# 16 - Neural Networks for Classification

Recall that a neural network is a 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 classifying the area in Italy an olive oil comes from.



The neural network model for classification problems looks like this:

where is the probability that belongs to class given the set of predictor values . This model has 1 hidden layer but it is possible to have additional hidden layers (in JMP e.g.). For a classification problem the final activation function could NOT be linear as the predictions must be probabilities, i.e. between 0 and 1! In general the activation functions for neural networks are usually chosen from the following choices.

Activation Functions -



As mentioned above for classification problems the final prediction needs to be a probability and hence be restricted to be between 0 and 1. Thus is generally going to be the logistic activation or simply the logistic function, however the threshold and hyperbolic tangent could also be used. The threshold function could be used for a binary classification problem (-1 denotes one class and +1 denotes the other) and the (hyperbolic tangent + 1)/2 would take on values between 0 and 1, making them valid probabilities. It should be clear that these models are highly parameterized and thus have the potential to over fit the training data. Cross-validation is therefore critical to make sure that the predictive performance of the models is adequate.

**Example 16.1 – Italian Olive Oils**

> library(nnet) 🡨 There are other packages that allow more flexible neural network models   
 than the nnet function in the nnet library.

> names(OliveOils)

[1] "Region.name" "Area.name" "Region" "Area" "palmitic" "palmitoleic" "strearic" "oleic"

[9] "linoleic" "eicosanoic" "linolenic" "eicosenoic"  
  
Form training and tests sets, size 429 and 143 respectively.

> sam = sample(1:572,429,replace=F)

> Olive.train = OliveOils[sam,c(1,3,4)]

> Olive.test = OliveOils[-sam,c(1,3,4)]

**General Comments about Classification Neural Network Parameters**

* Increased size increases flexibility by adding hidden nodes, this can be critical when the number of possible categories is large.
* Increasing the decay will increase the length of the search, generally resulting in a better solution and hence model, though overfitting is an issue.
* Increasing the maximum number of iterations (maxit) is necessary especially if you increase the model size and/or the decay parameter.

> olive.nn<-nnet(Area.name~.,data=Olive.train,size=9,decay=.025,maxit=5000)

# weights: 162

initial value 1135.733158

iter 10 value 223.982116

iter 20 value 118.718825

iter 30 value 77.189140

iter 40 value 67.122399

iter 50 value 62.277726

iter 60 value 59.135737

iter 70 value 57.422192

iter 80 value 56.152411

iter 90 value 55.682681

iter 100 value 55.507056

iter 110 value 55.419962

iter 120 value 55.298828

iter 130 value 54.840218

iter 140 value 54.260595

iter 150 value 53.858588

iter 160 value 53.632600

iter 170 value 52.394640

iter 180 value 51.711774

iter 190 value 51.234710

iter 200 value 50.985742

iter 210 value 50.865543

iter 220 value 50.758722

iter 230 value 50.485761

iter 240 value 50.070691

iter 250 value 49.959243

iter 260 value 49.924597

iter 270 value 49.905078

iter 280 value 49.899314

iter 290 value 49.895285

iter 300 value 49.889335

iter 310 value 49.882619

iter 320 value 49.880260

iter 330 value 49.880093

iter 340 value 49.880061

iter 350 value 49.880046

iter 350 value 49.880046

iter 350 value 49.880046

final value 49.880046

converged

> olive.nn

a 7-9-9 network with 162 weights

inputs: palmitic palmitoleic strearic oleic linoleic eicosanoic linolenic

output(s): Area.name

options were - softmax modelling decay=0.025

Estimated Probabilities from Neural Network ~ P(Area | )

> predict(olive.nn)[1:5,]

Calabria Coastal-Sardinia East-Liguria Inland-Sardinia North-Apulia

389 1.139807e-09 3.237642e-03 2.556212e-04 9.944179e-01 1.296157e-03

367 1.362286e-09 4.169660e-03 1.287757e-03 9.920853e-01 1.348859e-03

572 5.426047e-04 4.505087e-04 2.471932e-01 4.611739e-03 6.910849e-03

531 4.040190e-06 4.981115e-05 3.487181e-03 5.535082e-05 3.176028e-05

378 2.416832e-09 9.762054e-01 4.867926e-06 2.166963e-02 1.567489e-04

Sicily South-Apulia Umbria West-Liguria

389 6.819229e-04 9.435564e-05 8.909214e-08 1.626639e-05

367 9.068451e-04 1.729733e-04 1.520462e-07 2.849803e-05

572 6.359606e-03 2.022353e-03 2.383067e-03 7.295260e-01

531 6.125910e-05 6.909839e-04 8.075130e-05 9.955389e-01

378 3.829648e-05 1.807280e-03 1.103909e-07 1.176798e-04

> predict(olive.nn,type="class")[1:5]

[1] "Inland-Sardinia" "Inland-Sardinia" "West-Liguria" "West-Liguria"

[5] "Coastal-Sardinia"

**Misclassification Function for Classification Neural Networks**

> misclass.nnet <- function(fit,y) {

temp <- table(predict(fit,type="class"),y)

cat("Table of Misclassification\n")

cat("(row = predicted, col = actual)\n")

print(temp)

cat("\n\n")

numcor <- sum(diag(temp))

numinc <- length(y) - numcor

mcr <- numinc/length(y)

cat(paste("Misclassification Rate = ",format(mcr,digits=3)))

cat("\n")

}

Input arguments are the model fit and the response used. The rows of the resulting matrix are the predictions the columns are the actual categories.

Misclassification Rate for Area Grown Neural Network

> misclass.nnet(olive.nn,Olive.train$Area.name)

Table of Misclassification

(row = predicted, col = actual)

y

Calabria Coastal-Sardinia East-Liguria Inland-Sardinia North-Apulia

Calabria 42 0 0 0 0

Coastal-Sardinia 0 23 0 0 0

East-Liguria 0 0 41 0 0

Inland-Sardinia 0 0 0 45 0

North-Apulia 0 0 0 0 21

Sicily 0 0 0 0 0

South-Apulia 0 0 0 0 0

Umbria 0 0 0 0 0

West-Liguria 0 0 0 0 0

y

Sicily South-Apulia Umbria West-Liguria

Calabria 0 0 0 0

Coastal-Sardinia 0 0 0 0

East-Liguria 0 0 0 0

Inland-Sardinia 0 0 0 0

North-Apulia 0 0 0 0

Sicily 27 1 0 0

South-Apulia 0 151 0 0

Umbria 0 0 43 0

West-Liguria 0 0 0 35

Misclassification Rate = 0.00233 🡨 **.**233% misclassified!

**Cross-Validation Function for Classification Neural Networks**

> cnnet.cv = function (fit, y, data, B = 25, p = 0.667, size = 5, decay = 0.001, maxit = 5000,trace=T)

{

n <- length(y)

cv <- rep(0, B)

nin <- floor(n \* p)

out <- n - nin

for (i in 1:B) {

sam <- sample(1:n, nin)

temp <- data[sam, ]

fit2 <- nnet(formula(fit), data = temp, size = size,

decay = decay, maxit = maxit,trace=trace)

ynew <- predict(fit2, newdata = data[-sam, ], type = "class")

tab <- table(y[-sam], ynew)

mc <- out - sum(diag(tab))

cv[i] <- mc/out

}

cv

}

This function takes essentially the same arguments as the CV function for neural networks for regression. The size, decay parameter, and maximum iterations should be set to the same values you used to obtain the original fit. By default I have set those values 5, .001, and 1000 respectively. Sometimes during the process it is possible that the neural network will not converge in which case you will need to “throw out” misclassification rates that are ridiculously large relative to the rest. Also this function takes a long time to run, so do not use a large value for *B*!

> results=cnnet.cv(olive.nn,Olive.train$Area.name,data=Olive.train,  
 B=50,size=9,decay=.025,maxit=5000)  
> summary(results)

Min. 1st Qu. Median Mean 3rd Qu. Max.

0.03497 0.06469 0.08392 0.08280 0.09790 0.12590

> results = cnnet.cv(olive.nn,Olive.train$Area.name,data=Olive.train,  
 B=100,size=15,decay=.025,maxit=5000)

> summary(results)

Min. 1st Qu. Median Mean 3rd Qu. Max.

0.02797 0.06294 0.06993 0.07203 0.08392 0.12590

> results = cnnet.cv(olive.nn,Olive.train$Area.name,data=Olive.train,

B=100,size=15,decay=.05,maxit=5000)

> summary(results)

Min. 1st Qu. Median Mean 3rd Qu. Max.

0.02098 0.05594 0.06993 0.06839 0.07692 0.13290

> results = cnnet.cv(olive.nn,Olive.train$Area.name,data=Olive.train,

B=100,size=20,decay=.05,maxit=5000)

> summary(results)

Min. 1st Qu. Median Mean 3rd Qu. Max.

0.02098 0.05420 0.06643 0.07245 0.07692 0.60840 🡨 case where it did not converge!

> results = cnnet.cv(olive.nn,Olive.train$Area.name,data=Olive.train,

B=100,size=12,decay=.05,maxit=5000)  
> summary(results)

Appears “optimal”

Min. 1st Qu. Median Mean 3rd Qu. Max.

0.03497 0.05594 0.06993 0.06867 0.07692 0.11890

> results=cnnet.cv(olive.nn,Olive.train$Area.name,data=Olive.train,size=13,  
 decay=.05,maxit=5000,B=100)  
> summary(results)

Min. 1st Qu. Median Mean 3rd Qu. Max.

0.02797 0.05594 0.06993 0.07091 0.08392 0.11190

We will now predict for the test cases using the Olive.test set.

> olive.nn = nnet(Area.name~.,data=Olive.train,size=12,decay=.05,maxit=5000)  
> misclass.nnet(olive.nn,Olive.train$Area.name)

Table of Misclassification

(row = predicted, col = actual)

y

Calabria Coastal-Sardinia East-Liguria Inland-Sardinia North-Apulia

Calabria 41 0 0 0 0

Coastal-Sardinia 0 23 0 0 0

East-Liguria 0 0 41 0 0

Inland-Sardinia 0 0 0 45 0

North-Apulia 0 0 0 0 21

Sicily 0 0 0 0 0

South-Apulia 1 0 0 0 0

Umbria 0 0 0 0 0

West-Liguria 0 0 0 0 0

y

Sicily South-Apulia Umbria West-Liguria

Calabria 0 0 0 0

Coastal-Sardinia 0 0 0 0

East-Liguria 0 0 0 0

Inland-Sardinia 0 0 0 0

North-Apulia 0 0 0 0

Sicily 26 1 0 0

South-Apulia 1 151 0 0

Umbria 0 0 43 0

West-Liguria 0 0 0 35

Misclassification Rate = 0.00699  
  
> yhat = predict(olive.nn,newdata=Olive.test,type="class")

> misclass(yhat,Olive.test$Area.name)

Table of Misclassification

(row = predicted, col = actual)

y

fit Calabria Coastal-Sardinia East-Liguria Inland-Sardinia North-Apulia

Calabria 12 0 0 0 0

Coastal-Sardinia 0 9 0 0 0

East-Liguria 1 0 9 0 0

Inland-Sardinia 0 1 0 19 0

North-Apulia 0 0 0 1 3

Sicily 1 0 0 0 0

South-Apulia 0 0 0 0 0

Umbria 0 0 0 0 1

West-Liguria 0 0 0 0 0

y

fit Sicily South-Apulia Umbria West-Liguria

Calabria 0 2 0 0

Coastal-Sardinia 0 0 0 0

East-Liguria 1 0 0 1

Inland-Sardinia 0 0 0 0

North-Apulia 0 0 0 0

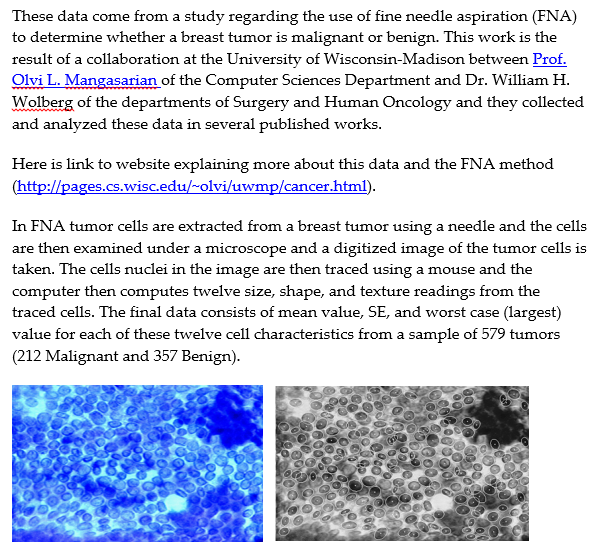
Sicily 8 2 0 0

South-Apulia 0 50 0 0

Umbria 0 0 8 0

West-Liguria 0 0 0 14

Misclassification Rate = 0.0769

**Example 16.2 – Breast Cancer Diagnosis using Final Needle Aspiration (FNA)**

> names(BreastDiag)

[1] "Id" "Diagnosis" "Radius" "Texture" "Perimeter" "Area"

[7] "Smoothness" "Compactness" "Concavity" "ConcavePts" "Symmetry" "FracDim"

[13] "serad" "setex" "seperi" "searea" "sesmoo" "secomp"

[19] "seconc" "seconpts" "sesym" "sefd" "wrad" "wtex"

[25] "wperi" "warea" "wsmoo" "wcomp" "wconc" "wconpts"

[31] "wsym" "wfd"

> dim(BreastDiag)

[1] 569 32

> n = nrow(BreastDiag)

> n

[1] 569

> ind = sample(1:n)

> trainval = floor(n\*.6)

> testval = floor(n\*.2)

> BC.train = BreastDiag[ind[1:trainval],-1]

> BC.test = BreastDiag[ind[(trainval+1):(trainval+testval)],-1]

> BC.valid = BreastDiag[ind[(trainval+testval+1):n],-1]

Check dimensions to make sure all this worked

> dim(BC.train)

[1] 341 31

> dim(BC.test)

[1] 113 31

> dim(BC.valid)

[1] 115 31

> bc.nn = nnet(Diagnosis~.,data=BC.train,size=1,decay=.05,maxit=5000)

# weights: 33

initial value 238.042969

iter 10 value 107.282768

iter 20 value 92.951081

iter 30 value 73.288123

iter 40 value 62.824208

iter 50 value 48.154052

iter 60 value 28.683344

iter 70 value 23.714517

iter 80 value 23.700881

final value 23.700531

converged

> yhat = predict(bc.nn,newdata=BC.valid,type="class")

> misclass(yhat,BC.valid$Diagnosis)

Table of Misclassification

(row = predicted, col = actual)

y

fit B M

B 78 3

M 2 32

Misclassification Rate = 0.0435

> bc.nn = nnet(Diagnosis~.,data=BC.train,size=1,decay=.025,maxit=5000)

# weights: 33

initial value 229.137214

iter 10 value 213.136605

iter 20 value 113.177711

iter 30 value 107.843320

iter 40 value 97.352735

iter 50 value 95.821745

iter 60 value 60.071680

iter 70 value 52.310584

iter 80 value 41.307812

iter 90 value 26.270056

iter 100 value 17.846390

iter 110 value 17.608037

iter 120 value 17.607481

final value 17.607480

converged

> yhat = predict(bc.nn,newdata=BC.valid,type="class")

> misclass(yhat,BC.valid$Diagnosis)

Table of Misclassification

(row = predicted, col = actual)

y

fit B M

B 78 3

M 2 32  
  
Misclassification Rate = 0.0696

> bc.nn = nnet(Diagnosis~.,data=BC.train,size=1,decay=.01,maxit=5000)

# weights: 33

initial value 231.356470

iter 10 value 227.582640

iter 20 value 199.646623

iter 30 value 89.473908

iter 40 value 83.683339

iter 50 value 80.847525

iter 60 value 80.667082

iter 70 value 80.651168

iter 80 value 80.650047

final value 80.650043

converged

> yhat = predict(bc.nn,newdata=BC.valid,type="class")

> misclass(yhat,BC.valid$Diagnosis)

Table of Misclassification

(row = predicted, col = actual)

y

fit B M

B 80 8

M 0 27

Misclassification Rate = 0.0435

> bc.nn = nnet(Diagnosis~.,data=BC.train,size=2,decay=.05,maxit=5000)

# weights: 65

initial value 258.555097

iter 10 value 104.842714

iter 20 value 97.133928

iter 30 value 93.191638

iter 40 value 67.589459

iter 50 value 52.475276

iter 60 value 26.674979

iter 70 value 23.713939

iter 80 value 23.682110

iter 90 value 23.118995

iter 100 value 20.063019

iter 110 value 18.696817

iter 120 value 18.133749

iter 130 value 18.035818

final value 18.035775

converged

> yhat = predict(bc.nn,newdata=BC.valid,type="class")

> misclass(yhat,BC.valid$Diagnosis)

Table of Misclassification

(row = predicted, col = actual)

y

fit B M

B 79 3

M 1 32

Misclassification Rate = 0.0348

> bc.nn = nnet(Diagnosis~.,data=BC.train,size=2,decay=.025,maxit=5000)

# weights: 65

initial value 250.341830

iter 10 value 227.597697

iter 20 value 227.457969

iter 30 value 216.577281

iter 40 value 128.690673

iter 50 value 107.751863

iter 60 value 97.204637

iter 70 value 62.310209

iter 80 value 36.714397

iter 90 value 26.519342

iter 100 value 20.402546

iter 110 value 17.233374

iter 120 value 16.468071

iter 130 value 16.404572

iter 140 value 16.355124

iter 150 value 16.311390

iter 160 value 16.253751

iter 170 value 16.227197

iter 180 value 16.219677

final value 16.151290

converged

> yhat = predict(bc.nn,newdata=BC.valid,type="class")

> misclass(yhat,BC.valid$Diagnosis)

Table of Misclassification

(row = predicted, col = actual)

y

fit B M

B 78 3

M 2 32

Misclassification Rate = 0.0435

> bc.nn = nnet(Diagnosis~.,data=BC.train,size=2,decay=.01,maxit=5000)

# weights: 65

initial value 228.429432

iter 10 value 227.600782

iter 20 value 194.403199

iter 30 value 127.183025

iter 40 value 126.351005

iter 50 value 113.054875

…  
iter 160 value 11.285575

iter 170 value 11.284807

iter 180 value 11.284787

iter 180 value 11.284787

iter 180 value 11.284787

final value 11.284787

converged

> yhat = predict(bc.nn,newdata=BC.valid,type="class")

> misclass(yhat,BC.valid$Diagnosis)

Table of Misclassification

(row = predicted, col = actual)

y

fit B M

B 78 2

M 2 33

Misclassification Rate = 0.0348

> bc.nn = nnet(Diagnosis~.,data=BC.train,size=3,decay=.01,maxit=5000)

# weights: 97

initial value 300.647538

iter 10 value 227.740290

iter 20 value 181.445811

iter 30 value 124.360033

iter 40 value 114.879777

iter 50 value 111.265831

iter 60 value 110.076387

iter 70 value 107.547163

iter 80 value 101.331972

iter 90 value 92.028990

iter 100 value 90.402311

iter 110 value 86.065467

…

iter 250 value 5.878389

iter 260 value 5.856474

iter 270 value 5.853542

iter 280 value 5.853154

iter 290 value 5.853041

iter 300 value 5.853010

iter 310 value 5.853003

final value 5.853003

converged

> yhat = predict(bc.nn,newdata=BC.valid,type="class")

> misclass(yhat,BC.valid$Diagnosis)

Table of Misclassification

(row = predicted, col = actual)

y

fit B M

B 78 6

M 2 29

Misclassification Rate = 0.0696

It appears that a 2 node hidden layer with decay = 0.01 is “optimal”. Finally we can predict for the test cases (BC.test) that have not been used at this point to characterize accuracy of future neural network predictions using this model.

> ytest = predict(bc.nn,newdata=BC.test,type="class")

> misclass(ytest,BC.test$Diagnosis)

Table of Misclassification

(row = predicted, col = actual)

y

fit B M

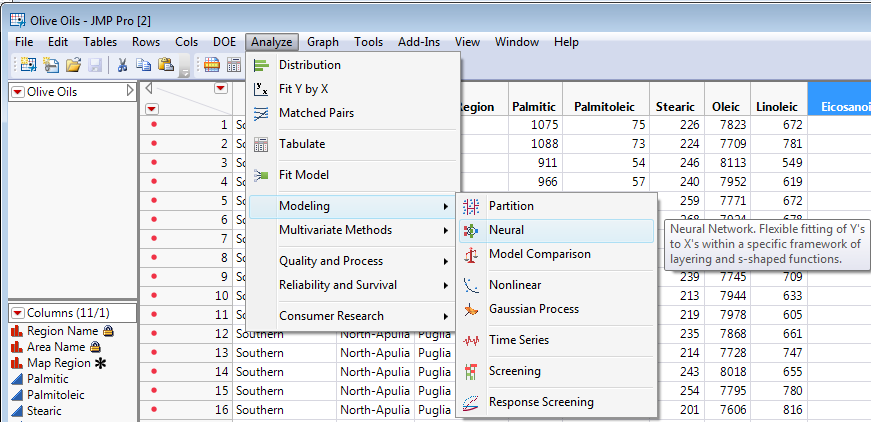
B 65 2

M 3 43

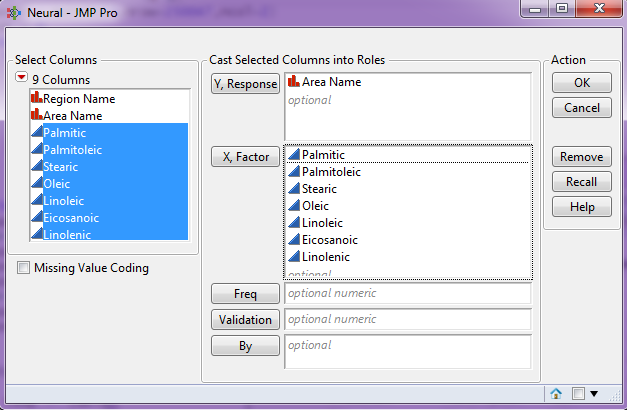
Misclassification Rate = 0.0442

**16.2 - Neural Networks in JMP**

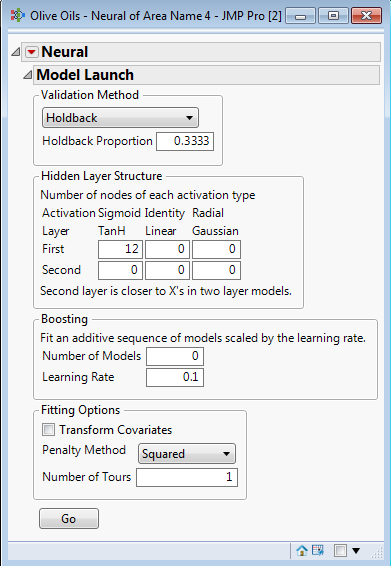
Neural network modeling in JMP is done by first selecting **Analyze > Modeling > Neural** as shown below.



The model specification dialog box is shown below



Next we choose the number of hidden nodes to use and the method of cross-validation to use in the model development process. Holdback will set aside a certain percentage of the observations and use those as a test set, the remaining observations will be used as a training set. K-Fold will use -fold cross-validation to determine the predictive abilities of the model. Recall that -fold CV will break the data into chunks. One of the chunks is left out and model is developed from the remaining pieces. The left out chunk is then predicted back and this process is repeated times until each chunk has been left out.

Below is the neural network specification dialog box.  


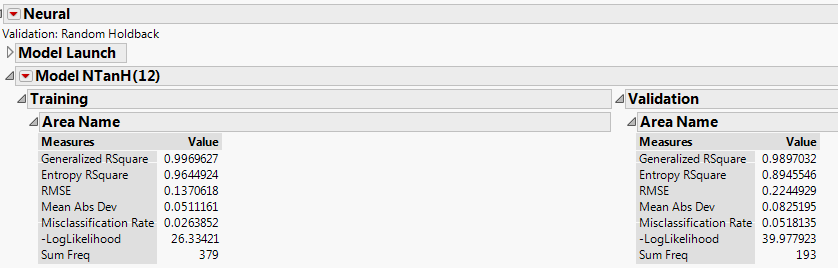
***Boosting*** with non-variable weights is used. Bootstrap samples are drawn with observations misclassified in the previous model given more weight.

Number of Models

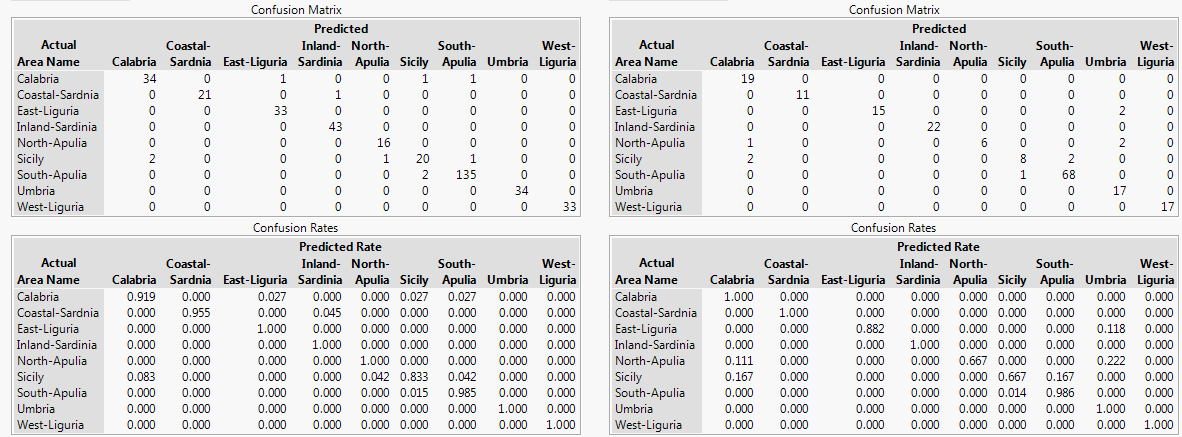
Learning Rate

Cross-validation options.

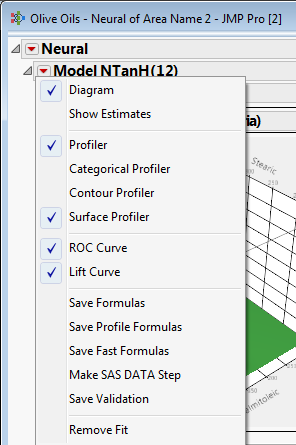
High-level model summary



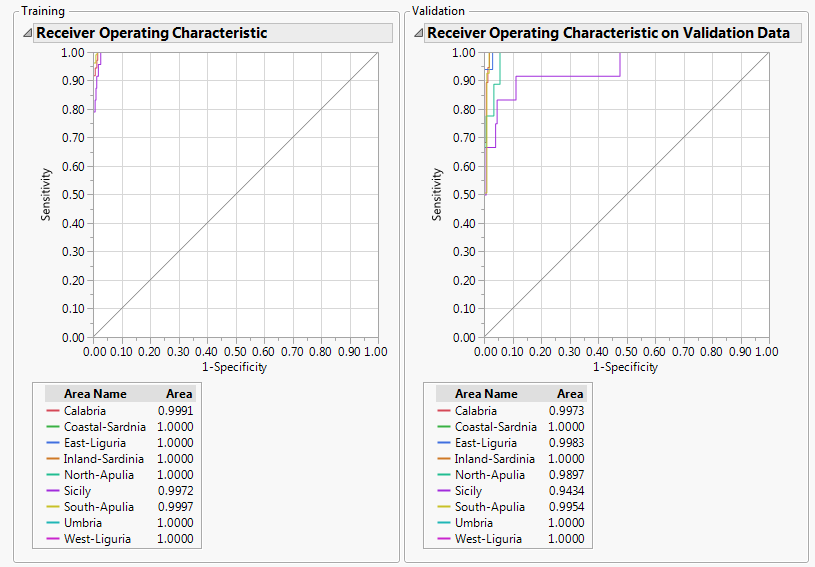
Confusion Matrices and Prediction Accuracy by Growing Area  
for Training and Validation (Holdback) Sets



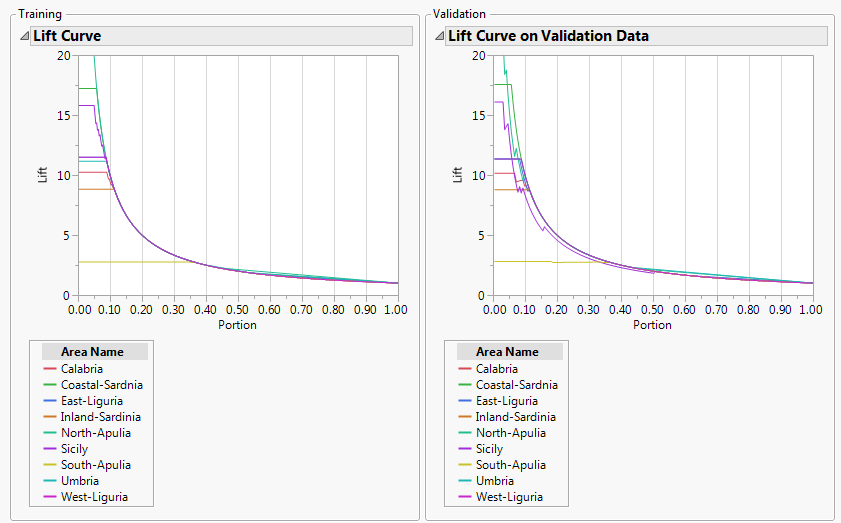
Useful options for visualizing the model and model performance are available in the drop down menu for the model.



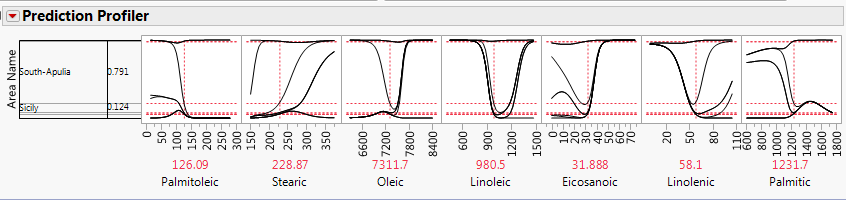
ROC Curves and AUC values are outstanding



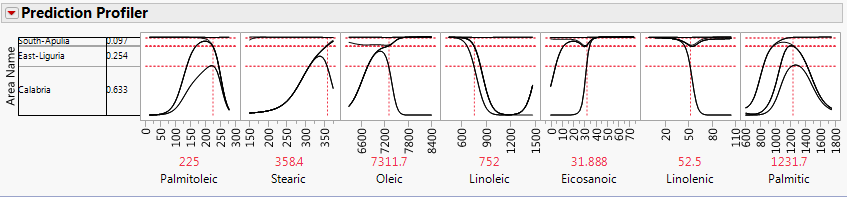
The Lift Curves are also very good.



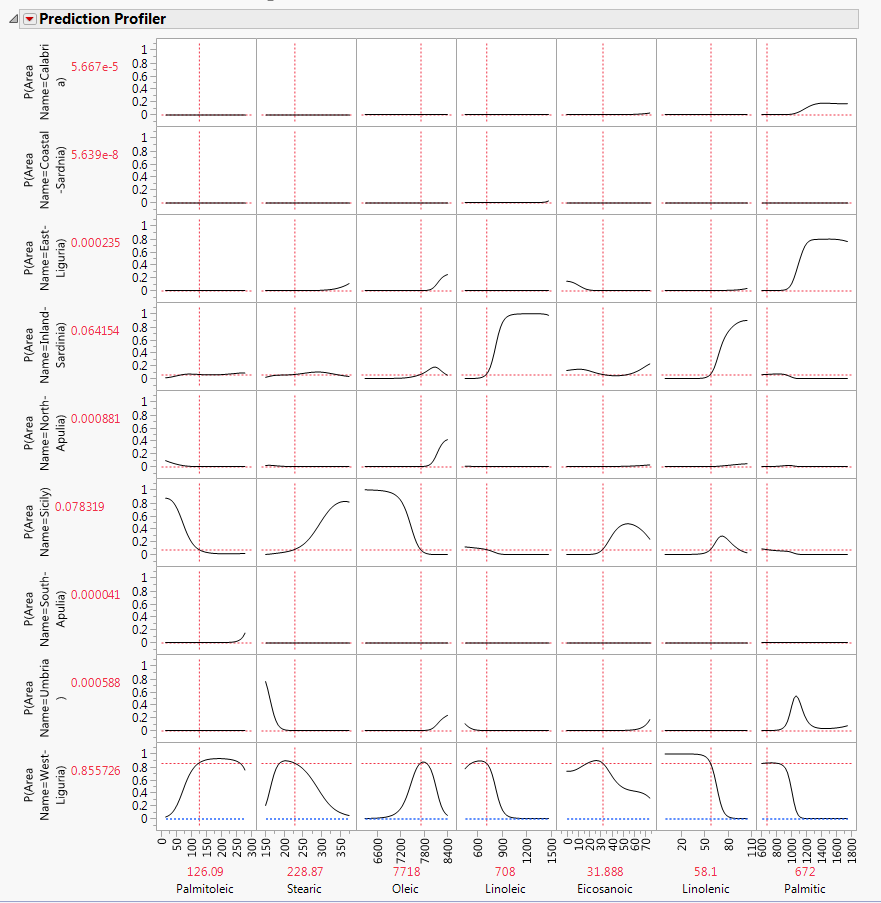
Categorical profiler allows you to examine for a particular choice of values for the covariates. Here the predicted class is South Apulia.



Another view with the predictor values changed. Here the predicted class is Calabria.

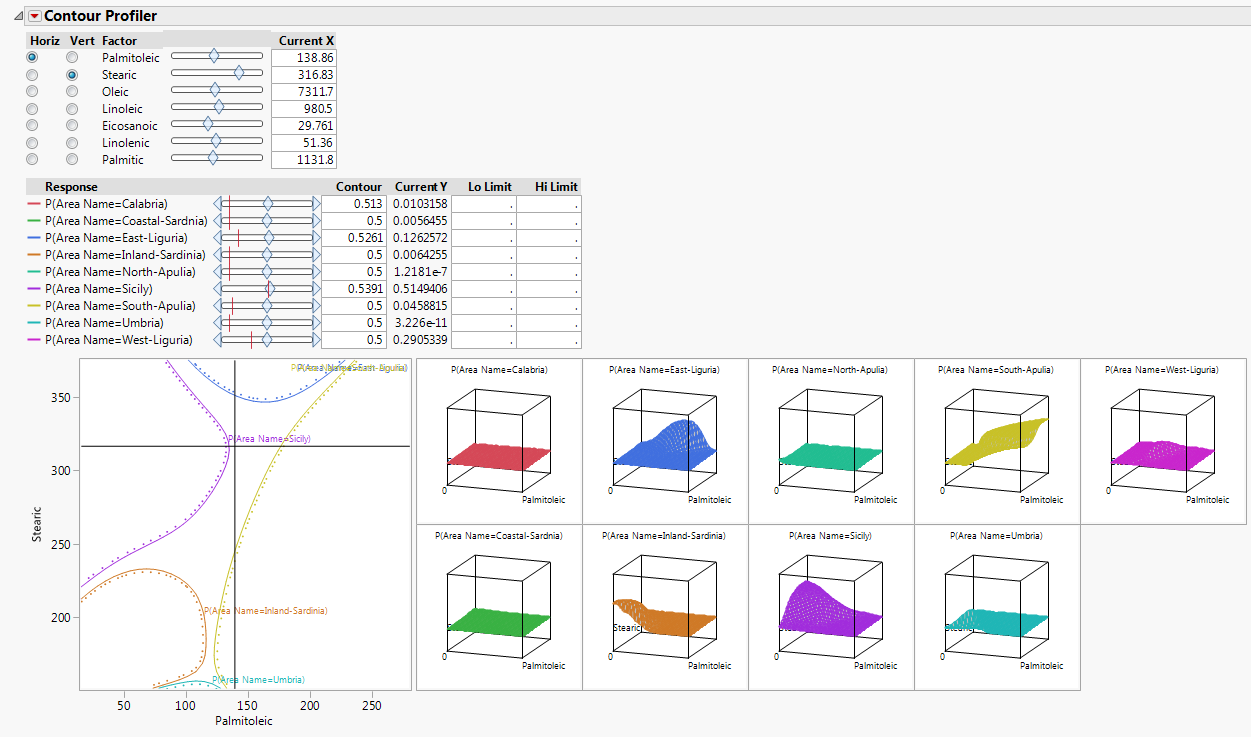


The profiler breaks the into a separate series of profilers for each growing area separately.

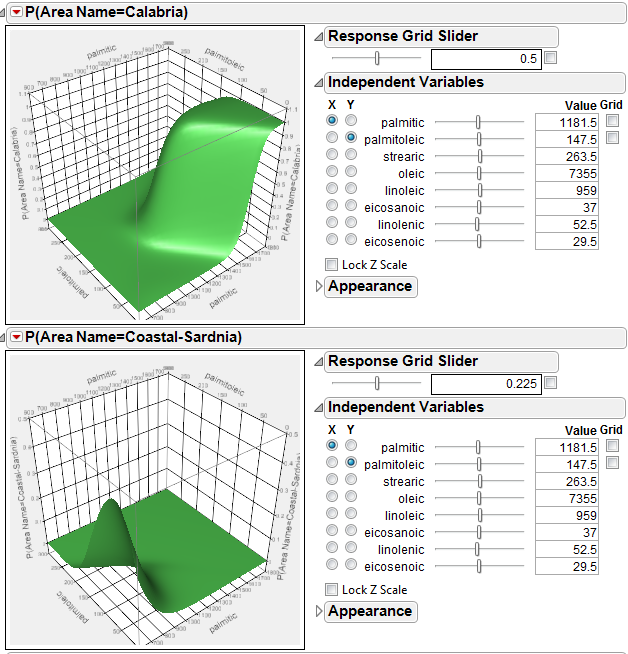


The predicted class is clearly West-Liguria.

You can also examine the predicted probabilities as a surface using the Surface and Contour Profilers. Below is the **Contour Profiler**.



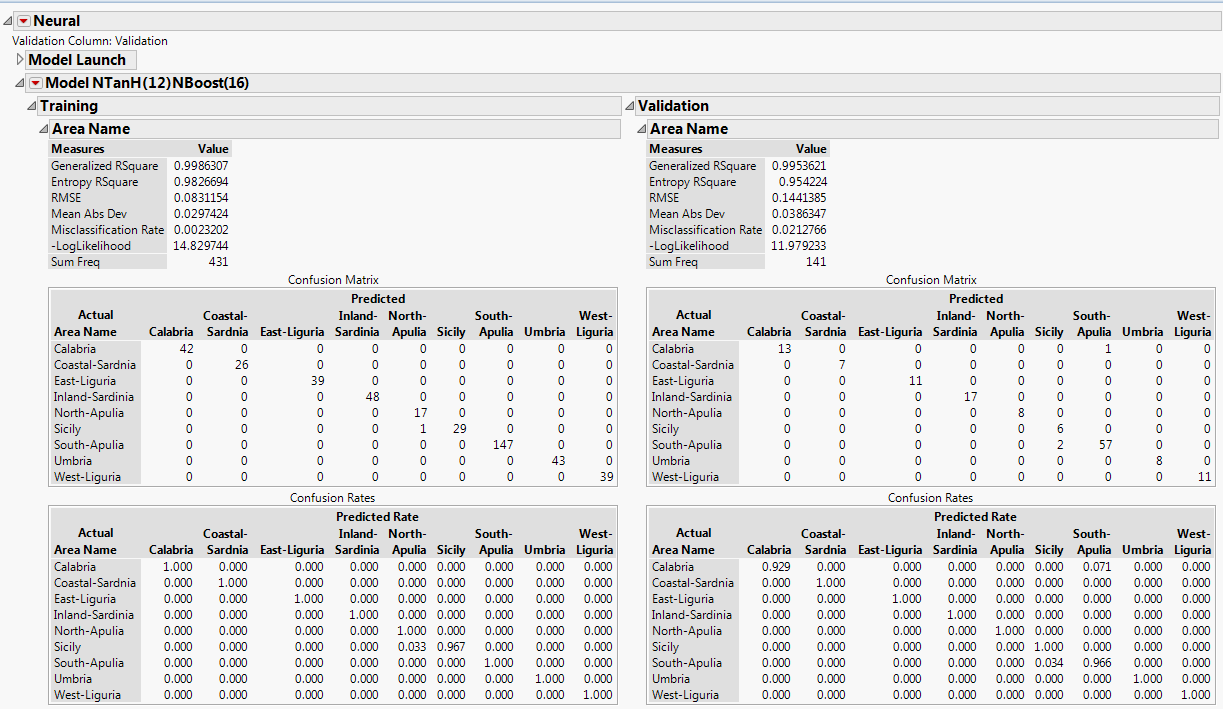
The **Surface Profiler** allow you consider separate surfaces for each category, two variables at a time.



You can save the prediction formulae to the JMP spreadsheet which will allow you to add X values for a “test” set of data and obtain predictions for future observations by appending those rows to the training data set.

You can also use JMP to fit a boosted neural network by specifying the number of models to average and the weight to use on the subsequent neural networks fit to the weighted observations at each stage. Also I used a validation column with 75% training and 25% test cases.

Below is a summary of the boosted neural network with a single layer of 12 hidden nodes with up to 30 models used in the boosting sequence fit to the olive oil data. We can see the misclassification rate on the training data and the validation set are both very good. To further refine this model you could consider changing the learning rate (here set to .10) and altering the number of layers and hidden nodes per layer. As only 16 models where actually used in the boosting sequence, altering the number of models from 30 would probably not change the results.



Only 2.12% of the validation cases were misclassified. This is superior to anything achieved using the R implementation of neural networks.