XGBoost Add-In for JMP Pro

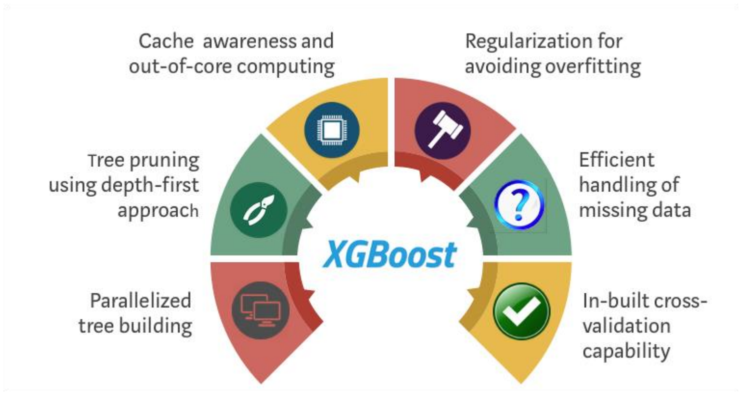
Overview

The XGBoost Add-In for JMP Pro provides a point-and-click interface to the popular XGBoost open-source library for predictive modeling with extreme gradient boosted trees. Value-added functionality includes:

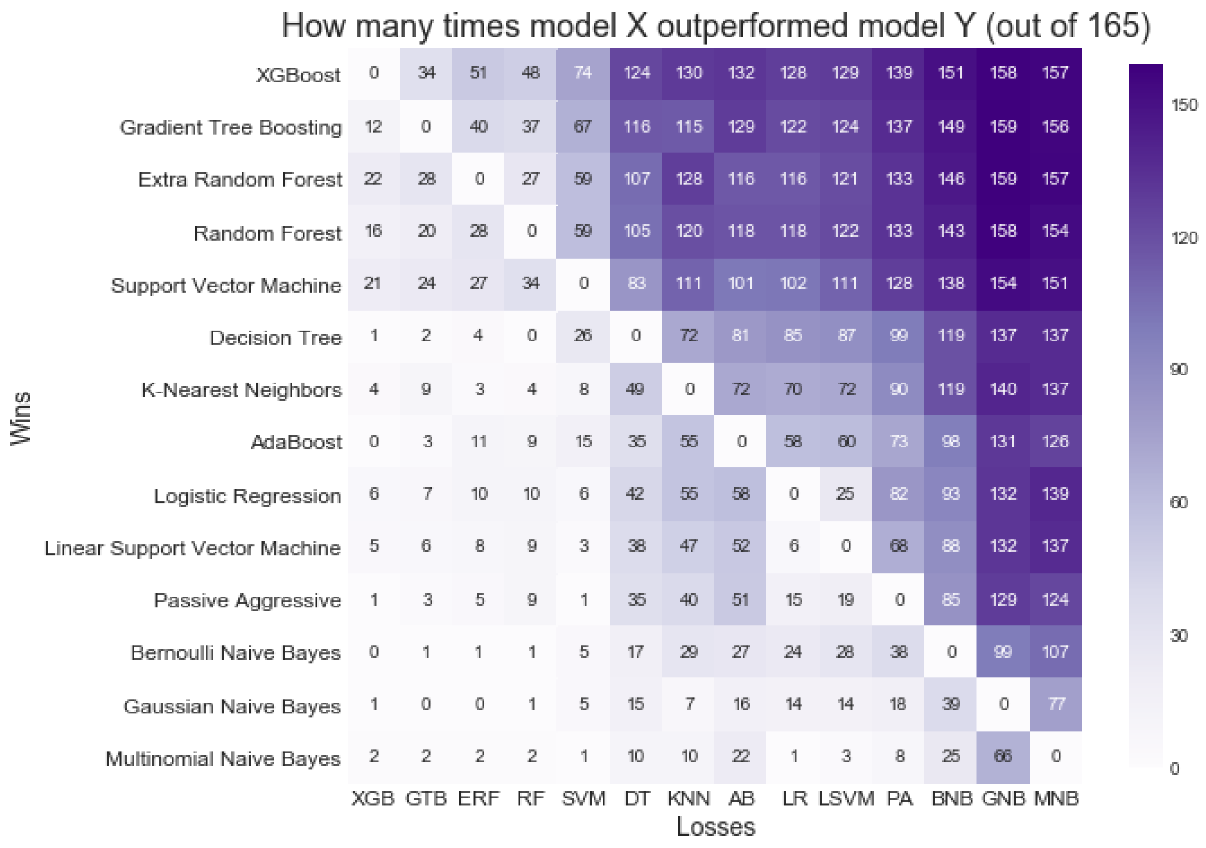
- Repeated k-fold cross validation with out-of-fold predictions, plus a separate routine to create optimized k-fold validation columns
- Ability to fit multiple Y responses in one run
- Automated parameter search via JMP Design of Experiments (DOE) Fast Flexible Filling Design
- Interactive graphical and statistical outputs
- Model comparison interface
- Profiling
- Export of JMP Scripting Language (JSL) and Python code for reproducibility

What is XGBoost? Why Use It?

XGBoost is a scalable, portable, distributed, open-source C++ library for gradient boosted tree prediction written by the dmlc team; see https://github.com/dmlc/xgboost and XGBoost: A Scalable Tree Boosting System. The original theory and applications were developed by Leo Breiman and Jerry Friedman in the late 1990s. XGBoost sprang from a research project at the University of Washington around 2015 and is now sponsored by Amazon, NVIDIA, and others.


Cache awareness and out-of-core computing
Regularization for avoiding overfitting
Tree pruning using depth-first approach
Efficient handling of missing data
XGBoost
Parallelized tree building
In-built cross-validation capability

XGBoost has grown dramatically in popularity due to successes in nearly every major Kaggle competition and others with tabular data over the past five years. It has also been the top performer in several published studies, including the following result from https://github.com/rhiever/sklearn-benchmarks



Wins \ Losses	XGB	GTB	ERF	RF	SVM	DT	KNN	AB	LR	LSVM	PA	BNB	GNB	MNB
XGBoost	0	34	51	48	74	124	130	132	128	129	139	151	158	157
Gradient Tree Boosting	12	0	40	37	67	116	115	129	122	124	137	149	159	156
Extra Random Forest	22	28	0	27	59	107	128	116	116	121	133	146	159	157
Random Forest	16	20	28	0	59	105	120	118	118	122	133	143	158	154
Support Vector Machine	21	24	27	34	0	83	111	101	102	111	128	138	154	151
Decision Tree	1	2	4	0	26	0	72	81	85	87	99	119	137	137
K-Nearest Neighbors	4	9	3	4	8	49	0	72	70	72	90	119	140	137
AdaBoost	0	3	11	9	15	35	55	0	58	60	73	98	131	126
Logistic Regression	6	7	10	10	6	42	55	58	0	25	82	93	132	139
Linear Support Vector Machine	5	6	8	9	3	38	47	52	6	0	68	88	132	137
Passive Aggressive	1	3	5	9	1	35	40	51	15	19	0	85	129	124
Bernoulli Naive Bayes	0	1	1	1	5	17	29	27	24	28	38	0	99	107
Gaussian Naive Bayes	1	0	0	1	5	15	7	16	14	14	18	39	0	77
Multinomial Naive Bayes	2	2	2	2	1	10	10	22	1	3	8	25	66	0

We have done extensive internal testing of XGBoost within JMP R&D and obtained results like those above.

Two alternative open-source libraries with similar methodologies, LightGBM from Microsoft and CatBoost from Yandex, have also enjoyed strong and growing popularity and further validate the effectiveness of the advanced boosted tree methods available in XGBoost.

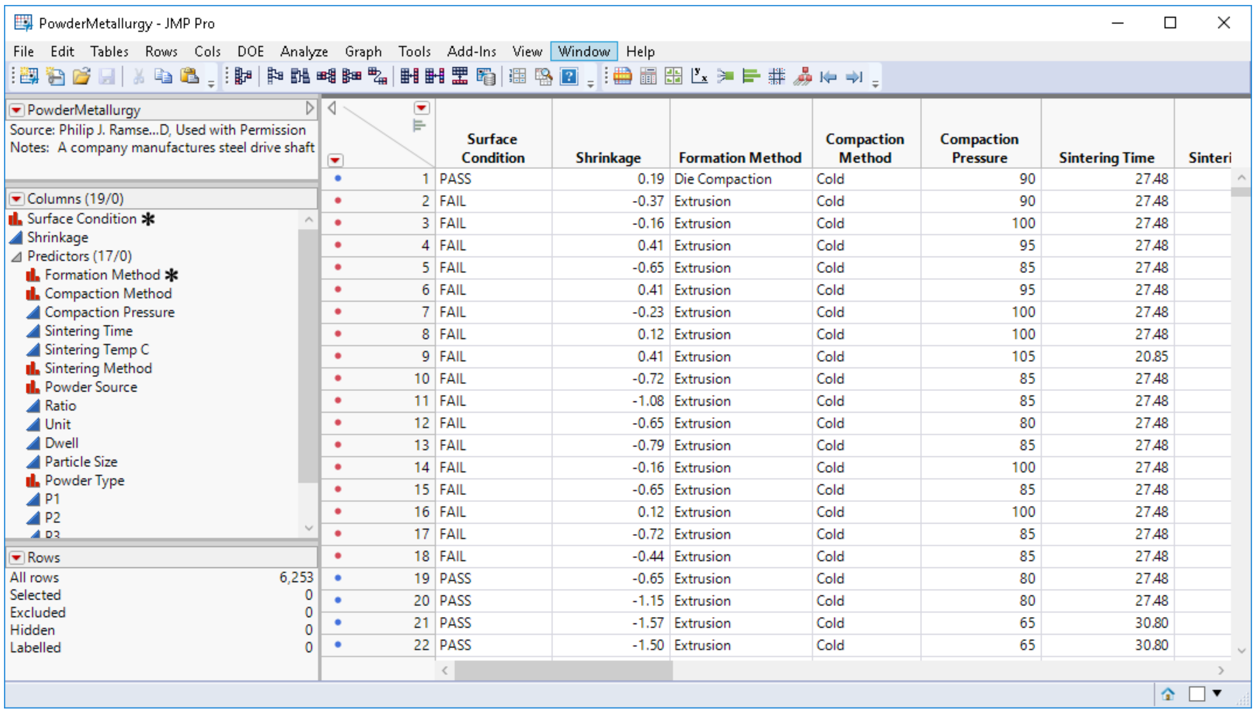
Data scientists typically run XGBoost using a higher-level language like Python or R. This add-in provides a mouse-driven interface with no programming required, enabling you to make predictions rapidly, reliably, and interactively within the dynamic JMP framework.

Installation

You must have a functional copy of JMP 15 Pro or later. After opening it, drag the XGBoost.jmpaddin file onto it and click Yes to install. The installed add-in is then located under the Add-Ins menu.

Powder Metallurgy Example

This example is based on a data set kindly provided by Philip J Ramsey. Click **Add-ins > XGBoost > Example Data > Power Metallurgy** to open a JMP table containing the data:


PowderMetallurgy - JMP Pro
File Edit Tables Rows Cols DOE Analyze Graph Tools Add-Ins View Window Help

PowderMetallurgy
Source: Philip J. Ramse...D, Used with Permission
Notes: A company manufactures steel drive shaft

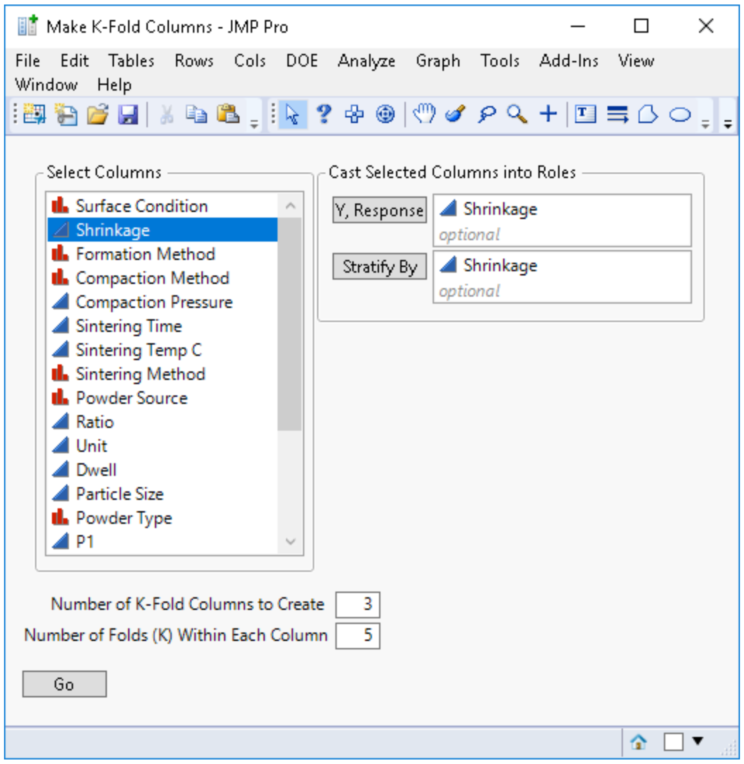
Columns (19/0)
- Surface Condition ✱
- Shrinkage
- Predictors (17/0)
  - Formation Method ✱
  - Compaction Method
  - Compaction Pressure
  - Sintering Time
  - Sintering Temp C
  - Sintering Method
  - Powder Source
  - Ratio
  - Unit
  - Dwell
  - Particle Size
  - Powder Type
  - P1
  - P2
  - P3

Rows	
All rows	6,253
Selected	0
Excluded	0
Hidden	0
Labelled	0

	Surface Condition	Shrinkage	Formation Method	Compaction Method	Compaction Pressure	Sintering Time	Sinteri
1	PASS	0.19	Die Compaction	Cold	90	27.48	
2	FAIL	-0.37	Extrusion	Cold	90	27.48	
3	FAIL	-0.16	Extrusion	Cold	100	27.48	
4	FAIL	0.41	Extrusion	Cold	95	27.48	
5	FAIL	-0.65	Extrusion	Cold	85	27.48	
6	FAIL	0.41	Extrusion	Cold	95	27.48	
7	FAIL	-0.23	Extrusion	Cold	100	27.48	
8	FAIL	0.12	Extrusion	Cold	100	27.48	
9	FAIL	0.41	Extrusion	Cold	105	20.85	
10	FAIL	-0.72	Extrusion	Cold	85	27.48	
11	FAIL	-1.08	Extrusion	Cold	85	27.48	
12	FAIL	-0.65	Extrusion	Cold	80	27.48	
13	FAIL	-0.79	Extrusion	Cold	85	27.48	
14	FAIL	-0.16	Extrusion	Cold	100	27.48	
15	FAIL	-0.65	Extrusion	Cold	85	27.48	
16	FAIL	0.12	Extrusion	Cold	100	27.48	
17	FAIL	-0.72	Extrusion	Cold	85	27.48	
18	FAIL	-0.44	Extrusion	Cold	85	27.48	
19	PASS	-0.65	Extrusion	Cold	80	27.48	
20	PASS	-1.15	Extrusion	Cold	80	27.48	
21	PASS	-1.57	Extrusion	Cold	65	30.80	
22	PASS	-1.50	Extrusion	Cold	65	30.80	

A company manufactures steel drive shafts for the automotive industry using powder metallurgy technology. The company has several manufacturing processes they can use, depending upon customer specifications. Their customers require the drive shafts to be within specification of a target diameter and are typically well within spec limits. Furthermore, the surface of the finished parts must be free of pitting (porosity) and if pitting is observed on any of the manufactured parts then the entire production run is scrapped. Although failure of a production run is infrequent, the cost of scrap can run to as much as $100,000. A team has been assembled to attempt to find the cause of pitting and to minimize the probability of future failures. For the last 6253 production runs, data on key process variables and features of the parts from each production run have been recorded. The data contain both a binary target (Surface Condition) and a continuous response (Shrinkage), which is highly associated with Surface Condition.

Before fitting an XGBoost model to these data, we strongly recommend making one or more columns to use for k-fold cross validation. Click **Add-Ins > XGBoost > Make K-Fold Columns** and specify Shrinkage in both the **Y-Response** and **Stratify By** fields.


Make K-Fold Columns - JMP Pro

File Edit Tables Rows Cols DOE Analyze Graph Tools Add-Ins View
Window Help

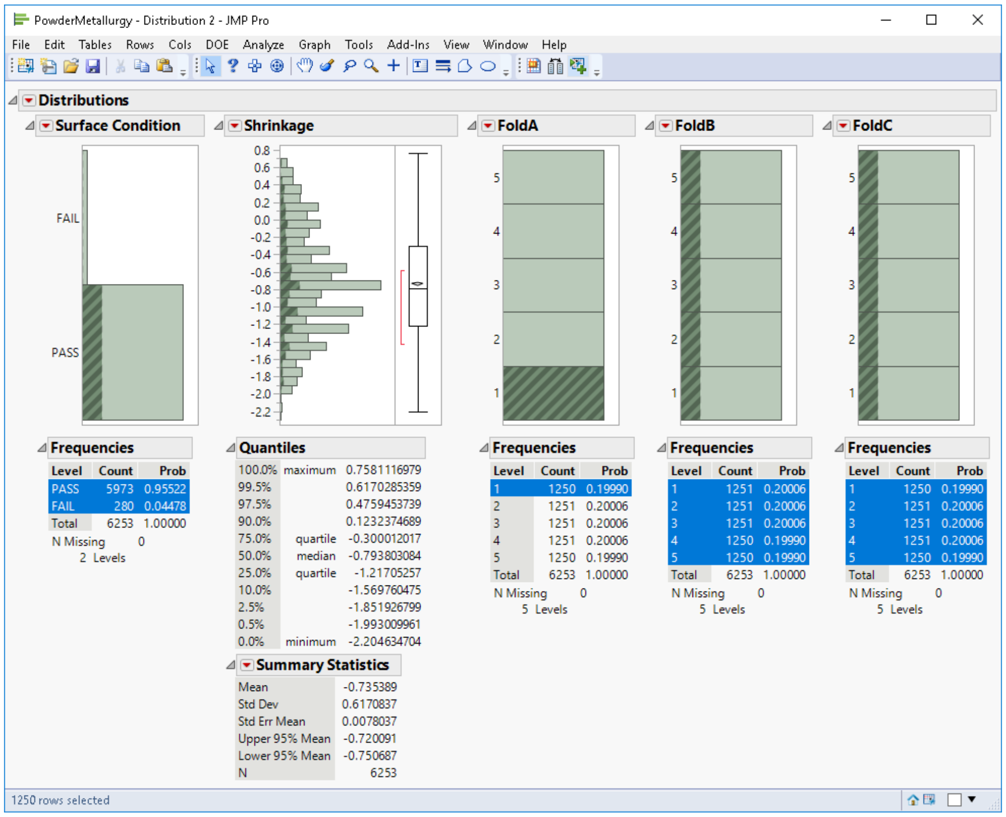
Select Columns
- Surface Condition
- Shrinkage
- Formation Method
- Compaction Method
- Compaction Pressure
- Sintering Time
- Sintering Temp C
- Sintering Method
- Powder Source
- Ratio
- Unit
- Dwell
- Particle Size
- Powder Type
- P1

Cast Selected Columns into Roles
- Y, Response: Shrinkage (optional)
- Stratify By: Shrinkage (optional)

Number of K-Fold Columns to Create: 3
Number of Folds (K) Within Each Column: 5

This routine checks **Y, Response** variables for missingness and creates folds for rows with no missing data. When you specify **Stratify By** variables, it balances representation of these variables across the folds, which can improve out-of-fold model performance. You can change the number of k-fold columns to create as well as K itself with the options at the bottom of the dialog. By default, three columns are created with K=5. Click **Go** to create the columns, which should take just over five seconds.

Make K-Fold Columns uses JMP DOE to create an optimal set of validation columns orthogonal to each other and balanced across ranges of the stratification variables. This enables maximum information gain from a repeated k-fold analysis. Upon completion, three new columns (FoldA, FoldB, and FoldC) are added to the end of the table. You can verify their orthogonality by clicking **Analyze > Distribution** and selecting Surface Condition, Shrinkage, and the three new fold columns.



Surface Condition — Frequencies

Level	Count	Prob
PASS	5973	0.95522
FAIL	280	0.04478
Total	6253	1.00000

N Missing 0
2 Levels

Shrinkage — Quantiles

100.0%	maximum	0.7581116979
99.5%		0.6170285359
97.5%		0.4759453739
90.0%		0.1232374689
75.0%	quartile	-0.300012017
50.0%	median	-0.793803084
25.0%	quartile	-1.21705257
10.0%		-1.569760475
2.5%		-1.851926799
0.5%		-1.993009961
0.0%	minimum	-2.204634704

Summary Statistics

Mean	-0.735389
Std Dev	0.6170837
Std Err Mean	0.0078037
Upper 95% Mean	-0.720091
Lower 95% Mean	-0.750687
N	6253

FoldA — Frequencies

Level	Count	Prob
1	1250	0.19990
2	1251	0.20006
3	1251	0.20006
4	1251	0.20006
5	1250	0.19990
Total	6253	1.00000

N Missing 0
5 Levels

FoldB — Frequencies

Level	Count	Prob
1	1251	0.20006
2	1251	0.20006
3	1251	0.20006
4	1250	0.19990
5	1250	0.19990
Total	6253	1.00000

N Missing 0
5 Levels

FoldC — Frequencies

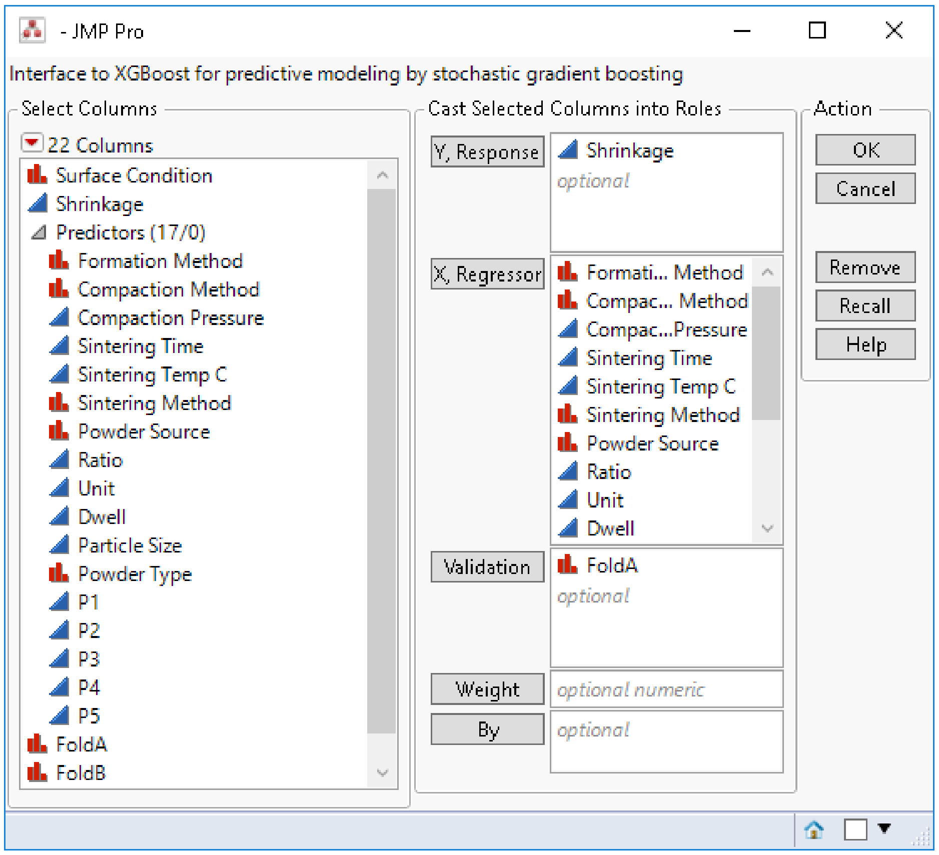
Level	Count	Prob
1	1250	0.19990
2	1251	0.20006
3	1251	0.20006
4	1251	0.20006
5	1250	0.19990
Total	6253	1.00000

N Missing 0
5 Levels

1250 rows selected

Click any of the histogram bars and note the balance in other variables. We see here also that Surface Condition failure rate is around 4.5% and Shrinkage is roughly symmetric.

We are now ready to fit an XGBoost model and will predict the continuous target Shrinkage. Click **Add-Ins > XGBoost > XGBoost**, specify Shrinkage as **Y, Response**, FoldA as **Validation** (we will not use FoldB or FoldC for this analysis), and the columns within the Predictors group as the **X, Regressor** variables:



- JMP Pro

Interface to XGBoost for predictive modeling by stochastic gradient boosting

Select Columns

22 Columns
- Surface Condition
- Shrinkage
- Predictors (17/0)
  - Formation Method
  - Compaction Method
  - Compaction Pressure
  - Sintering Time
  - Sintering Temp C
  - Sintering Method
  - Powder Source
  - Ratio
  - Unit
  - Dwell
  - Particle Size
  - Powder Type
  - P1
  - P2
  - P3
  - P4
  - P5
- FoldA
- FoldB

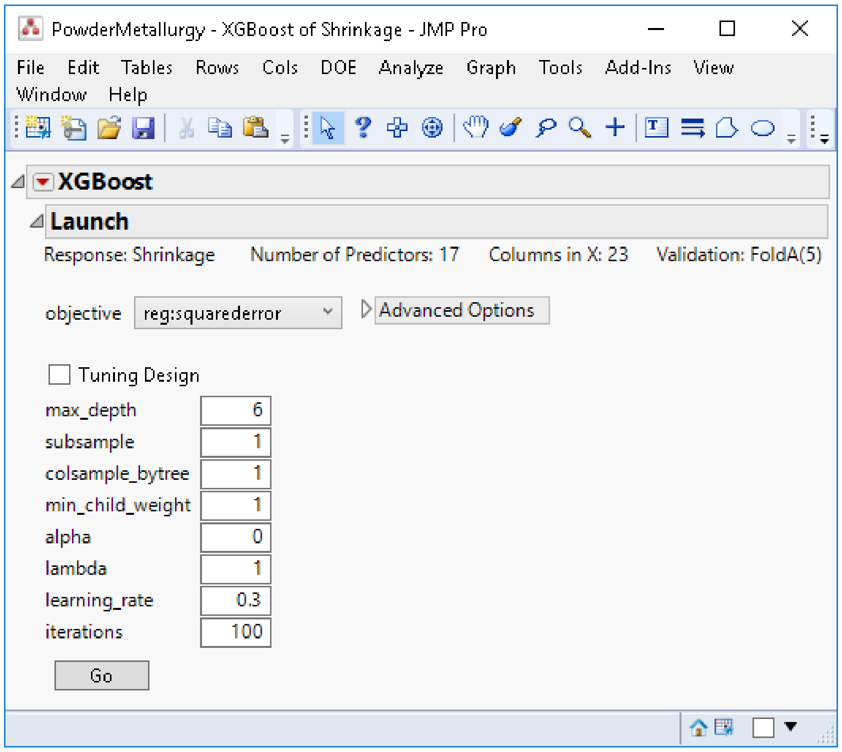
Cast Selected Columns into Roles

Y, Response	Shrinkage; optional
X, Regressor	Formati... Method; Compac... Method; Compac...Pressure; Sintering Time; Sintering Temp C; Sintering Method; Powder Source; Ratio; Unit; Dwell
Validation	FoldA; optional
Weight	optional numeric
By	optional

The **Validation** variable and cross-validation work somewhat differently in the XGBoost add-in as compared to other JMP predictive modeling platforms. In the add-in, specifying a nominal variable like FoldA as **Validation** automatically indicates that k-fold cross-validation is to be performed using each distinct level of the variable as a holdout set. The add-in fits models corresponding to each fold and uses out-of-fold predictions for all validation graphical and statistical output.

If you want to perform a single holdout analysis instead of k-fold, change the type of the **Validation** variable to Continuous and make sure it has values 0 for training and 1 for validation. You can create such variables easily from a Nominal variable by selecting it then clicking **Cols > Utilities > Make Indicator Columns**. You can optionally specify multiple validation columns in order to perform repeated k-fold cross validation. For this example we will do a basic 5-fold analysis via FoldA.

Click **OK** to bring up the main fitting dialog and verify information along the top row is correct.


PowderMetallurgy - XGBoost of Shrinkage - JMP Pro
File Edit Tables Rows Cols DOE Analyze Graph Tools Add-Ins View Window Help
XGBoost
Launch
Go

Response: Shrinkage   Number of Predictors: 17   Columns in X: 23   Validation: FoldA(5)

objective reg:squarederror   Advanced Options

Tuning Design

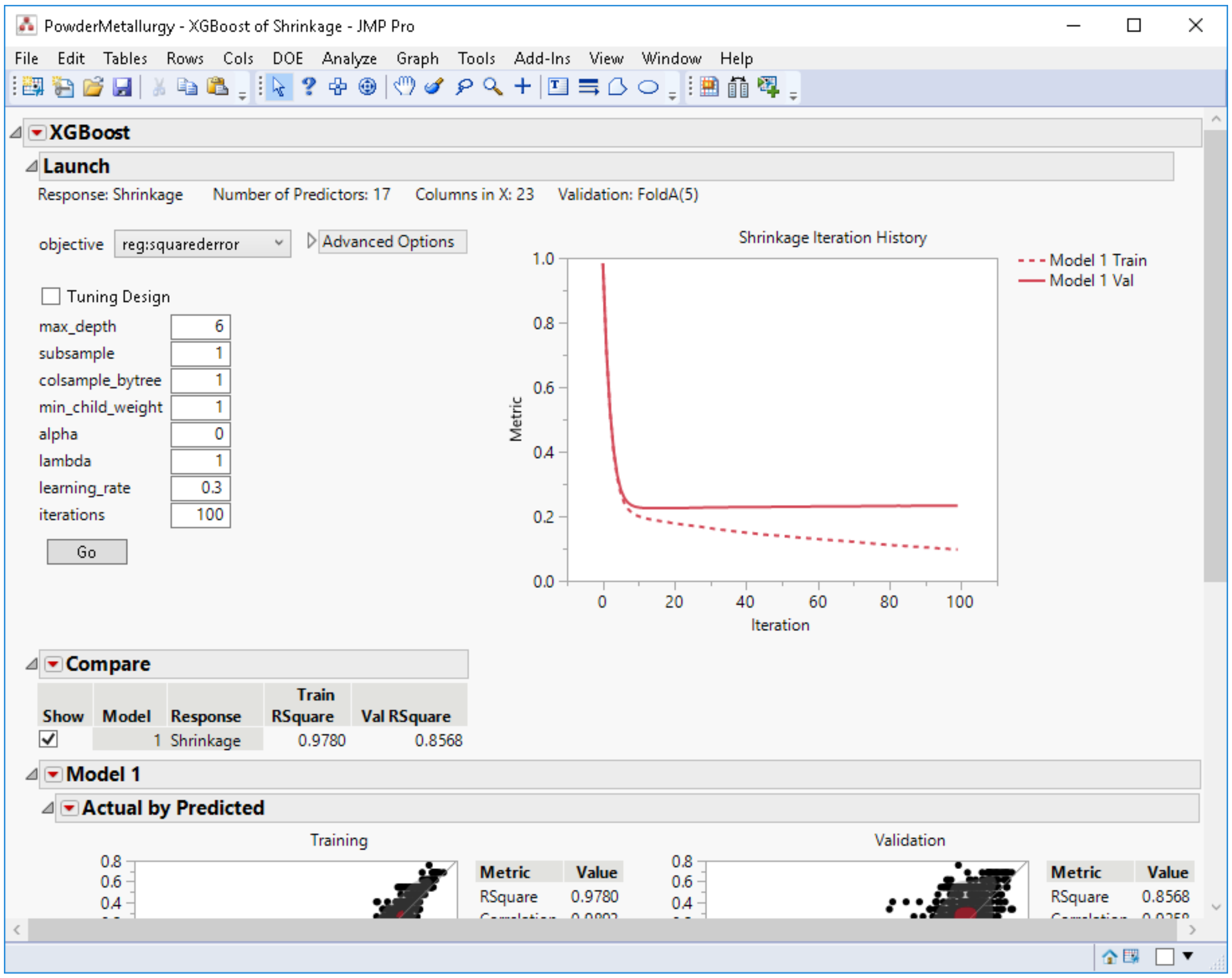
max_depth	6
subsample	1
colsample_bytree	1
min_child_weight	1
alpha	0
lambda	1
learning_rate	0.3
iterations	100

Here we have 17 predictors. However, 0/1 indicator coding (also known as one-hot coding) is used for all nominal predictor variables, so the total number of X columns is 23. Nominal variables with only two levels result in a single column, and those with more than two levels produce indicator columns for each distinct level.

“FoldA(5)” indicates that 5 levels are found for FoldA and 5-fold cross-validation is to be performed.

In the **Launch** section you can specify parameter values for the XGBoost model. The key parameters are listed along the left-hand side, at default values, and many others are available by clicking the disclosure icon next to **Advanced Options**. For our first fit we use the default values, so just click **Go** to fit the model.

You should briefly see a progress window tracking the five model fits corresponding to each fold, and then results should appear as follows:


PowderMetallurgy - XGBoost of Shrinkage - JMP Pro
File Edit Tables Rows Cols DOE Analyze Graph Tools Add-Ins View Window Help
Shrinkage Iteration History
Model 1 Train
Model 1 Val
Metric
Iteration
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
Actual by Predicted
Training
Validation

XGBoost

Launch

Response: Shrinkage   Number of Predictors: 17   Columns in X: 23   Validation: FoldA(5)

objective: reg:squarederror   Advanced Options

☐ Tuning Design

max_depth	6
subsample	1
colsample_bytree	1
min_child_weight	1
alpha	0
lambda	1
learning_rate	0.3
iterations	100

Go

Compare

Show	Model	Response	Train RSquare	Val RSquare
✓	1	Shrinkage	0.9780	0.8568

Model 1

Actual by Predicted

Metric	Value
RSquare	0.9780

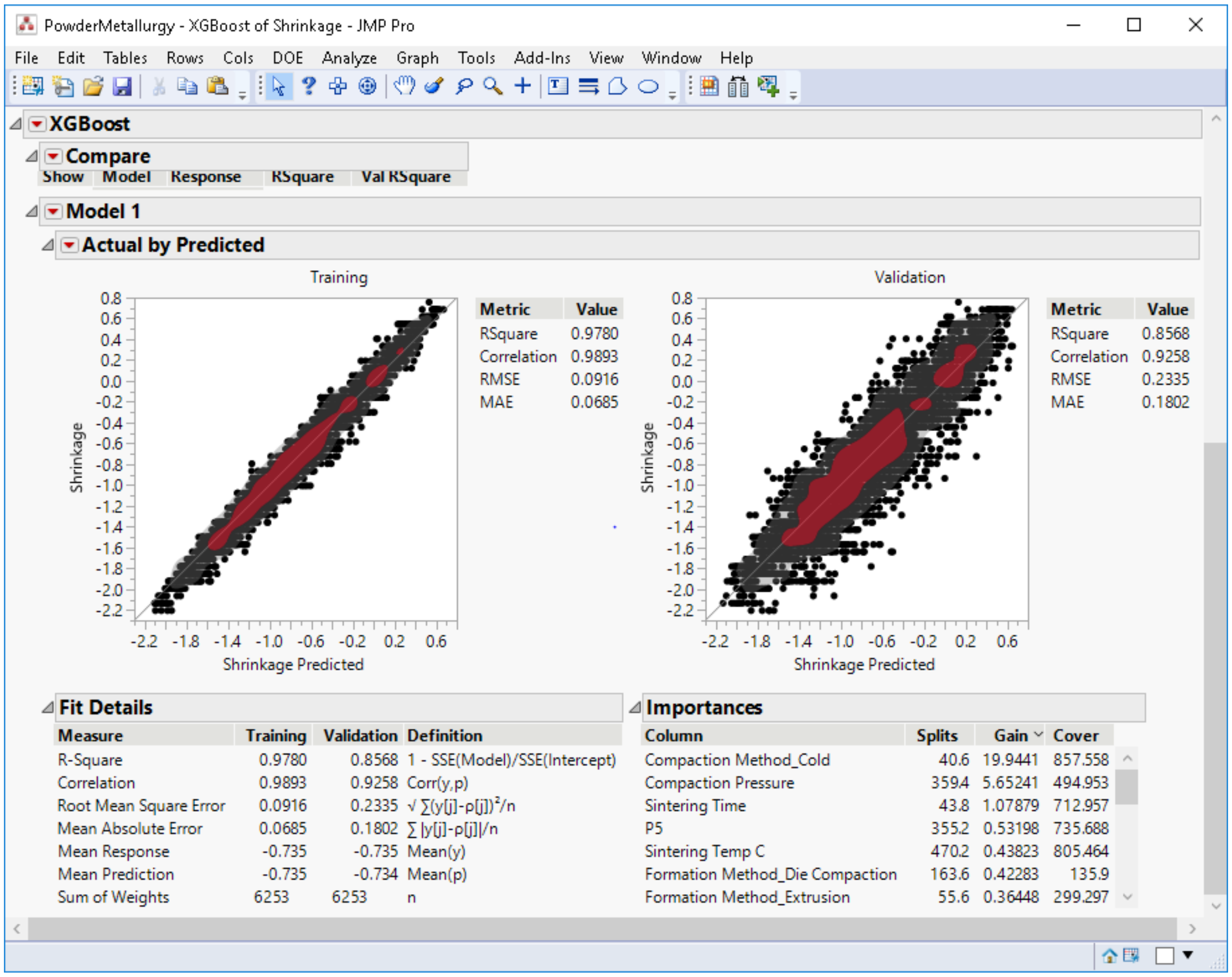
Metric	Value
RSquare	0.8568

The graph in the upper right displays the iteration history of the evaluation metric (here root mean squared error) for both the training (dotted line) and validation (solid line) sets, averaged over the five folds. Note that XGBoost distinguishes between this metric and the actual objective function that is minimized during gradient boosting, here **reg:squarederror**. You can change the objective function with the pulldown menu near the upper left corner, and you can change the evaluation metric within **Advanced Options**. Refer to the XGBoost documentation for further explanation.

The training evaluation metric is lower than validation, as is typical in machine learning, but validation results are more representative of how we expect the model to perform on future data. The **Iteration History** graph can be helpful to assess potential overfitting by studying the gap between the curves. Here the validation metric plateaus at around 10 iterations, so the additional 90 are not really needed at the specified learning rate of 0.3. However, the training metric continues to decline. Since the validation metric does not increase, the prediction results for 100 iterations should still be reasonable and can likely be improved with model tuning.

The **Compare** section tracks model fitting statistics. You can add statistics or remove model fits by selecting options from the **Compare** red triangle menu.

Scrolling to the bottom of the window reveals more detailed graphical and numerical results for the model fit:


PowderMetallurgy - XGBoost of Shrinkage - JMP Pro
File Edit Tables Rows Cols DOE Analyze Graph Tools Add-Ins View Window Help
XGBoost
Compare
Show Model Response RSquare Val RSquare
Model 1
Actual by Predicted
Training
Validation
Shrinkage
Shrinkage Predicted

Training

Metric	Value
RSquare	0.9780
Correlation	0.9893
RMSE	0.0916
MAE	0.0685

Validation

Metric	Value
RSquare	0.8568
Correlation	0.9258
RMSE	0.2335
MAE	0.1802

Fit Details

Measure	Training	Validation	Definition
R-Square	0.9780	0.8568	1 - SSE(Model)/SSE(Intercept)
Correlation	0.9893	0.9258	Corr(y,p)
Root Mean Square Error	0.0916	0.2335	$\sqrt{\sum(y[j]-p[j])^2/n}$
Mean Absolute Error	0.0685	0.1802	$\sum \lvert y[j]-p[j]\rvert/n$
Mean Response	-0.735	-0.735	Mean(y)
Mean Prediction	-0.735	-0.734	Mean(p)
Sum of Weights	6253	6253	n

Importances

Column	Splits	Gain	Cover
Compaction Method_Cold	40.6	19.9441	857.558
Compaction Pressure	359.4	5.65241	494.953
Sintering Time	43.8	1.07879	712.957
P5	355.2	0.53198	735.688
Sintering Temp C	470.2	0.43823	805.464
Formation Method_Die Compaction	163.6	0.42283	135.9
Formation Method_Extrusion	55.6	0.36448	299.297

Your results might differ a little from those shown here due to random number seeding in the creation of FoldA and in the XGBoost algorithm. The two graphs plot actual by predicted values for the training and validation sets, along with density contours and dynamic linkage to the main table. The training predictions are more strongly correlated with the true values than are the validation predictions, but the validation predictions provide a more realistic assessment of model performance. The difference between the two is a key motivation for using k-fold cross-validation.

The validation predictions are out-of-fold, that is, each predicted value comes from the fold for which that observation was held out. K-fold is desirable since each observation is held-out exactly once, as compared to classic single-holdout methods in which only a fraction of the observations is held out once, or bootstrapping routines for which the holdout percentages are unbalanced. In the **Fit Details** table, note there are 6253 observations (Sum of Weights) for both training and validation, and the validation errors are much larger than those for training.

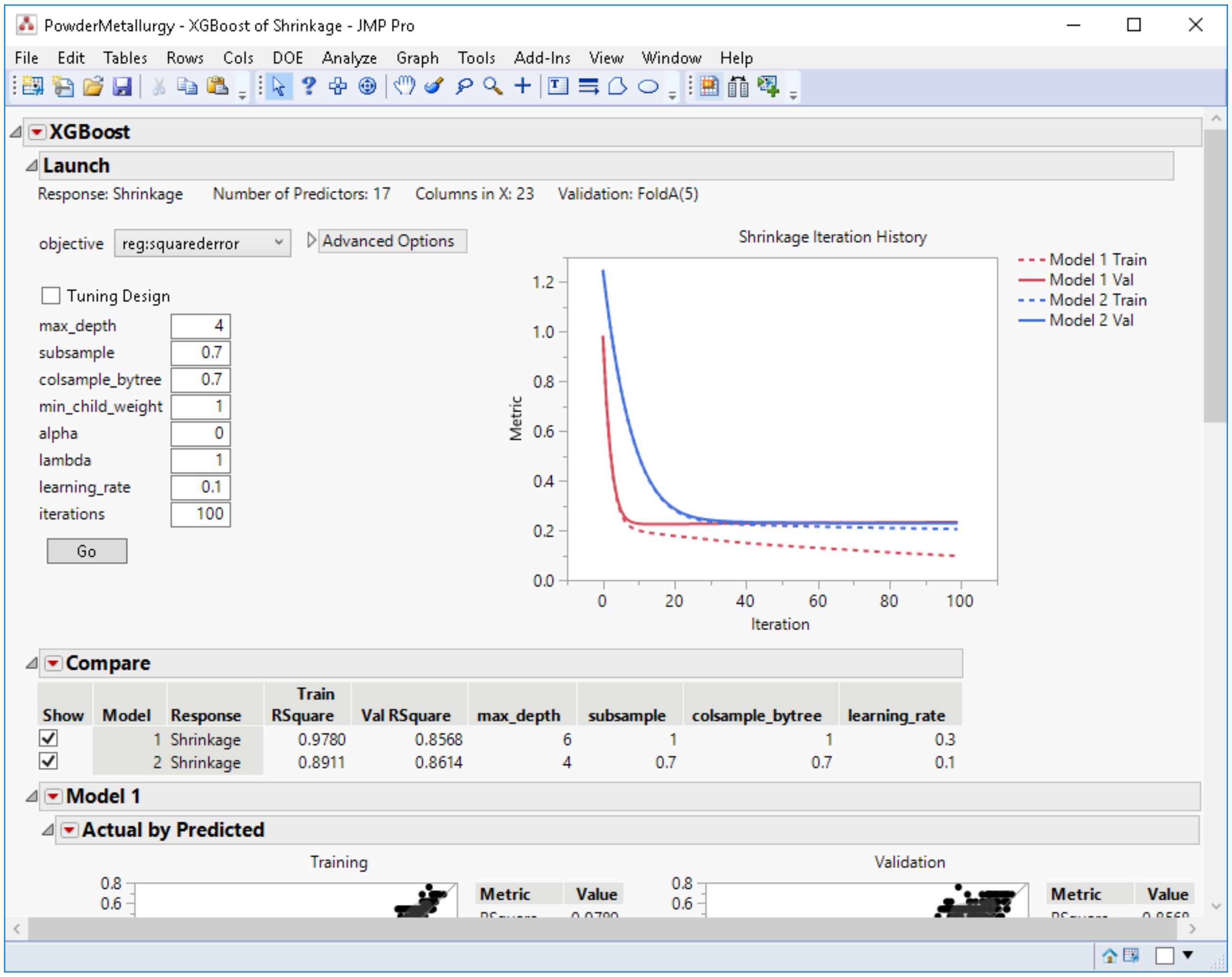
For continuous responses like Shrinkage, common performance measures include R-Square, Correlation, Root Mean Square Error (RMSE), and Mean Absolute Error (MAE). Which one you focus on depends on your objectives and how the predictions are to be utilized.

The **Importances** table contains three importance statistics for each of the predictors, each averaged over the five model fits. These are:

- **Splits**, the number of times that variable is used in a split
- **Gain**, the average improvement in objective function for splits involving that variable
- **Cover**, amount of data covered by splits involving that variable

You can sort the table by clicking each variable name and click again to switch from ascending to descending. Here Compaction Method with level Cold and Compaction Pressure are the most important predictors in terms of gain. To explore importances further, in the table **right click > Make into Data Table** to create a JMP table of these values and use other JMP platforms to analyze them. For example, take log transforms (**select columns > right click > New Formula Column > Transform > Log**) and then **Analyze > Multivariate** to plot logged values against each other.

In attempt to improve the model and decrease overfitting, let's try a less aggressive model. Back in the **Launch** section, change **max_depth** to 4, **subsample** to 0.7, **colsample_by_tree** to 0.7, and **learning_rate** to 0.1. Click **Go** to fit this second model:


PowderMetallurgy - XGBoost of Shrinkage - JMP Pro
File Edit Tables Rows Cols DOE Analyze Graph Tools Add-Ins View Window Help
XGBoost
Launch
Advanced Options
Go
Shrinkage Iteration History
Metric
Iteration
Model 1 Train
Model 1 Val
Model 2 Train
Model 2 Val
Compare
Model 1
Actual by Predicted
Training
Validation

Response: Shrinkage Number of Predictors: 17 Columns in X: 23 Validation: FoldA(5)

objective: reg:squarederror

☐ Tuning Design

max_depth	4
subsample	0.7
colsample_bytree	0.7
min_child_weight	1
alpha	0
lambda	1
learning_rate	0.1
iterations	100

Show	Model	Response	Train RSquare	Val RSquare	max_depth	subsample	colsample_bytree	learning_rate
✓	1	Shrinkage	0.9780	0.8568	6	1	1	0.3
✓	2	Shrinkage	0.8911	0.8614	4	0.7	0.7	0.1

A second row has been added to the **Compare** table corresponding to the new model fit, displaying the default metrics along with model parameters that have changed from the first model fit. Note the training model fit statistics for this new model are more like validation, and that the validation R-Square has improved a little.

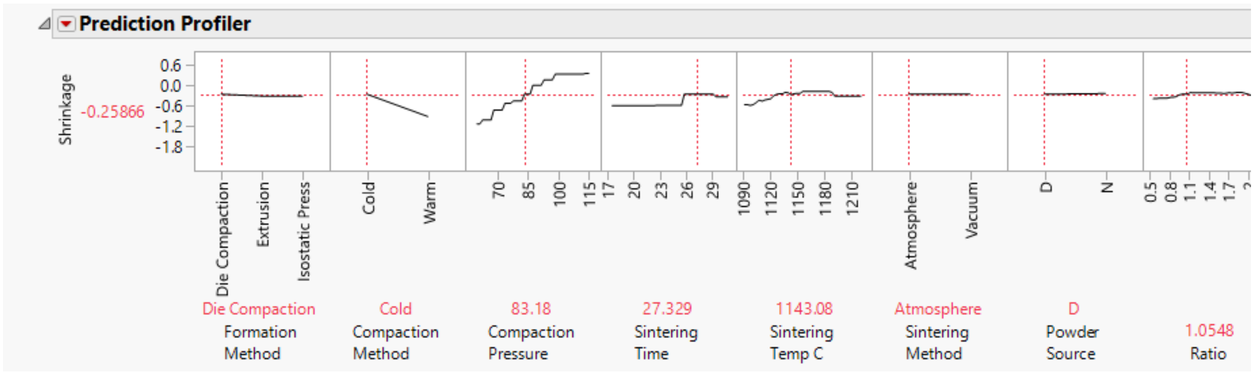
One of the strengths of XGBoost is its large number of parameters but tuning them appropriately can be a challenge at first. A key initial step is to become familiar with what each parameter means. For this refer to the XGBoost parameter documentation. The names of the parameters in the add-in match those in this document.

As you tune your model, the parameters listed on the left are typically the most critical, so begin with these, then progress to those available under **Advanced Options**. There are also many blogs and tutorials about XGBoost model tuning online readily found by searching.

To assist you further with parameter tuning, the Tuning Design checkbox in the **Launch** section enables you to automatically fit a collection of models with parameters chosen using JMP DOE's Fast Flexible Filling Design. The algorithm behind this method covers the parameter space in certain optimal ways. After checking the **Tuning Design** box, you must specify lower and upper bounds for each parameter you want to vary. Set the lower bound equal to the upper bound for all parameters you wish to remain constant. Choose the number of runs you wish to make, then click **Go** to fit all the models. The full analysis can take some time depending on the size of your data and number of runs you choose. Upon completion, sort results in the **Compare** table by various metrics to compare models and scroll down to see specific results for models of interest.

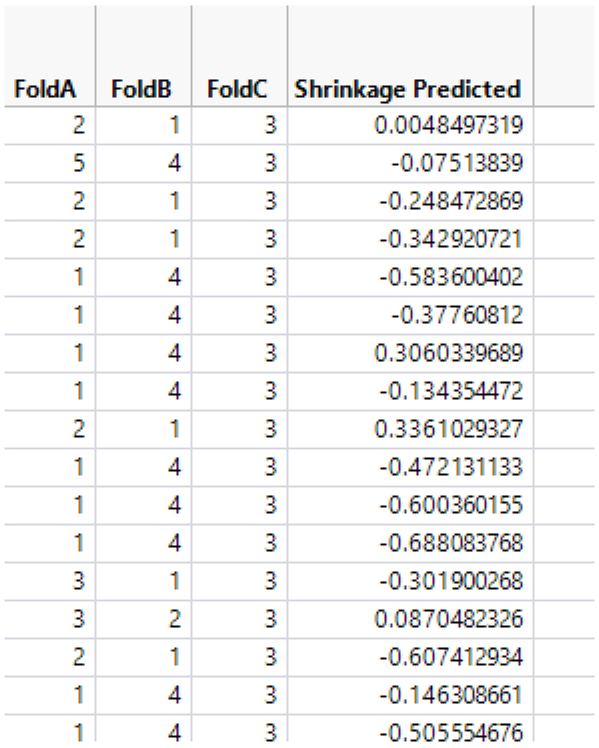
For each model fit, several more actions are available under the red triangle menu for the model. These include profiling, saving predicted values, and generating full Python code.

For example, clicking the **Model 2 red triangle > Profiler** from the analysis above produces the following profiles:



We see the two most important variables, Compaction Method and Compaction Pressure, have the steepest slopes, and the shapes of their profiles show their relationship to Shrinkage. The stair-step shape of the Compaction Pressure profile results from the various tree splits created during gradient boosting.

Clicking **Model 2 red triangle > Save Predicteds** should add a column to the JMP table as follows:



FoldA	FoldB	FoldC	Shrinkage Predicted
2	1	3	0.0048497319
5	4	3	-0.07513839
2	1	3	-0.248472869
2	1	3	-0.342920721
1	4	3	-0.583600402
1	4	3	-0.37760812
1	4	3	0.3060339689
1	4	3	-0.134354472
2	1	3	0.3361029327
1	4	3	-0.472131133
1	4	3	-0.600360155
1	4	3	-0.688083768
3	1	3	-0.301900268
3	2	3	0.0870482326
2	1	3	-0.607412934
1	4	3	-0.146308661
1	4	3	-0.505554676

These are out-of-fold predictions and are the same as those plotted in the Validation Actual versus Predicted plot above. If any rows had had a missing value for the target, the prediction is the average over all model fits, a technique known as bagging.

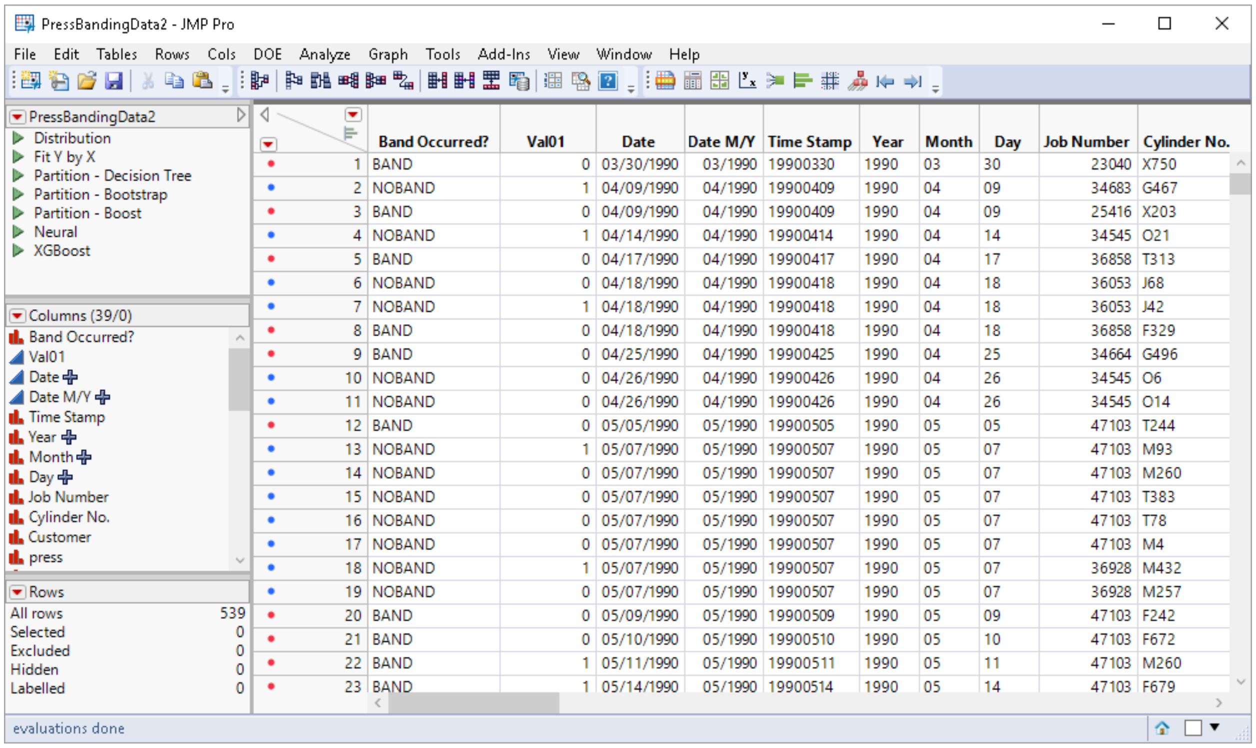
One suggested continuation of this example is to further tune the model, and then to begin again with a reduced predictor set and/or newly engineered features with a primary goal of improving validation performance. You can subsequently average out-of-fold predictions from two or more diverse models to create an ensemble model, which will tend to perform better than any single model.

You can alternatively use tuned parameters in a new model fit on all of the data, with no validation.

Another way to proceed is to use the binary target Surface Condition, being mindful of its rarity. The next example has a binary target, which triggers several changes in the output.

Press Banding Example

This data set is from the UC Irvine Machine Learning Repository generously donated by Bob Evans. It is also described in Grayson, Gardner, and Stephens, *Building Better Models with JMP Pro*, SAS Press, page 166 and a version is available as Bands Data.jmp in the JMP Sample Library. Click **Add-ins > XGBoost > Example Data > Press Banding** to open a JMP table containing the data:



All rows	539
Selected	0
Excluded	0
Hidden	0
Labelled	0

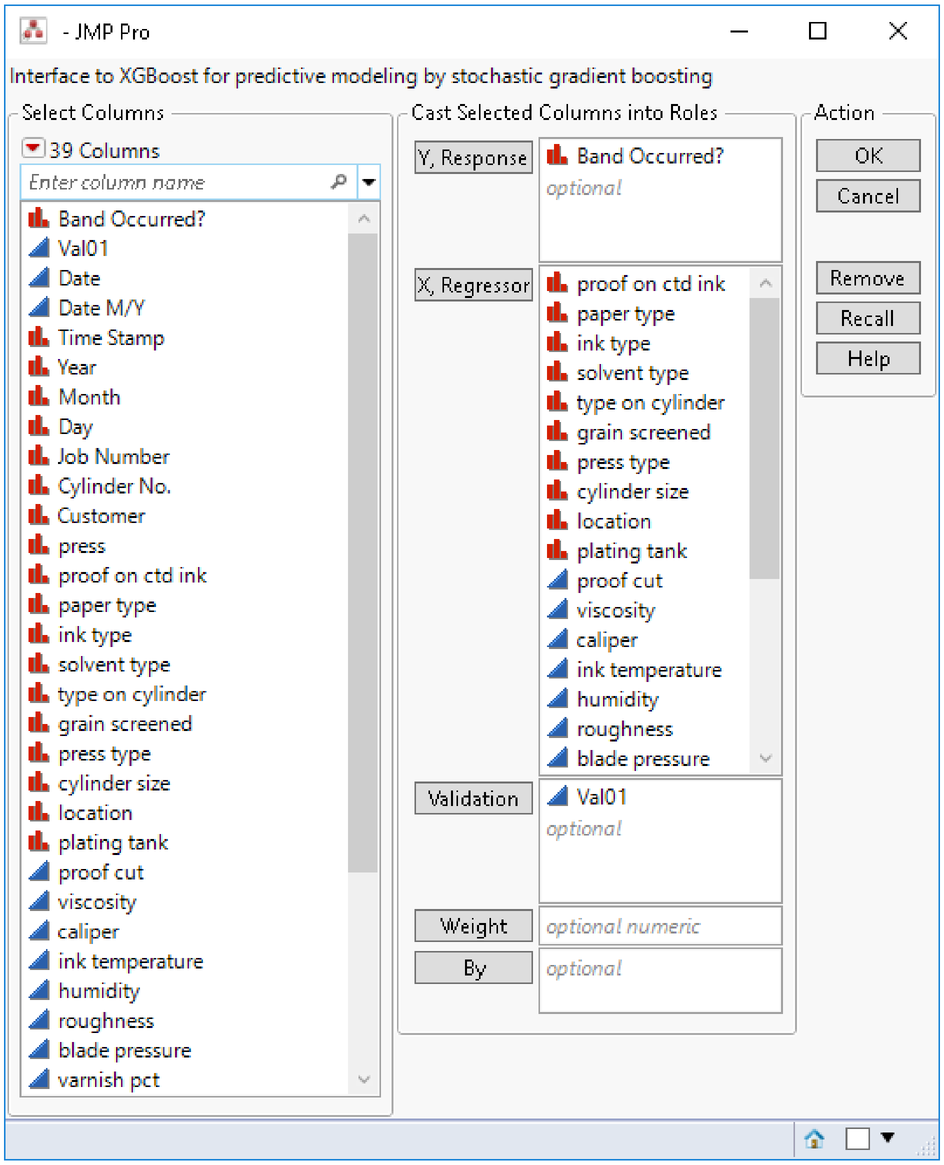
		Band Occurred?	Val01	Date	Date M/Y	Time Stamp	Year	Month	Day	Job Number	Cylinder No.
●	1	BAND	0	03/30/1990	03/1990	19900330	1990	03	30	23040	X750
●	2	NOBAND	1	04/09/1990	04/1990	19900409	1990	04	09	34683	G467
●	3	BAND	0	04/09/1990	04/1990	19900409	1990	04	09	25416	X203
●	4	NOBAND	1	04/14/1990	04/1990	19900414	1990	04	14	34545	O21
●	5	BAND	0	04/17/1990	04/1990	19900417	1990	04	17	36858	T313
●	6	NOBAND	0	04/18/1990	04/1990	19900418	1990	04	18	36053	J68
●	7	NOBAND	1	04/18/1990	04/1990	19900418	1990	04	18	36053	J42
●	8	BAND	0	04/18/1990	04/1990	19900418	1990	04	18	36858	F329
●	9	BAND	0	04/25/1990	04/1990	19900425	1990	04	25	34664	G496
●	10	NOBAND	0	04/26/1990	04/1990	19900426	1990	04	26	34545	O6
●	11	NOBAND	0	04/26/1990	04/1990	19900426	1990	04	26	34545	O14
●	12	BAND	0	05/05/1990	05/1990	19900505	1990	05	05	47103	T244
●	13	NOBAND	1	05/07/1990	05/1990	19900507	1990	05	07	47103	M93
●	14	NOBAND	0	05/07/1990	05/1990	19900507	1990	05	07	47103	M260
●	15	NOBAND	0	05/07/1990	05/1990	19900507	1990	05	07	47103	T383
●	16	NOBAND	0	05/07/1990	05/1990	19900507	1990	05	07	47103	T78
●	17	NOBAND	0	05/07/1990	05/1990	19900507	1990	05	07	47103	M4
●	18	NOBAND	1	05/07/1990	05/1990	19900507	1990	05	07	36928	M432
●	19	NOBAND	0	05/07/1990	05/1990	19900507	1990	05	07	36928	M257
●	20	BAND	0	05/09/1990	05/1990	19900509	1990	05	09	47103	F242
●	21	BAND	0	05/10/1990	05/1990	19900510	1990	05	10	47103	F672
●	22	BAND	1	05/11/1990	05/1990	19900511	1990	05	11	47103	M260
●	23	BAND	1	05/14/1990	05/1990	19900514	1990	05	14	47103	F679

A company uses a rotogravure printing process to print high volume commercial magazines. The process involves engraving an image onto a cylinder, running the cylinder through an ink bath, removing excess ink, then transferring the image to paper. The images are removed from the cylinders after each job and the cylinders are reused. With highly competitive pressures from online publishing, the company must maximize efficiencies in this process and avoid a defect called banding, which consists of grooves that appear in the cylinder during a print run. Banding can result in ruined product, long periods of downtime, and unhappy customers, all at substantial cost. Data are compiled on 539 production runs, with the goal of identifying conditions that lead to banding and predicting when it will occur. The binary target is named Banding Occurred? in the JMP table. Several supplemental columns are available and those towards the end of the table are predictors.

Most of the predictors have some missing values. XGBoost automatically accommodates them by using missingness as a distinct category when making decisions on where to make splits. This contrasts with informative missing approaches that create two columns for each original one, one with missing values imputed (typically with the mean) and the other a binary column indicating missingness.

In order to facilitate quick comparisons with results from other JMP platforms, we use a single holdout validation set as indicated by the Val01 variable near the beginning of the table. Note this variable must be Continuous for the XGBoost add-in to create a single holdout set. (If it were Nominal, XGBoost would perform a 2-fold cross validation analysis.)

Click **Add-Ins > XGBoost > XGBoost**, specify Band Occurred? as **Y, Response**, Val01 as **Validation**, and columns including and following proof on ctd ink as **X, Regressors**:



Interface to XGBoost for predictive modeling by stochastic gradient boosting

Select Columns

39 Columns

Enter column name

Band Occurred?
Val01
Date
Date M/Y
Time Stamp
Year
Month
Day
Job Number
Cylinder No.
Customer
press
proof on ctd ink
paper type
ink type
solvent type
type on cylinder
grain screened
press type
cylinder size
location
plating tank
proof cut
viscosity
caliper
ink temperature
humidity
roughness
blade pressure
varnish pct

Cast Selected Columns into Roles

Y, Response: Band Occurred?
optional

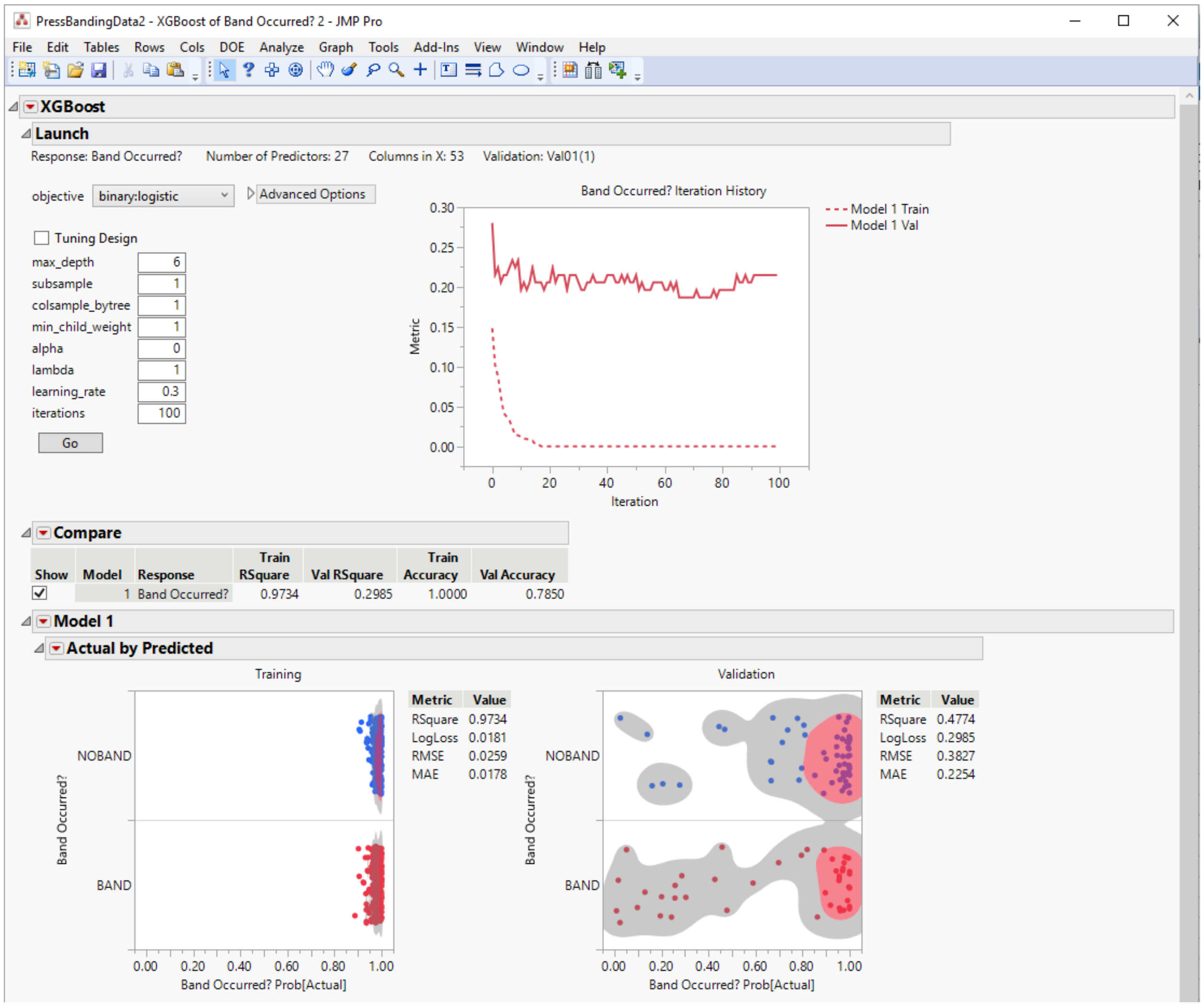
X, Regressor:
proof on ctd ink
paper type
ink type
solvent type
type on cylinder
grain screened
press type
cylinder size
location
plating tank
proof cut
viscosity
caliper
ink temperature
humidity
roughness
blade pressure

Validation: Val01
optional

Weight: optional numeric

By: optional

Click **OK** and then **Go** in the subsequent window to fit the model using default parameters. The top half of the results are as follows:


PressBandingData2 - XGBoost of Band Occurred? 2 - JMP Pro

File Edit Tables Rows Cols DOE Analyze Graph Tools Add-Ins View Window Help

XGBoost

Launch

Response: Band Occurred? Number of Predictors: 27 Columns in X: 53 Validation: Val01(1)

objective: binary:logistic

Tuning Design: ☐

max_depth	6
subsample	1
colsample_bytree	1
min_child_weight	1
alpha	0
lambda	1
learning_rate	0.3
iterations	100

Compare

Show	Model	Response	Train RSquare	Val RSquare	Train Accuracy	Val Accuracy
✓	1	Band Occurred?	0.9734	0.2985	1.0000	0.7850

Model 1

Actual by Predicted

Training

Metric	Value
RSquare	0.9734
LogLoss	0.0181
RMSE	0.0259
MAE	0.0178

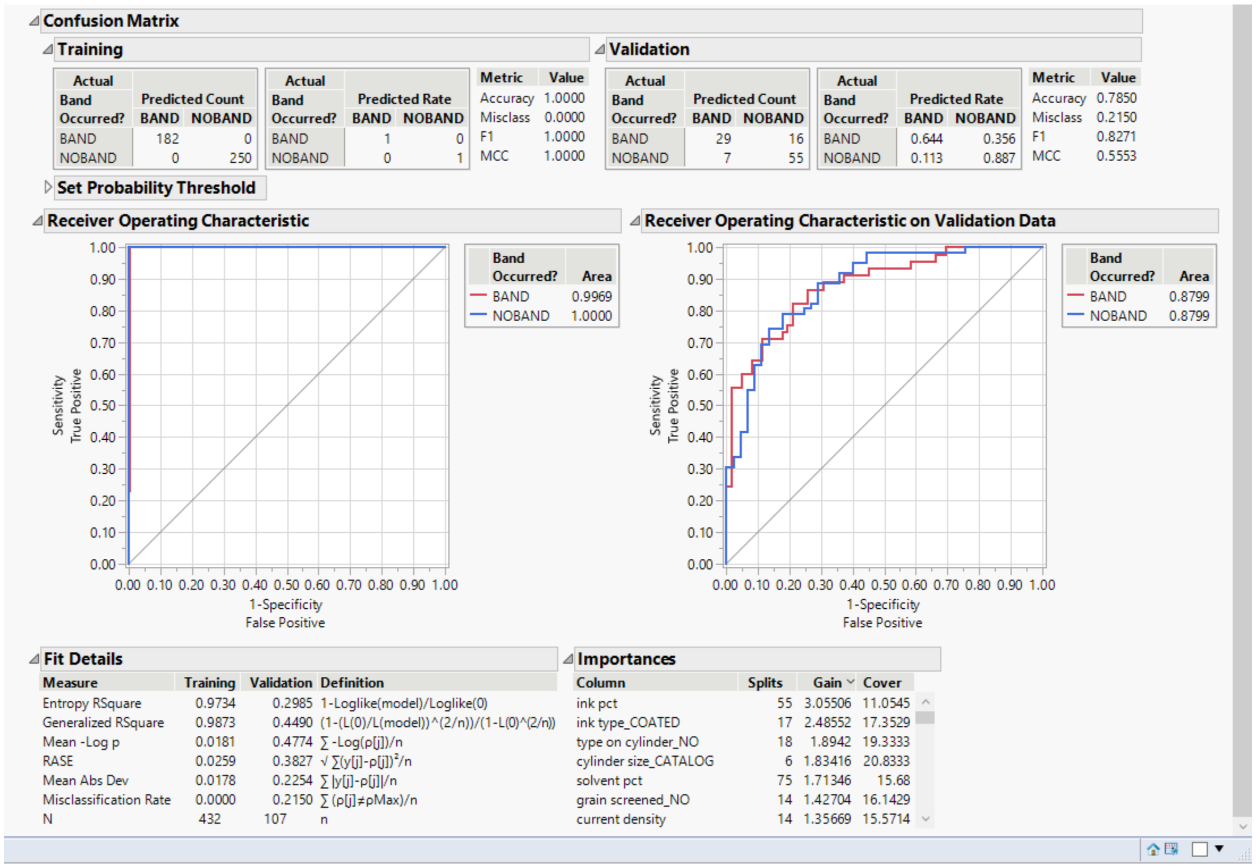
Validation

Metric	Value
RSquare	0.4774
LogLoss	0.2985
RMSE	0.3827
MAE	0.2254

By default, with a binary response, logistic log loss is used both as the objective function and the evaluation metric. A large gap is evident between the training and validation iteration histories, signifying a risk of overfitting. This is also reflected in the plots of actual by predicted values as well as the performance metric statistics. In the **Compare** table, the **Accuracy** statistic (which equals 1 minus misclassification rate) shows that 100% of the training values are predicted correctly, but only 78.5% of the validation ones. Such a large discrepancy is often indication that the model is overfitting, and a natural next step would be to try models with less complexity, for example, reducing **max_depth** from 6 to 3. For sake of illustration here we proceed with the current results.

In the **Actual by Predicted** plots, **Prob(Actual)** on the horizontal axis is the predicted probability of the true known value, and the plots are divided vertically by the two classes. The points are colored blue and red according to the original row state colors in the main table. Points to the far right represent correct predictions whereas those to the left represent prediction errors. In the **Validation** graph, you can select points on the left and return to the main table to explore rows the model is predicting incorrectly out-of-fold. Such an error analysis can reveal outliers or suggest new features that should be added to the model.

The bottom half of the XGBoost output is as follows:



Confusion Matrix

Training

Actual Band Occurred?	Predicted Count BAND	Predicted Count NOBAND
BAND	182	0
NOBAND	0	250

Actual Band Occurred?	Predicted Rate BAND	Predicted Rate NOBAND
BAND	1	0
NOBAND	0	1

Metric	Value
Accuracy	1.0000
Misclass	0.0000
F1	1.0000
MCC	1.0000

Validation

Actual Band Occurred?	Predicted Count BAND	Predicted Count NOBAND
BAND	29	16
NOBAND	7	55

Actual Band Occurred?	Predicted Rate BAND	Predicted Rate NOBAND
BAND	0.644	0.356
NOBAND	0.113	0.887

Metric	Value
Accuracy	0.7850
Misclass	0.2150
F1	0.8271
MCC	0.5553

Set Probability Threshold

Receiver Operating Characteristic

Band Occurred?	Area
BAND	0.9969
NOBAND	1.0000

Receiver Operating Characteristic on Validation Data

Band Occurred?	Area
BAND	0.8799
NOBAND	0.8799

Fit Details

Measure	Training	Validation	Definition
Entropy RSquare	0.9734	0.2985	1-Loglike(model)/Loglike(0)
Generalized RSquare	0.9873	0.4490	(1-(L(0)/L(model))^(2/n))/(1-L(0)^(2/n))
Mean -Log p	0.0181	0.4774	∑ -Log(ρ[j])/n
RASE	0.0259	0.3827	√ ∑(y[j]-ρ[j])²/n
Mean Abs Dev	0.0178	0.2254	∑ |y[j]-ρ[j]|/n
Misclassification Rate	0.0000	0.2150	∑ (ρ[j]≠ρMax)/n
N	432	107	n

Importances

Column	Splits	Gain	Cover
ink pct	55	3.05506	11.0545
ink type_COATED	17	2.48552	17.3529
type on cylinder_NO	18	1.8942	19.3333
cylinder size_CATALOG	6	1.83416	20.8333
solvent pct	75	1.71346	15.68
grain screened_NO	14	1.42704	16.1429
current density	14	1.35669	15.5714

For a binary response, it is common to construct both confusion matrices and receiver operating characteristic (ROC) curves, along with associated statistics. Here again we see large differences between training and validation results, and the perfect predictions for the training data are far too optimistic. You can update statistics like Accuracy that depend upon a probability cutoff within the **Set Probability Threshold** section.

The **Importances** table shows that ink pct has the highest split and gain scores, and ink type_COATED is second in terms of gain. This latter variable is actually a 0/1 indicator variable created from the original nominal variable ink type.

One possible way of continuing this example is compare results from other JMP Pro platforms. In the main table, some scripts are saved to help you get started. Near the upper left corner of the table click the corresponding green triangles to run various other models and compare outputs.

Known Problems with the Current Version of the Add-In

The initial release of the add-in with JMP Pro 15.0 has the following known defects discovered since release:

1. For binary targets, the ROC curves for training data can sometimes be wrong. When this happens, one or both curves are incorrectly near the 45-degree line.
2. After fitting many models (for example with a Tuning Design), then clicking the **Model red triangle > Remove All but This Fit**, JMP can sometimes crash completely due to a memory freeing problem. Make sure to save your data table regularly if you are doing a lot of fitting.
3. All nominal predictor variables are automatically 0/1 indicator (one-hot) encoded. Such variables with a large number of levels (e.g. more than 1000) can slow computations significantly. We are currently considering best ways to handle such high-cardinality variables, including ways to collapse rare categories, and welcome suggestions.
4. JSL scripting of the XGBoost Fit object is not fully functional.
5. With very large data sets, **Model red triangle > Save Predicteds** can run out of memory.

If you encounter problems differing from those above, please contact us as described at the end of this document.

Technical Details

Customized code within JMP Pro directly interfaces to the C++ API available in the XGBoost library. Fixes and enhancements are forthcoming in future releases of JMP Pro and the add-in itself. The add-in comes with a precompiled dll for Windows and dylib files for Mac.

Model fits are saved to disk in standard XGBoost format and reused to create predictions. See your JMP log (View > Log) for exact locations. Prediction formulas are naturally complex, especially with k-fold cross validation. When you click **Model red triangle > Save Prediction Formula**, several intermediate columns are added to the table to facilitate final assemblage of the out-of-fold predictions.

FAQ

1. *Why develop an XGBoost add-in?*
   XGBoost is a popular open-source machine learning framework. By employing this add-in, JMP Pro users can add XGBoost models to the suite of built-in models available in JMP Pro.

2. *Which version of JMP do I need?*
   JMP Pro 15 (or higher).

3. *How does XGBoost differ from the existing Boosted Tree platform in JMP Pro?*
   While the underlying algorithms are based on a similar approach, XGBoost has a more comprehensive set of parameters and objective functions. In addition, the two handle nominal variables differently.

4. *Will the add-in utilize my GPU?*
   On Windows, the dll included with the add-in has been compiled to take advantage of appropriate NVIDIA GPU cards via its CUDA library. To try it, select **Advanced Options > tree_method > gpu_hist**. On the Mac, no GPU capabilities are currently available since Macs typically do not have NVIDIA cards.

5. *Can I use my own compiled xgboost.dll or libxgboost.dylib?*
   While we have not tested this thoroughly, it should be possible to swap in your own properly compiled executable within the lib directory in the add-in install location. Click **View > Add-Ins** then click XGBoost to see a link to this location on your machine.

Contact

If you are able, please complete the online survey. It should take around five minutes and your feedback is warmly appreciated. Please also participate in the JMP Community XGBoost group and keep an eye on it for recent news and posts from others.

This is a great time to influence directions of this add-in, and we thank you for taking the time to help improve it. You may also contact russ.wolfinger@jmp.com directly with advanced questions or ideas. If you encounter a bug or crash, please, if possible, send Russ an accompanying data set and steps for reproducing it.