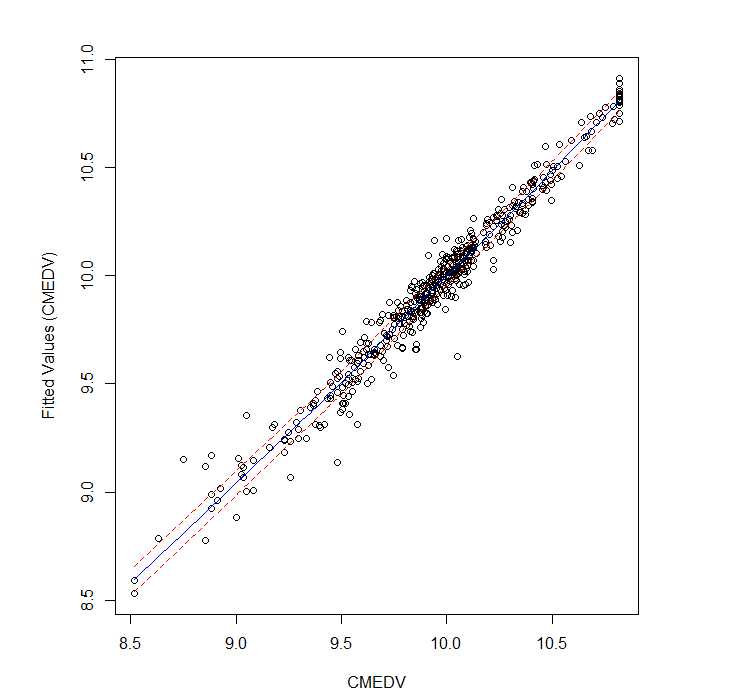
**Weights:**



Recall the skip-layer neural network regression model looks like this:

What are the following weights/parameters from the output above?

> trendscat(boston.trans$CMEDV,fitted(bos.nn),xlab="CMEDV", ylab="Fitted Values (CMEDV)")



> cor(boston.trans$CMEDV,fitted(bos.nn))^2

[,1]

[1,] 0.963475893 🡨

Probably the best R-squares from all the flexible regression methods we have examined, however this model almost certainly over fits the training data.  If we think of the weights as parameters to be estimated this model essentially uses 164 degrees of freedom! Cross-validation or an estimate of prediction accuracy (RMSEP, etc.) is a must.    
 **MC Split-Sample CV Function for Regression Neural Network Models**

nnet.sscv = function(x,y,fit,p=.667,B=100,size=3,decay=fit$decay,skip=T,  
linout=T,maxit=10000){  
 n = length(y)

MSEP = rep(0,B)

MAEP = rep(0,B)

MAPEP = rep(0,B)

ss = floor(n\*p)  
 for (i in 1:B){

sam = sample(1:n,ss,replace=F)

fit2 = nnet(x[sam,],y[sam],size=size,linout=linout,skip=skip,decay=decay,  
 maxit=maxit,trace=F)  
 ynew = predict(fit2,newdata=x[-sam,])

MSEP[i]=mean((y[-sam]-ynew)^2)

MAEP[i]=mean(abs(y[-sam]-ynew))

MAPEP[i]=mean(abs(y[-sam]-ynew)/y[-sam])

}

RMSEP = sqrt(mean(MSEP))

MAE = mean(MAEP)

MAPE = mean(MAPEP)

cat("RMSEP\n")

cat("===============\n")

cat(RMSEP,"\n\n")

cat("MAE\n")

cat("===============\n")

cat(MAE,"\n\n")

cat("MAPE\n")

cat("===============\n")

cat(MAPE,"\n\n")

temp = data.frame(MSEP=MSEP,MAEP=MAEP,MAPEP=MAPEP)

return(temp)

}

Form matrix of predictors/inputs and the response vector (**y**).

> bos.x = boston.trans[,-1]

> bos.y = boston.trans[,1]

> results.nn10 = nnet.sscv(bos.x,bos.y,bos.nn,size=10)

RMSEP

===============

0.2290404665

MAE

===============

0.1445892243

MAPE

===============

0.01463486423

5 hidden nodes (*h = 5*)

> results.nn5 = nnet.sscv(bos.x,y,bos.nn,size=5)

RMSEP

===============

0.1826515418

MAE

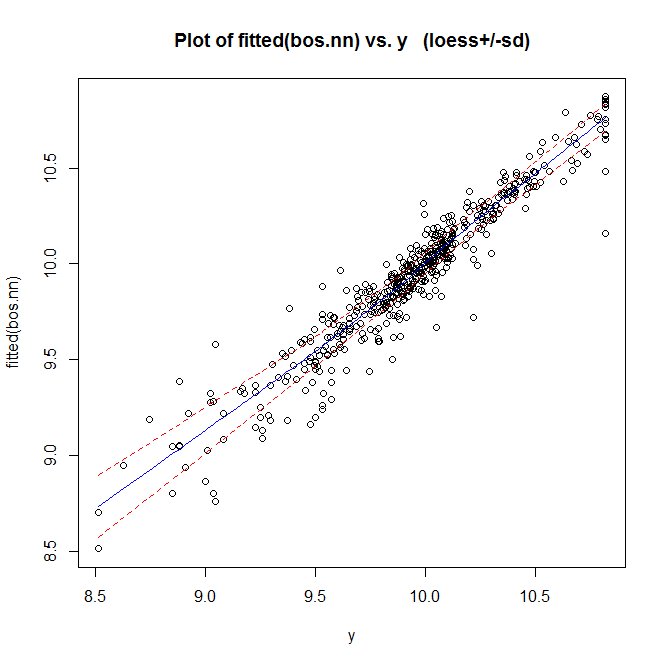
===============

0.1240026842

MAPE

===============

0.01258781841



Predicted values vs. Actual

*Neural Network with h = 5*

> trendscat(boston.trans$CMEDV,fitted(bos.nn5))

> cor(boston.trans$CMEDV,fitted(bos.nn5))^2

[,1]

[1,] 0.8932313038 🡨

3 hidden nodes (*h = 3*)  
> bos.nn3 = nnet(CMEDV~.,data=boston.trans,size=3,linout=T,skip=T,  
decay=0.001,maxit=10000)

> results.nn3 = nnet.sscv(bos.x,bos.y,bos.nn,size=3)

RMSEP

===============

0.1788908047

MAE

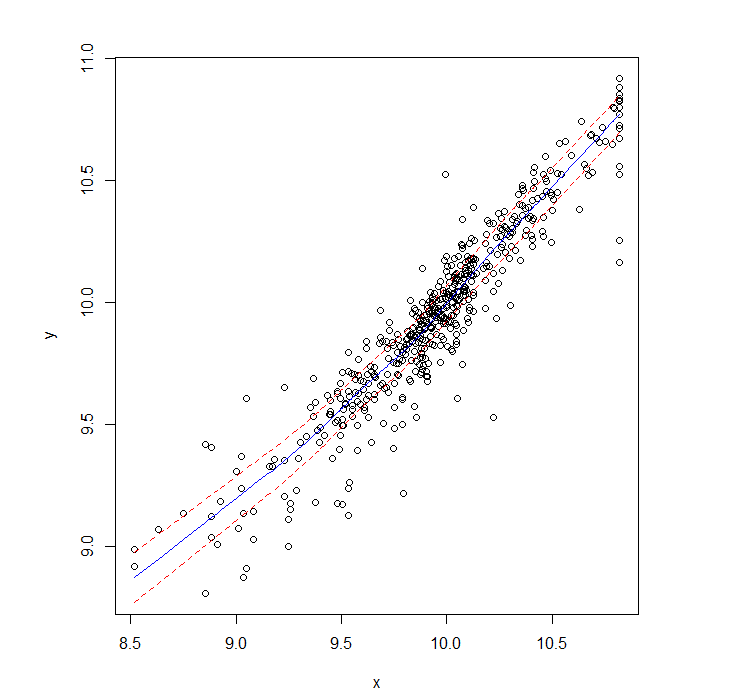
===============

0.1250348487

MAPE

===============

0.01271548165



Predicted values vs. Actual

*Neural Network with h = 3*

> trendscat(boston.trans$CMEDV,fitted(bos.nn3))

> cor(boston.trans$CMEDV,fitted(bos.nn3))^2

[,1]

[1,] 0.875159503 🡨

Naïve estimate of RMSEP from 13-3-1 neural network  
  
> sqrt(mean(resid(bos.nn3)^2))

[1] 0.1441122409

As the response (CMEDV) has been log transformed in the models above, we must also consider the prediction accuracy measures in the original scale ($). A slight alteration to the code in the nnet.sscv function highlighted below accomplishes this.

for (i in 1:B){

sam = sample(1:n,ss,replace=F)

fit2 = nnet(x[sam,],y[sam],size=size,linout=linout,skip=skip,  
 decay=decay,maxit=maxit,trace=F)

ynew = predict(fit2,newdata=x[-sam,])

ystar = exp(y)

ynew = exp(ynew)

MSEP[i]=mean((ystar[-sam]-ynew)^2)

MAEP[i]=mean(abs(ystar[-sam]-ynew))

MAPEP[i]=mean(abs(ystar[-sam]-ynew)/ystar[-sam])

}

The Monte Carlo split-sample cross-validation results for the response in the original scale are shown for one-hidden layer neural nets with nodes.

13-10-1 network   
> nnet10log = nnet.sscvlog(bos.x,bos.y,bos.nn,size=10)

RMSEP

===============

13072.0154

MAE

===============

3552.704401

MAPE

===============

0.1563114918

13-5-1 network   
> nnet5log = nnet.sscvlog(bos.x,bos.y,bos.nn,size=5)

RMSEP

===============

4542.967143

MAE

===============

2700.825668

MAPE

===============

0.1266975304

13-3-1 network   
> nnet3log = nnet.sscvlog(bos.x,bos.y,bos.nn,size=3)

RMSEP

===============

4068.10027

MAE

===============

2681.606937

MAPE

===============

0.1267905159

13-3-1 network – all variables in ORIGINAL SCALE (i.e. no transformations!)  
> nnet3orig = nnet.sscv(bos.x,bos.y,bos.orig,size=3)

RMSEP

===============

4867.085064

MAE

===============

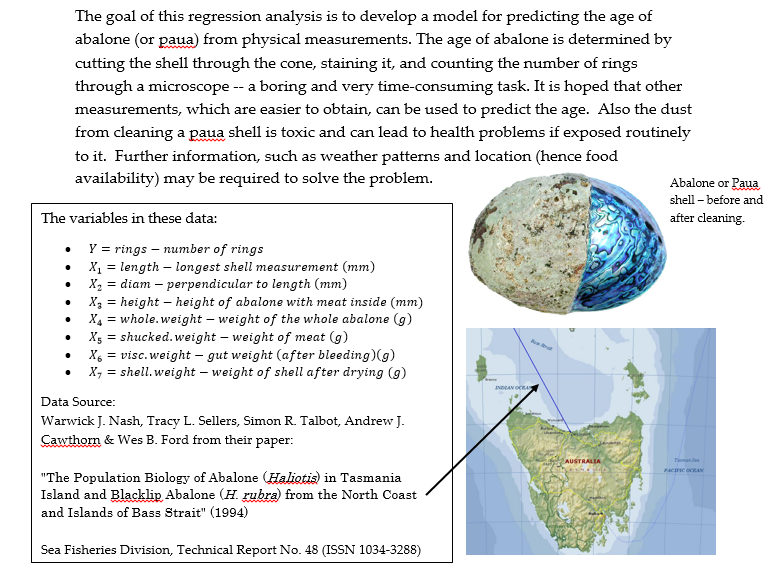
3373.971904

MAPE

===============

0.1687000727

**Example 7.2 – Age of Abalone**

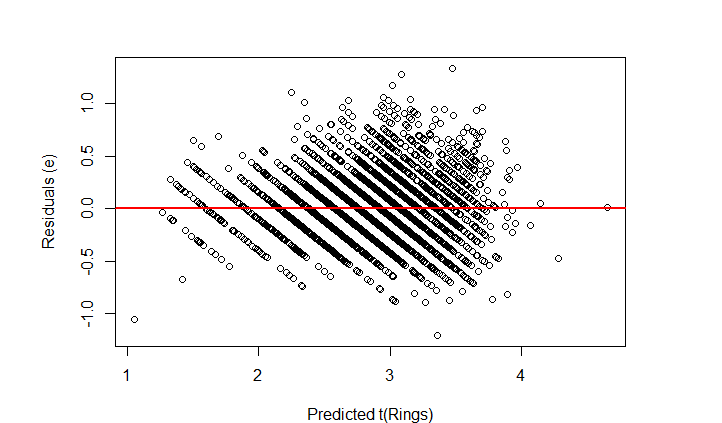
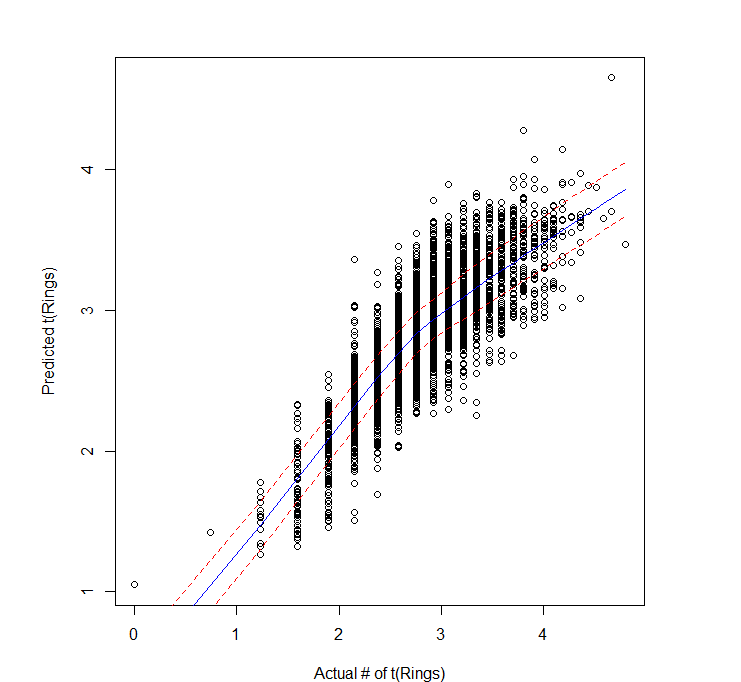


Using the same preprocessing as we did with this example in Section 6, we will develop a neural network model for predicting the age of an abalone.

> Abalone$height = Abalone$height+.001

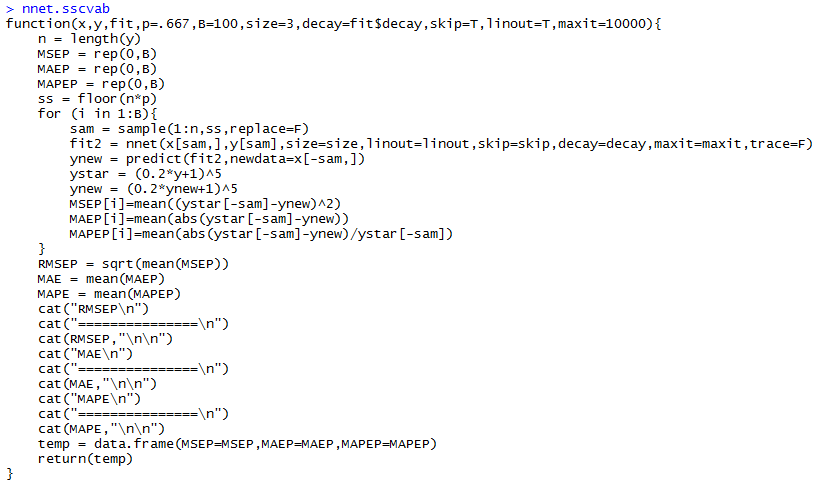
> Abalone.PP = preProcess(Abalone,method="BoxCox")  
> Abalone.PP = predict(Abalone.PP,Abalone)  
> ab.nn = nnet(rings~.,data=Abalone.PP,size=10,linout=T,skip=T,maxit=20000,decay=0.001)

> plot(Abalone.PP$rings,fitted(ab.nn),xlab="Actual # of t(Rings)",

ylab="Predicted t(Rings)")  


Predicted values ( vs. actual () and residuals () vs. predicted values (.

We will use Monte Carlo split-sample cross-validation to choose a model size (i.e. number of hidden nodes).

  
  
> results10 = nnet.sscvab(X,y,nnet.ab,B=20,size=10,maxit=50000,decay=0.001)

Back-transforming Box-Cox Transformations

or if a log-transformation is used ()

Here for the number of growth rings.

RMSEP

===============

2.192256605

MAE

===============

1.501620536

MAPE

===============

0.1478346995

When cross-validating neural networks in R it is important to inspect individual results to make sure the models in the Monte Carlo loops are converging.  
> results10

MSEP MAEP MAPEP

1 4.979206575 1.535119874 0.1478446865

2 4.294512414 1.463880751 0.1439444469

3 4.702296327 1.498502330 0.1457980815

4 4.478168746 1.512246928 0.1477585221

5 5.009387813 1.531553258 0.1502114096

There is one model with elevated CV results, but nothing that is particularly troubling.

6 4.803444672 1.512394981 0.1484857646

7 4.642164012 1.503079780 0.1467056083

8 4.470057094 1.488789614 0.1515410219

9 4.483435695 1.423178157 0.1438279999

10 4.378793028 1.468209111 0.1452918548

11 4.936520973 1.539216814 0.1512320984

12 4.389926280 1.477366036 0.1478061410

13 7.371581926 1.573640614 0.1524241058

14 4.405711674 1.466473481 0.1440095589

15 4.957017730 1.491229677 0.1506556385

16 4.238000061 1.455545196 0.1444471516

17 4.796194281 1.503103336 0.1465307574

18 5.484317447 1.601435278 0.1549825999

19 4.805709606 1.509816867 0.1463876097

20 4.493334069 1.477628638 0.1468089335

> results8 = nnet.sscvab(X,y,nnet.ab,B=20,size=8,maxit=50000,decay=0.001)

RMSEP

===============

2.150189607

MAE

===============

1.49311196

MAPE

===============

0.146747925

> results6 = nnet.sscvab(X,y,nnet.ab,B=20,size=6,maxit=50000,decay=0.001)

RMSEP

===============

2.123917623

MAE

===============

1.479688867

MAPE

===============

0.1456880488

> results4 = nnet.sscvab(X,y,nnet.ab,B=20,size=4,maxit=50000,decay=0.001)

RMSEP

===============

2.089436543

The model with 4 hidden nodes appears to be the best from the MC split-sample results.

MAE

===============

1.463094066

MAPE

===============

0.1443794935

> results3 = nnet.sscvab(X,y,nnet.ab,B=20,size=3,maxit=50000,decay=0.001)

RMSEP

===============

2.116362006

MAE

===============

1.482192706

MAPE

===============

0.1458066396

**Example 7.3: CA Homes**

These data come from a study of median home values in census tracts in California.

> read.table(file.choose(),header=T,sep=”,”)

> head(CAhomes) 🡨 displays the first 6 rows of a data frame in R.

MEDV MEDINC MEDAGE TotRooms TotBeds Pop Households Latitude Longitude

1 452600 8.3252 41 880 129 322 126 37.88 -122.23

2 358500 8.3014 21 7099 1106 2401 1138 37.86 -122.22

3 352100 7.2574 52 1467 190 496 177 37.85 -122.24

4 341300 5.6431 52 1274 235 558 219 37.85 -122.25

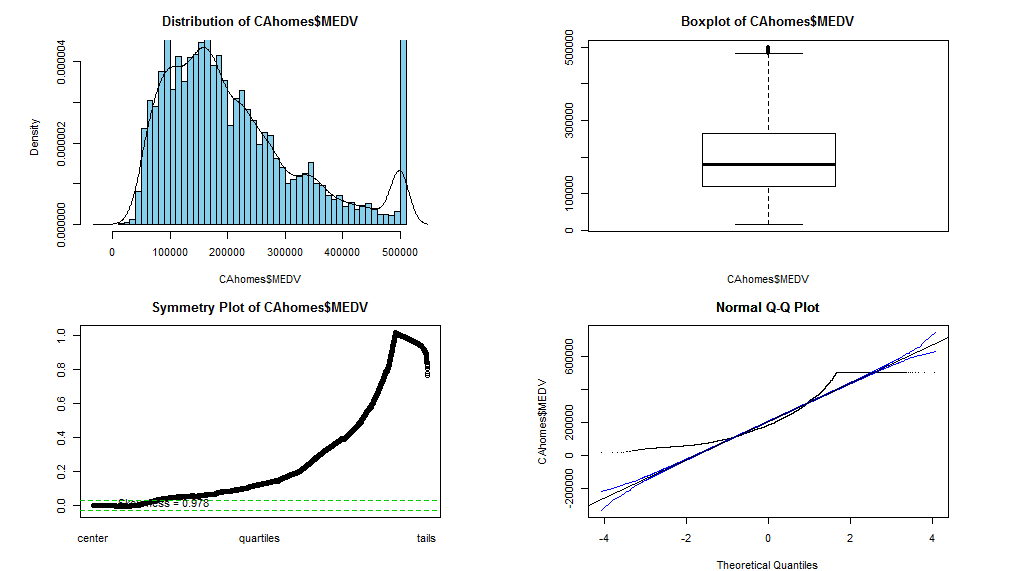
5 342200 3.8462 52 1627 280 565 259 37.85 -122.25

6 269700 4.0368 52 919 213 413 193 37.85 -122.25   
The response is the median home price/value (MEDV) and the potential predictors are:

* MEDINC – median household income in the census tract
* MEDAGE– median home age in the census tract.
* TotRooms – total number of rooms in all the homes combined in the census tract.
* TotBeds – total number of bedrooms in all the homes combined in the census tract.
* Pop – number of people in the census tract.
* Households – total number of households in the census tract.
* Latitude – latitude of the centroid of the census tract
* Longitude – longitude of the centroid of the census tract.

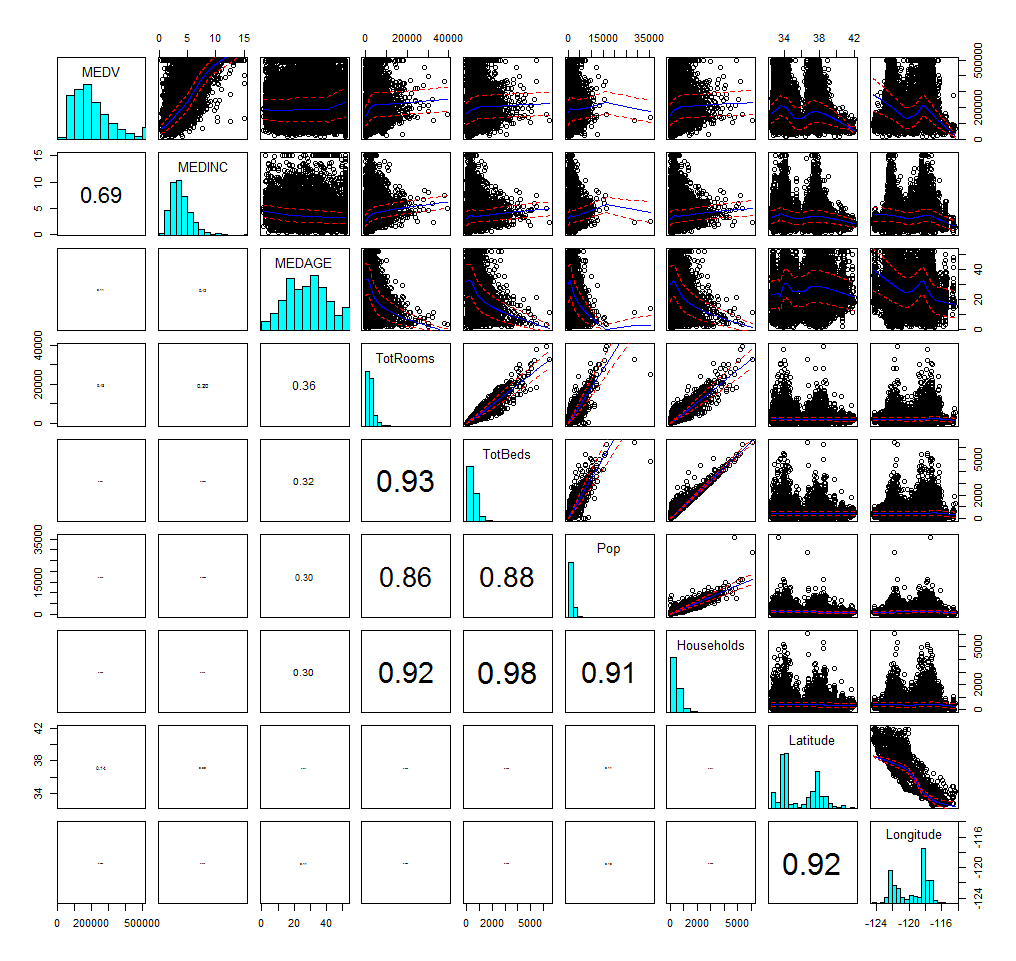
Presented below are some plots of these data:

> Statplot(CAhomes$MEDV)

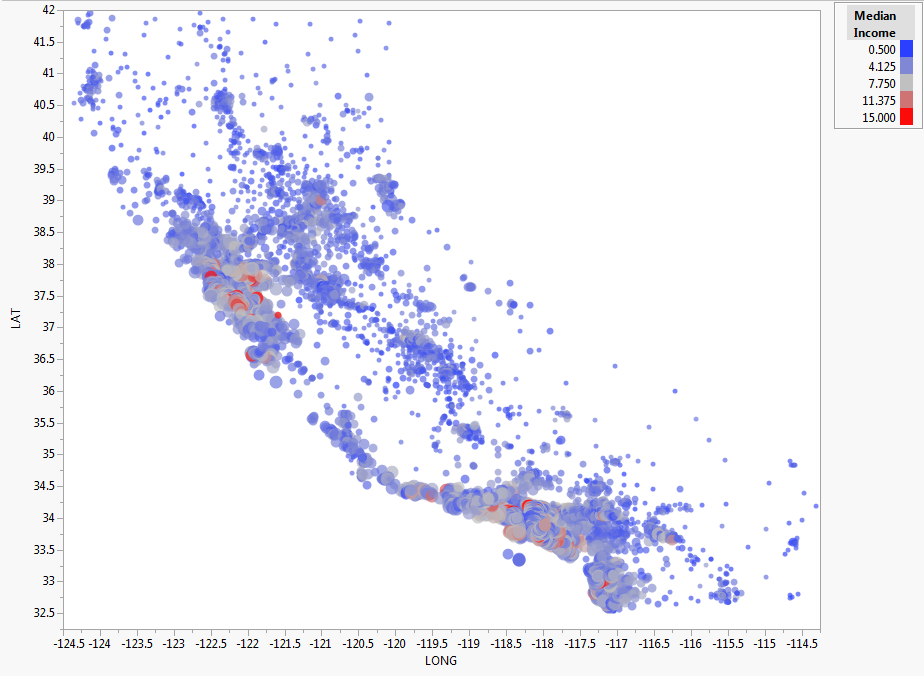


What interesting features do you see in the response?

> pairs.plus(CAhomes) **🡨 WARNING: TAKES A LONG TIME TO RUN!**



**Using Bubble Plots in JMP (under Graph menu)**



Here circles are proportional in size to the response and the color denotes the median household income.

Without doing any preprocessing of the response and/or the predictors we can fit a neural network to predict the median home value in a census tract using the census tract level predictors.

> X = CAhomes[,-1]

> y = CAhomes[,1]

> ca.nnet = nnet(X,y,size=6,linout=T,skip=T,maxit=10000,decay=.001)

# weights: 69

initial value 1155054841839622.500000

iter 10 value 180176476664334.437500

iter 20 value 143205254195815.500000

iter 30 value 94815590611921.406250

iter 40 value 94805464731121.609375

iter 50 value 94602493096295.328125

iter 60 value 94434274690428.843750

iter 70 value 94383275084861.390625

iter 80 value 94373838660981.296875

iter 90 value 94352897115920.890625

iter 100 value 94344043542937.640625

iter 110 value 94298251944650.421875

iter 120 value 94099849636283.406250

iter 130 value 94036450323079.875000

iter 140 value 93797377684854.312500

iter 150 value 93697914437649.546875

iter 160 value 93433613476027.984375

iter 170 value 93398377336154.593750

iter 180 value 92980400623365.890625

iter 190 value 92512810126913.015625

iter 200 value 92002250948180.640625

iter 210 value 91850999131736.437500

iter 220 value 91456684623887.828125

iter 230 value 91392579048187.343750

iter 240 value 91064866712578.375000

iter 250 value 90991063375381.437500

iter 260 value 90873991849937.062500

iter 270 value 90849965960191.328125

iter 270 value 90849965960191.328125

iter 280 value 90825629314687.984375

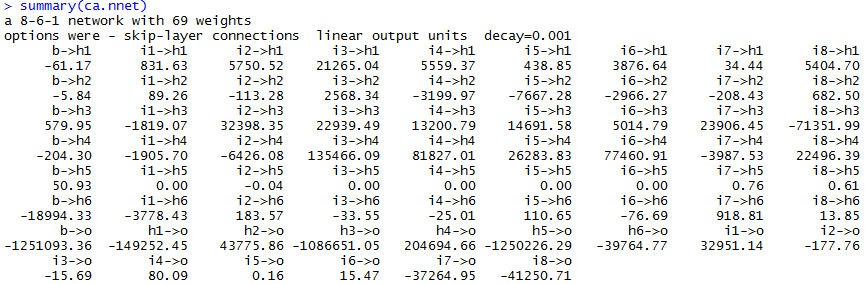
iter 290 value 90816126987550.437500

iter 300 value 90815506792120.015625

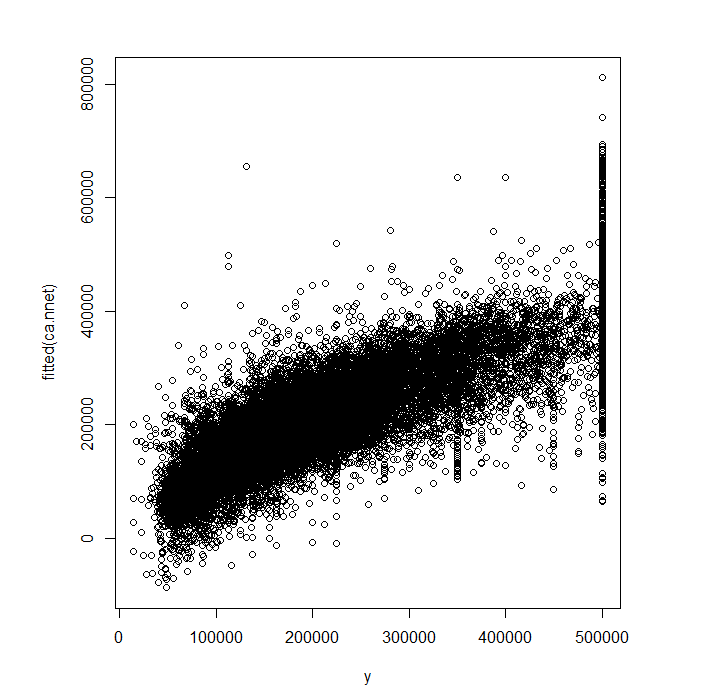
iter 310 value 90814395800305.937500

final value 90814373948204.828125

converged



> plot(y,fitted(ca.nnet))



> cor(y,fitted(ca.nnet))

[,1]

[1,] 0.8267136674

> cor(y,fitted(ca.nnet))^2

[,1]

[1,] 0.6834554879 🡨 is 68.34%

Try using as the response…

> logy = log(y)

> ca.nnet2 = nnet(X,logy,size=6,linout=T,skip=T,maxit=10000,decay=0.01)

# weights: 69

initial value 53507862150.151787

iter 10 value 4629141470.195857

iter 20 value 995924050.679593  
. . . … … … … … …

iter 750 value 1776.152891

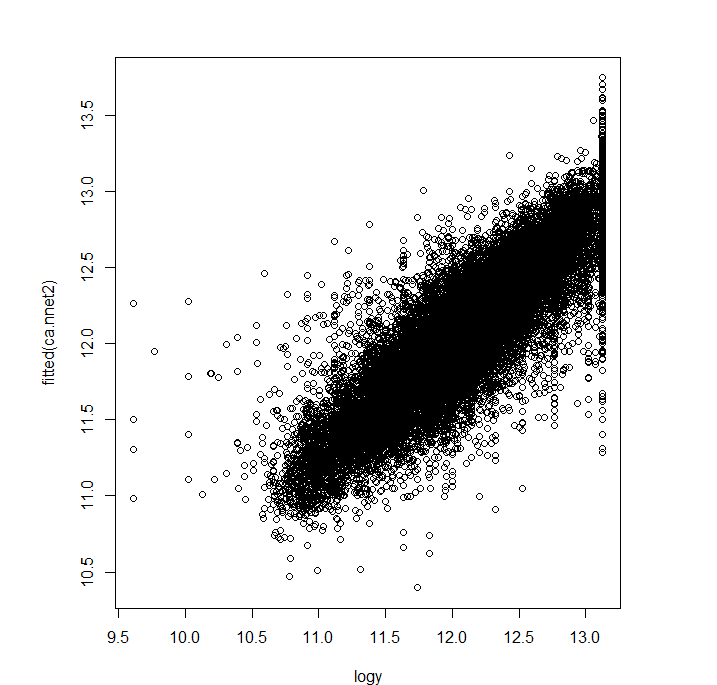
iter 760 value 1776.127757

iter 770 value 1776.125791

iter 780 value 1776.124947

final value 1776.124764

converged

> plot(logy,fitted(ca.nnet2))  


> cor(logy,fitted(ca.nnet2))

[,1]

[1,] 0.8591545838

> cor(logy,fitted(ca.nnet2))^2

[,1]

[1,] 0.7381465989 🡨

**Some Monte Carlo Split-Sample Cross-validation Results**

> results.nn6 = nnet.sscv(X,y,ca.nnet,B=25,size=6)

RMSEP

===============

68498.88789

MAE

===============

49876.5076

MAPE

===============

0.2987564039

> results.nn3 = nnet.sscv(X,y,ca.nnet,B=25,size=3)

RMSEP

===============

69271.51735

MAE

===============

50433.33891

MAPE

===============

0.3025012267

**and log transforming all numeric covariates except latitude and longitude**

> results.nn6 = nnet.sscvlog(Xt,yt,ca.nnet3,B=10,size=6,maxit=50000,decay=0.01)

RMSEP

===============

66203.5973

MAE

===============

40953.40909

MAPE

===============

0.2167377705

**IMPORTANT NOTE:** Prior to transforming the as well the log model would get stuck at non-optimal local minimum in searching for the weights. I could not get the NNET with to converge consistently when running nnet.sscvlog. It is always a good idea to run NNETs a few times to see if there are convergence issues. I did not identify the issues with the log-response NNET until I tried cross-validating and got very weird results, especially for the RMSEP. This happens become some neural nets fit in the cross-validation loop did not converge to a reasonable solution.

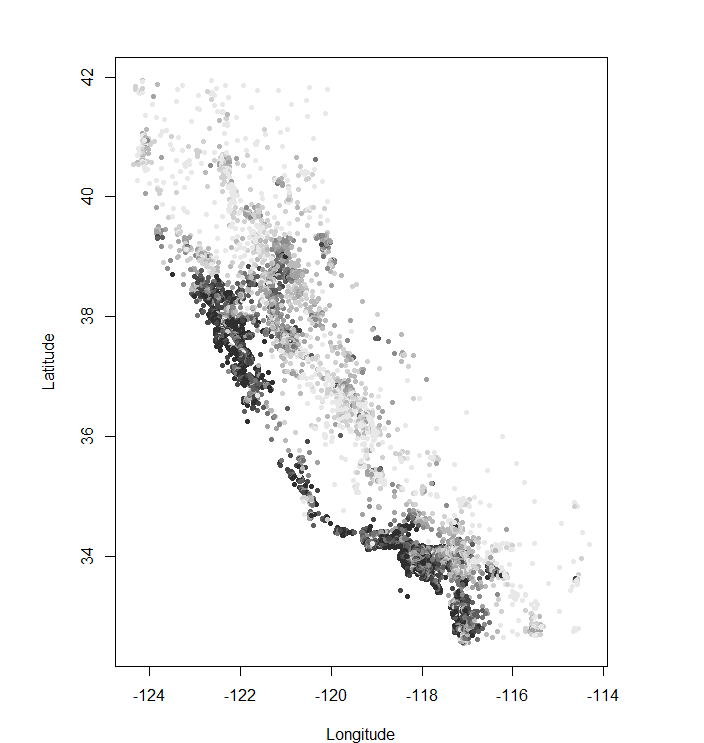
For this example transforming the predictors to improve symmetry/normality was important - and I suspect this will generally be the case for NNETs when the inputs () have very skewed distributions as is the case for the California Census Tract data.

**A fancy plots in R** – (not as easy as JMP)

> price.deciles = quantile(CAhomes$MedHP,0:10/10)

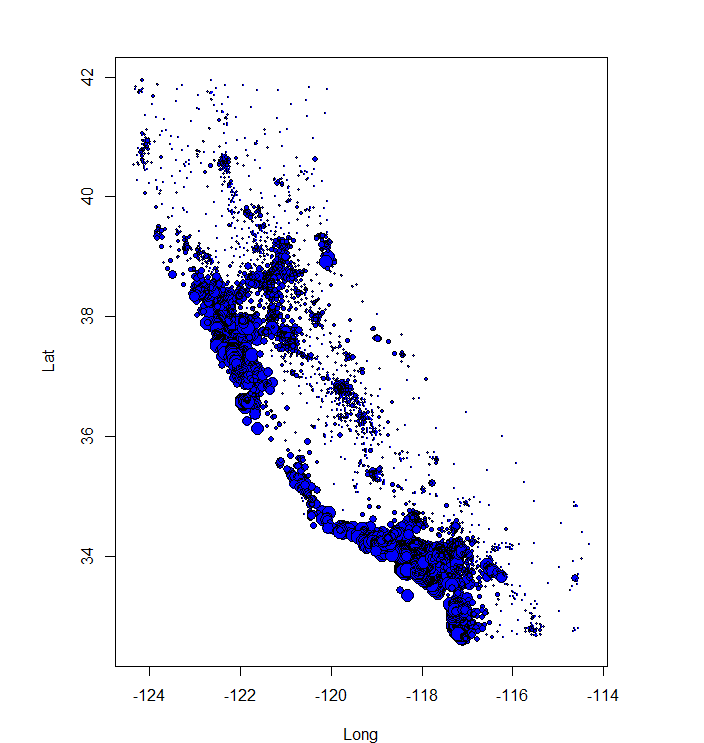
> cut.prices = cut(CAhomes$MedHP,price.deciles,include.lowest=T)

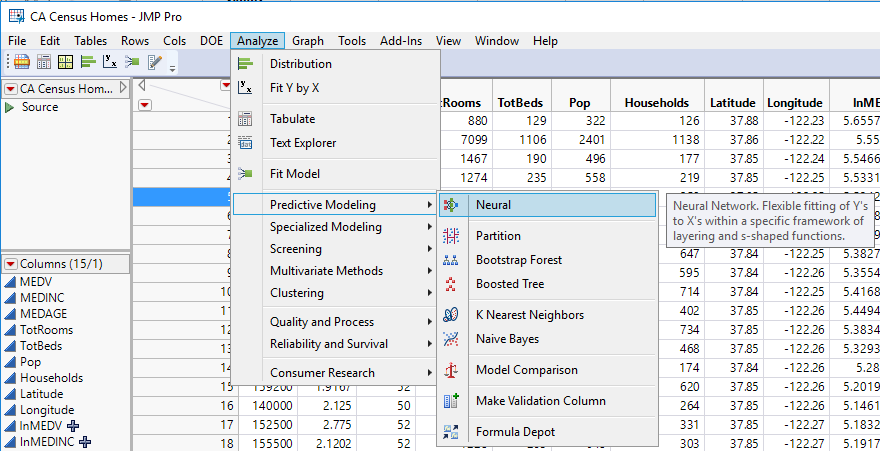
> plot(Longitude,Latitude,col=grey(10:2/11)[cut.prices],pch=20,  
xlab=”Longitude”,ylab=”Latitude”)

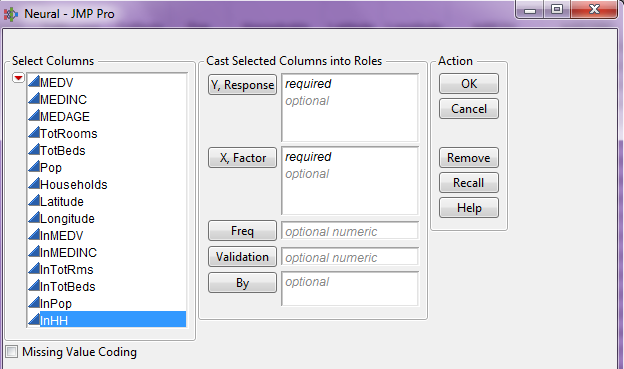


> plot(CAhomes$Longitude,CAhomes$Latitude,pch=".",xlab=”Long”,ylab=”Lat”)

> symbols(CAhomes$Longitude,CAhomes$Latitude,circles=CAhomes$MEDV,bg="blue",  
add=T,inches=.1)

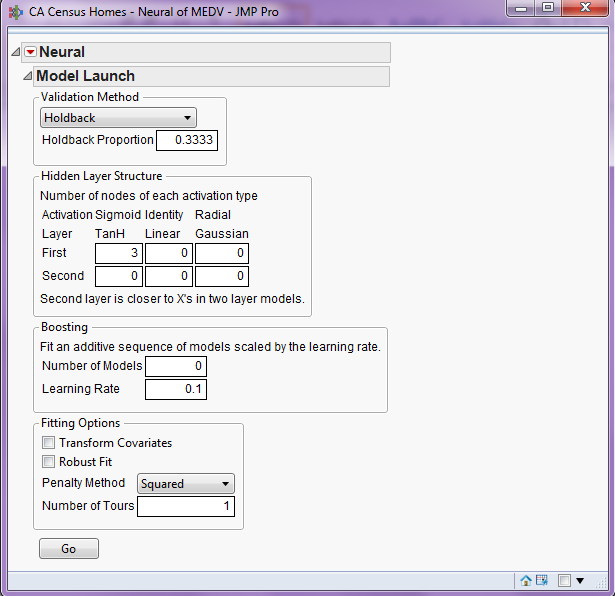


**Neural Networks in JMP**To fit a neural network in JMP select the **Neural** option from within the **Analyze Predictive Modeling > Neural** menu as shown below.

The model dialog box for fitting a neural network is shown below.  


The response (Y) goes in the **Y,Response** box and maybe either continuous for a regression problem or categorical for a classification problem. The potential predictors go in the X,Factor box and maybe a mixture of variable types.

The neural network model building platform is shown on the following page. There are numerous options that can be set which control different aspects of the model fitting process such as the number of hidden layers (1 or 2), type of “squash” function, cross-validation proportion, robustness (outlier protection), regularization (similar to ridge and Lasso), predictor transformations, and number of models to fit (which protects against the randomness of the optimization algorithm.



**Penalty Method** – includes a penalty as in regularized/shrinkage regression methods we have examined previously, the penalty options are:

* **Squared** = ~ analagous to ridge regression (discussed later). Use this if you generally believe that all predictors are likely to be useful.
* **Absolute** = ~ analogous to Lasso regression (discussed later). Use this if you generally believe that some the predictors are not useful and would like to essentially remove them from the neural network model.
* **Weight Decay** = – another way to penalize parameters/weights to effectively zero out some predictors.
* **No Penalty** – there is no penalty on estimated weights.

***Absolute*** and ***Weight Decay*** tend to work well when the number of potential predictors is large as it tends to “0” some out.

**Number of Tours** – the number of times to fit the model to find the “optimal” weights. Due to the fact the algorithm utilizes some randomness in fitting the model, you will not get the results fit every time even when all the aspects of the fitting process are the same. JMP returns the model with best fit amongst those found.

**Holdback Proportion** – fraction to use in test set (33.33% by default)

**Hidden Layer Structure** – for two hidden layer networks, the Second layer is actual the first layer coming from the predictors. The “squash” or activation functions are the hyperbolic tangent, linear or Gaussian.

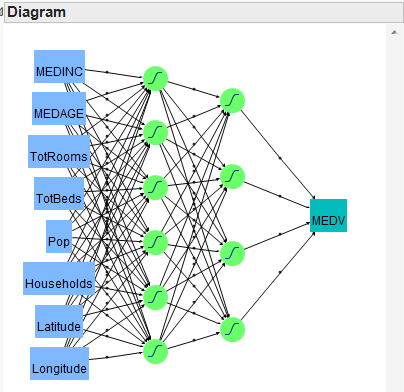
**Boosting** – will be discussed later in the course.

**Transform Covariates** – checking this will perform transformations of continuous predictors to approximate normality.

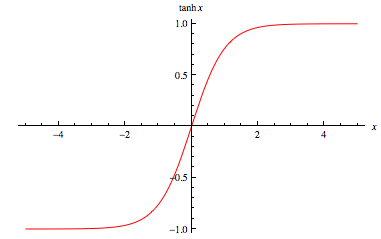
**Robust Fit** – will provide protection against outliers by minimizing the sum of absolute residuals, rather than the squared residuals.

As an example of fitting neural networks in JMP we again consider the California census tract home value data.

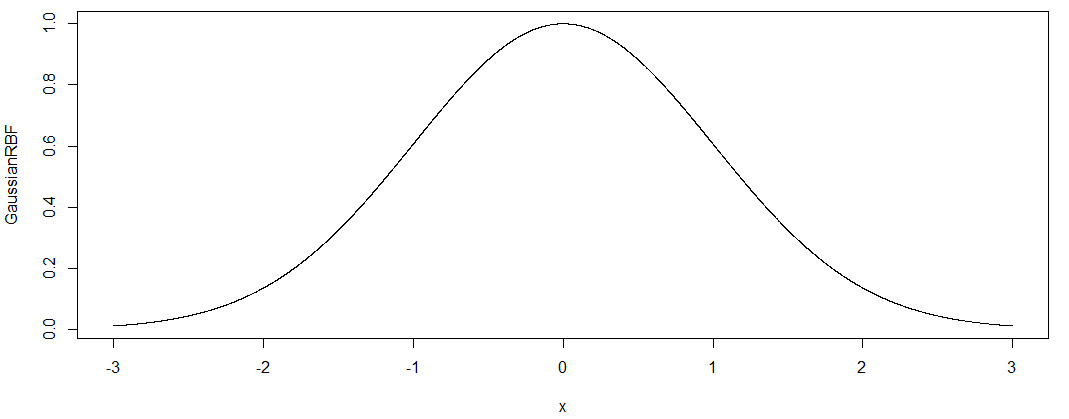
Below are some neural networks to these data. I did not give much thought into these models and I am sure there are other viable models for these data. The basic model diagram of the models summarized below is shown below.



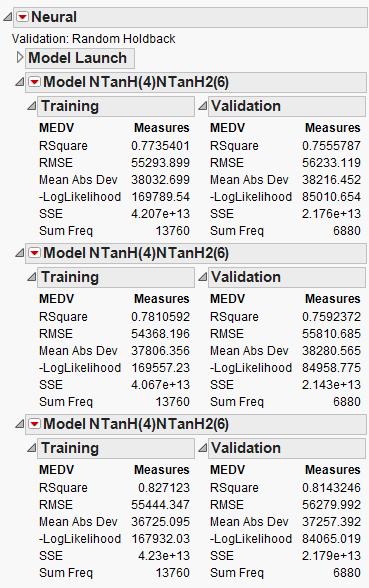
Here I have used two hidden layers in the neural network, the first consists of 6 nodes and second consists of 4 nodes. At each node the activation function is given by:



Gaussian Activation Function

The choice of 6 and 4 was completely arbitrary, but the results achieved seem reasonable as we shall see below.

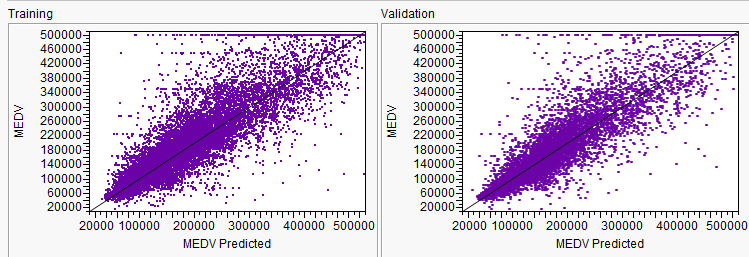
The basic form of all models are the same, I did use some of the optional settings to fine tune the fitting process.

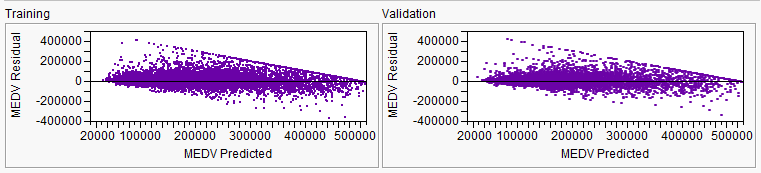


Model 3 – Same model as those above with both Transform Covariates and Robust Fit options checked.

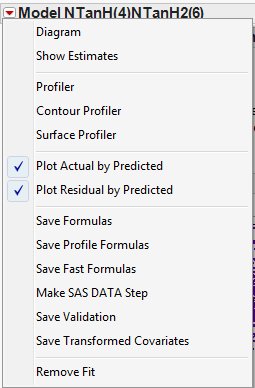
Model 2 - Same model as above with Transform Covariates checked.

Model 1 - Base model fit using the default settings.

Plot of Actual vs. Predicted for Training and Validation/Test Sets (Model 3)  


Plot of Residuals vs. Predicted for Training and Validation/Test Sets (Model 3)  


If consider Model 3 to be our final model, then we can use JMP to explore the fitted equation interactively and save certain results of the fit.



**Diagram** – displays diagram of neural network.

**Show Estimates** – shows the weights/coefficients using a convention similar to R.

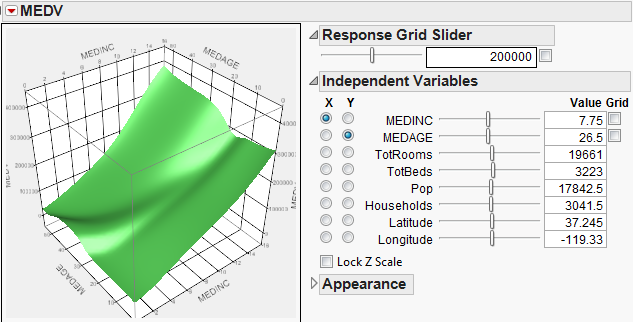
**Profiler** – explore the fitted surface using univariate sliders.

**Contour Profiler** – explore the fitted “surface” using a contour plot as a function of two predictors at a time.

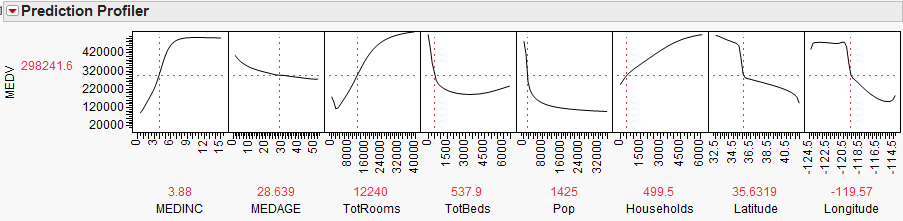
**Surface Profiler** – display the fitted “surface” using a 3-D surface a function of two predictors at a time.

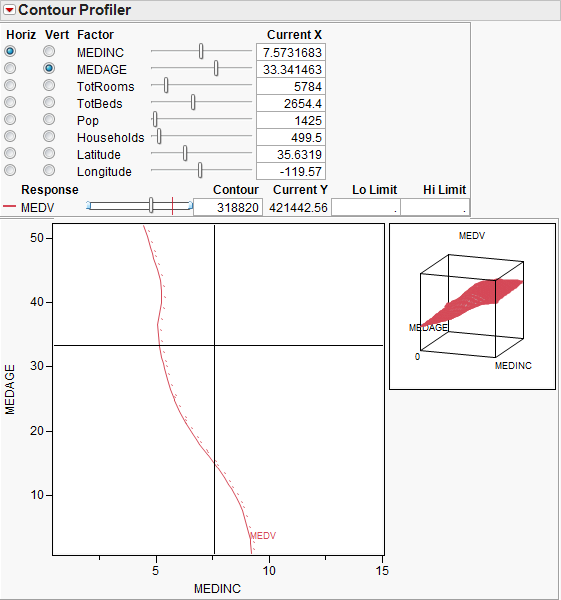
**Save Formulas** – save formulae for hidden layer nodes and the response. They can be used to make future predictions.

Save Transformed Covariates – save the covariate transformations found before fitting neural network.

**Surface Profiler**   
**Profiler –** the profiler shows the estimated response curves as a function of each term

**The Surface Profiler** allows you interactively explore the fitted surface as a function of two predictors/terms holding the others held constant.



**Contour Profiler –** similar to the surface plot, but shows fitted surface as a contour plot instead. 

**Transformed Covariates** – each has been transformed to be approximately normally distributed prior to fitting the neural network. (These are the terms created automatically for each predictor if this option is checked when fitting the neural network).

