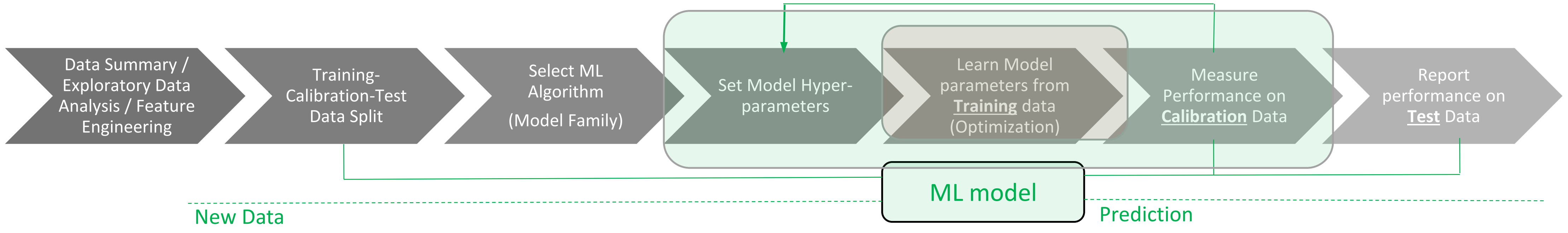
Ensemble Learning

Praphul Chandra

1. James, Gareth, et al. *An introduction to statistical learning*. Vol. 6. New York: springer, 2013.
2. Friedman, Jerome, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning.* Vol. 1. Springer, Berlin: Springer series in statistics, 2001.
3. Kuhn, Max, and Kjell Johnson. *Applied predictive modeling*. New York: Springer, 2013.

Machine Learning Framework

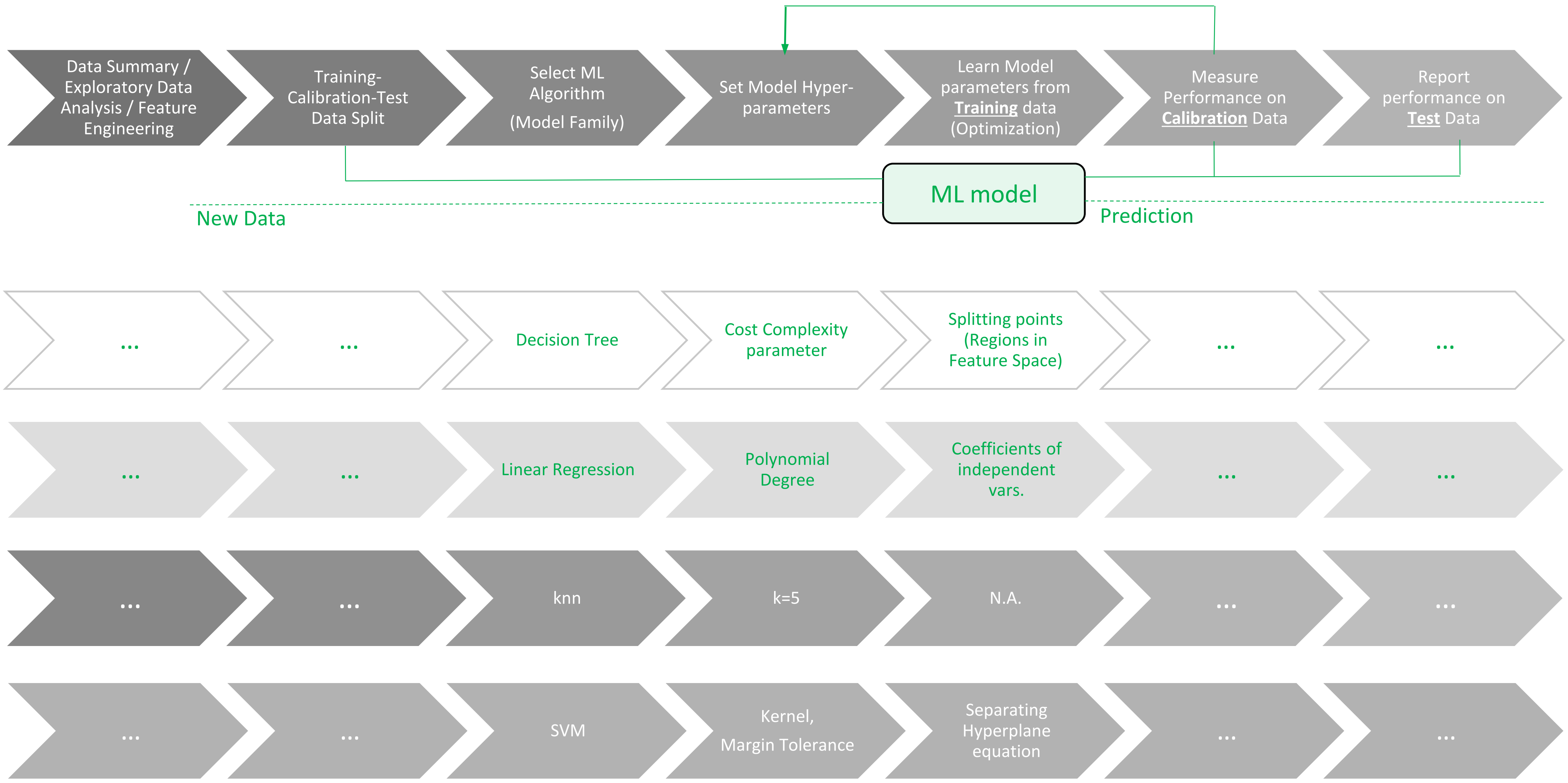
Machine Learning Framework


Data Summary / Exploratory Data Analysis / Feature Engineering
Training-Calibration-Test Data Split
Select ML Algorithm (Model Family)
Set Model Hyper-parameters
Learn Model parameters from Training data (Optimization)
Measure Performance on Calibration Data
Report performance on Test Data
ML model
New Data
Prediction

- Hyperparameter Optimization
  - Optimal model complexity
  - **Iterate** over Hyperparameters + CV (grid search)
- Parameter Optimization
  - Minimize the Loss Function : L(Y, f(X))
  - Given the model (hyperparameter)
  - Solved using (convex) **optimization** techniques

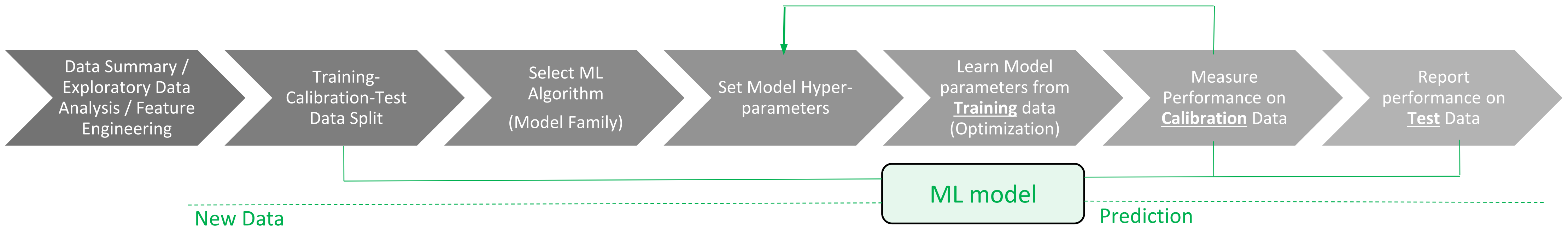
- knn
  - Hyperparameter: Lower k ➔ Higher complexity
  - Parameter: N.A.
- Decision Tree
  - Hyperparameter : Lower α ➔ Higher complexity
  - Parameter: Split points
- Linear Regression
  - Hyperparamet: degree ➔ Higher complexity
  - Parameter: Coefficients of independent variables
- Support Vector Machine
  - Hyperparamater: Kernel, slack
  - Parameter : Coefficients (in the new space)

Machine Learning Framework

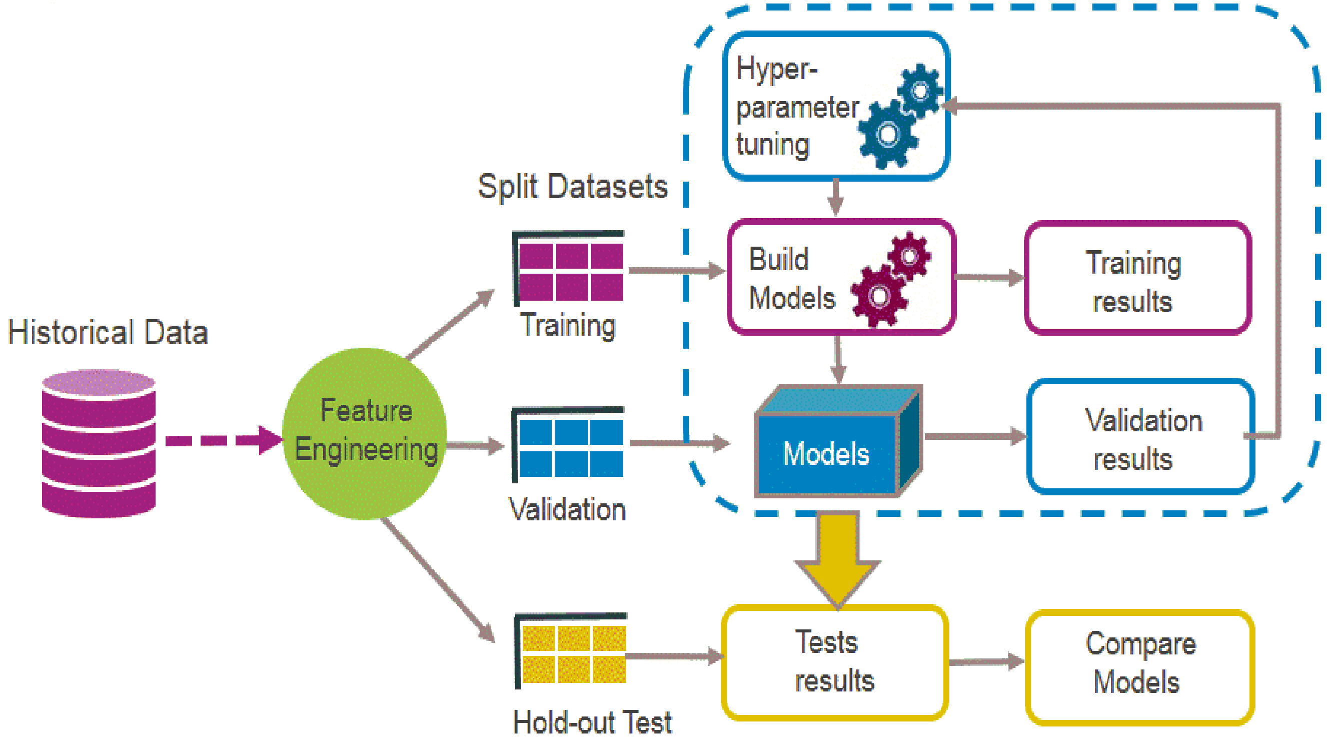


Data Summary / Exploratory Data Analysis / Feature Engineering	Training-Calibration-Test Data Split	Select ML Algorithm (Model Family)	Set Model Hyper-parameters	Learn Model parameters from **Training** data (Optimization)	Measure Performance on **Calibration** Data	Report performance on **Test** Data
…	…	Decision Tree	Cost Complexity parameter	Splitting points (Regions in Feature Space)	…	…
…	…	Linear Regression	Polynomial Degree	Coefficients of independent vars.	…	…
…	…	knn	k=5	N.A.	…	…
…	…	SVM	Kernel, Margin Tolerance	Separating Hyperplane equation	…	…

Machine Learning Framework (cont’d)



Data Summary / Exploratory Data Analysis / Feature Engineering
Training-Calibration-Test Data Split
Select ML Algorithm (Model Family)
Set Model Hyper-parameters
Learn Model parameters from Training data (Optimization)
Measure Performance on Calibration Data
Report performance on Test Data
ML model
New Data
Prediction



Ensemble Learning

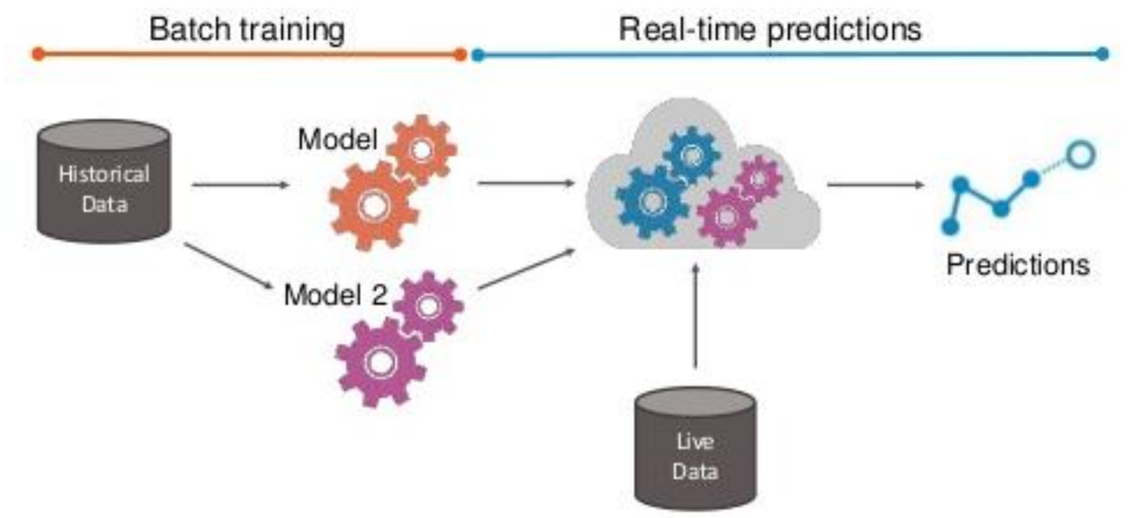
Bagging, Boosting, Stacking etc.

Ensemble Learning : The Big !dea

- ***"Why build 1 when you can have 2 at twice the price?"***
    - Improve model accuracy
    - Different models optimal for different data subsets
    - Reduce variance (Bias-Variance tradeoff)
    - 3?, 4?, 100?

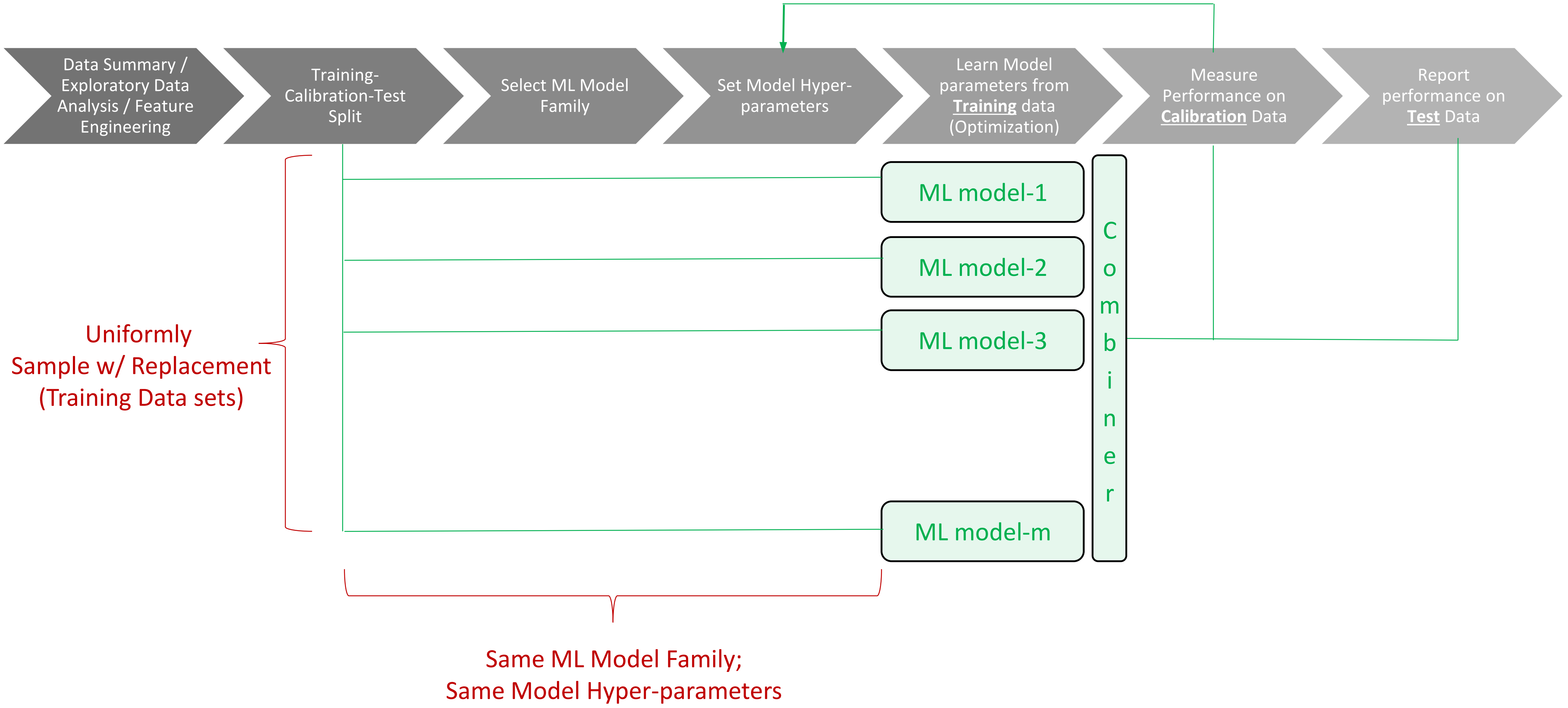
- Aggregating (Combining) results of multiple models
    - Regression: (Weighted) Mean, Median, Max, Min
    - Classification : (Weighted) Majority Voting, Borda

- Building "different" models
    - Model families: knn, decision tree, linear regression
    - Different training data!
    - Different features!



Bagging

Bagging: ML Framework

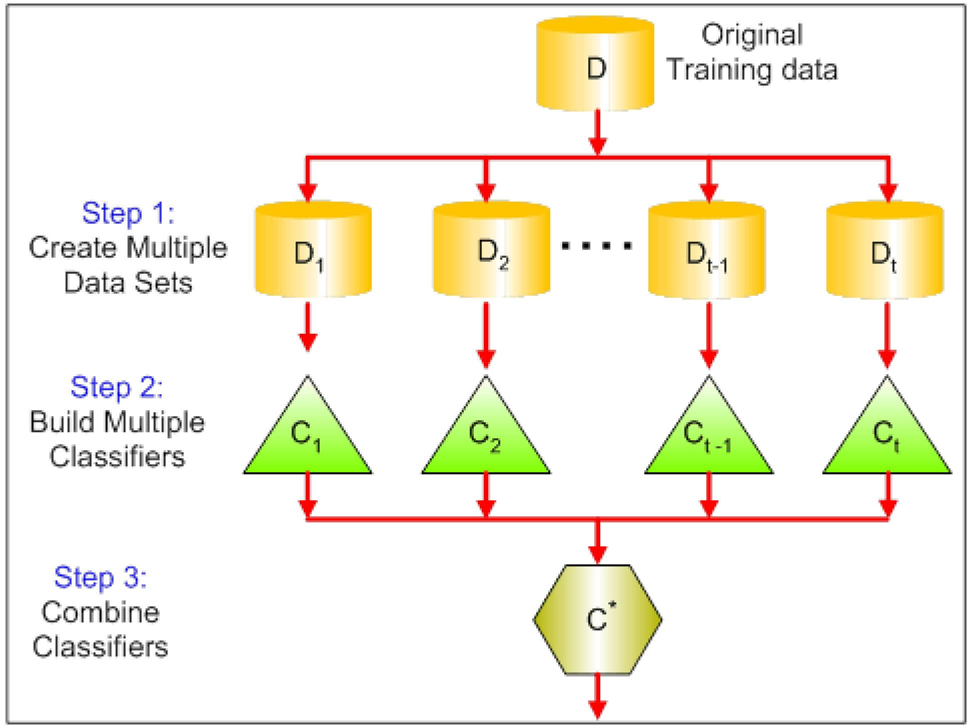

Data Summary / Exploratory Data Analysis / Feature Engineering
Training-Calibration-Test Split
Select ML Model Family
Set Model Hyper-parameters
Learn Model parameters from **Training** data (Optimization)
Measure Performance on **Calibration** Data
Report performance on **Test** Data
ML model-1
ML model-2
ML model-3
ML model-m
Combiner
Uniformly Sample w/ Replacement (Training Data sets)
Same ML Model Family; Same Model Hyper-parameters

Bagging: Bootstrap Aggregation

- Different training data!
  - Sample Training Data with Replacement
  - Same algorithm on different subsets of training data

- Bagging
  - Algorithm independent : general purpose technique
  - Well suited for high variance algorithms
  - Variance Reduction: Averaging a set of observations reduces variance
  - Choose # of classifiers to build (B)

- Application
  - Use with high variance algorithms (DT, NN)
  - Easy to parallelize
  - Limitation: Loss of Interpretability
  - Limitation: What if one of the features dominates?

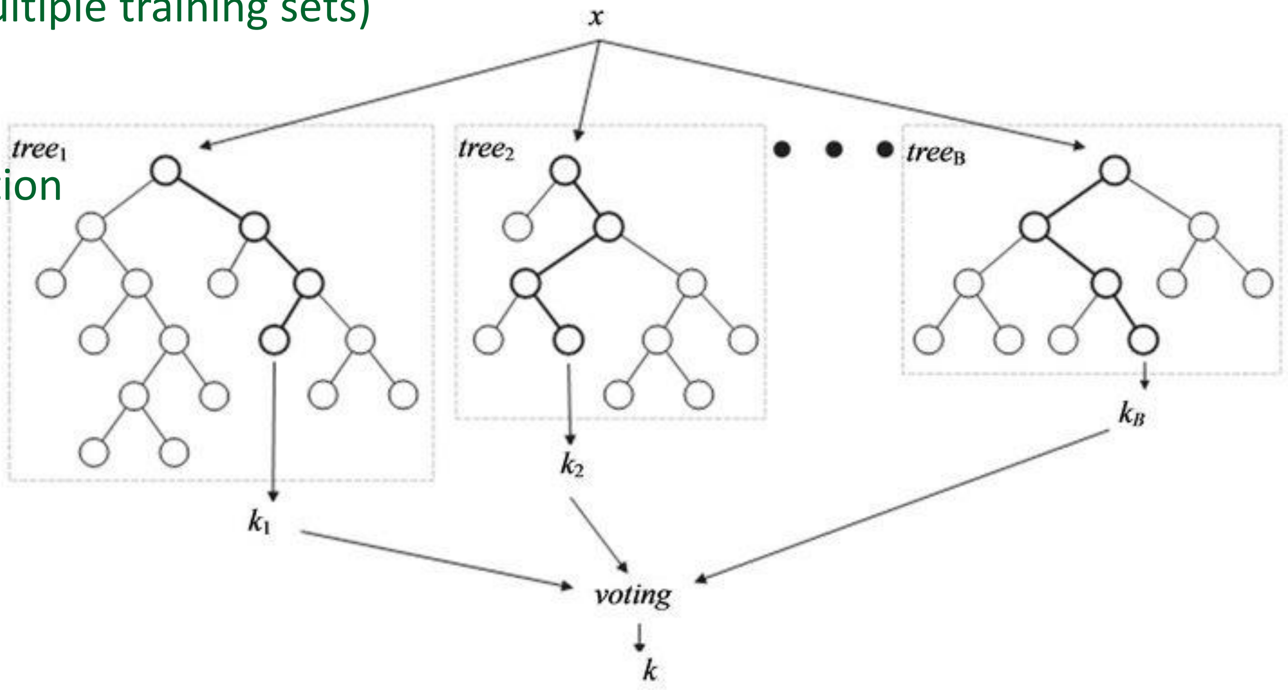

D
Original Training data
Step 1: Create Multiple Data Sets
D_1
D_2
D_{t-1}
D_t
Step 2: Build Multiple Classifiers
C_1
C_2
C_{t-1}
C_t
Step 3: Combine Classifiers
C^*

Bagged Trees

- Decision Trees have high variance
  - The resulting tree (model) depends on the underlying training data
  - (Unpruned) Decision Trees tend to overfit
  - One option: Cost Complexity Pruning

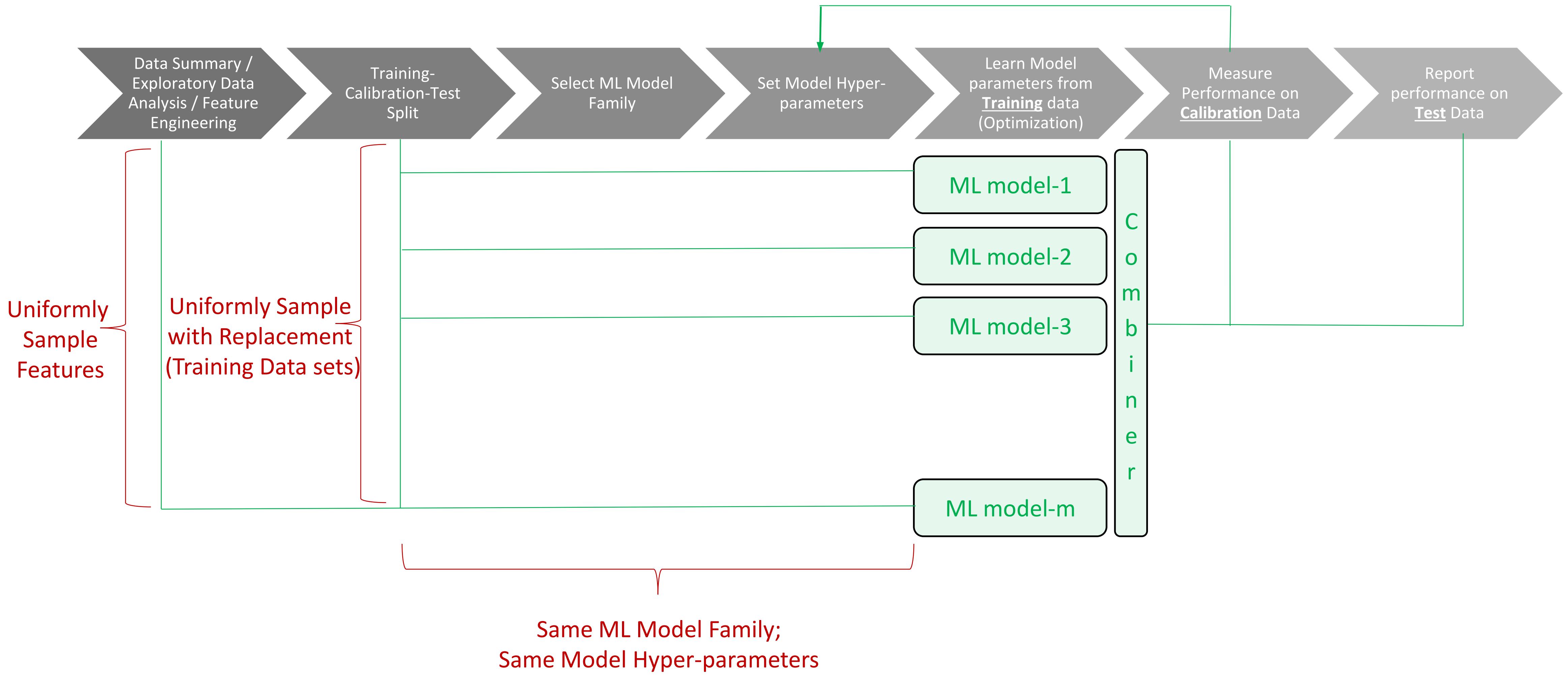
- Bag Trees
  - Sample with replacement (1 Training set → Multiple training sets)
  - Train model on each bootstrapped training set
  - Multiple trees; each different : A garden ☺
  - Each DT predicts; Mean / Majority vote prediction
  - Choose # of trees to build (B)

- Advantages
  - Reduce model variance / instability
  - But Harder to interpret (rules ?)



Random (Feature) Sub-spaces

Random subspaces: ML Framework



Data Summary / Exploratory Data Analysis / Feature Engineering
Training-Calibration-Test Split
Select ML Model Family
Set Model Hyper-parameters
Learn Model parameters from Training data (Optimization)
Measure Performance on Calibration Data
Report performance on Test Data
ML model-1
ML model-2
ML model-3
ML model-m
Combiner
Uniformly Sample Features
Uniformly Sample with Replacement (Training Data sets)
Same ML Model Family; Same Model Hyper-parameters

Random Feature Subspaces

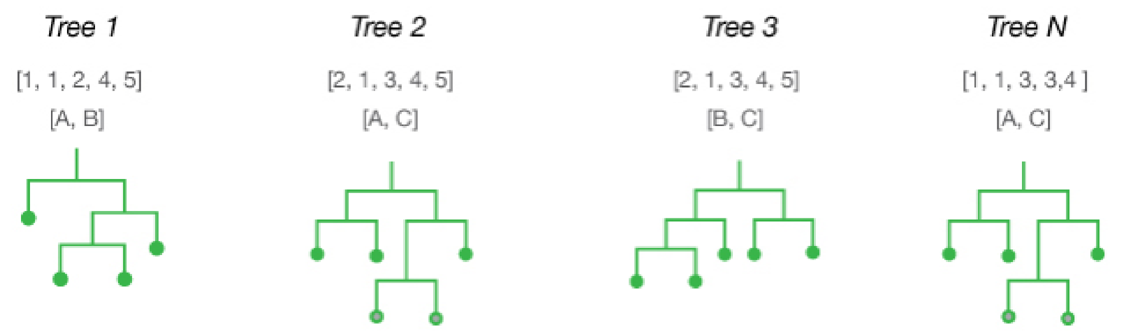
- Try different features for different training data
  - Sample Training Data with Replacement
  - Build each model using a random subset of the features!
  - Same algorithm on different subsets of training data

- Why?
  - Think “regularization”
  - If there is one very strong predictor & other moderately strong predictors..
  - All models will give high importance to the strong predictor $\rightarrow$ All models in the ensemble will be similar
  - Choose # of classifiers to build (B) and # of features to sample (m) from p available features
  - Recommended heuristics: m = sqrt(p) *(m = p : Approach reduces to Bagged Trees)*

- Advantages
  - De-correlates the models in the ensemble
  - Improve accuracy of prediction

Random Feature Spaces in Decision Trees : Random Forests

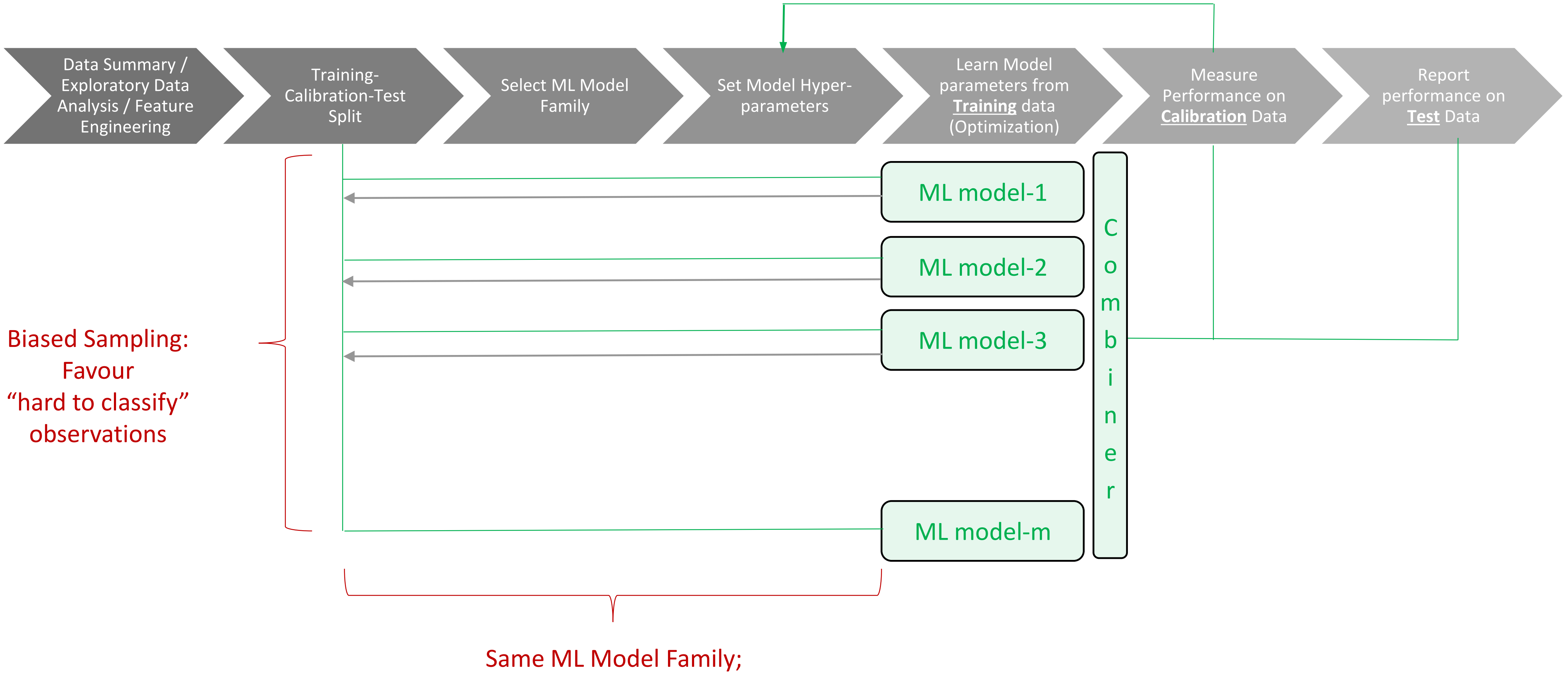
- Decision Trees have high variance
  - The resulting tree (model) depends on the underlying training data
  - Bagged Trees can help reduce variance; Random Forests even more so…
- Random Forests
  - Sample with replacement (1 Training set → Multiple training sets) : Train model on each training set
  - Each tree uses a random subset of feature : A random forest
  - Each DT predicts; Mean / Majority vote prediction
  - Faster than bagging (fewer splits to evaluate per tree)
  - Choose # of trees to build (B) and # of features to sample (m) from p available features



The diagram above assumes five observations [1, 2, 3, 4, 5] and three predictor variables [A, B, C]. It shows the construction of four simple (but different) trees. The observations have been sampled with replacement which means that some observations can occur more than once. In the scenario above, two predictor variables are used to grow each tree, rather than using the entire set of predictor variables.

Boosting

Boosting: ML Framework

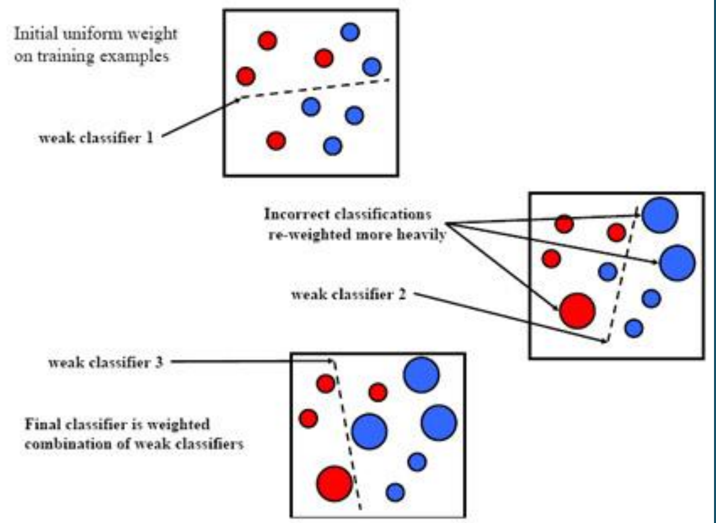

Data Summary / Exploratory Data Analysis / Feature Engineering
Training-Calibration-Test Split
Select ML Model Family
Set Model Hyper-parameters
Learn Model parameters from **Training** data (Optimization)
Measure Performance on **Calibration** Data
Report performance on **Test** Data
ML model-1
ML model-2
ML model-3
ML model-m
Combiner
Biased Sampling: Favour "hard to classify" observations
Same ML Model Family;

Boosting

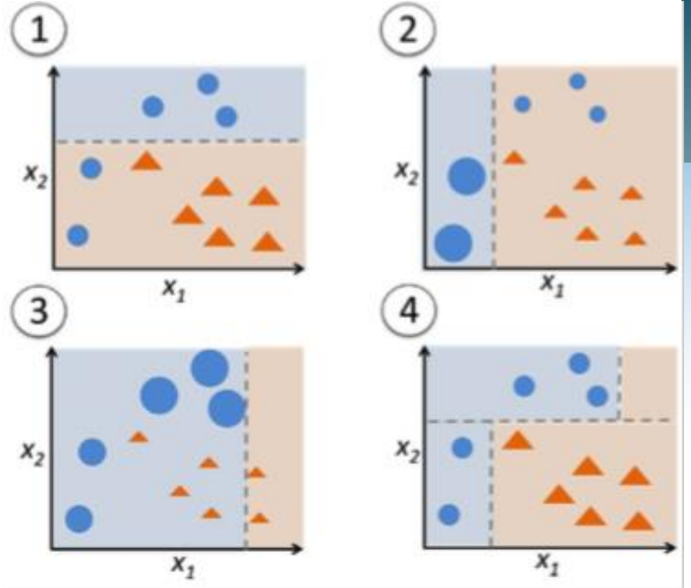
- A general method for improving the accuracy of any given learning algorithm
- Key Intuition
  - finding many rough rules of thumb can be easier than finding a single, highly accurate prediction rule
- Approach Outline
  - Start with a ML algorithm for finding the rough rules of thumb (a.k.a. “weak” or “base” algorithm)
  - Call the base algorithm repeatedly, each time feeding it a different subset of the training examples
    - (To be more precise, a different distribution or weighting over the training examples).
  - Each time it is called, the base learning algorithm generates a new weak prediction rule,
  - After many rounds, the boosting algorithm must combine these weak rules into a single prediction rule that, hopefully, will be much more accurate than any one of the weak rules
- Two key details
  - How should each distribution be chosen on each round?
  - How should the weak rules be combined into a single rule?

Boosting : Adaboost

- How should each distribution be chosen on each round?
  - place the most weight on examples most often misclassified by preceding weak rules;
  - ➔ force the base learner to focus its attention on the “hardest” examples.
- How should the weak rules be combined into a single rule?
  - take a (weighted) majority vote of their predictions is natural and effective
- Strategic resampling
  - Model-1 is trained with a random subset of the training data.
  - Model-2 is trained with the most informative subset, given Model-1
    - Model-2 is trained on a training data which gives higher preference to instances misclassified by Model-1.
  - Model-3 is trained with the most informative subset, given Model-1 & Model-2
  - The classifiers are combined through a three-way majority vote.
- Series Not Parallel
  - Train a series of classifiers
  - Each classifier focusses on (gives higher weightage to) instances that the previous ones got wrong
  - Then, use a weighted average of predictions


Initial uniform weight on training examples
weak classifier 1
Incorrect classifications re-weighted more heavily
weak classifier 2
weak classifier 3
Final classifier is weighted combination of weak classifiers

$$H(x) = sign(\alpha_1 h_1(x) + \alpha_2 h_2(x) + \alpha_3 h_3(x))$$



Adaboost: pseudocode

```
Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in X$, $y_i \in Y = \{-1, +1\}$
Initialize $D_1(i) = 1/m$.
For $t = 1, \ldots, T$:
    • Train base learner using distribution $D_t$.
    • Get base classifier $h_t : X \to \mathbb{R}$.
    • Choose $\alpha_t \in \mathbb{R}$.
    • Update:
          $D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$
      where $Z_t$ is a normalization factor (chosen so that $D_{t+1}$ will be a distribution).
Output the final classifier:
    $H(x) = \mathrm{sign}\left( \sum_{t=1}^{T} \alpha_t h_t(x) \right)$.
```

Schapire, Robert E. "The boosting approach to machine learning: An overview." *Nonlinear estimation and classification*. Springer New York, 2003. 149-171.

Boosting (cont'd)

Extending the design pattern of AdaBoost : exp loss etc.

Statistical Decision Theory: Summary $Y = f(X)$

f	(X)	$L(Y, f(X))$
• Constant • Linear • Non-Linear • Polynomial • Piecewise • Splines & Kinks • Additive	• Global • Local • Kernel • Basis Transformation • Expansion • Reduction • Learn (Dictionary) • Manifold	• Distance Measure • L2, L1, etc. • Hinge Loss • Overfitting • Regularization • Penalize roughness

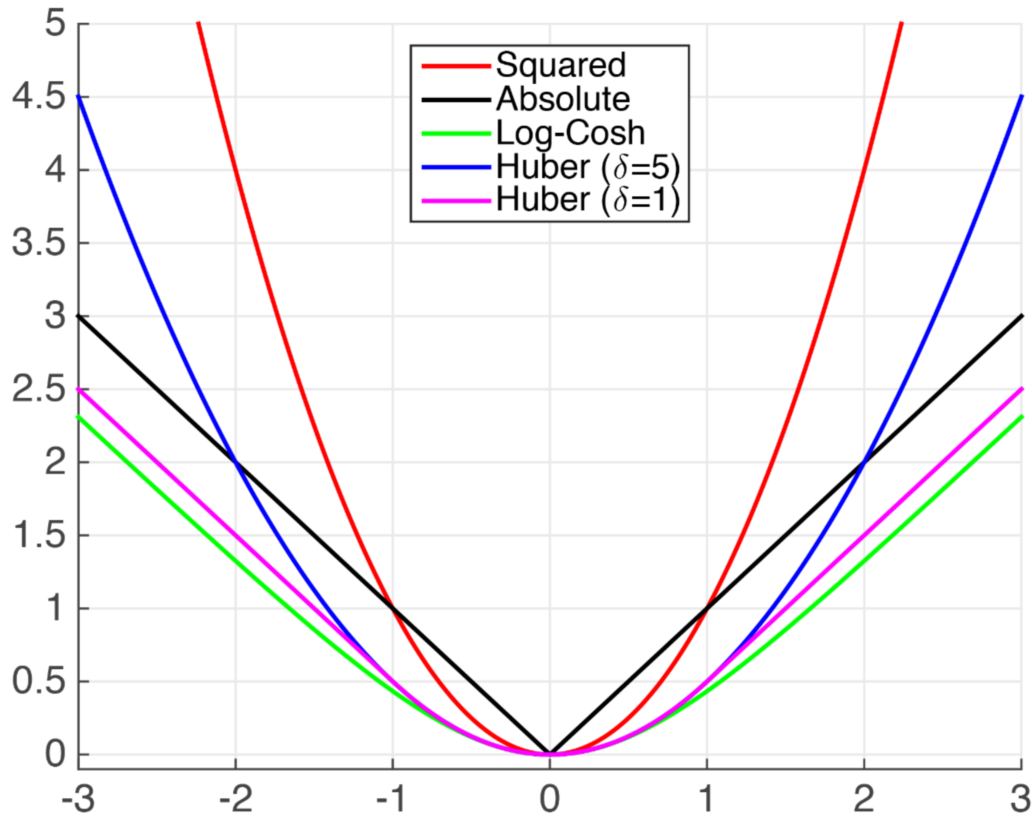
Loss functions for Regression

$$L_2(\Theta) = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{n}(y_i - f_\Theta(X_i))^2$$

$$L_1(\Theta) = \sum_{i=1}^{n}|y_i - \hat{y}_i| = \sum_{i=1}^{n}|y_i - f_\Theta(X_i)|$$

$$L_{huber}(\Theta) = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \quad \text{if} \quad |y_i - \hat{y}_i \leq \delta|$$

$$\sum_{i=1}^{n}\delta|y_i - \hat{y}_i| - \frac{\delta^2}{2} \quad \text{if} \quad |y_i - \hat{y}_i >$$



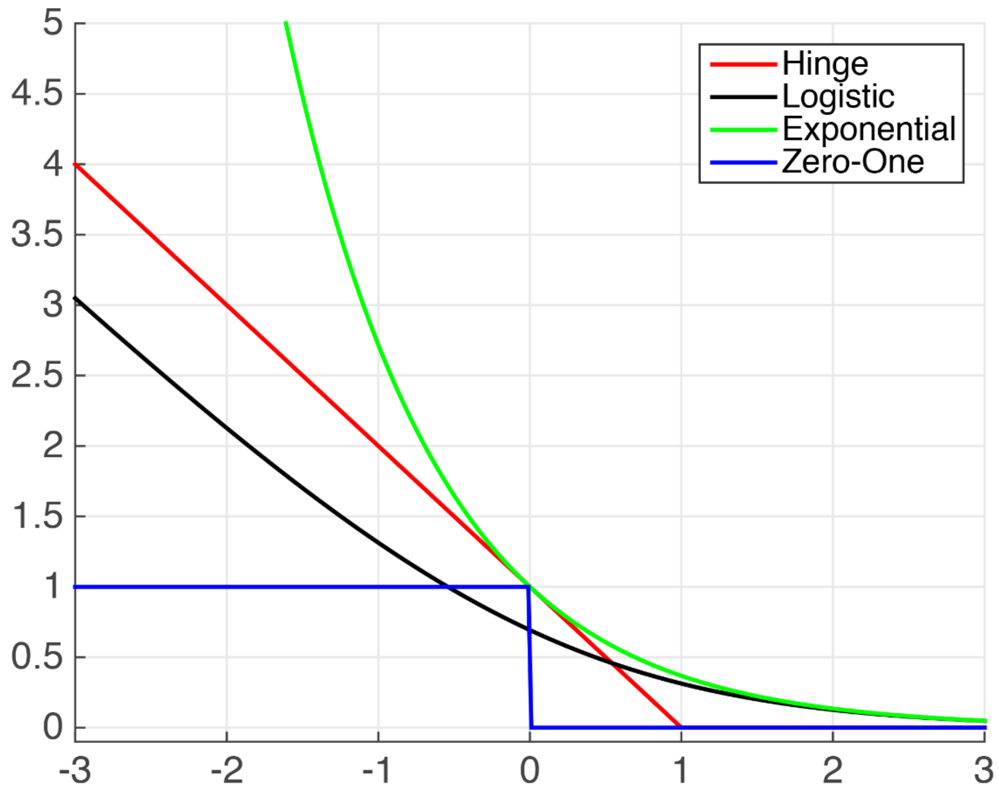
http://www.cs.cornell.edu/courses/cs4780/2015fa/web/lecturenotes/lecturenote10.html

Loss functions for Binary {-1,1} Classification

$$L_{logistic}(\Theta) = \frac{1}{\ln 2}\sum_{i=1}^{n} \ln(1+\exp(-y_i\hat{y}_i))$$

$$L_{Bernoulli}(\Theta) = \sum_{i=1}^{n} \ln(1+\exp(-2y_i\hat{y}_i)); \quad L_{AdaBoost}(\Theta) = \sum_{i=1}^{n} \exp(-y_i\hat{y}_i)$$

$$L_{cross-entropy}(\Theta) = -\frac{1}{2}\sum_{i=1}^{n}[(1+y_i)\ln(\hat{y}_i) + (1-y_i)\ln(1-\hat{y}_i)]$$



Gradient Boosting

- An iterative algorithm
  - Like Adaboost

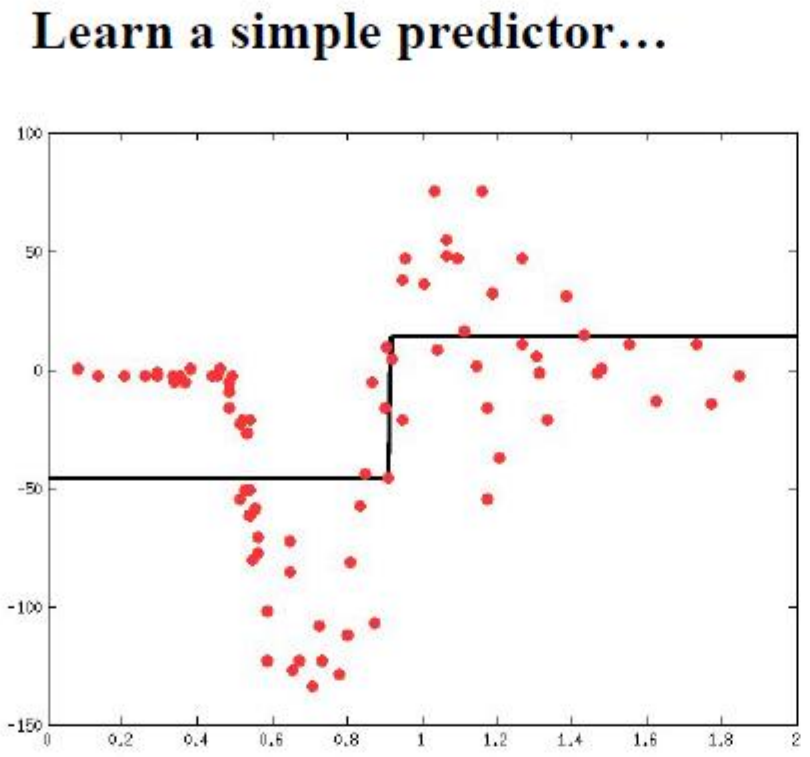
$$\begin{aligned} L(Y, f(X)) &= L(Y, \hat{Y}) = (Y - f(X))^2 \\ f^{(t+1)}(X) &= f^{(t)}(X) + h_t(X) \end{aligned}$$

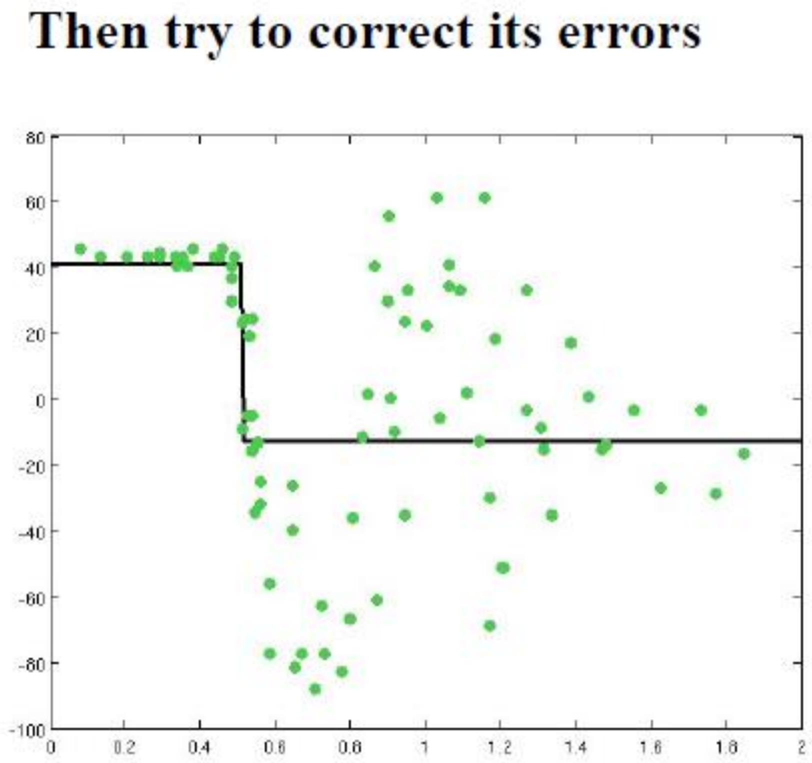
- At each iteration,
  - “fix” the error of the last iteration
  - Fit a function to the residuals of the last iteration (assume squared loss)
  - Learn to correct the predecessors

$$\begin{aligned} \text{Perfect } h \quad &\Rightarrow \quad f^{(t+1)}(X) = f^{(t)}(X) + h(X) = y \\ &\Rightarrow \quad h(X) = y - f^{(t)}(X) \end{aligned}$$

$$\begin{aligned} f^{(0)}(X) &= 0 \quad \text{or some constant} \\ f^{(1)}(X) &= f^{(0)}(X) + h_1(X) = h_1(X) \\ f^{(2)}(X) &= f^{(1)}(X) + h_2(X) = h_1(X) + h_2(X) \\ f^{(t)}(X) &= \sum_{k=1}^{t} h_k(X) \end{aligned}$$

Error residual

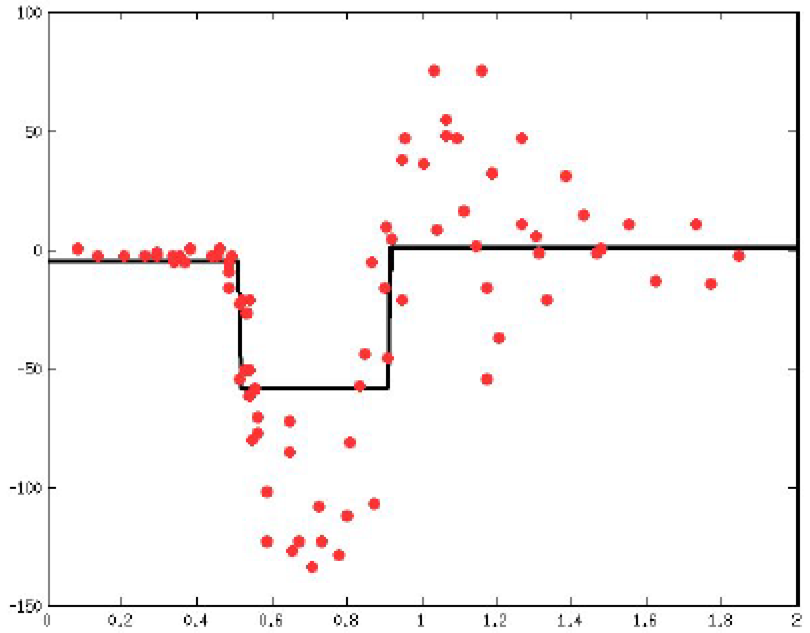




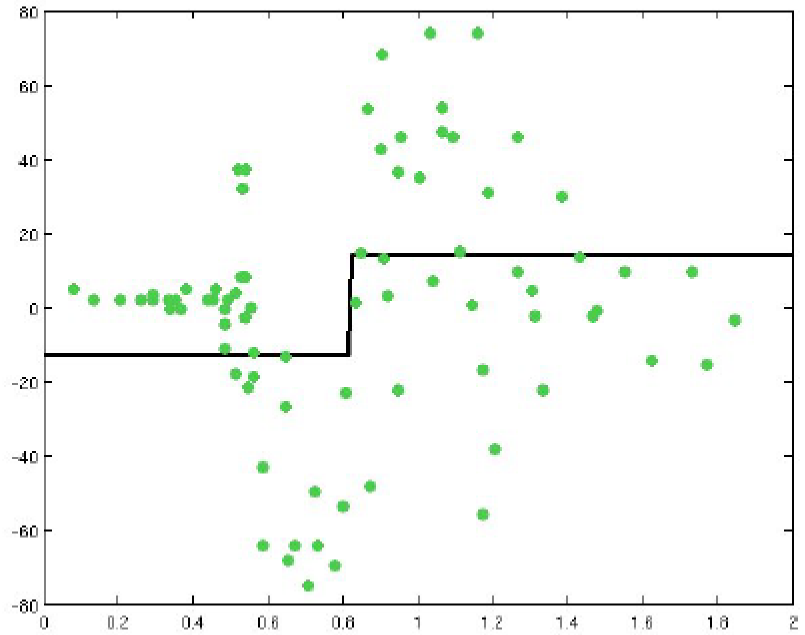
Excellent video: http://www.youtube.com/watch?v=sRktKszFmSk
Tutorial: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3885826/

Model gets complex with each addition

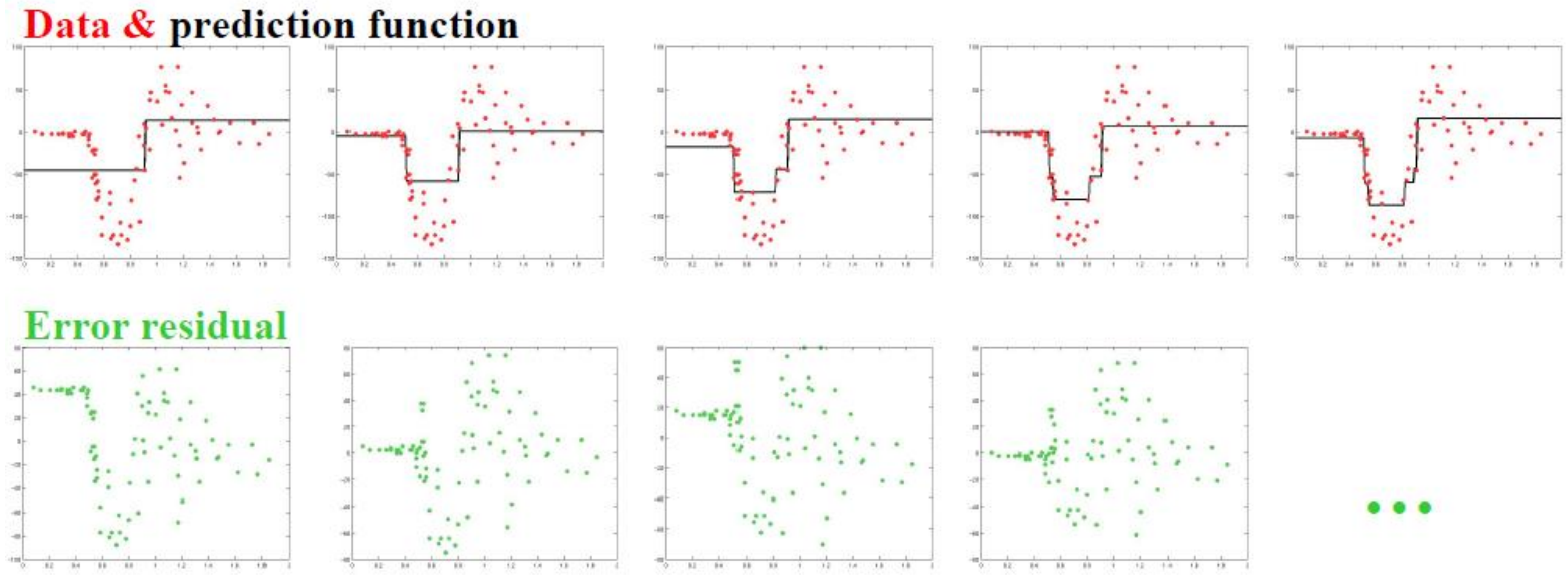
Combining gives a better predictor…



Can try to correct its errors also, & repeat



Model gets complex with each addition …



Base learners

- Linear models:
  - Ordinary linear regression
  - Ridge penalized linear regression
  - Random effects
- Smooth models:
  - P-splines
  - Radial basis functions
- Decision trees
  - Decision tree stumps
  - Decision trees with arbitrary interaction depth
- Other models:
  - Markov Random Fields
  - Wavelets
  - Custom base-learner functions

GBM Overview

Choice of the base-learner model $h(x, \theta)$ (regression, trees etc.)
Choice of the loss-function $\Psi(y, f)$ (least squares, logistic etc.)

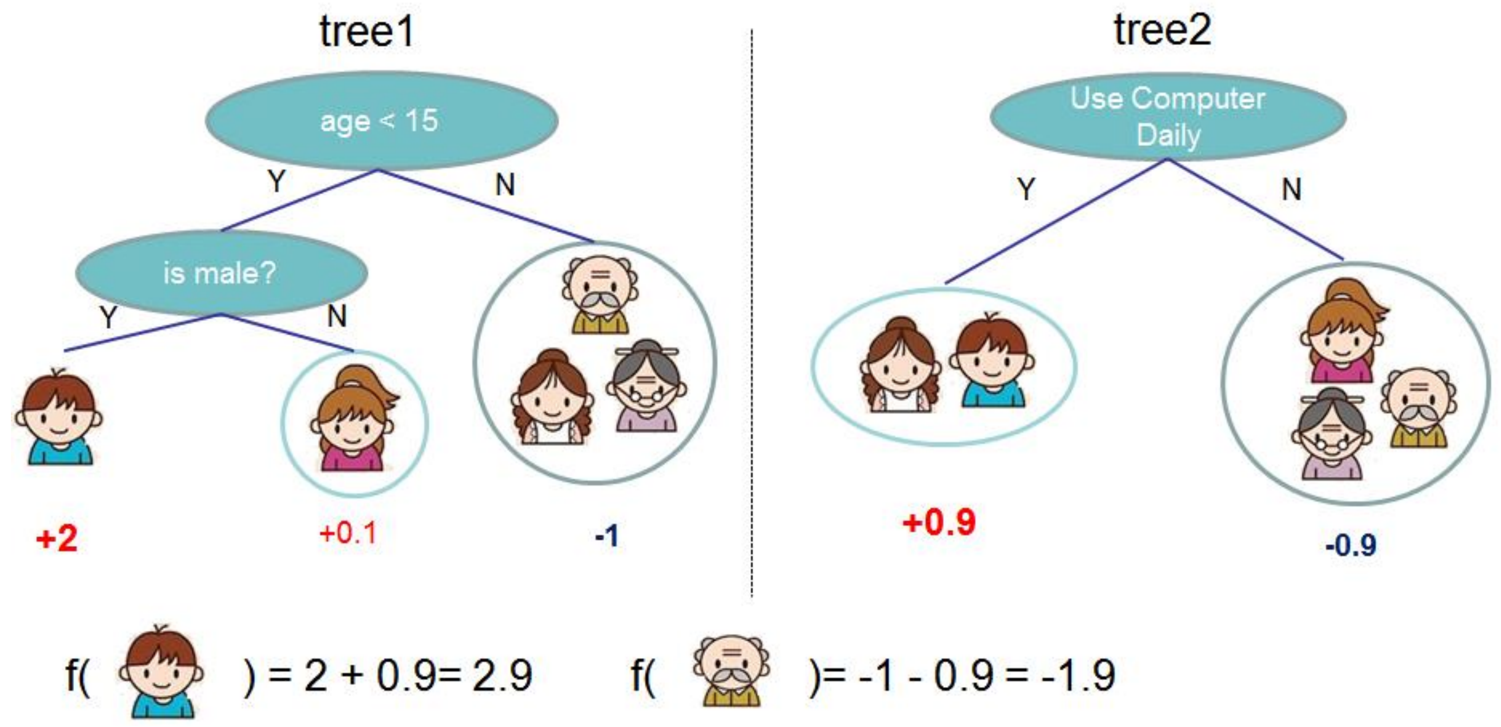
Algorithm: 1: initialize $\hat{f}0$ with a constant

```
for $t = 1$ to $M$ do
compute the negative gradient $g_t(x)$
fit a new base-learner function $h(x, \theta_t)$
find the best gradient descent step-size $\rho_t$:
```

$$\widehat{f}_t \leftarrow \widehat{f}_{t-1} + \rho_t h(x, \theta_t)$$

Boosted Trees : Gradient Boosting using Decision Trees as Base Learner

- Boosted Trees
  - Tree-1 is trained with a random subset of the available training data.
  - Tree-2 is trained on the residuals (instead of y) of one of the leaf nodes of Tree-1
  - Tree-3 is trained on the residuals (instead of y) of one of the leaf nodes of Tree-1
  - The three classifiers are combined through a three-way majority vote.
  - a.k.a. XGBoost



https://raw.githubusercontent.com/dmlc/web-data/master/xgboost/model/twocart.png

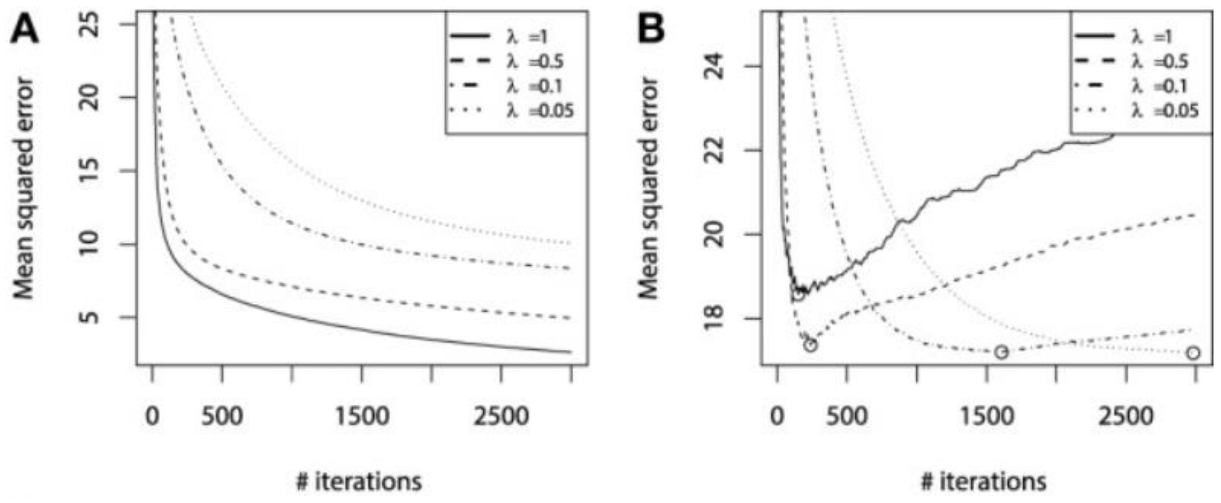
Boosted Trees : Gradient Boosting using Decision Trees as Base Learner

- Advantages
  - Dedicate models to harder samples.
  - But Not easy to parallelize

- Interpretability
  - Influence of a variable : number of times a variable is selected for splitting,
  - Captures weights of the influence with the empirical squared improvement I assigned to the model as a result of this split

Boosted Trees (cont'd)

- With each iteration,
  - Tree gets deeper ➔ Higher order interaction terms
  - No interaction -> y=f(x1,x2,x3…).
  - Binary interactions; y=f(x1x2,x2x3,…)
  - Ternary interactions -> y=f(x1x2x3…)
  - The depth of the tree is called interaction depth



(A) training set error; (B) validation set error.

- When to stop
  - Early stopping

- Modified objective function
  - Shrinkage parameter : Explicitly penalize depth : λ (sometimes θ_1)
  - Used for reducing, or shrinking, the impact of each additional fitted base-learner.
  - Penalizes the importance of each consecutive iteration.
  - !ntuition: better to improve a model by taking many small steps than by taking fewer large steps.

Boosted Trees (cont’d)

- Other methods to avoid overfitting: Sub-sampling
- At each learning iteration only a random part of the training data is used to fit a consecutive base-learner.
- The training data is typically sampled without replacement, however, replacement sampling, just as it is done in bootstrapping, is yet another possible design choice.
- The subsampling procedure requires a parameter called the “bag fraction.” Bag fraction is a positive value not greater than one, which specifies the ratio of the data to be used at each iteration. For example, bag = 0.1 corresponds to sampling and using only 10% of the data at each iteration.
- Another useful property of the subsampling is that it naturally adapts the GBM learning procedures to large datasets when there is no reason to use all the potentially enormous amounts of data at once.

Gradient Boost (in depth)

- Key Idea in ML
    - Minimize Prediction Error
    - Minimize (Expected) Prediction Error
    - Reason behind Training & Test data split
- Avoid Overfitting using regularization
    - Objective function for Optimization
    - Loss + Regularization
- Loss depends on
    - Choice of Model family (hyper-parameter)
    - Model parameters (parameters)
    - Loss and Regularization depend on parameter (given model family)
    - In boosting : loss at every iteration…
    - Objective function as a quadratic term and residual!

$$
\begin{aligned}
J(\Theta) &= L(\Theta) + \Omega(\Theta) \\
J^{(t)}(\Theta) &= \sum_{i=1}^{n} (y_i - \hat{y}_i^{(t)})^2 + \Omega(\Theta) \\
&= \sum_{i=1}^{n} (y_i - (\hat{y}_i^{(t-1)} + h_t(x_i)))^2 + \Omega(\Theta) \\
&= \sum_{i=1}^{n} y_i^2 + (\hat{y}_i^{(t-1)})^2 + h_t^2(x_i) + 2\hat{y}_i^{(t-1)} h_t(x_i) - 2y_i\hat{y}_i^{(t-1)} - 2y_i h_t(x_i) + \Omega(\\
&= \sum_{i=1}^{n} h_t^2(x_i) + 2\hat{y}_i^{(t-1)} h_t(x_i) - 2y_i h_t(x_i) + \Omega(\Theta) + \text{constant} \\
&= \sum_{i=1}^{n} 2h_t(x_i)(\hat{y}_i^{(t-1)} - y_i) + h_t^2(x_i) + \Omega(\Theta) + \text{constant}
\end{aligned}
$$

Gradient Boost (in depth)

- Gradient Boosting can support many loss functions
  - As long as it is differentiable
  - For non squared error loss, use the Taylor series expansion of the Loss function

$$J^{(t)}(\Theta) = \sum_{i=1}^{n} L(y_i, \hat{y}_i^{(t-1)}) + a_i h_t(x_i) + \frac{1}{2} b_i h_t^2(x_i) + \Omega(\Theta) + \text{constant}$$

$$a_i = \partial_{\hat{y}_i^{(t-1)}} L(y_i, \hat{y}_i^{(t-1)})$$

$$b_i = \partial^2_{\hat{y}_i^{(t-1)}} L(y_i, \hat{y}_i^{(t-1)})$$

$$J^{(t)}(\Theta) \quad = \quad \sum_{i=1}^{n} a_i h_t(x_i) + \frac{1}{2} b_i h_t^2(x_i) + \Omega(\Theta) + \text{constant2}$$

- Boosted Trees (in depth)
  - The key trick is that for a given tree structure, we push the statistics ai and bi for each element to the leaves they belong to, sum the statistics together and use the formula to calculate how good the tree is.
  - This score is like the impurity measure in DTs except that it also takes the complexity into account.

$$h(x_i) \quad = \quad \sum_{m=1}^{|T|} c_m \cdot 1_{(x_i \in R_m)}$$

$$\Omega(\Theta) \quad = \quad \theta_1 |T| + \frac{1}{2}\theta_2 \sum_{m=1}^{|T|} c_m^2$$

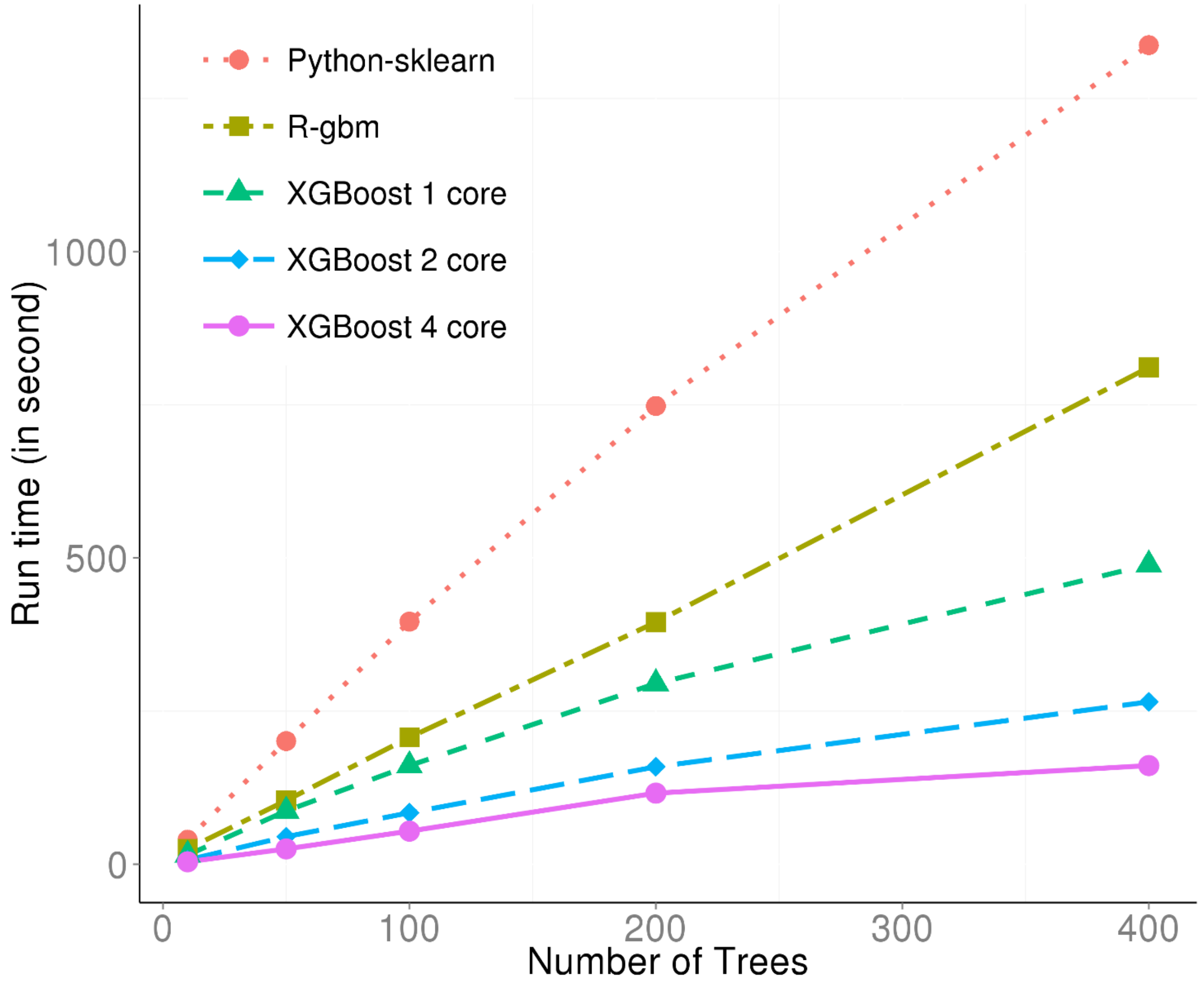
$$\begin{aligned} J^{(t)}(\Theta) &= \sum_{i=1}^{n} a_i c_m \cdot 1_{(x_i \in R_m)} + \frac{1}{2} b_i c_m^2 \cdot 1_{(x_i \in R_m)} + \theta_1 |T| + \frac{1}{2}\theta_2 \sum_{m=1}^{|T|} c_m^2 \\ &= \sum_{m=1}^{|T|} [(\sum_{i \in R_m} a_i) c_m + \frac{1}{2}(\sum_{i \in R_m} b_i + \theta_2) c_m^2] + \theta_1 |T| \\ &= \sum_{m=1}^{|T|} [A_m c_m + \frac{1}{2}(B_m + \theta_2) c_m^2] + \theta_1 |T| \end{aligned}$$

$$\Rightarrow \quad c_m^* = -\frac{A_m}{B_m + \theta_2}$$

$$J^{(t)}(\Theta) \quad = \quad -\frac{1}{2} \sum_{m=1}^{|T|} \frac{A_m^2}{B_m + \theta_2} + \theta_1 |T|$$

Packages for Gradient Boosting

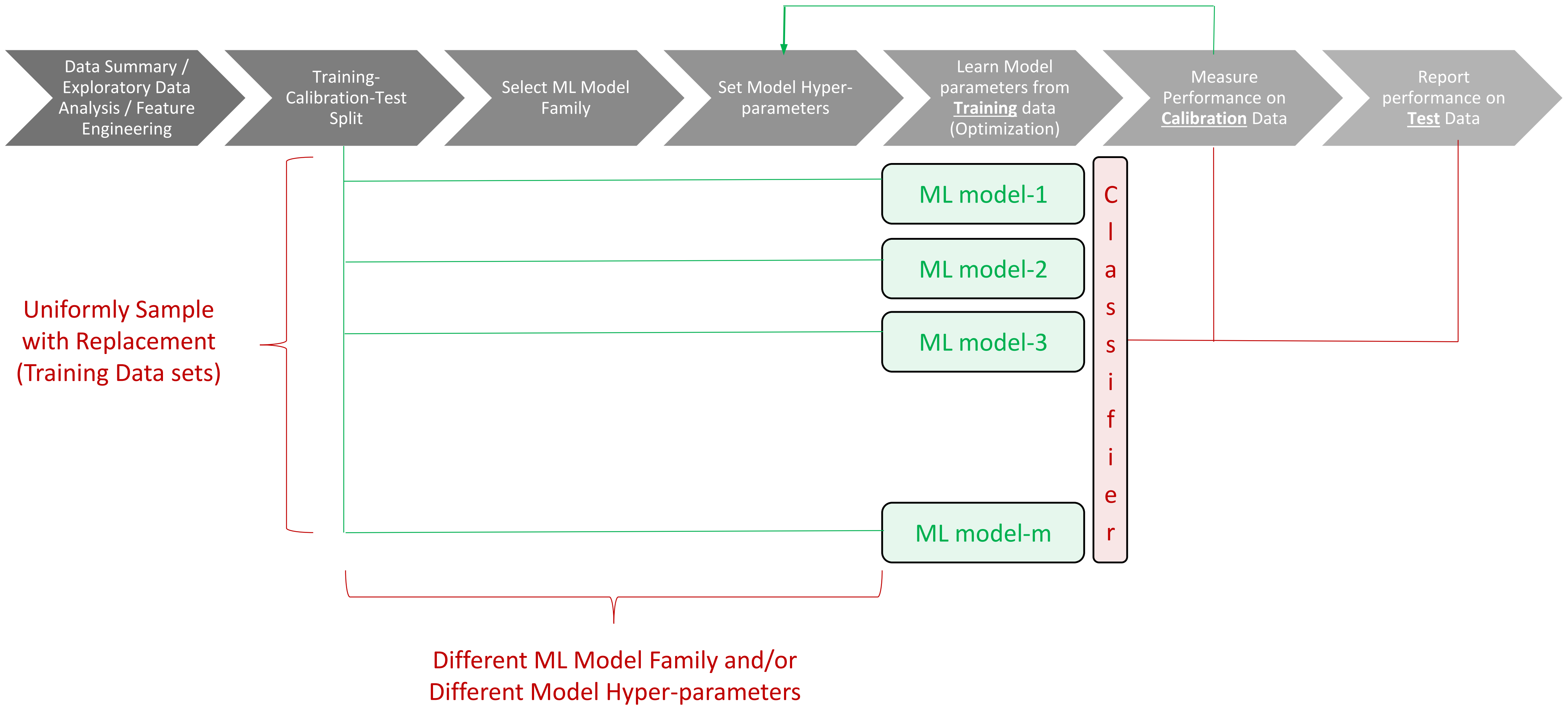
- XGBoost
  - Implemented in Python
  - Interface available in R
- GBM
  - Native R package



https://raw.githubusercontent.com/dmlc/web-data/master/xgboost/SpeedFigure.png

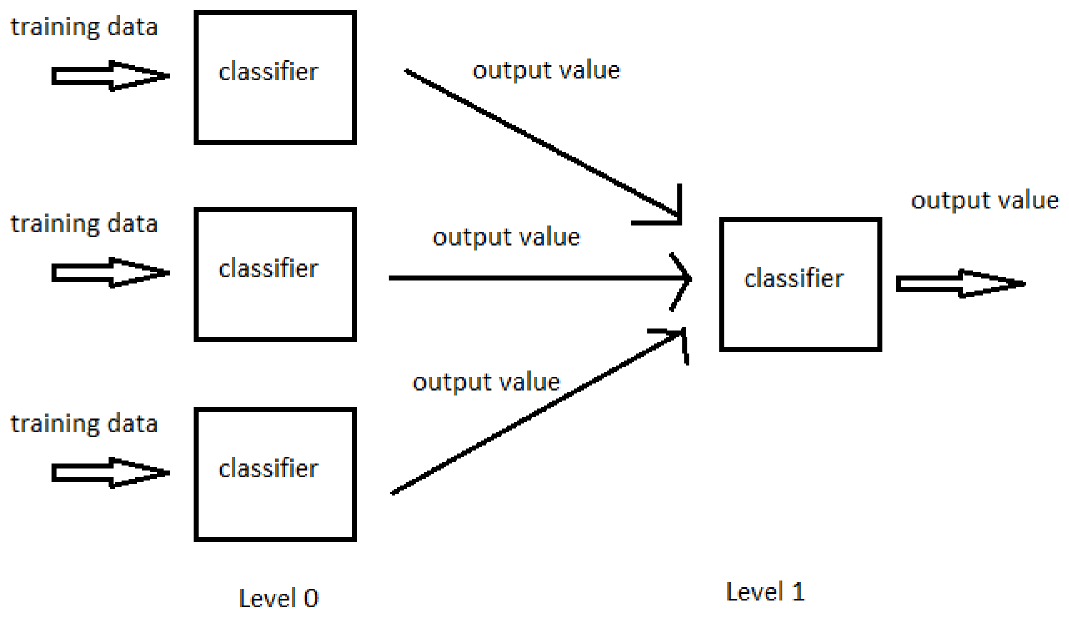
Stacking

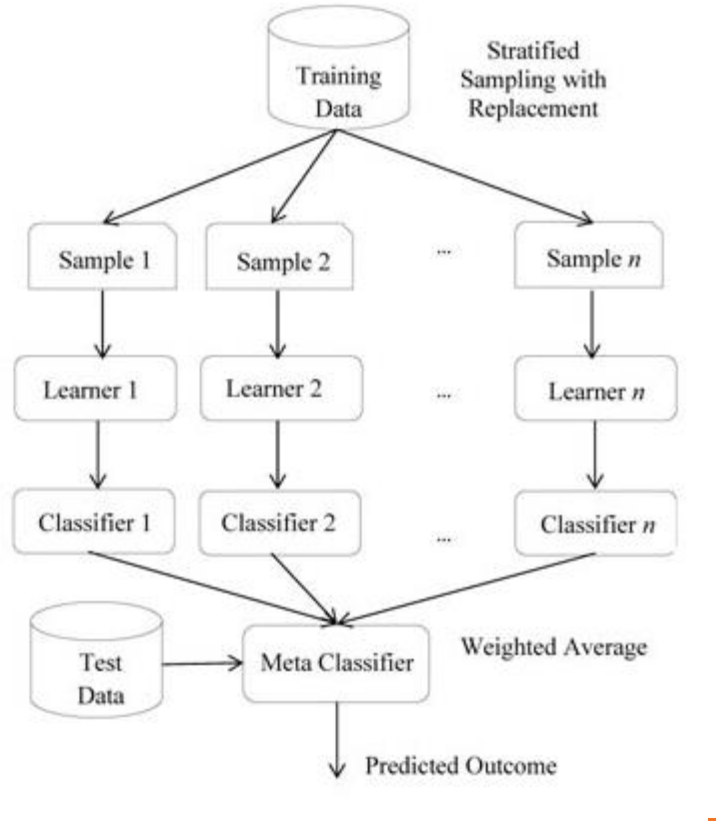
Stacking: ML Framework


Data Summary / Exploratory Data Analysis / Feature Engineering
Training-Calibration-Test Split
Select ML Model Family
Set Model Hyper-parameters
Learn Model parameters from **Training** data (Optimization)
Measure Performance on **Calibration** Data
Report performance on **Test** Data
Uniformly Sample with Replacement (Training Data sets)
ML model-1
ML model-2
ML model-3
ML model-m
Classifier
Different ML Model Family and/or Different Model Hyper-parameters

Stacking : Stacked Generalization : Blending

- Learn the combiner
  - Combiner: output of Level 0 classifiers will be used as training data for another classifier
  - Level 1 classifier "learns" the combining mechanism.
  - Combiner : A Meta-Classifier which takes as input the output of other classifiers to make a prediction





Summary

Summary

- ***"Why build 1 when you can have 2 at twice the price?"***
  - Improve model accuracy
  - Reduce variance (Bias-Variance tradeoff)

- Building "different" models
  - Bagging: Uniformly draw samples from training data with replacement
  - Random (Feature) Subspaces : Build models on subsets of feature on different subsets of training data
  - Boosting : A series of classifiers, each focusing on observations hard to classify by previous classifiers
  - Stacking: Learn the combination rule (instead of voting / mean)
  - Mixture of Experts : Input vector specific combination rule; Joint optimization of gates and experts

- Ensembling
  - Can be used with any model family / algorithms
  - Often leads to higher accuracy and / or lower variance

Backup