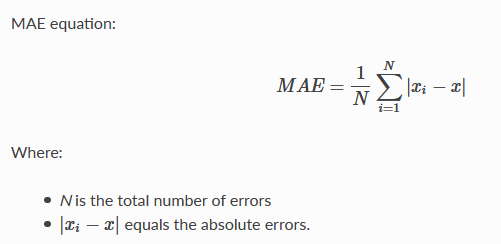
**Model Performance h2o:**

https://docs.h2o.ai/h2o/latest-stable/h2o-docs/performance-and-prediction.html

**Für eine Regression gibt es folgende Evaluationsmetriken in h2o:**

**mae: mean absolute error**

The mean absolute error is an average of the absolute errors. The MAE units are the same as the predicted target, which is useful for understanding whether the size of the error is of concern or not. The smaller the MAE the better the model’s performance. (**Tip**: MAE is robust to outliers. If you want a metric that is sensitive to outliers, try root mean squared error (RMSE).)



**residual\_deviance:**

- berechnet über 2(LL(Saturated Model) - LL(Proposed Model))

-> df\_Sat - df\_proposed

LL: log likelihood

Saturated Model: Annahme, dass jeder Datenpunkt seine eigenen Parameter hat

Proposed Model: Annahme, dass man die Datenpunkte über p parameter + intercept bestimmen kann

ist das Modell gut, sollte die Abweichung ~ Chi² entsprechen

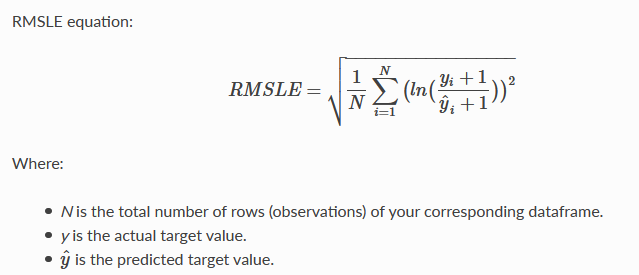
**r2:** R-Squared

The R2 value represents the degree that the predicted value and the actual value move in unison.

**mean\_residual\_deviance:**

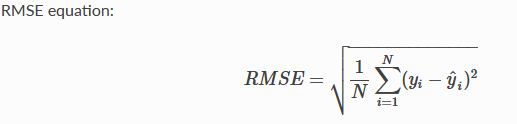
**rmsle:** Root mean squared log error

This metric measures the ratio between actual values and predicted values and takes the log of the predictions and actual values. Use this instead of RMSE if an under-prediction is worse than an over-prediction. You can also use this when you don’t want to penalize large differences when both of the values are large numbers.



**rmse:** Root mean squared error

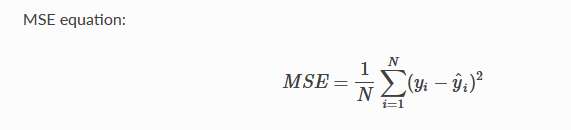
The RMSE metric evaluates how well a model can predict a continuous value. The RMSE units are the same as the predicted target, which is useful for understanding if the size of the error is of concern or not. The smaller the RMSE, the better the model’s performance. (**Tip**: RMSE is sensitive to outliers. If you want a more robust metric, try mean absolute error (MAE).)



**mse:** mean squared error (mittlere quadratische Abweichung)

The MSE metric measures the average of the squares of the errors or deviations. MSE takes the distances from the points to the regression line (these distances are the “errors”) and squaring them to remove any negative signs. MSE incorporates both the variance and the bias of the predictor.

MSE also gives more weight to larger differences. The bigger the error, the more it is penalized.

(**Tip**: MSE is sensitive to outliers. If you want a more robust metric, try mean absolute error (MAE).)

**Which metric should I use?**

Different metrics will show the performance of your model in different units. Let’s continue with our example where our target is to predict the number of days until an event. Some possible performance units are:

* **Same as target:** The unit of the metric is in days
  + ex: MAE = 5 means the model predictions are off by 5 days on average
* **Percent of target:** The unit of the metric is the percent of days
  + ex: MAPE = 10% means the model predictions are off by 10 percent on average
* **Square of target:** The unit of the metric is in days squared
  + ex: MSE = 25 means the model predictions are off by 5 days on average (square root of 25 = 5)

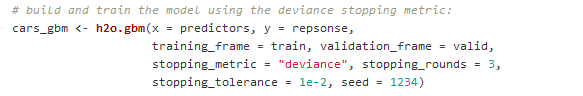
| **Metric** | **Units** | **Sensitive to Outliers** | **Tip** |
| --- | --- | --- | --- |
| R2 | scaled between 0 and 1 | No | use when you want performance scaled between 0 and 1 |
| MSE | square of target | Yes |  |
| RMSE | same as target | Yes |  |
| RMSLE | log of target | Yes |  |
| RMSPE | percent of target | Yes | use when target values are across different scales |
| MAE | same as target | No |  |
| MAPE | percent of target | No | use when target values are across different scales |
| SMAPE | percent of target divided by 2 | No | use when target values are close to 0 |

**Wie setze ich die stopping model metrics richtig?**

**Deviance**

The model will stop building if the deviance fails to continue to improve. Deviance is computed as follows:

Loss = Quadratic -> MSE==Deviance For Absolute/Laplace or Huber -> MSE != Deviance





**Model Performance Graphs:**

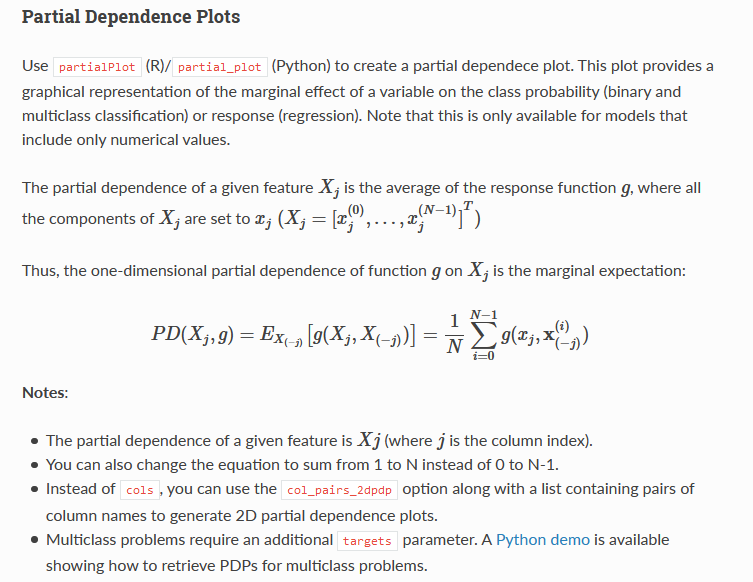
**Variable Importances**

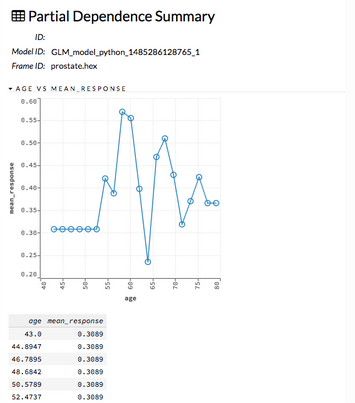
Variable importances represent the statistical significance of each variable in the data in terms of its affect on the model. Variables are listed in order of most to least importance. The percentage values represent the percentage of importance across all variables, scaled to 100%.

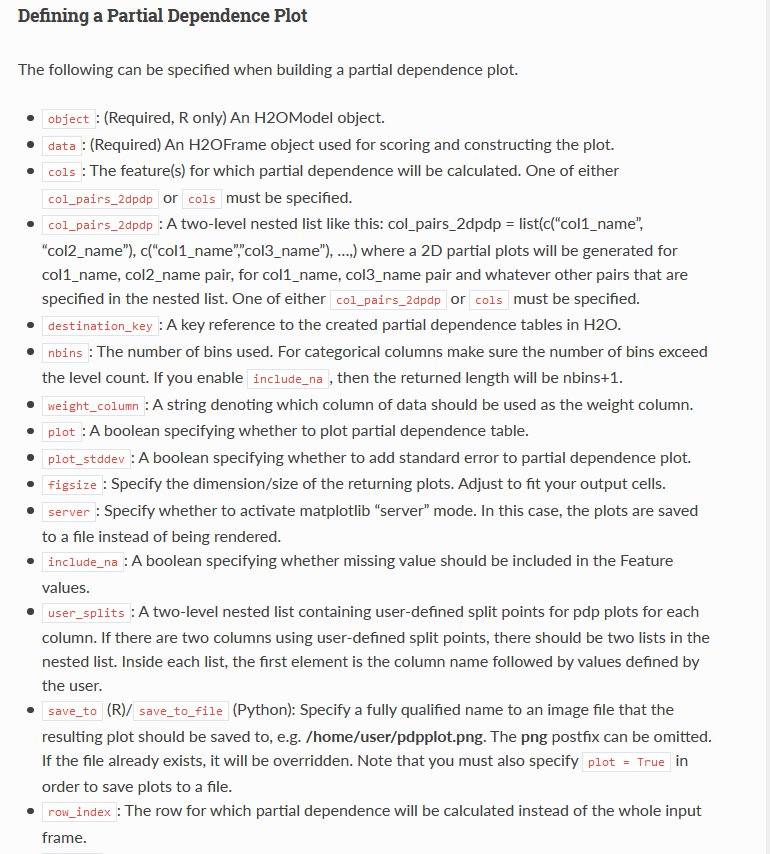
**Standardized Coefficient Magnitudes**

This chart represents the relationship of a specific feature to the response variable. Coefficients can be positive (orange) or negative (blue). A positive coefficient indicates a positive relationship between the feature and the response, where an increase in the feature corresponds with an increase in the response, while a negative coefficient represents a negative relationship between the feature and the response where an increase in the feature corresponds with a decrease in the response (or vice versa).

h2o.std\_coef\_plot(pros\_glm)







# Target Encoding

Target encoding is the process of replacing a categorical value with the mean of the target variable. Any non-categorical columns are automatically dropped by the target encoder model.

**Note**: You can also use target encoding to convert categorical columns to numeric. This can help improve machine learning accuracy since algorithms tend to have a hard time dealing with high cardinality columns. The jupyter notebook, [categorical predictors with tree based model](https://github.com/h2oai/h2o-tutorials/blob/master/best-practices/categorical-predictors/gbm_drf.ipynb), discusses two methods for dealing with high cardinality columns:

* Comparing model performance after removing high cardinality columns
* Parameter tuning (specifically tuning nbins\_cats and categorical\_encoding)

## Target Encoding Parameters

### blending

The blending parameter defines whether the target average should be weighted based on the count of the group. It is often the case, that some groups may have a small number of records and the target average will be unreliable. To prevent this, the blended average takes a weighted average of the group’s target value and the global target value.

### data\_leakage\_handling

To control data leakage, specify one of the following data leakage handling strategies:

* none (Python)/None (R): Do not holdout anything. Using whole frame for training
* k\_fold (Python)/KFold (R): Encodings for a fold are generated based on out-of-fold data.
* leave\_one\_out (Python)/LeaveOneOut (R): The current row’s response value is subtracted from the pre-calculated per-level frequencies.

### f

The smoothing value is used for blending when blending=True and to calculate lambda. Smoothing controls the rate of transition between the particular level’s posterior probability and the prior probability. For smoothing values approaching infinity, it becomes a hard threshold between the posterior and the prior probability. This value defaults to 20.

### fold\_column

Specify the name or column index of the fold column in the data. This defaults to NULL (no fold\_column).

### ignored\_columns

Specify the column or columns to ignore. Note that this command is only available in the Python client and in Flow. It is not available in R.

### k

Use k to specify the inflection point value. This value is used for blending when blending=True and to calculate lambda. This determines half of the minimal sample size for which we completely trust the estimate based on the sample in the particular level of the categorical variable. This value defaults value to 10.

### noise\_level

If random noise should be added to the target average, the noise\_level parameter can be used to specify the amount of noise to be added. This value defaults to 0.01 times the range of *y*

of random noise.

### response\_column

Use response\_column to specify the response column is the column that you are attempting to predict (y-axis).

### seed

A random seed used to generate draws from the uniform distribution for random noise. Defaults to -1.

### training\_frame

Specify the dataset that you want to use when you are ready to build a Target Encoding model.

### transform

Apply transformation to target encoded columns based on the encoding maps generated during training. Available parameters include:

* frame: The H2O frame to which you are applying target encoding transformations.
* data\_leakage\_handling: To control data leakage, specify one of the following data leakage handling strategies:
  + none (Python)/None (R): Do not holdout anything. Using whole frame for training
  + k\_fold (Python)/KFold (R): Encodings for a fold are generated based on out-of-fold data.
  + leave\_one\_out (Python)/LeaveOneOut (R): The current row’s response value is subtracted from the pre-calculated per-level frequencies.
* noise: A float value specifying the amount of random noise added to the target encoding. This helps prevent overfitting. Defaults to 0.01 \* range of y.
* seed: A random seed used to generate draws from the uniform distribution for random noise. Defaults to -1.

**Cross Validation:**

[K-fold cross-validation](https://en.wikipedia.org/wiki/Cross-validation_(statistics)#k-fold_cross-validation) is used to validate a model internally, i.e., estimate the model performance without having to sacrifice a validation split. Also, you avoid statistical issues with your validation split (it might be a “lucky” split, especially for imbalanced data). Good values for K are around 5 to 10. Comparing the K validation metrics is always a good idea, to check the stability of the estimation, before “trusting” the main model.

You have to make sure, however, that the holdout sets for each of the K models are good. For i.i.d. data, the random splitting of the data into K pieces (default behavior) or modulo-based splitting is fine. For temporal or otherwise structured data with distinct “events”, you have to make sure to split the folds based on the events. For example, if you have observations (e.g., user transactions) from K cities and you want to build models on users from only K-1 cities and validate them on the remaining city (if you want to study the generalization to new cities, for example), you will need to specify the parameter “fold\_column” to be the city column. Otherwise, you will have rows (users) from all K cities randomly blended into the K folds, and all K cross-validation models will see all K cities, making the validation less useful (or totally wrong, depending on the distribution of the data). This is known as “data leakage”: <https://youtu.be/NHw_aKO5KUM?t=889>

## How Cross-Validation is Calculated

In general, for all algos that support the nfolds parameter, H2O’s cross-validation works as follows:

For example, for nfolds=5, 6 models are built. The first 5 models (cross-validation models) are built on 80% of the training data, and a different 20% is held out for each of the 5 models. Then the main model is built on 100% of the training data. This main model is the model you get back from H2O in R, Python and Flow (though the CV models are also stored and available to access later).

This main model contains training metrics and cross-validation metrics (and optionally, validation metrics if a validation frame was provided). The main model also contains pointers to the 5 cross-validation models for further inspection.

All 5 cross-validation models contain training metrics (from the 80% training data) and validation metrics (from their 20% holdout/validation data). To compute their individual validation metrics, each of the 5 cross-validation models had to make predictions on their 20% of rows of the original training frame, and score against the true labels of the 20% holdout.

For the main model, this is how the cross-validation metrics are computed: The 5 holdout predictions are combined into one prediction for the full training dataset (i.e., predictions for every row of the training data, but the model making the prediction for a particular row has not seen that row during training). This “holdout prediction” is then scored against the true labels, and the overall cross-validation metrics are computed.

This approach has some implications. Scoring the holdout predictions freshly can result in different metrics than taking the average of the 5 validation metrics of the cross-validation models. For example, if the sizes of the holdout folds differ a lot (e.g., when a user-given fold\_column is used), then the average should probably be replaced with a weighted average. Also, if the cross-validation models map to slightly different probability spaces, which can happen for small DL models that converge to different local minima, then the confused rank ordering of the combined predictions would lead to a significantly different AUC than the average.

**Welche Parameter sind bei Entscheidungsbäumen zu beachten?**

Depth (Wie tief darf der Baum sein -> max splits)

Observations per Node (wie viele Fälle müssen in eine Node reinfallen, damit noch gesplitted wird?)

Loss metric (Die Loss metric muss sich um mind. Xxx erhöhen, damit noch gesplitted wird)

**Prinzip einer Gradient Boosting Machine**

Es werden squentiell Bäume trainiert, und jeder davon wird auf den Fehlern des vorhergigen Baumes aufgebaut