Using JMP to Compare Models from Various Environments

Lucas Beverlin

October 15, 2020

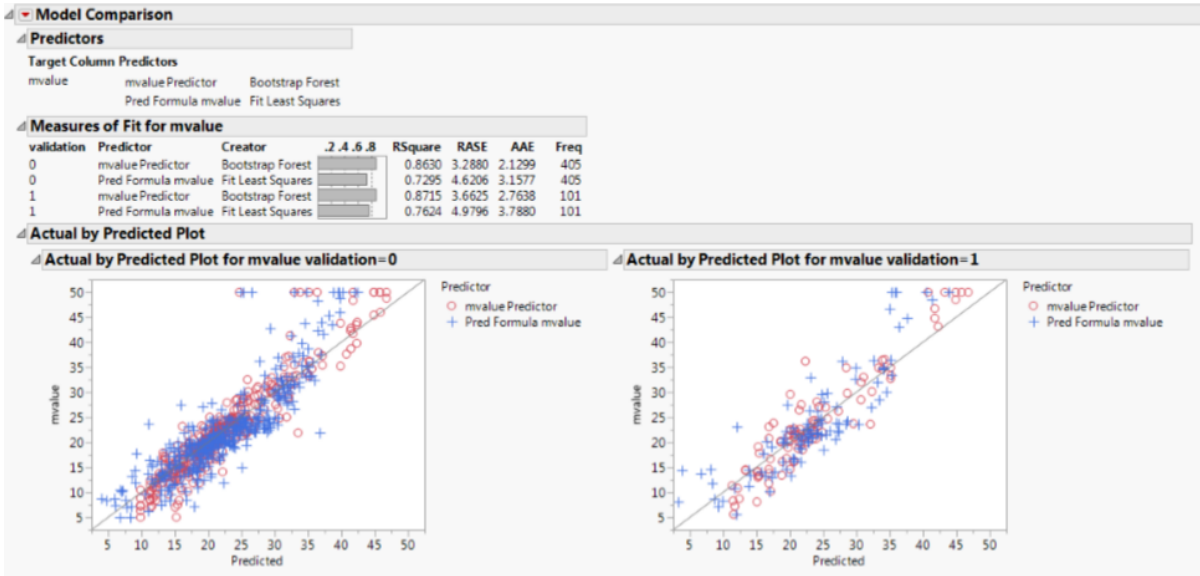
JMP Can Do Many Things

JMP 15 Pro has several platforms that can build a model:

- Fit Model
- Neural
- Partition
- Nonlinear
- And more!

And Compare the Models

The Model Comparison platform can be used to compare fits from various JMP platforms, to help the user determine which model is best.



(Figure 10.1 from the Predictive and Specialized Modeling pdf within the JMP 15 Pro help.)

But It Can't Do Everything

There are some models that JMP cannot fit, and some limitations to those that it can:

- A neural network with more than two hidden layers, e.g. autoencoders, convolutional neural networks, etc., and/or activation functions other than the usual 3 JMP Pro offers
- Projection pursuit regression
- Multivariate adaptive regression splines (MARS)

Software That Can Fit Other Models

- R
- Python
- MATLAB

Of course, there are many more, but for this talk we'll focus on these three. The ideas here can easily extend to basically any software!

The Power of Model Comparison

We can use the Model Comparison platform to compare model fits from other software too!

What you need:

- The model predictions from whatever you used to fit the model.
- If you split the data into training/validation/test sets, something that delineates those sets.

Calling R from JMP

JMP's scripting language (JSL) can create its own R session and run code from it.

```
dt = Open("C:\Program Files\SAS\JMPPRO\14\Samples\Data\Boston Housing.jmp");

Set Environment Variable( "R_HOME", "C:\Program Files\R\R-3.6.3" );//Handy if you have multiple versions of R!

R Init();
R Send(dt);
R Submit( "

lm1<-lm(y~.,data=dt)
preds<-predict(lm1,dt)

" );

R Get(preds)
```

https://www.r-project.org

Notes: I am using R 3.6.3 for this demonstration. JSL can call any version of R ¿= 2.9.1

Calling Python from JMP

JSL can create its own Python session and run code from it.

```
dt = Open("C:\Program Files\SAS\JMPPRO\14\Samples\Data\Boston Housing.jmp");

Python Init();
Python Send(dt);
Python Submit("
import numpy as np

X = dt.iloc[:, :-1].values
y = dt.iloc[:, 1].values

from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X, y)

y_pred = regressor.predict(X)

");

Python Get(y_pred)
```

https://python.org

Notes: While JSL can call the newest version of Python, which at the time of recording was 3.8.5, it cannot call versions of Python greater than 3.6.x if you use the Anaconda installer. Thus, I am using Python 3.6.5 for this demonstration. JMP recommends

Calling MATLAB from JMP

JSL can create its own MATLAB session and run code from it.

```
dt = Open("C:\Program Files\SAS\JMPPRO\14\Samples\Data\Boston Housing.jmp");

Set Environment Variable( "MATLABROOT", "C:\MATLAB\R2019b");

dtm = dt<<Get as Matrix;
MATLAB Init();
MATLAB Send(dtm);
MATLAB Submit("

    X=dtm(:,1:13)
    y=dtm(:,14)
    lsfit = fitlm(X,y,'Categorical',[4])
    preds = predict(lsfit,X)

");

MATLAB Get(preds)
```

https://www.mathworks.com/products/matlab.html

Notes: I am using Matlab R2019b for this demonstration. JSL can call any version of MATLAB ¿= R2012a (7.14.0).

Using JSL to Fit Models and Run Model Comparison

So how do we tie this all together?

1. Fit each model, sending each software the data, and which set (training/validation/test) each observation resides.
2. Output the model fits, and add them to the JMP data table. (Be sure to name them so that you can tell one from the other!)
3. Depending on the model, you may want some model diagnostics. Those can be outputted as well.
4. Use the Model Comparison platform to compare all the fits, and it will present tables for output showing the best in each set (training/validation/test).

Model Fitting

For the first step, we can do one of two things:

1. Tell JMP via JSL to call your software of choice, send it the data, and the code to fit it.
2. Create a dataset with a validation column, and schedule it to run on your software of choice, outputting what you need (predictions, diagnostic plots, etc.).
   Warnings:
   - Make sure all software has what it needs to fit the model!
   - Make sure all software has finished running before proceeding!
   - Make sure the output format is something JMP can read!
   - Make sure the predictions from each model correspond to the correct observations from the original data set!

What I Used to Fit Models

In R I used the following additional packages to fit a neural network:

- Keras
- Tensorflow

In Python, I used the following libraries:

- Keras
- Tensorflow
- numpy - to do a few calculations
- pandas - to manipulate the data a bit
- matplotlib - to create plots

In MATLAB, I used the deep learning toolbox.

Adding Predictions to the JMP Data Table

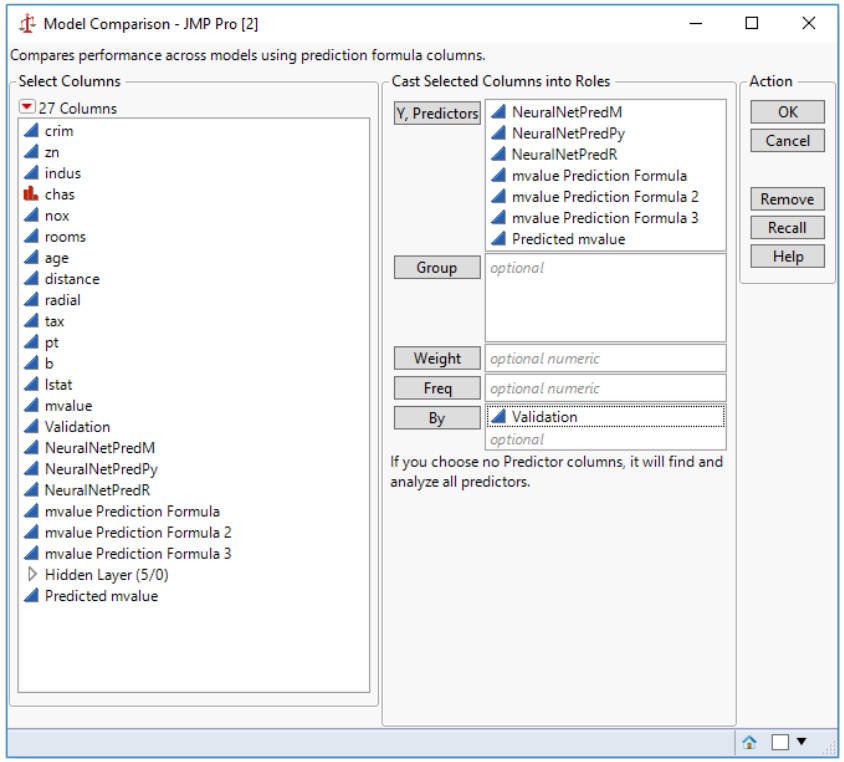
1. If you used JMP to call the software, you can use JSL code to retrieve the predictions and add them to the data table.
2. If the software ran on its own, and saved the output, you can ask JMP to read the output, and add it as a new column. We can also read the diagnostic plots, assuming they were saved as graphics files.

To help with identification later, I recommend adding the following property for each prediction column you add:

```
dt << Set Property("Predicting",{:mvalue, Creator("R")})
```

Here this will call the model's creator as R in the output. Of course, choose whatever string you wish to identify each model, particularly if multiple models come from the same software.

Model Comparison



Note that you can also use the Validation column here as a Group variable instead of a By variable.

Model Comparison Output

Training Diagnostics

Predictors

Measures of Fit for mvalue

Predictor	Creator	.2 .4 .6 .8	RSquare	RASE	AAE	Freq
NeuralNetPredM	MATLAB		0.7822	4.5525	3.0060	304
NeuralNetPredPy	Python		0.7623	4.7561	3.3869	304
NeuralNetPredR	R		0.6111	6.0837	4.2576	304
mvalue Prediction Formula	Fit Generalized Standard Least Squares		0.7464	4.9129	3.4574	304
mvalue Prediction Formula 2	Fit Generalized Lasso		0.7372	5.0008	3.4305	304
mvalue Prediction Formula 3	Fit Generalized Ridge		0.7295	5.0743	3.4247	304
Predicted mvalue	Neural		0.7781	4.5952	3.3754	304

Validation Diagnostics

Predictors

Measures of Fit for mvalue

Predictor	Creator	.2 .4 .6 .8	RSquare	RASE	AAE	Freq
NeuralNetPredM	MATLAB		0.7890	3.9142	2.8937	101
NeuralNetPredPy	Python		0.7171	4.5324	3.3725	101
NeuralNetPredR	R		0.6044	5.3600	3.5845	101
mvalue Prediction Formula	Fit Generalized Standard Least Squares		0.7077	4.6070	3.4992	101
mvalue Prediction Formula 2	Fit Generalized Lasso		0.7198	4.5105	3.3940	101
mvalue Prediction Formula 3	Fit Generalized Ridge		0.7339	4.3954	3.2904	101
Predicted mvalue	Neural		0.7853	3.9483	3.0428	101

Test Diagnostics

Predictors

Measures of Fit for mvalue

Predictor	Creator	.2 .4 .6 .8	RSquare	RASE	AAE	Freq
NeuralNetPredM	MATLAB		0.6542	4.6219	3.1106	101
NeuralNetPredPy	Python		0.8027	3.4916	2.8458	101
NeuralNetPredR	R		0.6171	4.8637	3.5895	101
mvalue Prediction Formula	Fit Generalized Standard Least Squares		0.6757	4.4761	3.3397	101
mvalue Prediction Formula 2	Fit Generalized Lasso		0.6898	4.3778	3.2233	101
mvalue Prediction Formula 3	Fit Generalized Ridge		0.7067	4.2565	3.1275	101
Predicted mvalue	Neural		0.6993	4.3100	3.2619	101

Here, had you used Validation as a Group variable, this would be one table instead of three.

Example: Boston Housing

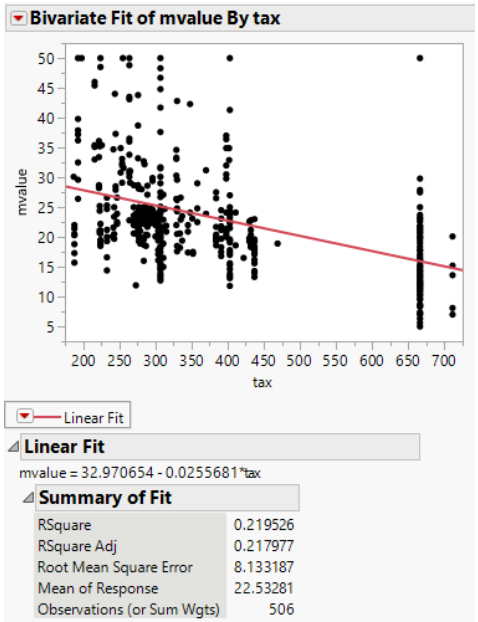
We will use the classic Boston Housing dataset. (Harrison & Rubinfeld 1978)

- mvalue - Median value of a house in thousands
- crim - Per capita crime rate by town
- zn - Proportion of residential land zoned for lots over 25,000 ft^2
- indus - Proportion of non-retail business acres per town
- chas - 1 if the tract borders the Charles River, and 0 otherwise
- nox - NOx concentration (parts per 10 million)
- rooms - Average number of rooms per dwelling
- age - Proportion of owner-occupied units built before 1940
- distance - Weighted distance to 5 Boston employment centers
- radial - Index of accessibility to radial highways
- tax - Full-value property tax rate per \$10,000
- pt - Pupil/teacher ratio by town
- b $= 1000(B - 0.63)^2$, where B $=$ proportion of African Americans by town
- lstat - % lower status of the population

Models to Consider

With the Model Comparison platform, we can compare all types of models to one dataset. In this presentation, we'll consider the following:

Linear Regression



Ridge Regression

Ridge regression is linear regression with the constraint that $\sum_{k=1}^{m} \beta_k^2 \leq c$, or, equivalently,

$$\hat{\beta} = \min_{\beta} \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{k=1}^{m} \beta_k x_{ki} \right)^2 + \lambda \sum_{k=1}^{m} \beta_k^2.$$

This method shrinks the coefficients toward 0.

The Lasso

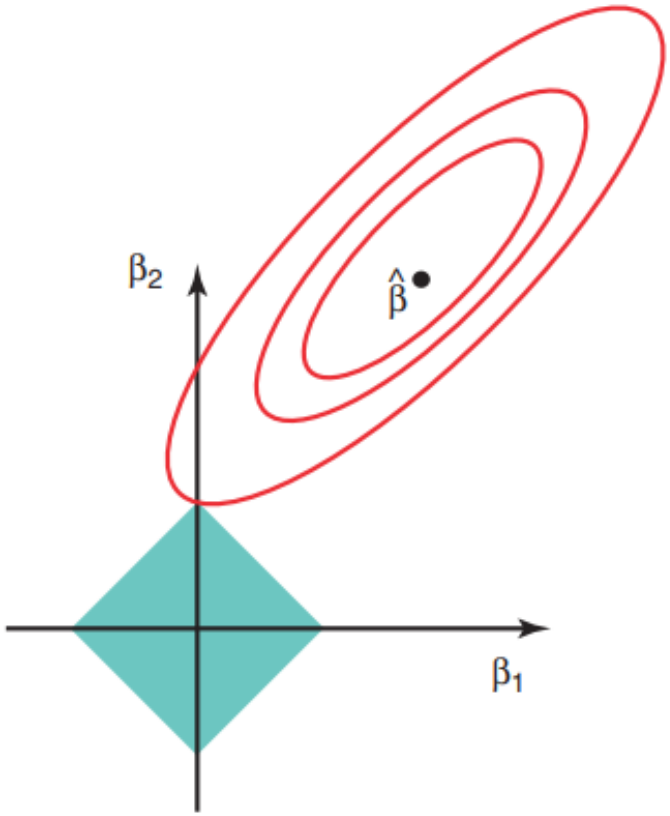
The lasso is linear regression with the constraint that $\sum_{k=1}^{m} |\beta_k| \leq c$, or, equivalently,

$$\hat{\beta} = \min_{\beta} \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{k=1}^{m} \beta_k x_{ki} \right)^2 + \lambda \sum_{k=1}^{m} |\beta_k| \,.$$

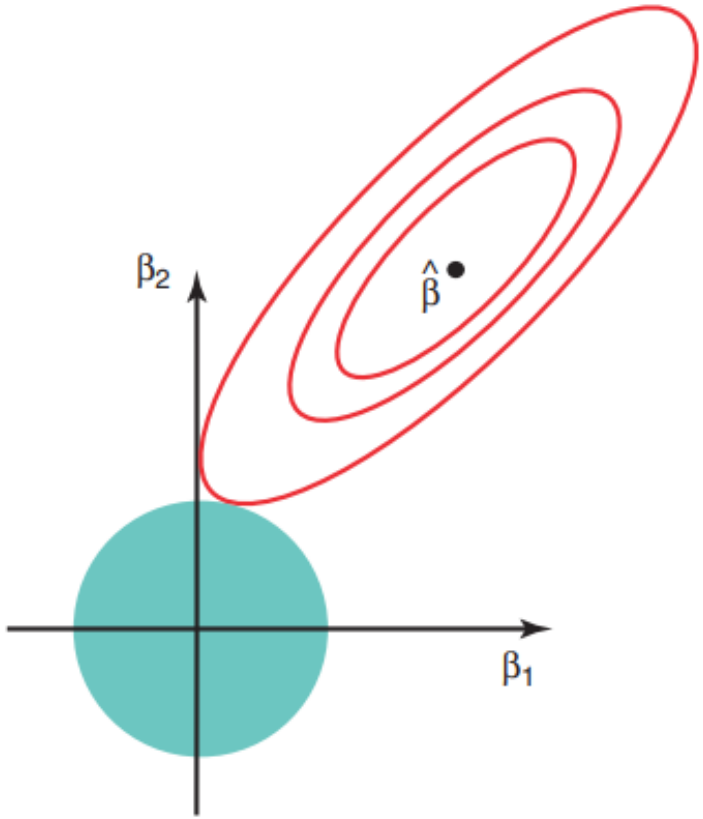
This method can eliminate inputs whose contribution to the response is weak.

The Lasso and Ridge Regression

Lasso

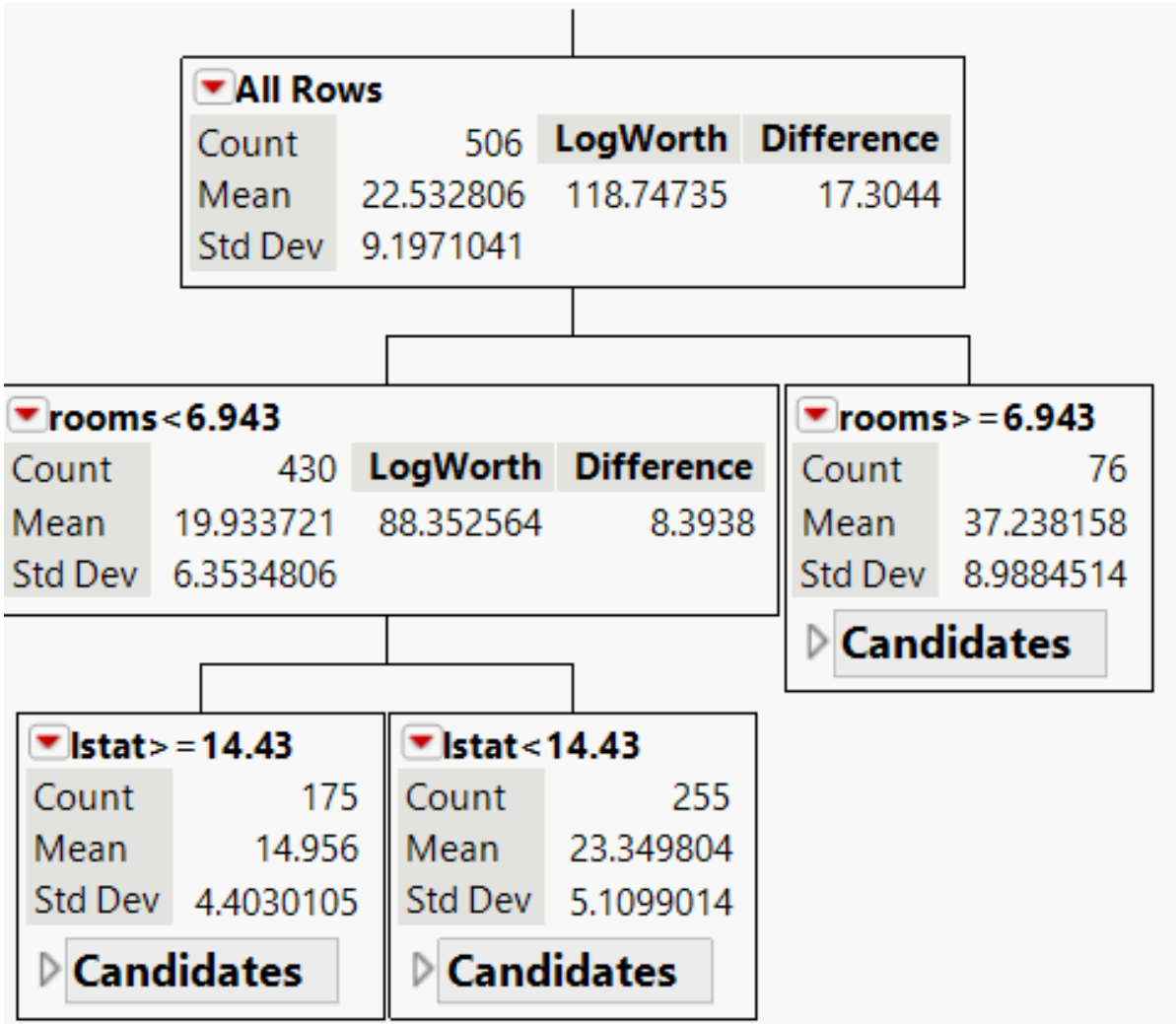


Ridge Regression



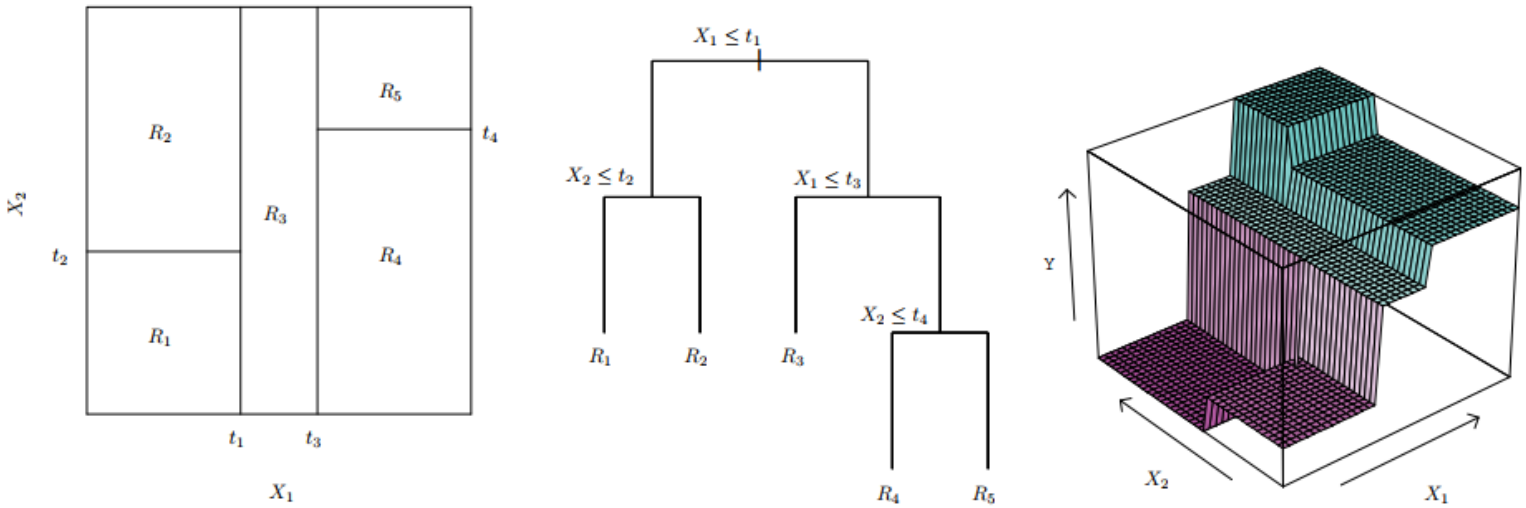
Thanks James, Witten, Hastie, and Tibshirani!

Regression Trees



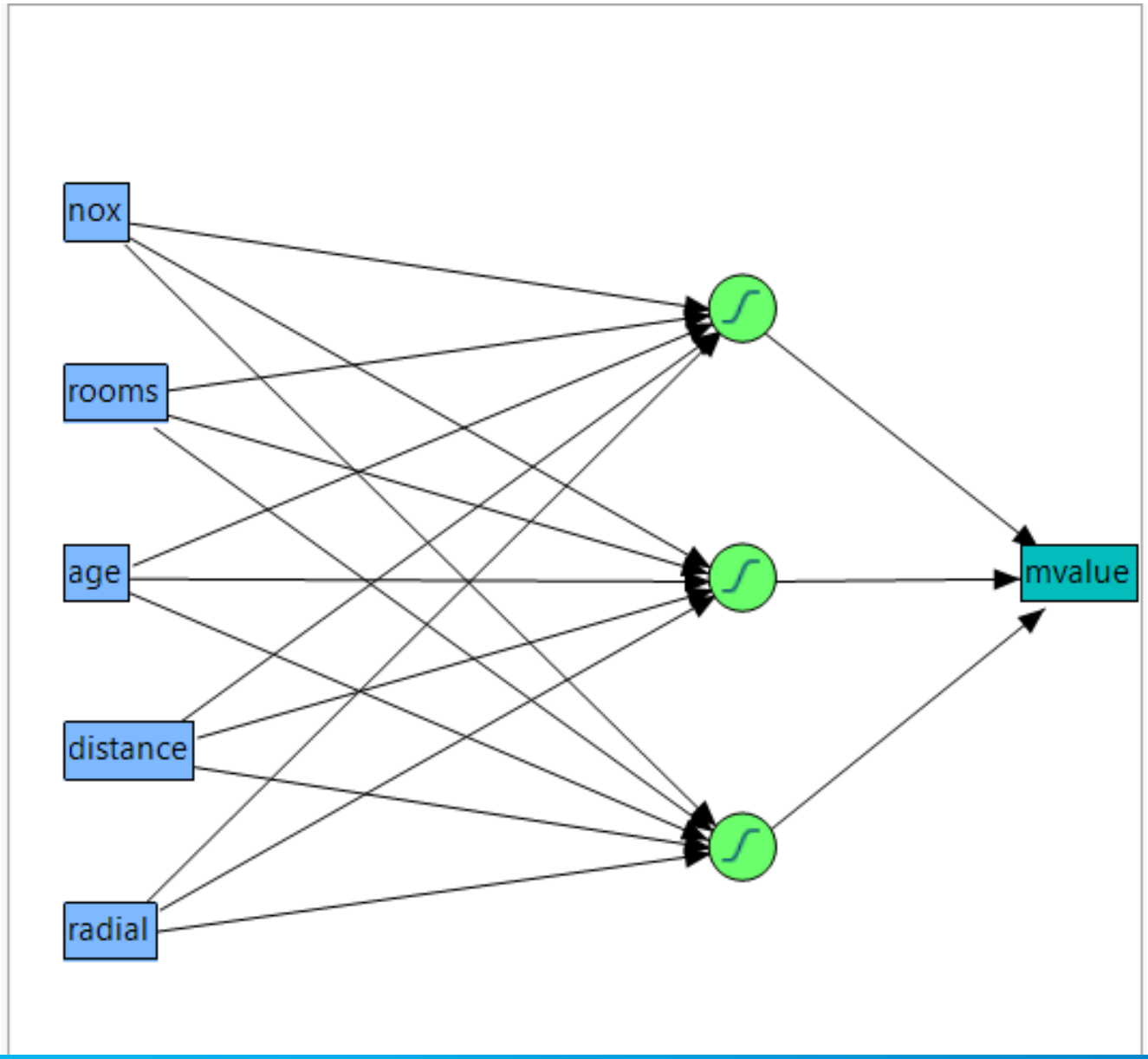
Regression Trees

Regression trees are models where the input space is partitioned into areas, and the prediction is the average response of the observations in that area.



Thanks James, Witten, Tibshirani, and Friedman!

Neural Networks



Neural Networks

Neural networks are nonlinear models whose representation can be drawn as a series a neurons, much like how we believe the brain works.

- Each input can be passed to nodes in a hidden layer.
- At the hidden layer, the inputs are pushed through an activation function, and an output is calculated.
- Each output can be passed to a node in another hidden layer, or be an output of the network.

Notes on the Neural platform within JMP Pro:

- There are only three options for activation functions: Linear, Hyperbolic Tangent, and Gaussian Radial Basis.
- JMP Pro only allows for 2 hidden layers. Deep learning neural networks, such as autoencoders, often have dozens.

Projection Pursuit Regression

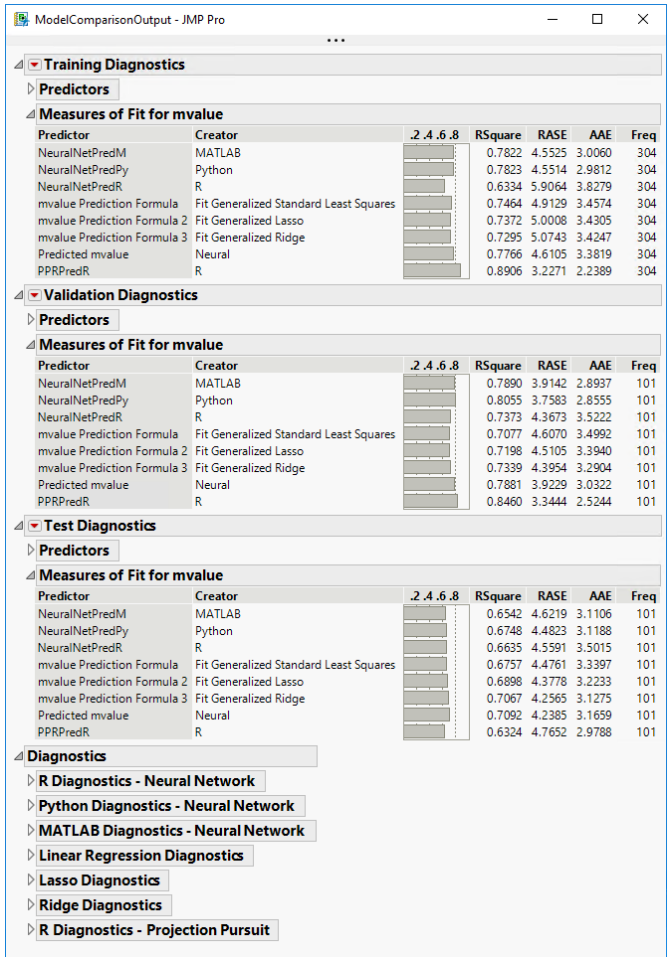
Projection pursuit regression, originally proposed by Friedman and Stuetzle (1981), is a model whose predictions take the form:

$$\mathbf{y} = \sum_{i=1}^{k} \beta_i f_i(\alpha' \mathbf{X}).$$

One can see that this is somewhat analogous to a neural network with one hidden layer with k nodes, each with activation function f_i. However, here, f_i is also estimated, as it is usually a smoother or a spline fit. (The f_i's are called ridge functions.) Of course, by altering the α and β_i, the optimal smooth will change, so fitting is usually done stagewise.

Model Comparison in Action - JMP Journal Output

Do note that I had difficulty setting the seeds for the neural networks fit in Keras for R and Python, so some of your results will differ slightly from mine.


ModelComparisonOutput - JMP Pro

Training Diagnostics

Predictors

Measures of Fit for mvalue

Predictor	Creator	.2 .4 .6 .8	RSquare	RASE	AAE	Freq
NeuralNetPredM	MATLAB		0.7822	4.5525	3.0060	304
NeuralNetPredPy	Python		0.7823	4.5514	2.9812	304
NeuralNetPredR	R		0.6334	5.9064	3.8279	304
mvalue Prediction Formula	Fit Generalized Standard Least Squares		0.7464	4.9129	3.4574	304
mvalue Prediction Formula 2	Fit Generalized Lasso		0.7372	5.0008	3.4305	304
mvalue Prediction Formula 3	Fit Generalized Ridge		0.7295	5.0743	3.4247	304
Predicted mvalue	Neural		0.7766	4.6105	3.3819	304
PPRPredR	R		0.8906	3.2271	2.2389	304

Validation Diagnostics

Predictors

Measures of Fit for mvalue

Predictor	Creator	.2 .4 .6 .8	RSquare	RASE	AAE	Freq
NeuralNetPredM	MATLAB		0.7890	3.9142	2.8937	101
NeuralNetPredPy	Python		0.8055	3.7583	2.8555	101
NeuralNetPredR	R		0.7373	4.3673	3.5222	101
mvalue Prediction Formula	Fit Generalized Standard Least Squares		0.7077	4.6070	3.4992	101
mvalue Prediction Formula 2	Fit Generalized Lasso		0.7198	4.5105	3.3940	101
mvalue Prediction Formula 3	Fit Generalized Ridge		0.7339	4.3954	3.2904	101
Predicted mvalue	Neural		0.7881	3.9229	3.0322	101
PPRPredR	R		0.8460	3.3444	2.5244	101

Test Diagnostics

Predictors

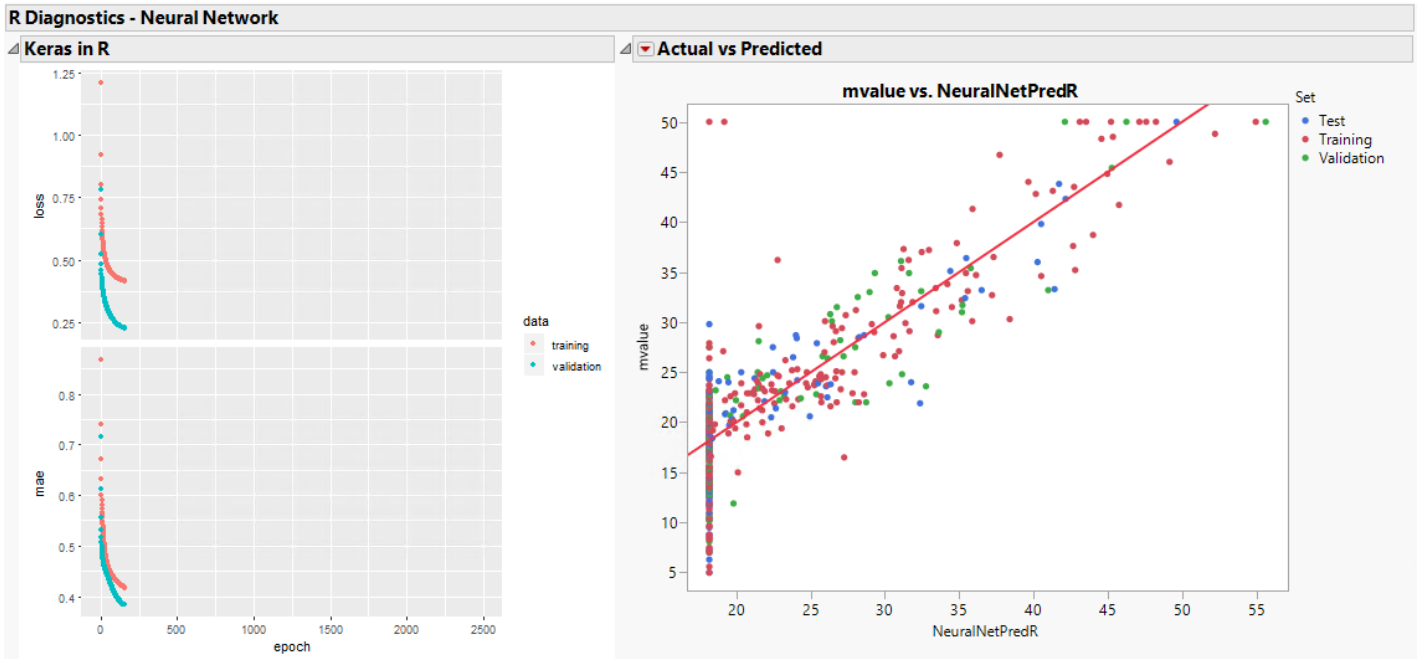
Measures of Fit for mvalue

Predictor	Creator	.2 .4 .6 .8	RSquare	RASE	AAE	Freq
NeuralNetPredM	MATLAB		0.6542	4.6219	3.1106	101
NeuralNetPredPy	Python		0.6748	4.4823	3.1188	101
NeuralNetPredR	R		0.6635	4.5591	3.5015	101
mvalue Prediction Formula	Fit Generalized Standard Least Squares		0.6757	4.4761	3.3397	101
mvalue Prediction Formula 2	Fit Generalized Lasso		0.6898	4.3778	3.2233	101
mvalue Prediction Formula 3	Fit Generalized Ridge		0.7067	4.2565	3.1275	101
Predicted mvalue	Neural		0.7092	4.2385	3.1659	101
PPRPredR	R		0.6324	4.7652	2.9788	101

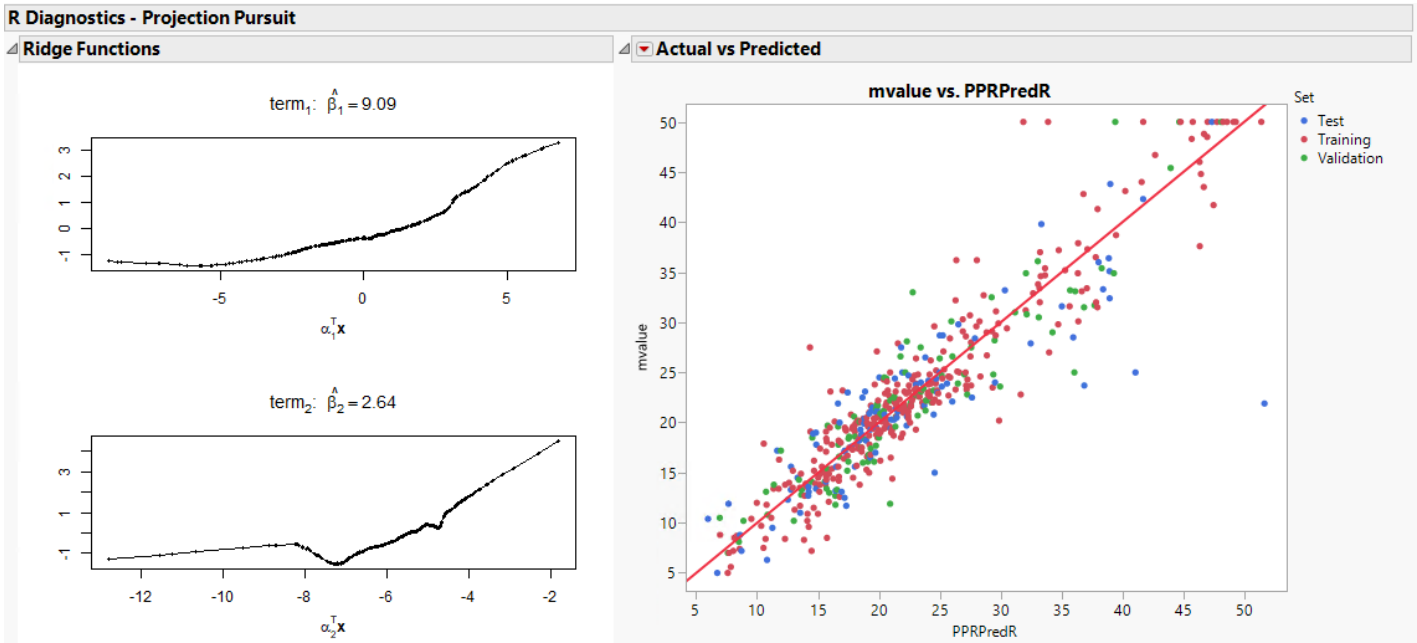
Diagnostics

- R Diagnostics - Neural Network
- Python Diagnostics - Neural Network
- MATLAB Diagnostics - Neural Network
- Linear Regression Diagnostics
- Lasso Diagnostics
- Ridge Diagnostics
- R Diagnostics - Projection Pursuit

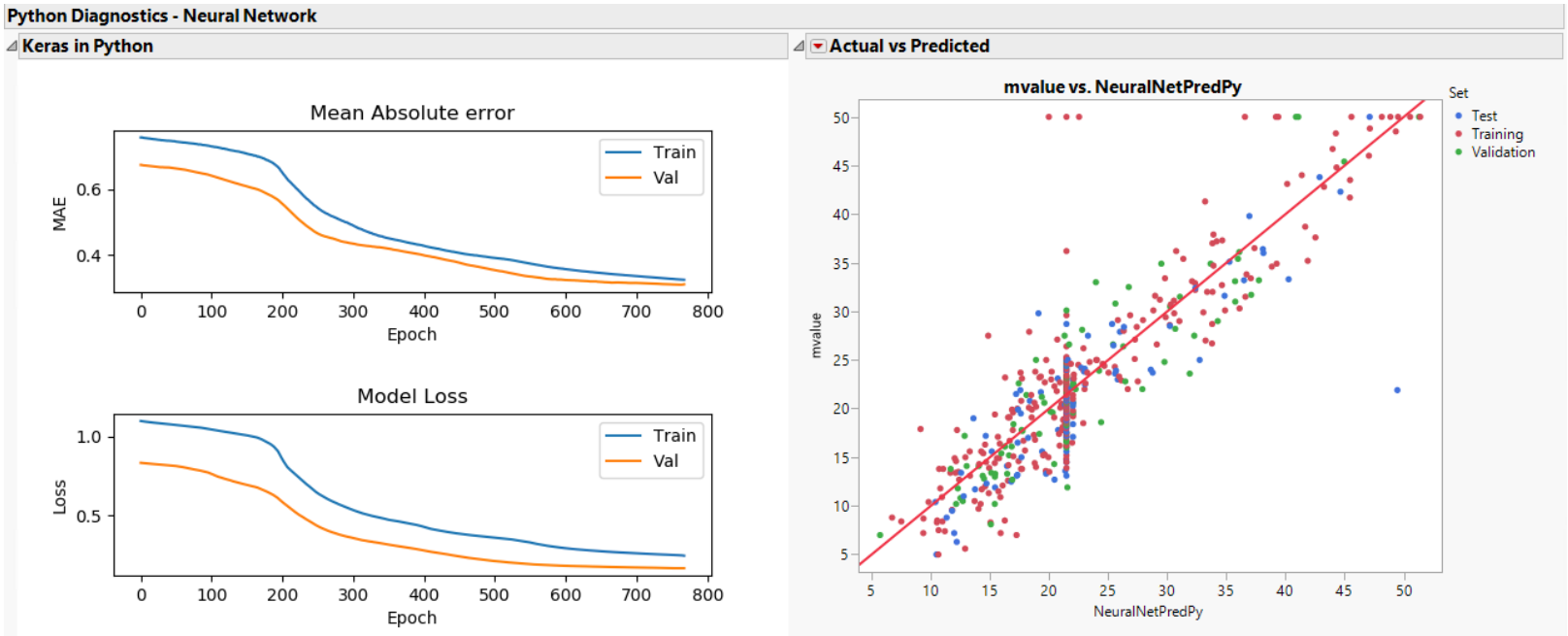
Model Comparison in Action - R Neural Network Diagnostic Plot



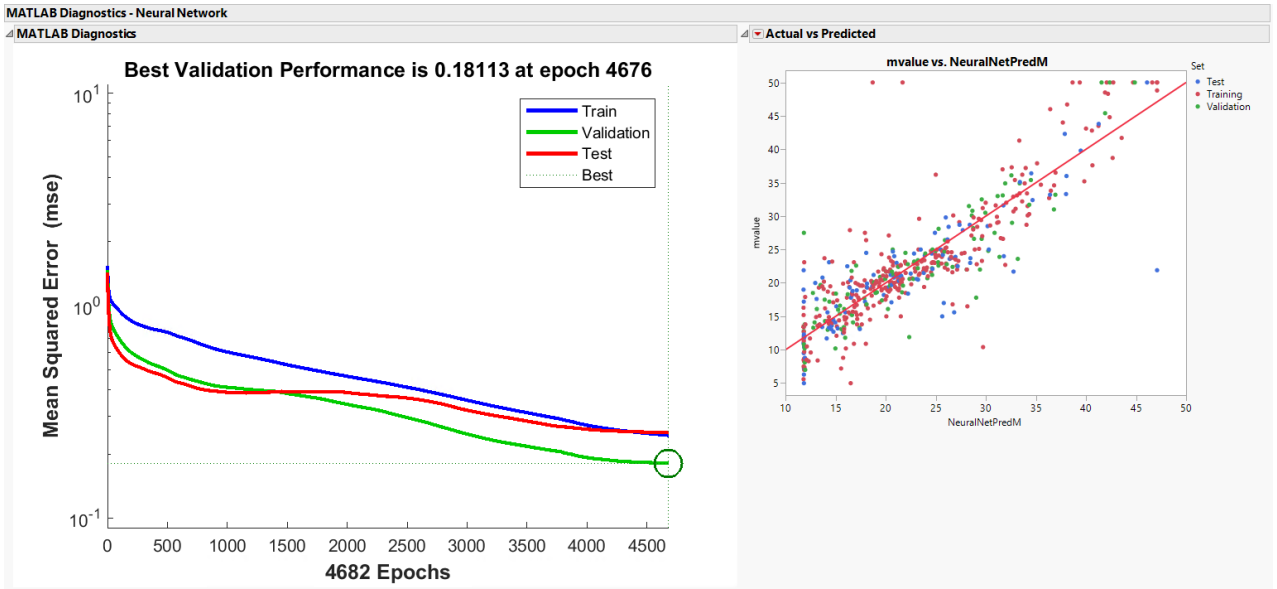
Model Comparison in Action - R Projection Pursuit Ridge Function Plots



Model Comparison in Action - Python Diagnostic Plot



Model Comparison in Action - MATLAB Diagnostic Plot



Model Comparison in Action - Linear Regression Diagnostic Plots and Output

Linear Regression Diagnostics

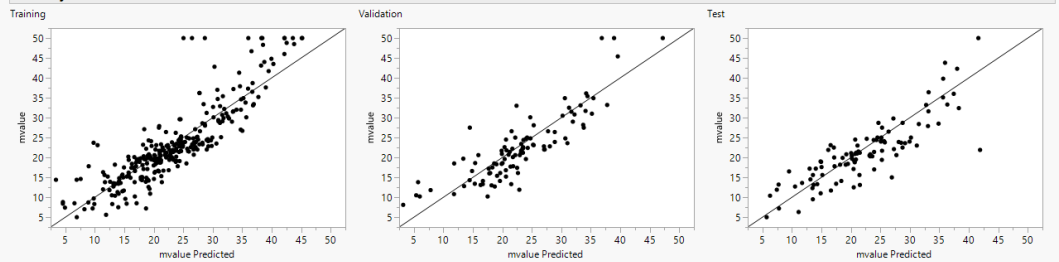
Regression Parameter Estimates

Parameter Estimates for Original Predictors

Term	Estimate	Std Error	Wald ChiSquare	Prob > ChiSquare	Lower 95%	Upper 95%
Intercept	36.112517	6.9717251	26.830885	<.0001*	22.448187	49.776847
crim	-0.139319	0.0383218	13.216935	0.0003*	-0.214429	-0.06421
zn	0.062282	0.0187403	11.045185	0.0009*	0.0255517	0.0990122
indus	0.0616746	0.0784578	0.6179313	0.4318	-0.0921	0.2154491
chas[0-1]	-3.9355	1.2158503	10.477062	0.0012*	-6.318522	-1.552477
nox	-17.50407	5.1447991	11.575534	0.0007*	-27.58769	-7.420444
rooms	4.3709758	0.5701532	58.772428	<.0001*	3.253496	5.4884556
age	0.0009407	0.0184103	0.0026106	0.9593	-0.035143	0.0370241
distance	-1.748226	0.2857906	37.419602	<.0001*	-2.308365	-1.188087
radial	0.3488901	0.0902008	14.960861	0.0001*	0.1720998	0.5256804
tax	-0.013031	0.0049001	7.0720366	0.0078*	-0.022635	-0.003427
pt	-0.873034	0.1834644	22.644299	<.0001*	-1.232618	-0.513451
b	0.010064	0.0035778	7.9123219	0.0049*	0.0030516	0.0170765
lstat	-0.572108	0.0676759	71.463958	<.0001*	-0.70475	-0.439465

Normal Distribution Parameters	Estimate	Std Error	Wald ChiSquare	Prob > ChiSquare	Lower 95%	Upper 95%
Scale	5.0300425	0.2114285	566	<.0001*	4.6156504	5.4444347

Actual by Predicted Plot



Model Comparison in Action - Lasso Diagnostic Plots and Output

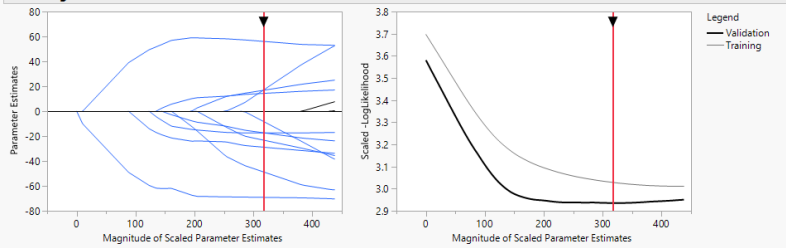
Lasso Parameter Estimates and Diagnostics

Parameter Estimates for Original Predictors

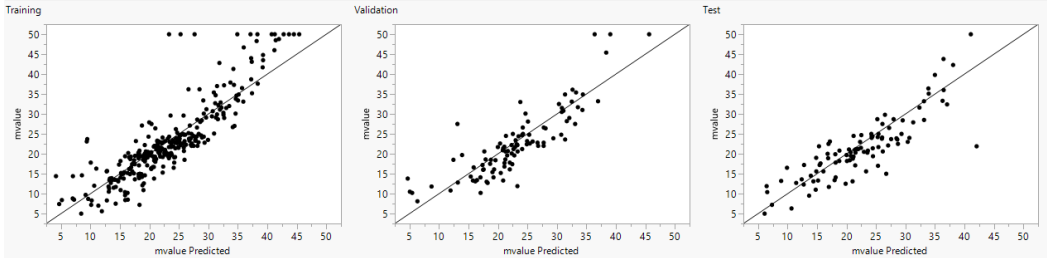
Term	Estimate	Std Error	Wald ChiSquare	Prob > ChiSquare	Lower 95%	Upper 95%
Intercept	26.675541	9.4865417	7.9069798	0.0049*	8.0822608	45.268821
crim	-0.101277	0.0315121	10.329182	0.0013*	-0.16304	-0.039514
zn	0.0420043	0.0177771	5.5829986	0.0181*	0.0071619	0.0768467
indus	0	0	0	1.0000	0	0
chas[0-1]	-4.019522	1.8030088	4.9699613	0.0258*	-7.553354	-0.485689
nox	-11.42086	4.5981566	6.1692157	0.0130*	-20.43308	-2.408637
rooms	4.6335552	0.9771665	22.484932	<.0001*	2.7183441	6.5487663
age	0	0	0	1.0000	0	0
distance	-1.349233	0.273822	24.279381	<.0001*	-1.885915	-0.812552
radial	0.1132661	0.0737758	2.3570677	0.1247	-0.031332	0.2578639
tax	-0.002793	0.0035891	0.6055647	0.4365	-0.009828	0.0042416
pt	-0.746198	0.1439871	26.857216	<.0001*	-1.028407	-0.463988
b	0.0084769	0.0038815	4.7696405	0.0290*	0.0008694	0.0160844
lstat	-0.561311	0.1216498	21.290381	<.0001*	-0.79974	-0.322881

Normal Distribution Parameters	Estimate	Std Error	Wald ChiSquare	Prob > ChiSquare	Lower 95%	Upper 95%
Scale	5.0007555	0.3865098	167.39802	<.0001*	4.2432103	5.7583008

Lasso Diagnostic Plots



Actual by Predicted Plot



Model Comparison in Action - Ridge Regression Diagnostic Plots and Output

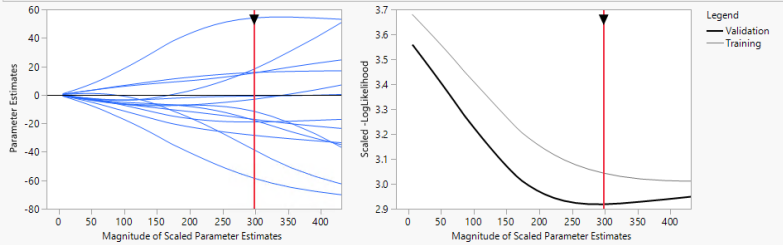
Ridge Parameter Estimates and Diagnostics

Parameter Estimates for Original Predictors

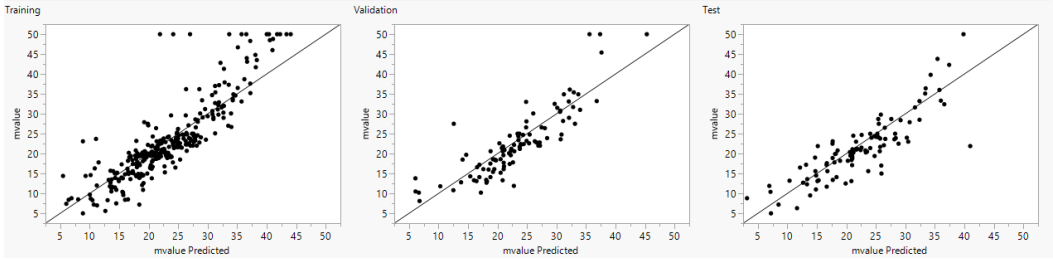
Term	Estimate	Std Error	Wald ChiSquare	Prob > ChiSquare	Lower 95%	Upper 95%
Intercept	24.73219	9.3704664	6.9663092	0.0083*	6.3664137	43.097967
crim	-0.100467	0.0311624	10.394043	0.0013*	-0.161544	-0.03939
zn	0.0385	0.0190199	4.0973658	0.0430*	0.0012217	0.0757784
indus	-0.024667	0.0611705	0.162611	0.6868	-0.144559	0.095225
chas[0-1]	-4.402922	1.9327746	5.189428	0.0227*	-8.19109	-0.614753
nox	-8.750434	4.5104149	3.7637971	0.0524	-17.59068	0.0898168
rooms	4.4710502	1.1060567	16.340462	<.0001*	2.3032189	6.6388814
age	-0.001622	0.0225744	0.0051654	0.9427	-0.045867	0.0426226
distance	-1.068877	0.265076	16.259777	<.0001*	-1.588417	-0.549338
radial	0.1201475	0.0754885	2.5331922	0.1115	-0.027807	0.2681022
tax	-0.003903	0.0035563	1.2045118	0.2724	-0.010873	0.0030672
pt	-0.734099	0.1412404	27.014192	<.0001*	-1.010925	-0.457273
b	0.0093097	0.003794	6.0211234	0.0141*	0.0018736	0.0167458
lstat	-0.474995	0.1331034	12.735033	0.0004*	-0.735873	-0.214117

Normal Distribution Parameters	Estimate	Std Error	Wald ChiSquare	Prob > ChiSquare	Lower 95%	Upper 95%
Scale	5.0742982	0.4041409	157.64725	<.0001*	4.2821966	5.8663998

Ridge Diagnostic Plots



Actual by Predicted Plot



Which Model Fit Best?

- Here we appear to see that the projection pursuit model fit the training data the best, as well as the validation data. However, it was worst on the test data!
- It appears that JMP's neural network fit the test data best.
- Here, I would recommend JMP's neural network since the fit is fairly consistent across all three sets of data.

Conclusions

- The Model Comparison platform is a powerful tool for comparing the predictive ability of multiple models.
- This flexibility can be extended to include models that were not fit with JMP!
- This can allow for several complex models to be fit simultaneously, and once the results are in, they can all be compared.

References

Friedman, J. and Stuetzle, W. "Projection Pursuit Regression," Journal of the American Statistical Association, Vol. 76, 817-823, 1981.

Harrison, D. and Rubinfeld, D.L. "Hedonic Prices and the Demand for Clear Air," J Environ. Economics & Management, Vol. 5, 81-102, 1978.

Hastie, T., Tibshirani R. and Friedman J. The Elements of Statistical Learning, Springer, New York, 12th printing, 2017.

James, G., Witten, D., Hastie, T. and Tibshirani, R. An Introduction to Statistical Learning with Applications in R. Springer, New York, 8th printing, 2017.