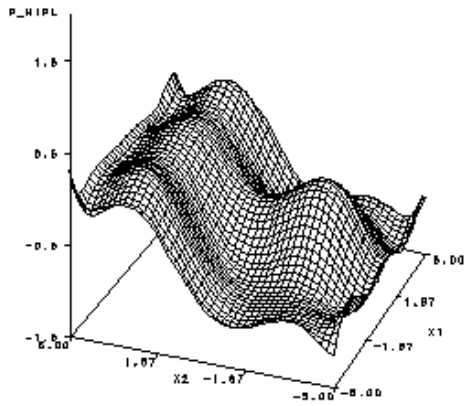
Machine Learning

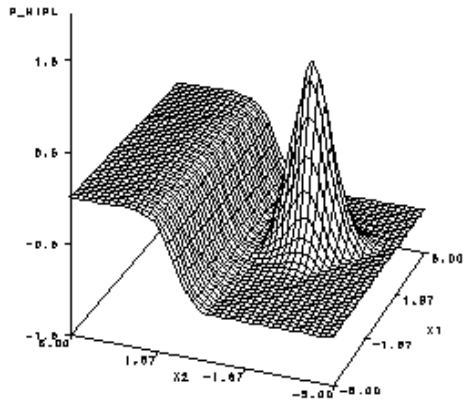
PATRICK HALL

DEPARTMENT OF DECISION SCIENCE

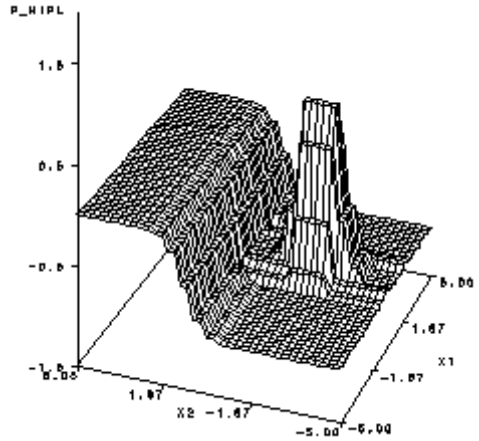
Decision Tree



Traditional regression



Neural network



Decision tree

Decision Tree: Pros & Cons

PROS	CONS
Accepted	Tendency to overfit
Understood	Many hyperparameters to tune
Interpretable	No parameters, standard errors or confidence limits
Few assumptions	Single decision trees can be unstable
Excellent for: discontinuous, nonlinear phenomena interactions; missing data; correlated variables variables on different scales	Usually poor performance in pattern recognition tasks vs. neural networks

Ensemble Models

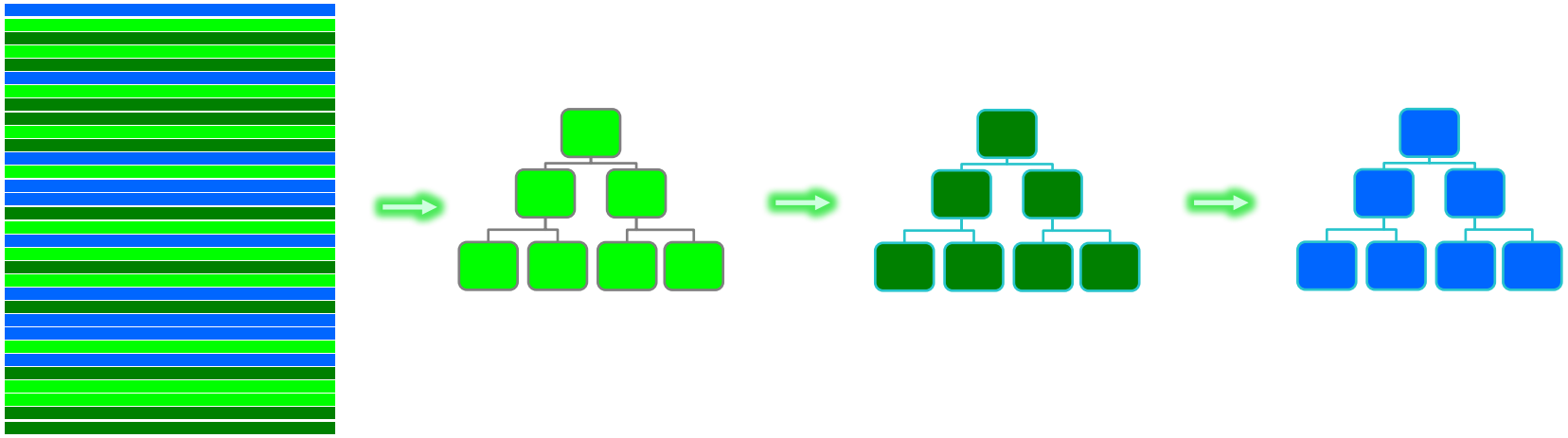
- Ensemble models combine the results of many other models, often called **base learners**
- Ensembles are often **more accurate than single models**
- There are several common approaches to ensembles:
  - Bootstrap aggregation (Bagging)
  - Boosting
  - Stacking (Super learner)

Ensemble Models: Intuition

- **Stability** - the predictions of ensemble models are stable w.r.t. minor perturbations of training data
- **Variable Hiding**– important variables are often correlated and can hide one-another (only the single most important variable from a group of important correlated variables will be used in many models); in different samples, many different important variables can shine through
- **Representative Samples** – some samples can be highly representative of new data

Decision Tree Boosting: GBM

Boosting is essentially a **sequential** process where each subsequent base learner attempts to improve on past results



(Sample with replacement)

Decision Tree Boosting

- Combine many “weak” classifiers (base learners) to produce a powerful “committee”
- Most commonly used boosting algorithm:
  - AdaBoost.M1 by Freund and Schapire (1997)
  - Algorithm – ELS pg. 339
- Method
  - Sequentially apply the weak classification algorithm to repeatedly modified versions of the data
  - Produce sequence of weak classifiers
  - Predictions are combined through a weighted majority vote to produce final prediction

Gradient Boosting Machine

- Function Estimation
- Numerical Optimization
- Commonly Used Loss Criteria
  - Least Square Regression
  - Least Absolute Deviation
  - Huber
  - Logistic binomial log-likelihood
- Regularization – learning rate (v) and number of components (M)
  - Natural regularization parameter: number of components M (analogous to “stepwise” regression)
  - Shrinkage strategy

GBM: Conceptual Framework

- Output variable y and vector input variable $x = \{x_1, ..., x_n\}$
- Training set $\{(y_i, x_i)\}_1^n$ of known values of x and corresponding value of y
- Find approximation $\hat{F}(x)$ to a function $F^*(x)$ that minimizes the expected value of some specified loss function $L(y, f(x))$

$$\hat{F} = arg\min_F E_{y,x} L(y, F(x)) = arg\min_F E_x[E_y(L(y, F(x)))|x]$$

- Loss function $L(y, F)$
  - Squared error: $(y - F)^2$
  - Absolute error: $|y - F|$
  - Negative binomial log-likelihood:$\log(1 + e^{-2yF})$ when y $\in \{-1, 1\}$

GBM: Conceptual Framework

- The gradient boosting method seeks an approximation $\hat{F}(x)$ in the form of an weighted sum of the base learners
- Restrict $F(x)$ to be a member of a parameterized class of functions $F(x; P)$
- The "additive" expansions form

$$F(x; \{\beta_m, \gamma_m\}_1^M) = \sum_{m=1}^{M} \beta_m b(x; \gamma_m)$$

- Models are fit by minimizing a loss function averaged over the training data - squared-error or a likelihood-based loss function

$$\min_{\{\beta_m, \gamma_m\}_1^M} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M} \beta_m b(x_i; \gamma_m)\right)$$

GBM: Conceptual Framework

- Recall, regression and classification trees partition the space of all joint predictor variable values into disjoint regions $R_j, j = 1, 2, ..., J$
- A constant γ_j is assigned to each region so that the predictive rule is

$$x \in R_j \Rightarrow f(x) = \gamma_j$$

- Thus a tree can be expressed as

$$T(x; \Theta) = \sum_{j=1}^{J} \gamma_j I(x \in R_j) \quad \text{with parameters} \quad \Theta = \{R_j, \gamma_j\}_1^J$$

GBM: Conceptual Framework

- Parameters can be found by minimizing the empirical risk

$$\hat{\Theta} = arg\min_{\Theta} \sum_{j=1}^{J} \sum_{x_i \in R_j} L(y_i, \gamma_j)$$

- Divide the optimization problem into two parts:
  - Find γ_j given Rj
  - Find R_j
- The boosted tree model is a sum of such trees

$$f_M(x) = \sum_{m=1}^{M} T(x; \Theta_m)$$

GBM: Conceptual Framework

- At each step in the forward stagewise procedure, solve

$$\hat{\Theta_m} = arg\min_{\Theta_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i; \Theta_m)) \quad \text{for the region set and constant}$$

$$\Theta_m = \{R_{jm}, \gamma_{jm}\}_1^{J_m} \quad \text{of the next tree, given the current model} \quad f_{m-1}(x)$$

- Given the regions R_{jm}, finding the optimal constants γ_{jm} in each region is typically straightforward:

$$\hat{\gamma_{jm}} = arg\min_{\gamma jm} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma_{jm})$$

GBM: Conceptual Framework

- For squared-error: regression tree that best predicts the current residual $y_i - f_{m-1}(x_i)$ where $\hat{\gamma_{jm}}$ is the mean of these residuals in each corresponding region
- Two-class classification and exponential loss
  - Minimize the weighted error rate $\sum_{i=1}^{N} w_i^{(m)} I(y_i \neq T(x_i; \Theta_m))$ with weight $w_i^{(m)} = e^{-y_i f_{m-1}(x_i)}$
  - scaled classification: $\beta_m T(x; \Theta_m)$ with the restriction that $\gamma_{jm} \in \{-1, 1\}$

GBM: Conceptual Framework

- Without the restriction - simplifies for exponential loss to a weighted exponential criterion for the new tree:

$$\hat{\Theta}_m = arg \min_{\Theta_m} \sum_{i=1}^{N} w_i^{(m)} e^{-y_i T(x_i;\Theta_m)}$$

- Apply a greedy recursive-partitioning algorithm using the above weighted exponential loss as a splitting criterion as log-odds in each corresponding region

$$\hat{\gamma}_{jm} = \frac{1}{2} \log \frac{\sum_{x_i \in R_{jm}} w_i^{(m)} I(y_i=1)}{\sum_{x_i \in R_{jm}} w_i^{(m)} I(y_i=-1)}$$

GBM: Conceptual Framework

- Numerical Optimization via Gradient Boosting
- The loss in using $f(x)$ to predict y on the training data is

$$L(f) = \sum_{i=1}^{N} L(y_i, f(x_i))$$

- To minimize $L(f)$ with respect to f can be viewed as a numerical optimization

$$\hat{f} = arg\min_{f} L(f)$$

- Solve above optimization as a sum of component vectors

$f_m = \sum_{m=0}^{M} h_m, h_m \in R^N, where f_0 = h_0$ is an initial guess and each successive f_m is induced

based on the current parameter vector f_{m-1}, sum of the previously induced updates

GBM: Conceptual Framework

- Steepest Descent chooses $h_m = -\rho_m g_m$ where ρ_m is scalar and $g_m \in R^N$ is the gradient of $L(f)$ evaluated at $f = f_{m-1}$; and the components of the gradient g_m are

$$g_{im} = [\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}]_{f(x_i)=f_{m-1}(x_i)}$$

The step length ρ_m is the solution to

$$\rho_m = arg \min_{\rho} L(f_m - \rho g_m)$$

The current solution is then updated

$$f_m = f_{m-1} - \rho_m g_m$$

and the process repeated at the next iteration. Steepest descent can be viewed as a very greedy strategy since $-\rho_m$ is the local direction in R^N for which $L(f)$ is the most rapid decreasing at $f = f_{m-1}$

GBM: Conceptual Framework

- Forward stagewise boosting is also a very greedy strategy. At each step the solution tree is the one that maximally reduces current model f_{m-1} and its fits $f_{m-1}(x_i)$. Note, the gradient of the steepest descent is defined only at the training data points whereas the ultimate goal is to generalize $f_M(x)$ to new data not represented in the training set. For Squared error:

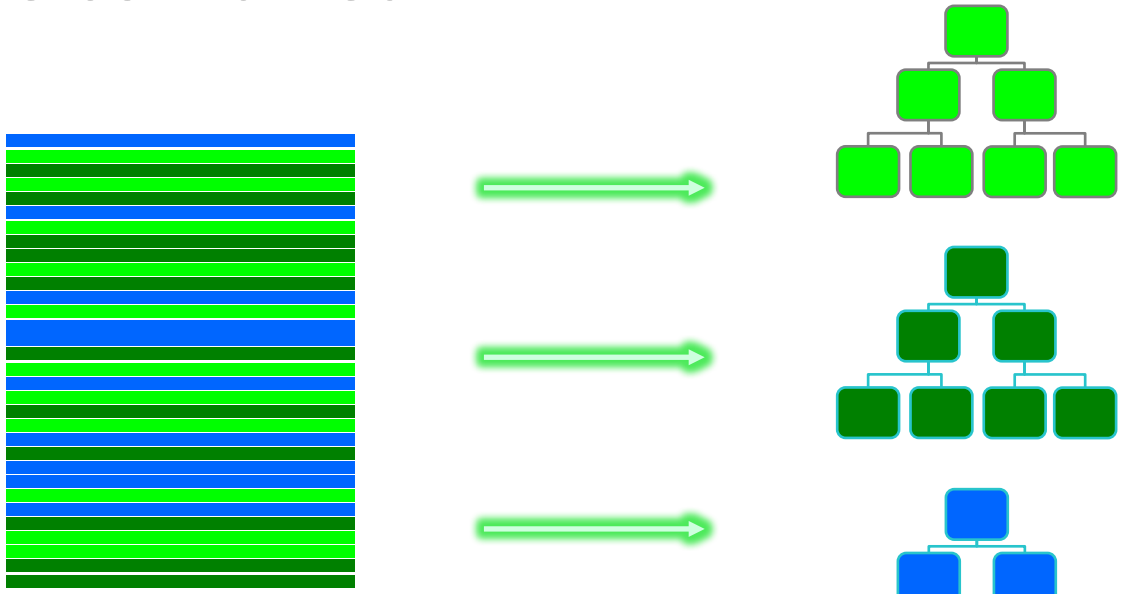
$$\tilde{\Theta}_m = arg \min_{\Theta} \sum_{i=1}^{N} (-g_{im} - T(x_i; \Theta))^2$$

For classification the loss function is the multinomial deviance. Each tree T_{km} is fit to its respective negative graident vector g_{km}:

$$-\rho_{ikm} = \frac{\partial L(y_i, f_{1m}(x_i), ..., f_{1m}(x_i))}{\partial f_{km}(x_i)} = I(y_i = G_k) - p_k(x_i)$$

Decision Tree Bagging: Random Forest

Bagging is essentially a **parallel** process where the results of base learners are combined



(Sample with replacement)

Decision Tree Bagging: Random Forest

- **Bagging or bootstrap aggregation**: technique for reducing the variance of an estimated prediction function
- Works well for high-variance and low-biased procedures like trees
- **Random Forest** essentially builds a large collection of **de-correlated trees** and then **averages** them
- Very similar to boosting yet **simpler to train and tune** and hence very popular and implemented in variety of packages

Decision Tree Bagging

- Essential idea in bagging is to average many noisy but approximately unbiased model and hence reduce the variance
- Since trees are noisy and each tree generated in bagging is identically distributed, the expectation of an average of B such trees is the same as the expectation of any of them This means that the bias of a bagged trees is the same as that of the individual trees with objective to reduce variance
- An average of B independent and identically distributed, IID, random variables each with variance of σ^2 has a variance of $\frac{1}{B}\sigma^2$. If the variables are simply ID with positive pairwise correlation ρ, the variance of the average is

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$$

Random Forest

- As B increases, the second term approaches 0, but the first term remains. Hence, the size of the correlation of pairs of bagged trees limits the benefits of averaging. The idea in random forest is to improve the variance reduction of bagging by reducing the correlation between the trees without increasing the variance too much via tree-growing process through random selection of the input variables.
- When growing a tree on a bootstrapped dataset, select $m \leq p$ of the input variables at random as candidates for splitting. Typical values for m are $\sqrt{p}$ or even as low as 1.
- After B such trees $\{T(x_i; \Theta_b)\}_1^B$ are grown, the random forest predictor is

$$\hat{f}_{rf}^B = \frac{1}{B} \sum_{b=1}^{B} T(x_i; \Theta_b)$$

Random Forest

Details of Random Forest

- For classification - "class votes" from each tree and then classifies using majority vote
  - Default value for m is $\lfloor \sqrt{p} \rfloor$ and the minimum node size is one
- For regression - prediction from each tree at a target point x are averaged
  - Default value for m is $\lfloor \frac{p}{3} \rfloor$ and the minimum node size is five

Out of Bag Samples

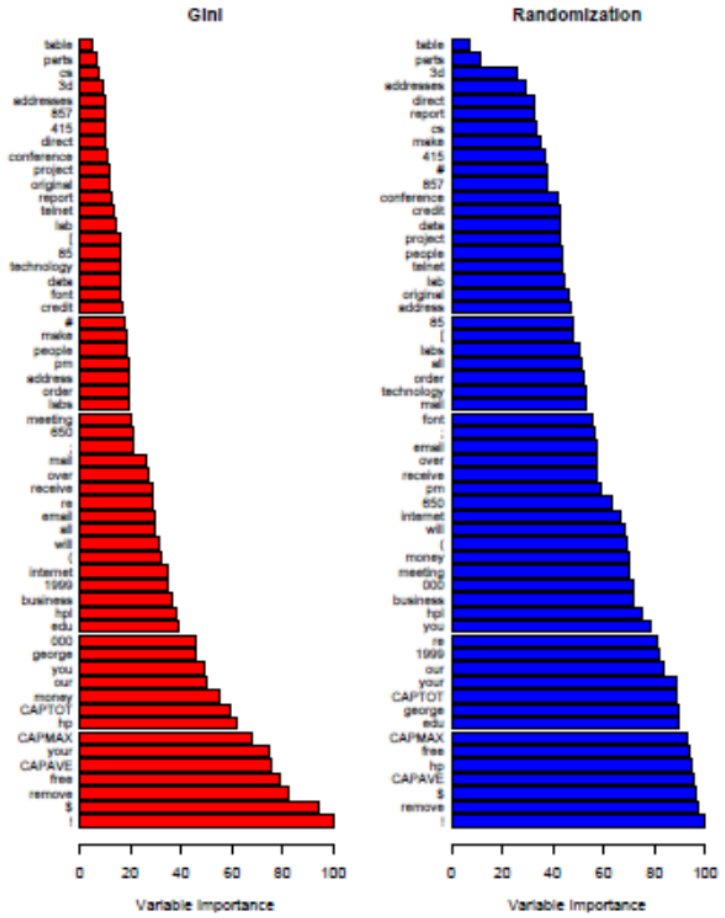
- For each observation $z_i = (x_i, y_i)$, construct its random forest predictor by averaging only those trees corresponding to bootstrap samples in which z_i does not appear

Random Forest

Variable Importance

- At each split in each tree, the improvements in the split-criterion is the importance measure attributed to the splitting variable and accumulated over all the trees in the forest separately for each variable
- Random Forest also use the OOB sample to construct a different variable-importance measure - measure the prediction strength of each variable. The decrease in accuracy as a result of this permuting is averaged over all trees.

Random Forest



Elements of Statistical Learning (pg.594) – Rankings of the two methods are similar but importances in the randomization plot are more uniform over the variables because randomization effectively voids the effect of a variable.

FIGURE 15.5. *Variable importance plots for a classification random forest grown on the* spam *data. The left plot bases the importance on the Gini splitting index, as in gradient boosting. The rankings compare well with the rankings produced by gradient boosting (Figure 10.6 on page 354). The right plot uses* OOB *randomization to compute variable importances, and tends to spread the importances more uniformly.*

Random Forest

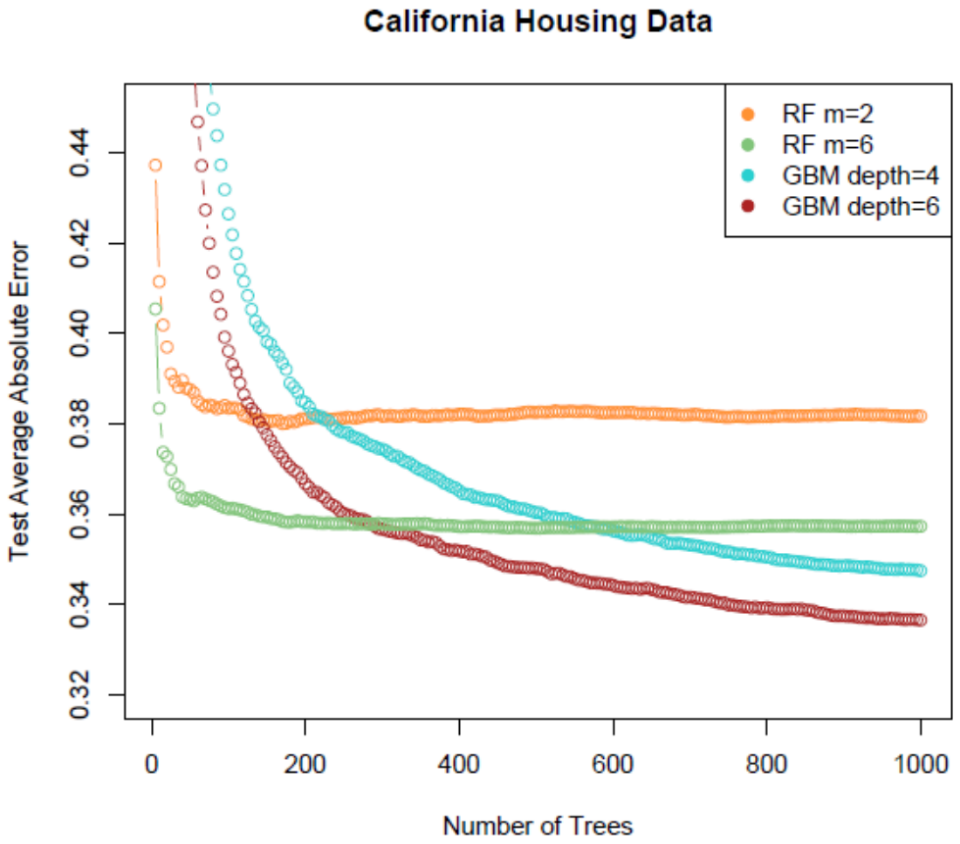
Overfitting Random Forest

- One claim is that random forest "cannot overfit" the data. While it is certainly true that increasing B does not cause the random forest sequence to overfit; random forest estimate approximates the expectation:

$$\hat{f}_{rf}(x) = E_{\Theta} T(x; \Theta) = \lim_{B \to \infty} \hat{f}(x)_{rf}^{B}$$

with an average over B realization of Θ. The limit can overfit the data - the average of fully grown trees can result in too rich a model and incur unnecessary variance.

Boosting vs. Bootstrapping



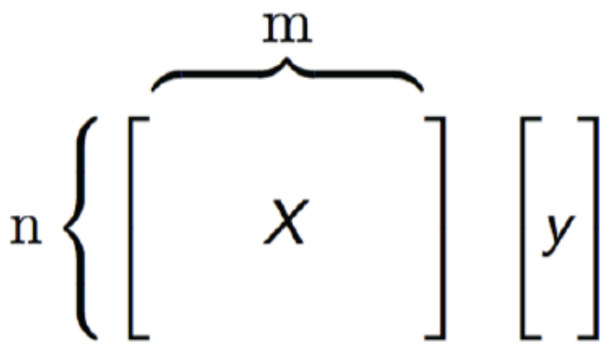
Elements of Statistical Learning (pg.591) – Comparison between random forest and boosting (with shrinkage) in a regression problem using the California housing data. Random forests stabilizes at about 200 trees, while 1000 trees boosting continues to improve.

FIGURE 15.3. *Random forests compared to gradient boosting on the California housing data. The curves represent mean absolute error on the test data as a function of the number of trees in the models. Two random forests are shown, with* $m = 2$ *and* $m = 6$*. The two gradient boosted models use a shrinkage parameter* $\nu = 0.05$ *in (10.41), and have interaction depths of* 4 *and* 6*. The boosted models outperform random forests.*

Stacking

- Use to ensemble a diverse group of **strong learners**
- Involves training a **second-level machine learning** algorithm called a "**meta-learner**" to learn the optimal combination of the base learners

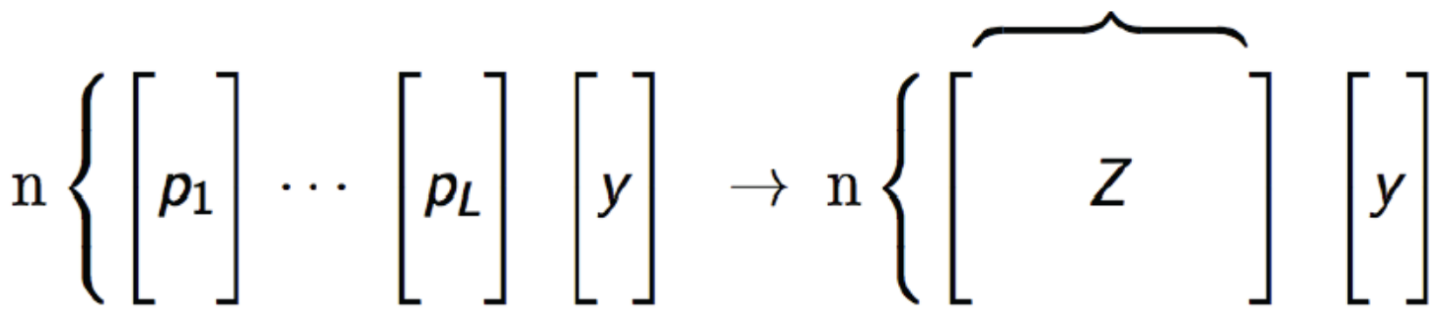
The Super Learner Algorithm – H2O.ai



"Level-zero" data

- Start with design matrix **X** and response matrix **Y**
- Specify **L base learners** with model parameters
- Specify a **metalearners**
- Perform **k-fold CV** on each of the **L learners**

The Super Learner Algorithm – H2O.ai



"Level-one" data

- Collect the prediction values from **k-fold CV** that was performed on each of the **L base learners**
- Column-bind these predictions vectors together to form a new **design matrix, Z**
- Train the metalearner using **(Z,y)**

The Super Learner Algorithm – H2O.ai

- The first phase of the Super Learner algorithm is computationally equivalent to the performing model selection via **cross-validation**
- The **metalearning** is just training another single model
- With Super Learner, no computational waste

Other blending strategies

- **Simple averaging** (for a categorical response, this is equivalent to simple soft voting, described by Zhou 2012) takes the average of the posterior probability for each response level across the models and then classifies the models based on the level that has the maximum average probability.
- **Top-t** ensemble selection (Lessmann et al. 2015), for various values of t (such as 5, 10, and 25), takes the top t models out of the M that are generated when the models are ranked by an accuracy measure and uses validation data to determine the best value for t. The code for this paper evaluated at all values of t: 1, …, M. You can think about this selection method as an alternative way to assess multiple combinations of your candidate models by adding one model at a time to a simple averaging ensemble.
- **Hill-climbing** ensemble selection (Caruana et al. 2004; Lessmann et al. 2015), like top-t ensemble selection, starts by ranking the models by an accuracy measure. The implementation in this paper calculates the improvement in accuracy of adding any given model, and includes in the ensemble the posterior probabilities of the model that most improves the misclassification rate in the validation set. The final ensemble is selected based on the misclassification rate in the test set.
- **Weighted averaging** (weighted soft voting; Zhou 2012) calculates a weighted average of the posterior probabilities for each response level, with a model-specific weight applied. Both top-t ensemble selection and hill-climbing ensemble selection can be considered weighted averaging techniques; top-t selects a subset of the available models and assigns an equal weight to them, whereas hill-climbing has a different weight for each model depending on how many times a particular model is included in the ensemble.

https://support.sas.com/resources/papers/proceedings16/SAS3120-2016.pdf