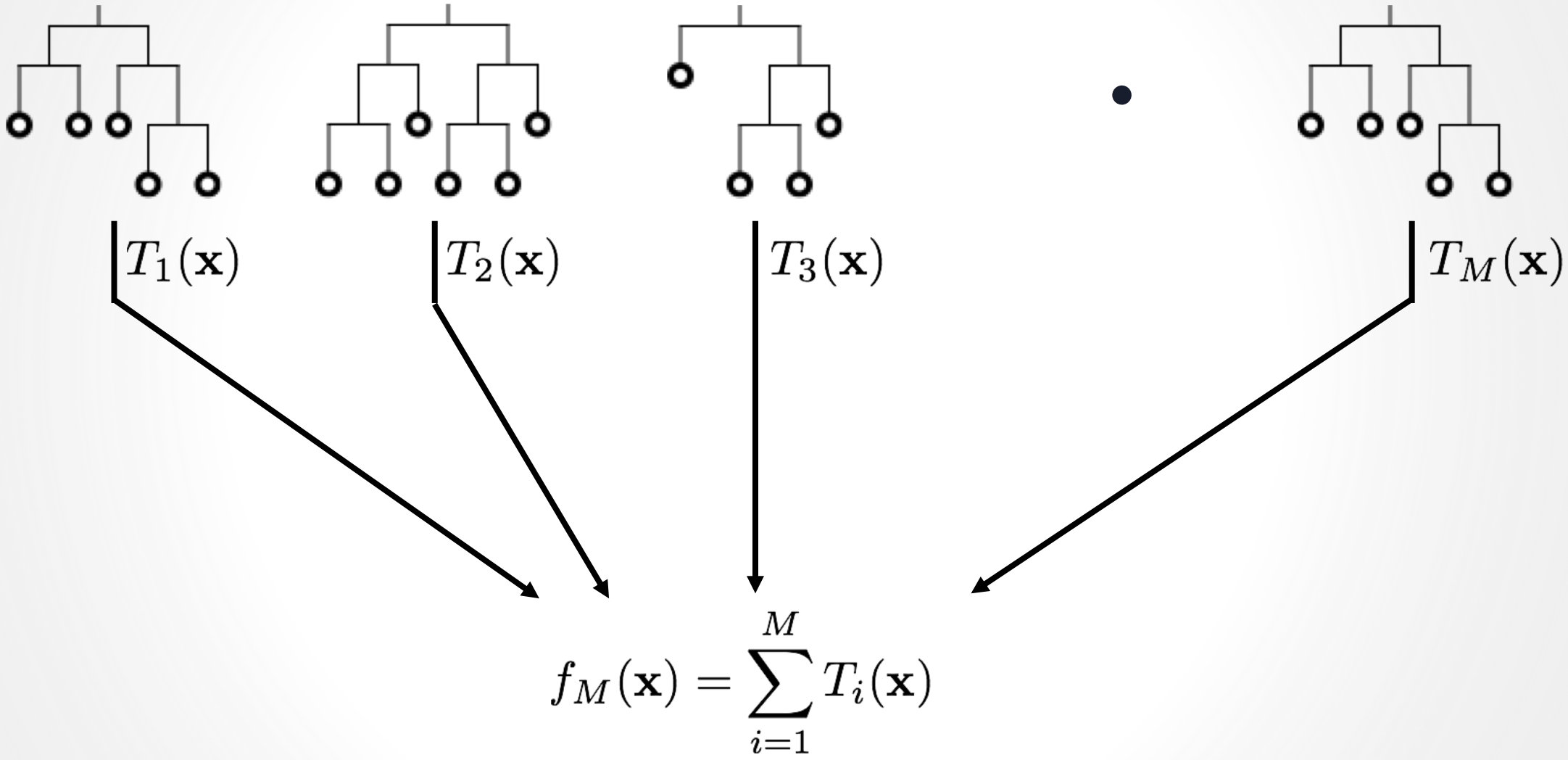
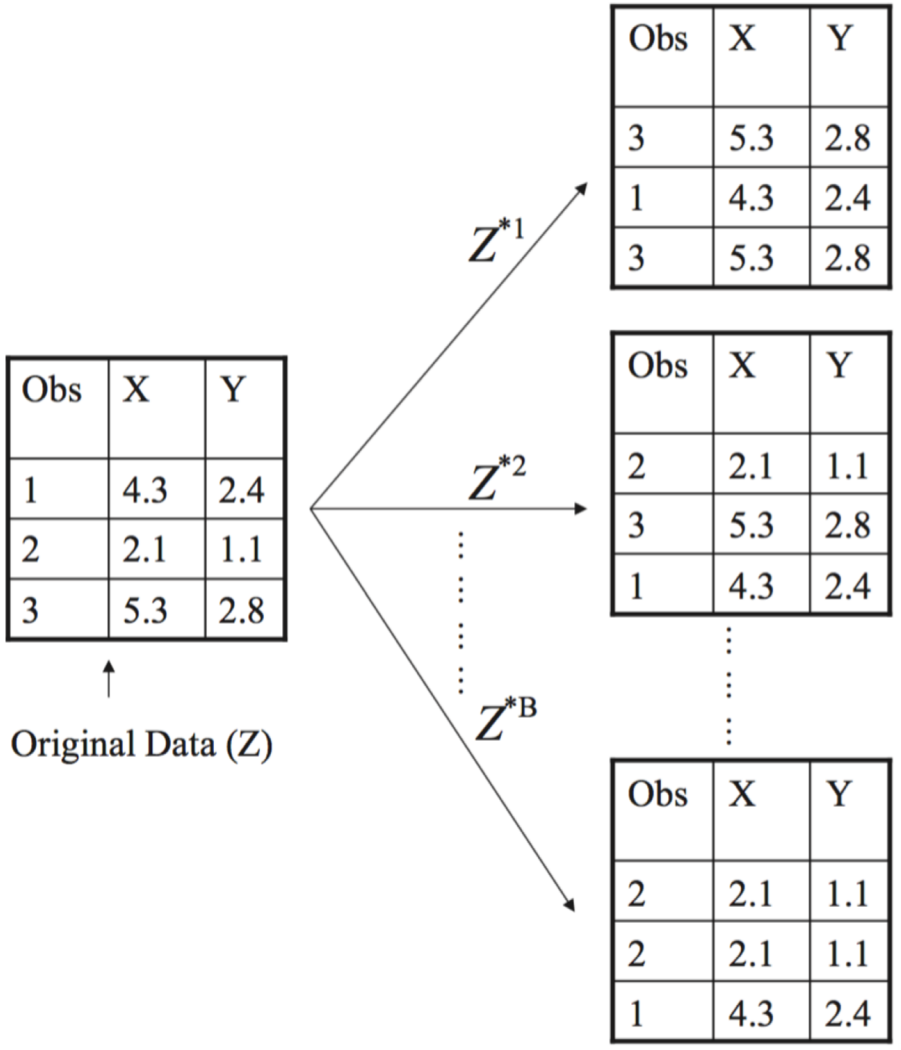
Supervised Learning:

# RANDOM FORESTS

# Random Forest



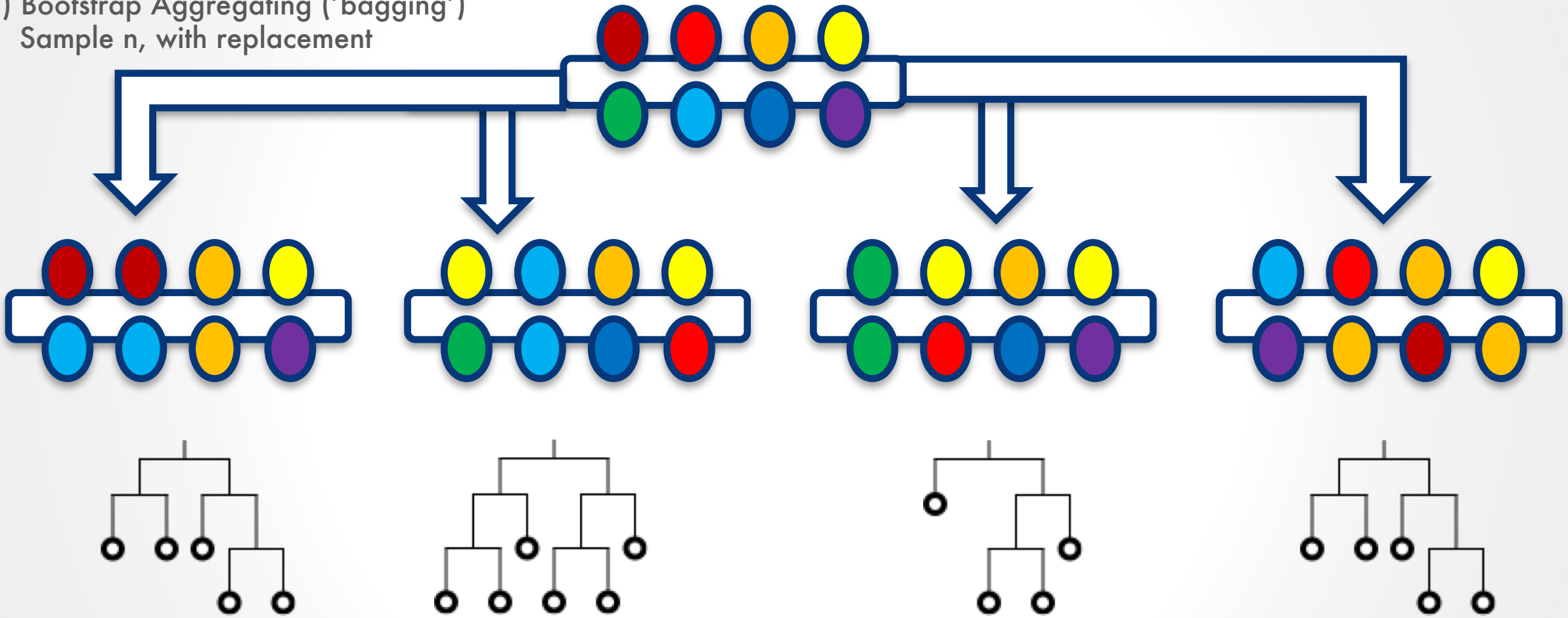
# The Bootstrap




James, Witten, Hastie. *An Introduction to Statistical Learning*

# RF Tree Variance: Bagging

1) Bootstrap Aggregating ('bagging')
   Sample n, with replacement



$$f_M(\mathbf{x}) = \sum_{i=1}^{M} T_i(\mathbf{x})$$

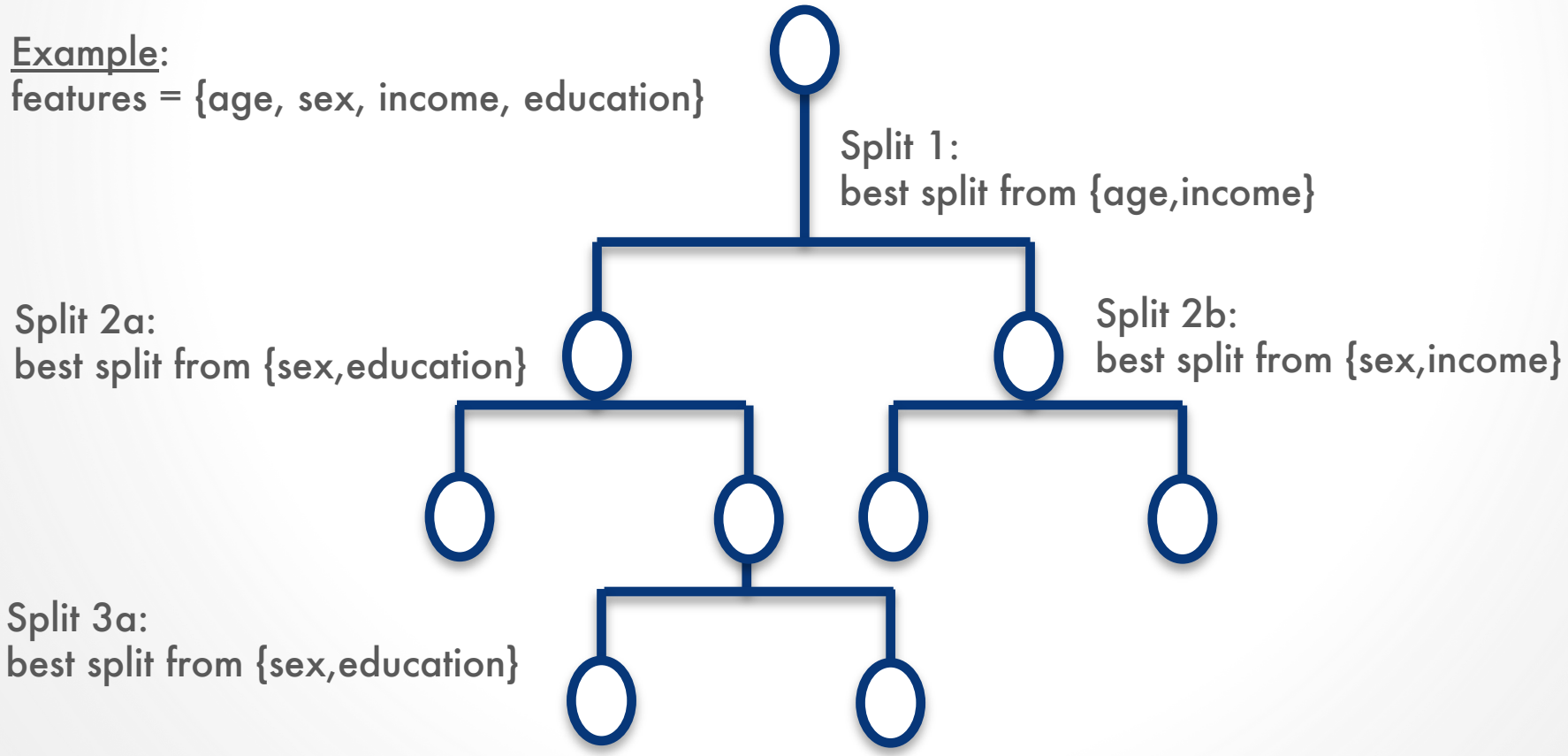
# RF Tree Variance: Splits

2) Do not consider all of the features for each split

- default is often $\sqrt{n}$
- also speeds up computational time

Additional methods for variance:

- do not consider all features for each *tree*
- H2O: random histograms

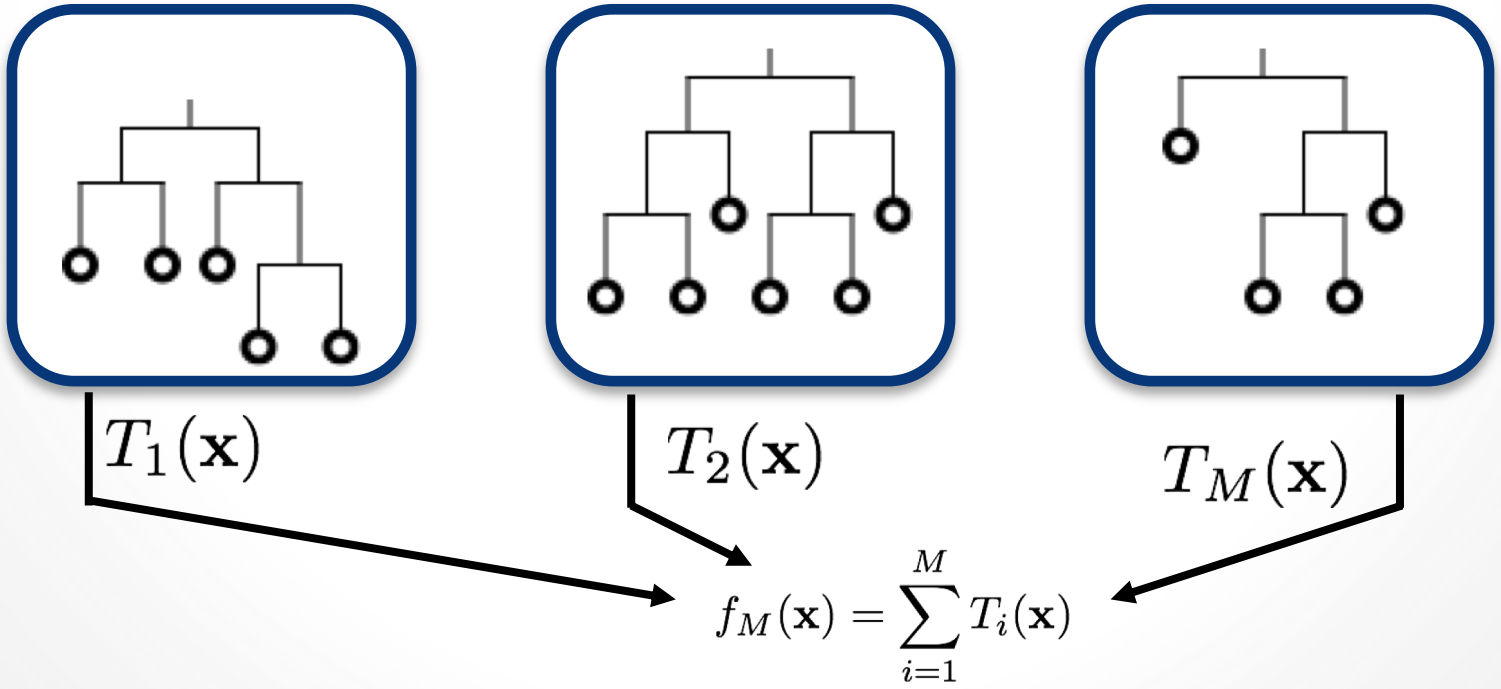


# Distributed RFs

Method 1: Parallelize by tree

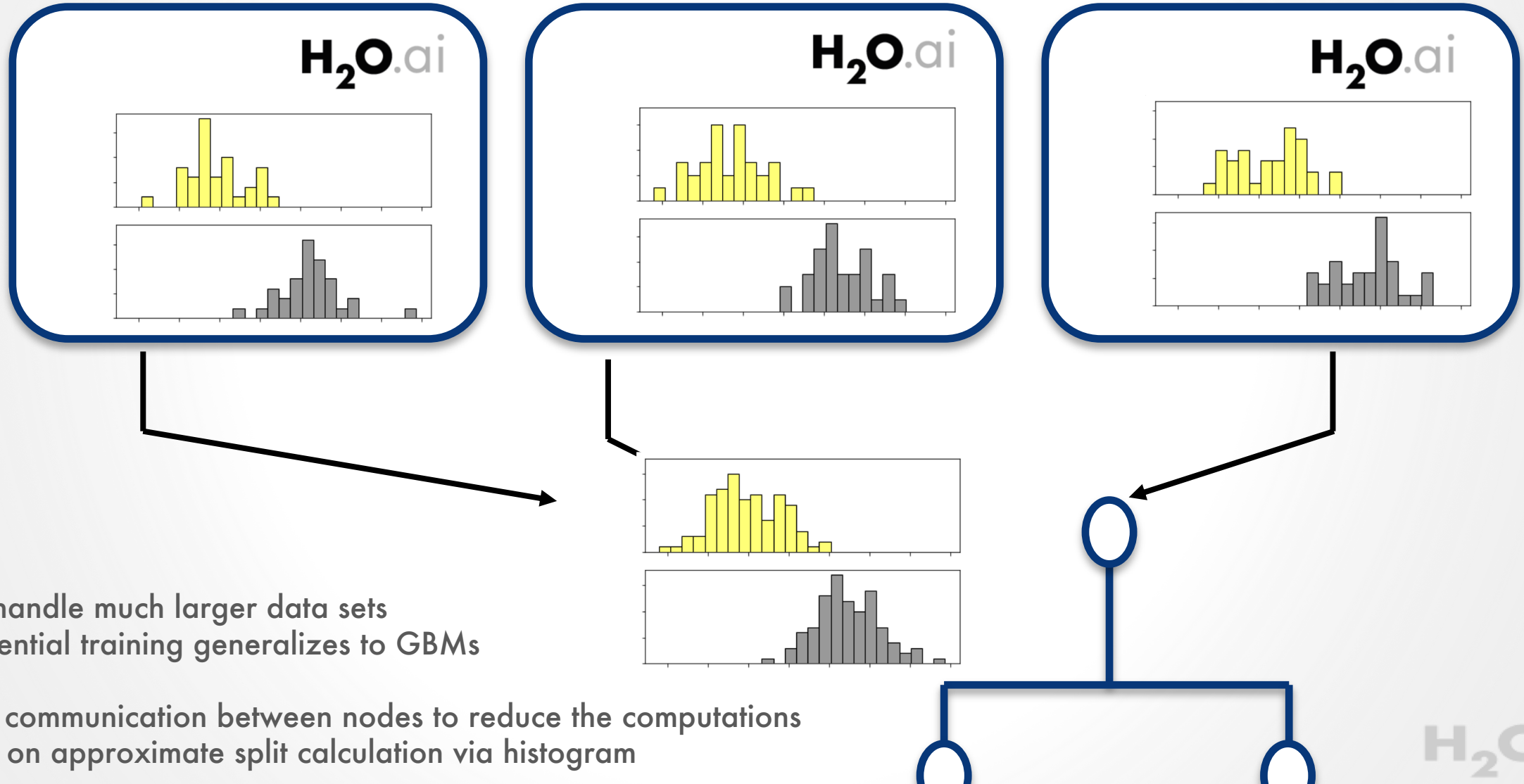
low communication between nodes (only to reduce as part of map/reduce)

✗ every node must have all data in memory
✗ does not generalize to GBMs



# Distributed RFs in H2O

Method 2: Parallelize by data




can handle much larger data sets
sequential training generalizes to GBMs

✗ more communication between nodes to reduce the computations
✗ relies on approximate split calculation via histogram

# RF Parameters: Setting Up a Model

Python API

```
h2o.estimators.random_forest.H2ORandomForestEstimator
- h2o.estimators.estimator_base.H2OEstimator
-- h2o.model.model_base.ModelBase
```

| | |
|---|---|
| `model_id:` | specify a custom name for the model to use as a reference. (default: randomly generated) |
| `checkpoint:` | load a previously generated model |
| `seed:` | specify a pseudorandom seed, for reproducibility |
| `verbose:` | print verbose scoring history to console |

# RF Parameters: Defining Data

- `training_frame:` H2OFrame used to train model
- `validation_frame:` H2OFrame used to validate model (optional)
- `y:` column name or index of target variable
- `x:` column names (of indices) of predictor variables (default: all but y and ignored_columns)

- `ignored_columns:` columns to ignore (e.g. ID column)
- `ignore_const_cols:` ignore constant columns (default: True)
- `weights_column:` column by which to weight the data points

- `categorical_encoding:` encoding scheme for categorical variables: {"enum", "one_hot_explicit", "binary", "eigen", "label_encoder", "sort_by_response"} (default: "enum")

# RF Parameters: Loss Function

- `distribution:` specifies the loss function
  - `'bernoulli'`
  - `'multinomial'`
  - `'gaussian'`
  - `'poisson'`
  - `'gamma'`
  - `'laplace'`
  - `'quantile'`
  - `'huber'`
  - `'tweedie'`

- `huber_alpha, quantile_alpha`

# Choosing a Distribution Function For Numeric Response

| Distribution | Loss | Details |
|---|---|---|
| Gaussian | Squared Error Loss | Sensitive to outliers |
| Laplace | Absolute Error Loss | Very robust to outliers |
| Huber | Hybrid of Squared and Absolute Error Loss | Robust to outliers |
| Poisson | Poisson Loss (Deviance) | Used for estimating counts |
| Gamma | Gamma Loss | Used for estimating total values |
| Tweedie | Tweedie Loss | Used for estimating densities |
| Quantile | Quantile Regression Loss | Used for estimating a specified percentile |

# RF Parameters: Ensemble

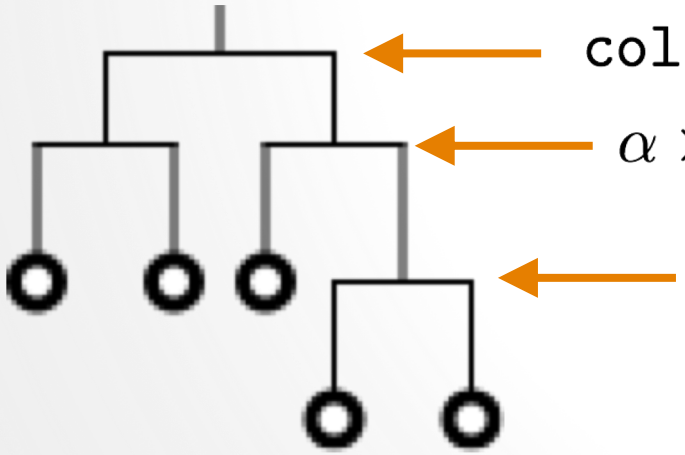
- `ntrees:` number of trees

- `sample_rate:` row sampling rate per tree (default: 0.6320000291)
- `col_sample_rate_per_tree:` column sampling rate per tree (default: 1)

# RF Parameters: Individual Trees

- **Column sampling for split:**
- **mtries:** number of columns to sample on each split
  (default: sqrt(n) for classification, n/3 for regression)
- **col_sample_rate_change_per_level:** factor by which to increase or decrease mtries per level of tree

$$\alpha = \texttt{col\_sample\_rate\_change\_per\_level}\ (0 \leq \alpha \leq 2)$$



$\texttt{col\_sample\_rate}$

$\alpha \times \texttt{col\_sample\_rate}$

$\alpha^2 \times \texttt{col\_sample\_rate}$

For level $n$ : $\alpha^{n-1} \times \texttt{col\_sample\_rate}$

# RF Parameters: Individual Trees

- **Column sampling for split:**
- **mtries:** number of columns to sample on each split (default: sqrt(n) for classification, n/3 for regression)
- **col_sample_rate_change_per_level:** factor by which to increase or decrease mtries per level of tree

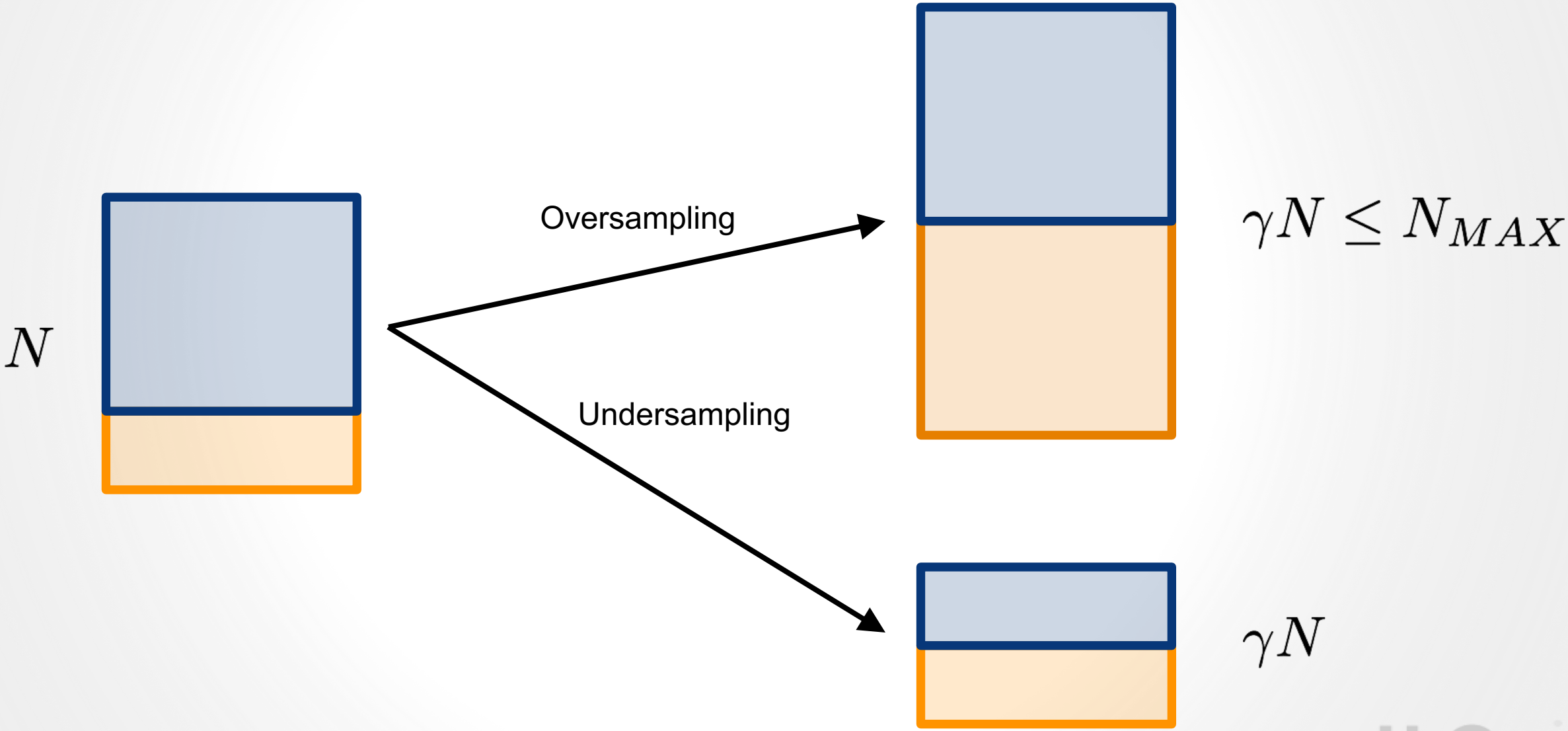
- **When to stop splitting?**
- **max_depth:** maximum depth of each tree
- **min_rows:** minimum rows in a leaf (i.e. stop splitting when data size is this small)
- **min_split_improvement:** **minimum relative improvement** in split criterion for a split to occur

- **Histogramming**
- **nbins**: number of bins for numeric variables (default: 20)
- **nbins_top_level**: can be used instead of nbins; nbins will then decrease by 2 each level
- **nbins_cats**: number of bins for categorical variables (default: 1024)
- **histogram_type**: method for binning {"Uniform Adaptive", "Random", "QuantilesGlobal", "RoundRobin"}

# RF Parameters: Class Imbalances

- **`balance_classes`**: balance training data class counts via over/under-sampling (default: False)
- **`class_sampling_factors`**: desired over/under-sampling ratios per class (in lexicographic order). If not specified, sampling fac- tors will be automatically
- **`max_after_balance_size`**: maximum relative size of the training data after balancing class counts (can be less than 1.0). Requires balance classes. Defaults to 5.0.
- **`sample_rate_per_class:`** variable row sampling rate per class

# RF Parameters: Class Imbalances



# RF Parameters: Scoring & Stopping

- **`score_each_iteration`**: score model after each tree (default: false)
- **`score_tree_interval`**: score model after n trees

- **`stopping_rounds`**: early stop if stopping metric's moving average does not improve for this many rounds
- **`stopping_metric`**: metric for early stopping (AUTO: logloss for classification, deviance for regression) Must be one of: "AUTO", "deviance", "logloss", "MSE", "RMSE", "MAE", "RMSLE", "AUC", "lift_top_group", "misclassification", "mean_per_class_error". Defaults to AUTO.
- **`stopping_tolerance`**: Relative tolerance for metric-based stopping criterion (stop if relative improvement is not at least this much) Defaults to 0.001.
- **`max_runtime_secs:`** maximum runtime to allow for model building (default: 0, disabled)

# RF Parameters: Cross Validation

- **`nfolds`**: number of folds for N-fold cross-validation (default: 0, disabled)
- **`keep_cross_validation_predictions`**: keep the predictions of the cross-validation models (default: False)
- **`keep_cross_validation_fold_assignment`**: keep the cross-validation fold assignment (default: False)
- **`fold_assignment`**: cross-validation fold assignment scheme, if fold column is not specified. The ”Stratified” option will stratify the folds based on the response variable, for classification problems. Must be one of: ”AUTO”, ”Random”, ”Modulo”, ”Strati- fied”. Defaults to AUTO.
- **`fold_column`**: column with cross-validation fold index assignment per observation.

# RF Misc. Parameters

- **binomial_double_trees**

- **offset_column**
- **build_tree_one_node**
- **calibrate_model**
- **calibration_frame**
- **max_hit_ratio_k**: Max. number (top K) of predictions to use for hit ratio computation (for multiclass only, 0 to disable) Defaults to 0.

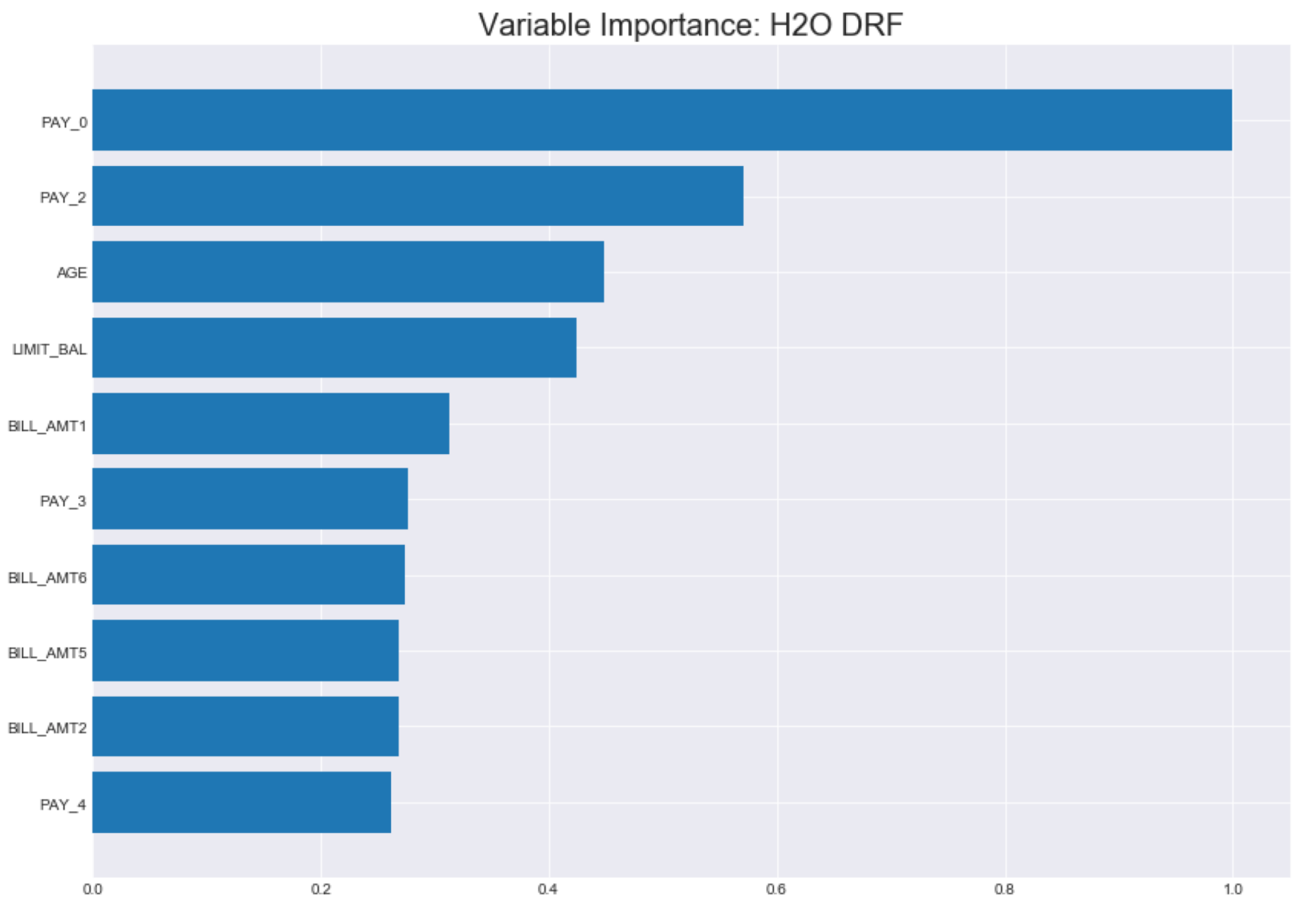
# Random Forests in H2O

`.download_pojo()`

```
class rf_Tree_4_class_0 {
  static final double score0(double[] data) {
    double pred =      (Double.isNaN(data[7]) || data[7 /* PAY_3 */] <1.5f ?
          (data[0 /* LIMIT_BAL */] <135683.5f ?
              (Double.isNaN(data[11]) || data[11 /* BILL_AMT1 */] <35312.5f ?
                  (Double.isNaN(data[17]) || data[17 /* PAY_AMT1 */] <3945.5f ?
                      0.7474255f :
                      0.8430034f) :
                  (Double.isNaN(data[4]) || data[4 /* AGE */] <41.5f ?
                      0.79907835f :
                      0.8511166f)) :
              (Double.isNaN(data[5]) || data[5 /* PAY_0 */] <1.5f ?
                  (data[17 /* PAY_AMT1 */] <2876.5f ?
                      0.8268641f :
                      0.9220396f) :
                  (Double.isNaN(data[20]) || data[20 /* PAY_AMT4 */] <11648.5f ?
                      0.44347826f :
                      0.13333334f))) :
          (Double.isNaN(data[9]) || data[9 /* PAY_5 */] <1.0f ?
              (Double.isNaN(data[14]) || data[14 /* BILL_AMT4 */] <165892.5f ?
                  (data[9 /* PAY_5 */] <-0.5f ?
                      0.6821192f :
                      0.5637931f) :
                  (data[15 /* BILL_AMT5 */] <178671.5f ?
                      0.0f :
                      0.42857143f)) :
              (data[3 /* MARRIAGE */] <1.5f ?
                  (Double.isNaN(data[22]) || data[22 /* PAY_AMT6 */] <11535.0f ?
                      0.3069307f :
                      0.8333333f) :
                  (data[12 /* BILL_AMT2 */] <4311.5f ?
                      0.25f :
                      0.4448276f))));
    return pred;
  } // constant pool size = 62B, number of visited nodes = 15, static init size = 0B
}
```

# RF Feature Importance

```
.varimp()
.varimp_plot()
```



# Pros and Cons of Random Forests

Pros

- Few tuning parameters
- Easy to parallelize

Cons

- Slow to score
- Not transparent