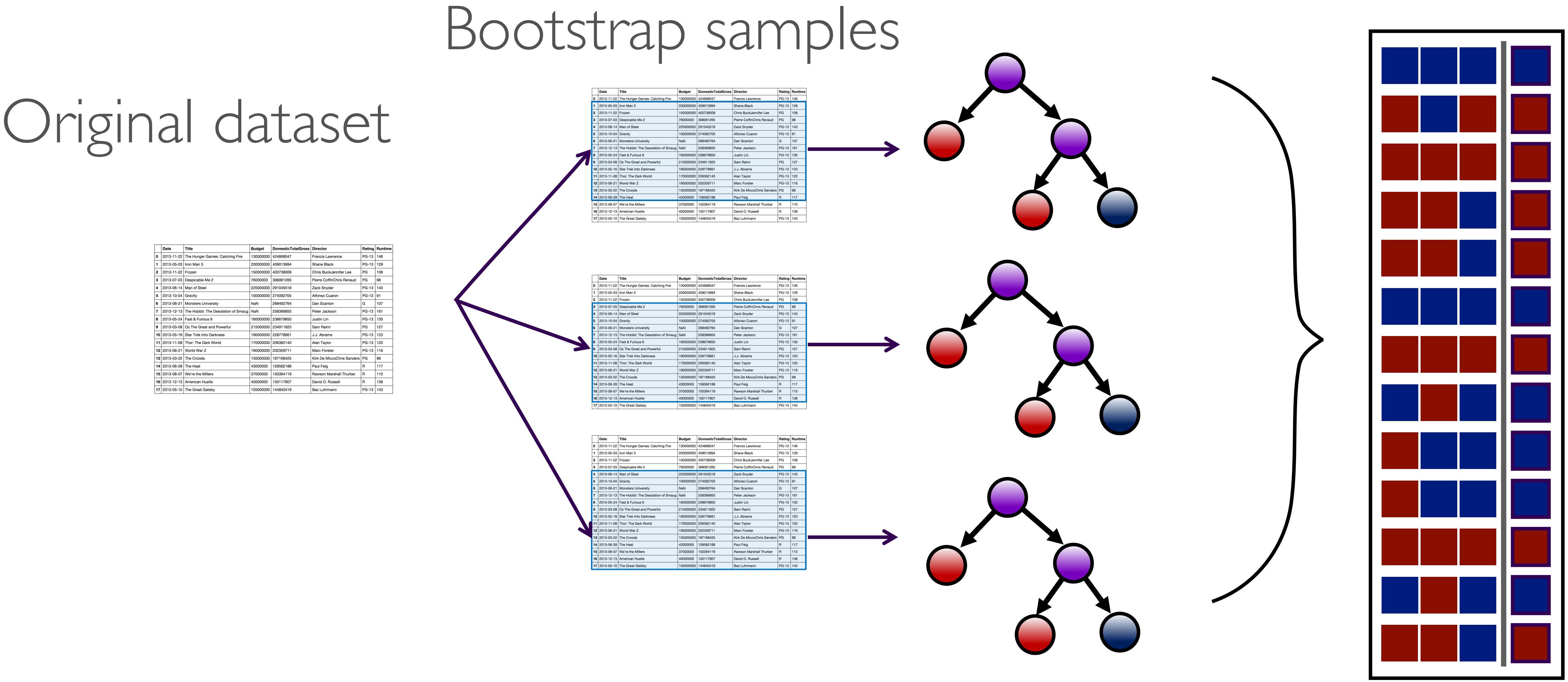
ENSEMBLE METHODS

- Ensemble of same classifiers
  - Bagging and random forest
  - Boosting and gradient boosted tree
- Ensemble of different classifiers

REVIEW: BAGGING



REVIEW: RANDOM FORESTS

- Bagged classifier using decision trees
  - Each split only considers a random group of features
  - Tree is grown to maximum size without pruning
  - Final predictions obtained by aggregating over the B trees

$$\hat{f}_{\mathrm{rf}}^{B}(\mathbf{x}) = \frac{1}{B} \sum_{b} T(\mathbf{x}; \theta_b)$$

- Reduce variance (at the cost of slight increase in bias compared to bagged trees)

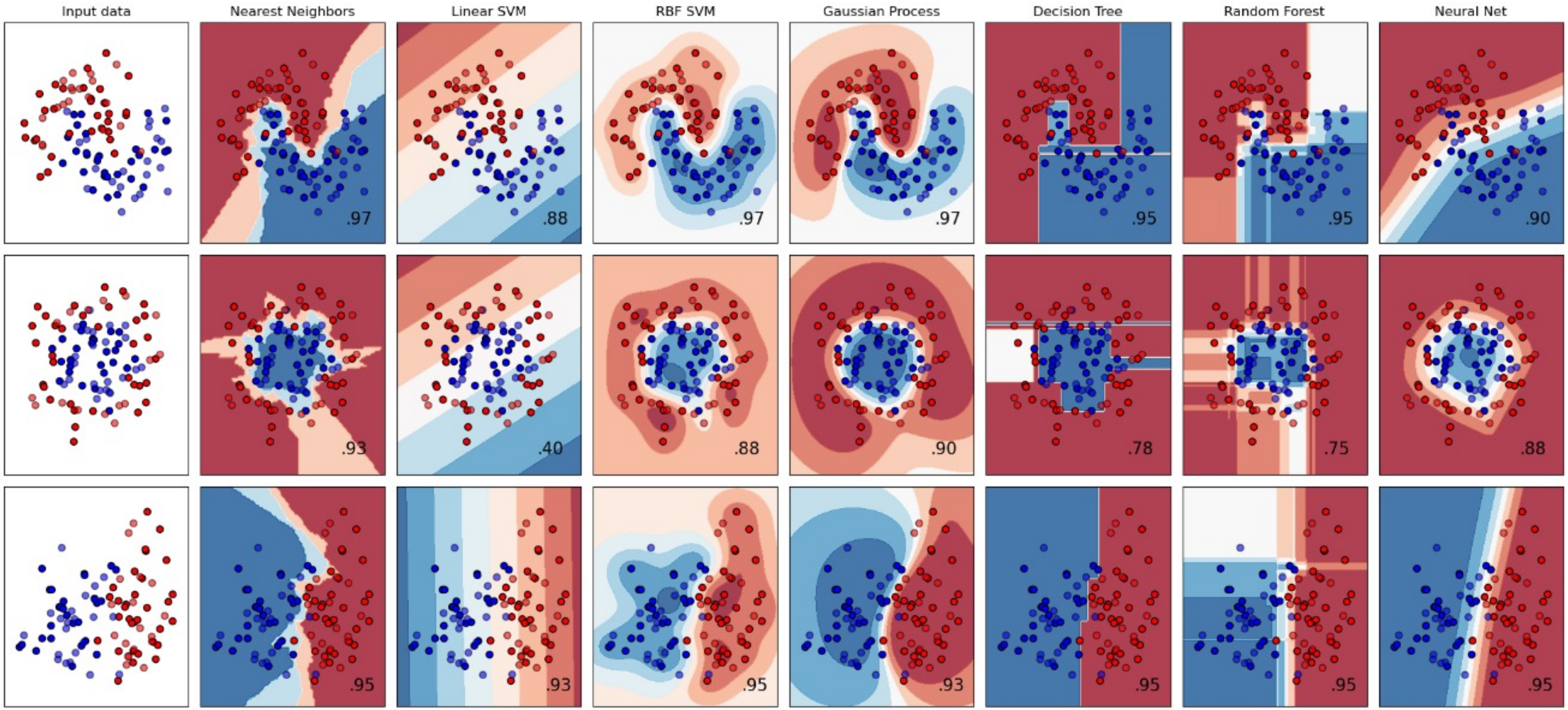
RANDOM FOREST: ADVANTAGES

- State of the art method, one of the most accurate general-purpose learners available
- Handles a large number of input variables without overfitting (variance reduction)
- Robust to errors and outliers
- Can model non-linear boundaries
- Gives variable importance and out of bag error rates
- Easy to train and tune, easily parallelized by training

RANDOM FOREST: DISADVANTAGES

- Loss of interpretability (no decision rules)
- Difficult to analyze as an algorithm and mathematical properties still largely unknown
- Large number of trees is memory-intensive
- Bias towards categorical variables with larger number of levels

RANDOM FOREST



https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html#sphx-glr-auto-examples-classification-plot-classifier-comparison-py

PREVIEW: HOMEWORK #5

- Almost Random Forest
- Instead of choosing a random subset of features for each split, choose a random subset of features that the tree will be created on (the same subset is used as candidates from all splits)

SKLEARN: RANDOM FOREST

- sklearn.ensemble.RandomForestClassifier
  - n_estimators, default=100
  - max_features: {“sqrt”, “log2”, None}, default=”sqrt”

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

ENSEMBLE

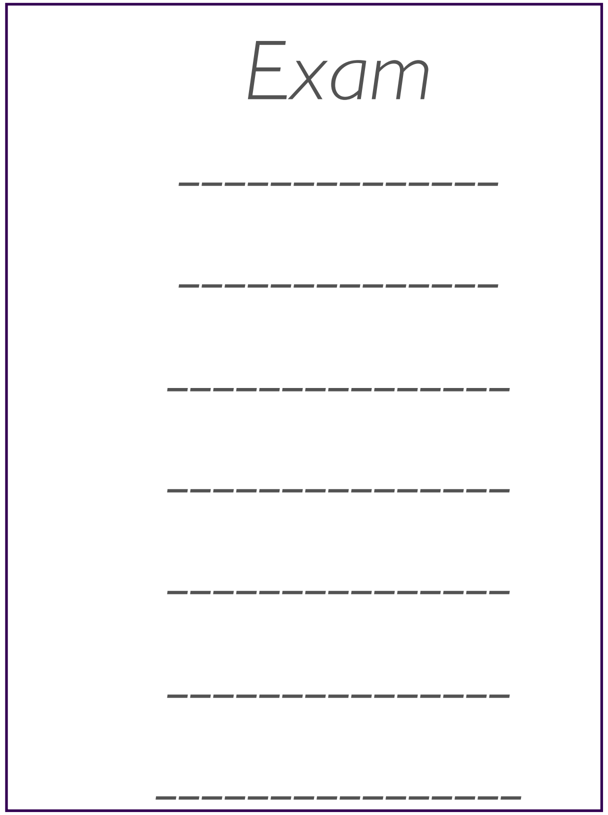
- Ensemble of same classifiers
  - Bagging and random forest
  - Boosting and gradient boosted tree
- Ensemble of different classifiers

BOOSTING AND GRADIENT BOOSTED TREE

CS 334: Machine Learning

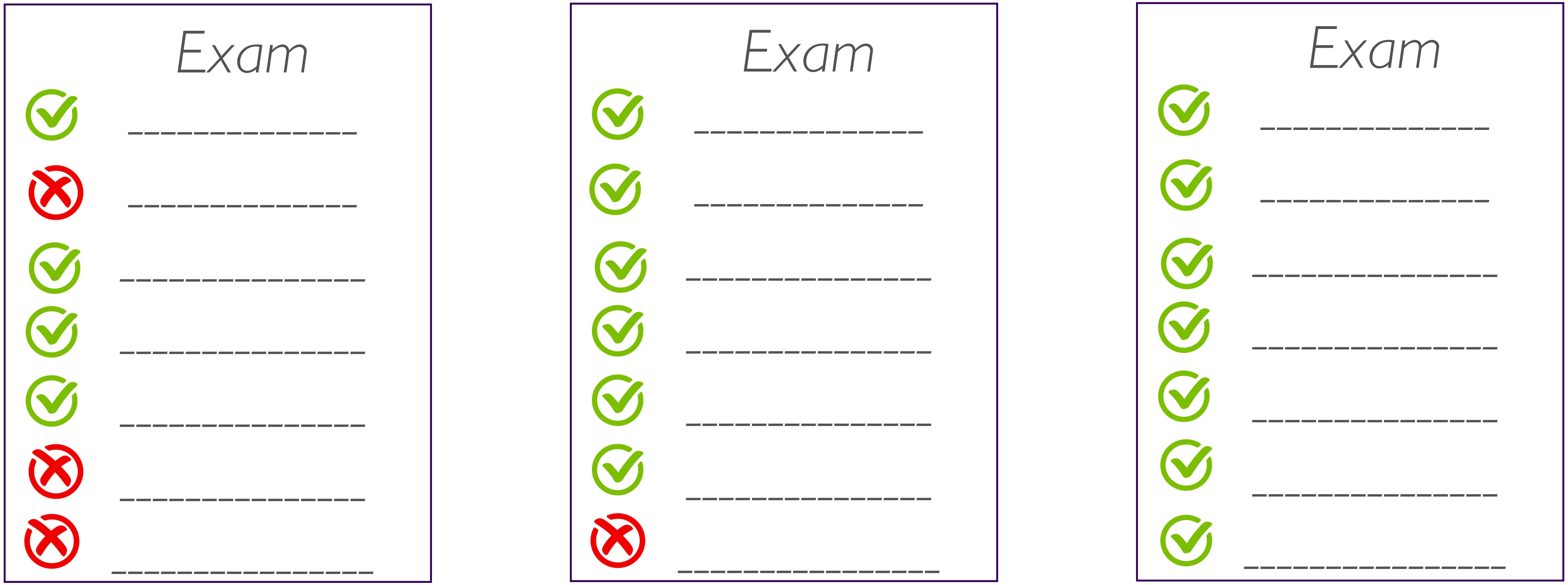
GROUP ACTIVITY

STUDY FOR AN EXAM



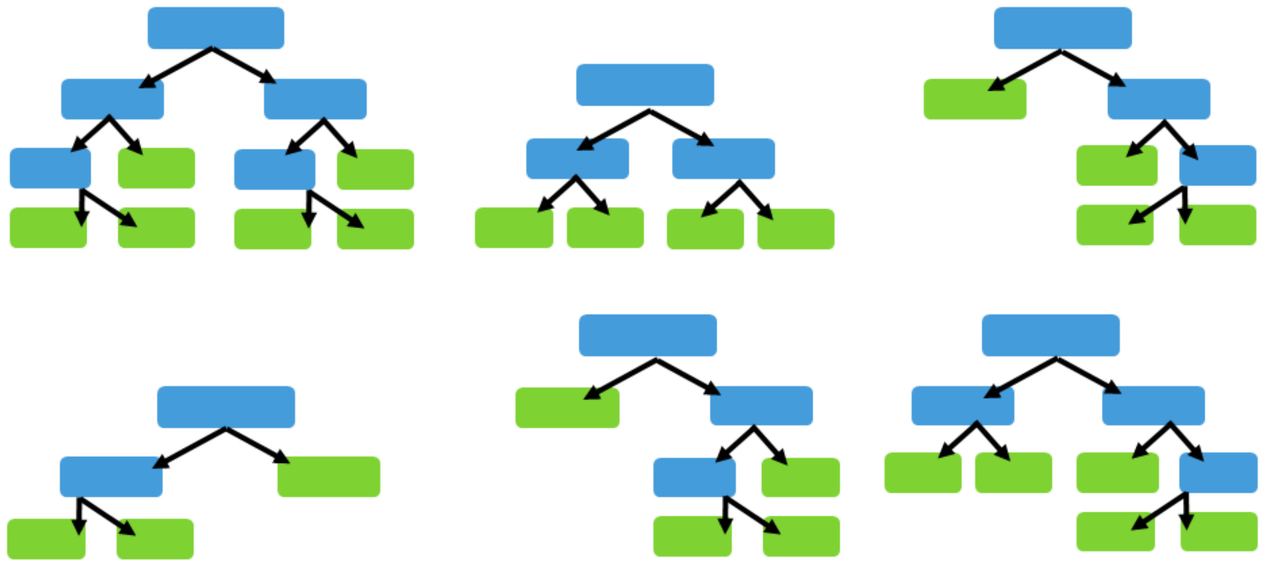
How would you study for an exam using a practice exam (given solution keys)?

STUDY FOR AN EXAM



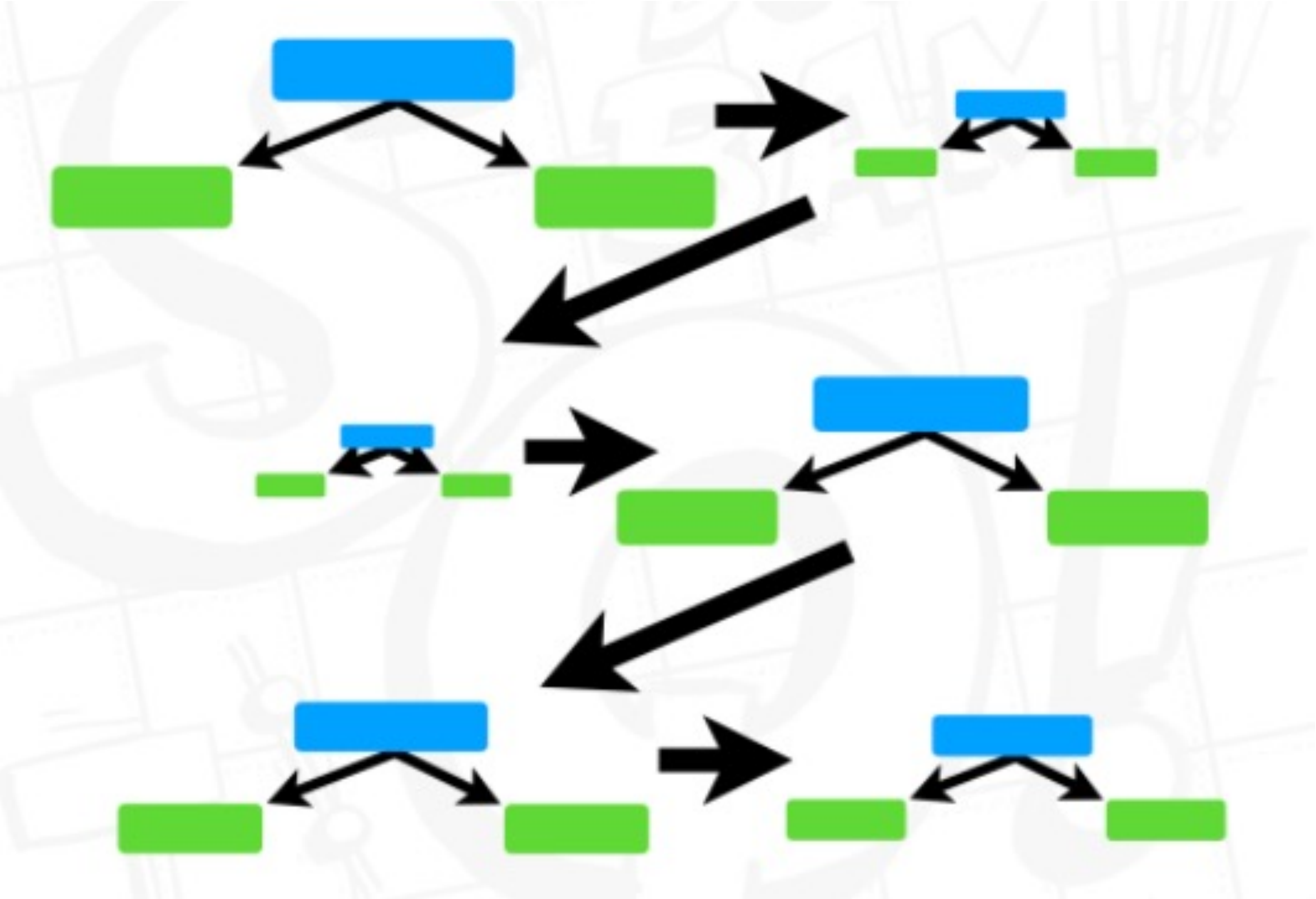
BAGGING

- Combine output of many classifiers (full sized trees)
- The learners are combined with equal weights
- Independently train each learner on a bootstrap



BOOSTING

- Combine output of many weak learners (stumps)
- The weak learners are combined with weights
- Sequentially train weak learners by compensating the shortcomings of previous learner



BOOSTING

- AdaBoost (1997): shortcomings are identified by high weight (misclassified) data points
- Gradient Boosting (1999): shortcomings are identified by residuals or gradients of the loss function (generalize AdaBoost)

ADABOOST

- Maintain a weight distribution over data points (importance of having the data point correctly classified)

- The weights are initialized as uniform and updated after each classifier is trained to guide next classifier (correctly classified examples have weight decreased and incorrect examples increased)

- Final classifier is weighted vote of individual classifiers, based on weighted error of each classifier

ADABOOST

Algorithm 32 $\textsc{AdaBoost}(\mathcal{W}, \mathcal{D}, K)$

$d^{(0)} \leftarrow \langle \frac{1}{N}, \frac{1}{N}, \dots, \frac{1}{N} \rangle$ // Initialize uniform importance to each example

for $k = 1 \dots K$ **do**

  $f^{(k)} \leftarrow \mathcal{W}(\mathcal{D}, d^{(k-1)})$ // Train kth classifier on weighted data

  $\hat{y}_n \leftarrow f^{(k)}(x_n), \forall n$ // Make predictions on training data

  $\hat{\epsilon}^{(k)} \leftarrow \sum_n d_n^{(k-1)} [y_n \neq \hat{y}_n]$ // Compute weighted training error

  $\alpha^{(k)} \leftarrow \frac{1}{2} \log \left(\frac{1-\hat{\epsilon}^{(k)}}{\hat{\epsilon}^{(k)}} \right)$ // Compute "adaptive" parameter

  $d_n^{(k)} \leftarrow \frac{1}{Z} d_n^{(k-1)} \exp[-\alpha^{(k)} y_n \hat{y}_n], \forall n$ // Re-weight examples and normalize

end for

return $f(\hat{x}) = \operatorname{sgn}\left[\sum_k \alpha^{(k)} f^{(k)}(\hat{x})\right]$ // Return (weighted) voted classifier

Weighted entropy/gini or weighted sampling

Assuming error < 0.5
Smaller error, great alpha

Decrease weight for correct samples, increase weight for incorrect samples

Smaller error -> more significant re-weighting, why?

ADABOOST

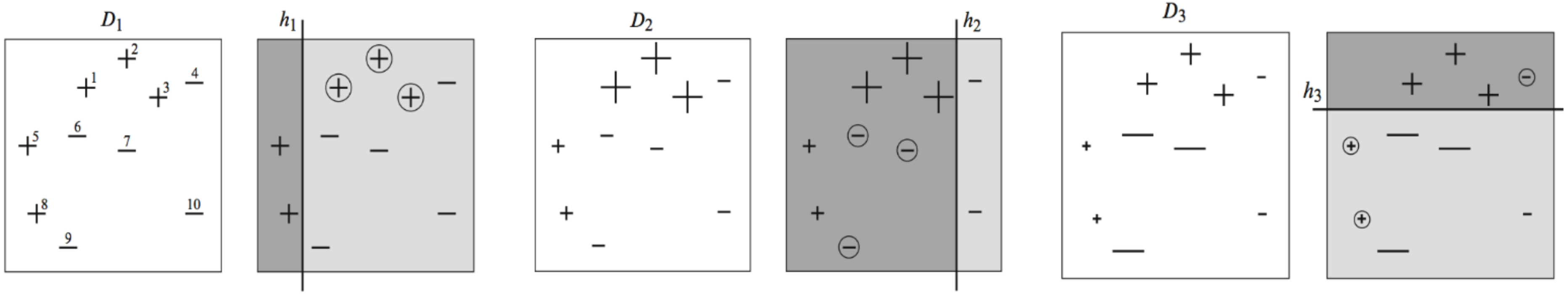
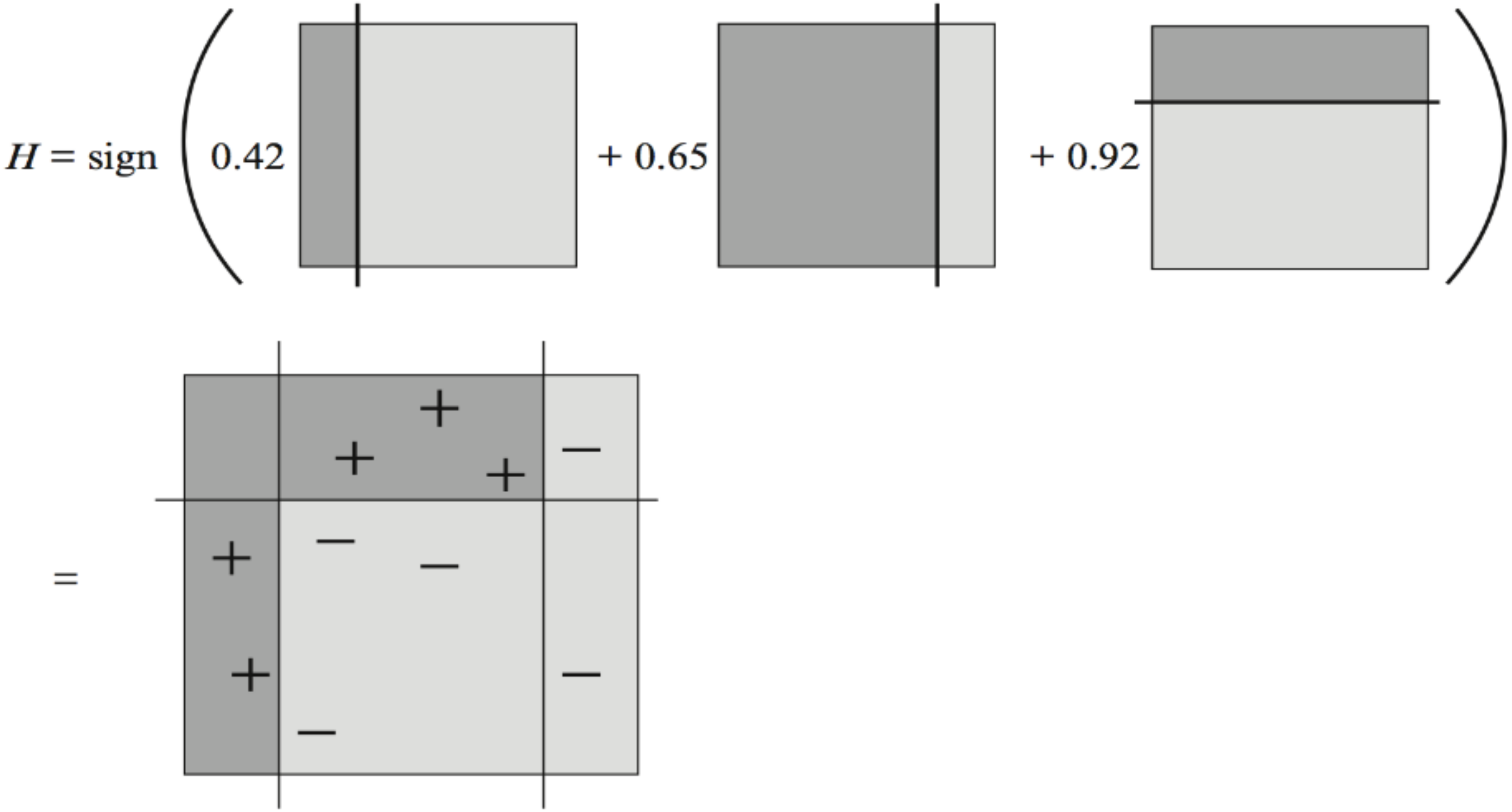


Figure: AdaBoost. Source: Figure 1.1 of [Schapire and Freund, 2012]

ADABOOST



BOOSTING

- AdaBoost: shortcomings are identified by high weight (misclassified) data points
- Gradient Boosting: shortcomings are identified by residuals or gradients of the loss function (generalize AdaBoost)

GRADIENT BOOSTING

- Gradient descent + boosting
- Powerful algorithm that can be used for regression, classification, ranking
- Data mining competition winner most likely uses this algorithm

GROUP ACTIVITY

ADDITIVE MODEL

Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88
1.6	Green	Female	76
1.5	Blue	Female	56
1.8	Red	Male	73
1.5	Green	Male	77
1.4	Blue	Female	57

- We have training data, $(x_1, y_1), \ldots, (x_n, y_n)$, and task to fit model $f(x)$ for y to minimize square loss
- A friend gives you a model f, with some mistakes
- You are not allowed to remove anything from f or change any parameter in f
- You are allowed to add additional models h to f, so new prediction is $f(x) + h(x)$

What would you do?

ADDITIVE MODEL

- Simple solution:

Height (m)	Favorite Color	Gender	Weight (kg)	Residual
1.6	Blue	Male	88	16.8
1.6	Green	Female	76	4.8
1.5	Blue	Female	56	-15.2
1.8	Red	Male	73	1.8
1.5	Green	Male	77	5.8
1.4	Blue	Female	57	-14.2

$$\begin{aligned} f(x_1) + h(x_1) &= y_1 \\ f(x_2) + h(x_2) &= y_2 \\ \vdots \qquad & \quad \vdots \\ f(x_n) + h(x_n) &= y_n \end{aligned} \quad \Leftrightarrow \quad \begin{aligned} h(x_1) &= y_1 - f(x_1) \\ h(x_2) &= y_2 - f(x_2) \\ \vdots \qquad & \quad \vdots \\ h(x_n) &= y_n - f(x_n) \end{aligned}$$

We can train a regression tree for residual h, then add the tree to original model f

GRADIENT BOOSTING: REGRESSION

- Initialize a constant model $f(x)$ for data $(x_1, y_1), \ldots, (x_n, y_n)$
- Fit a regression tree $h(x)$ to new data, $(x_1, y_1\text{-}f(x_1)), \ldots, (x_n, y_n\text{-}f(x_n))$
- $y_i\text{-}f(x_i)$ are pseudo residuals - parts that existing model f cannot do well
- Update $f(x) + h(x)$ as prediction, if it is not satisfactory, reiterate

GRADIENT BOOSTED TREE

Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88
1.6	Green	Female	76
1.5	Blue	Female	56
1.8	Red	Male	73
1.5	Green	Male	77
1.4	Blue	Female	57

GRADIENT BOOSTED TREE

71.2

Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88
1.6	Green	Female	76
1.5	Blue	Female	56
1.8	Red	Male	73
1.5	Green	Male	77
1.4	Blue	Female	57

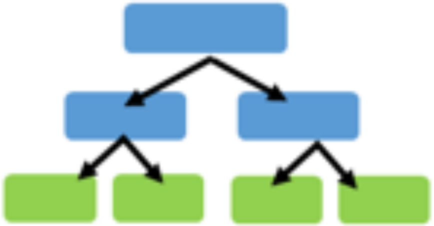
GRADIENT BOOSTED TREE

71.2

Height (m)	Favorite Color	Gender	Weight (kg)	Residual
1.6	Blue	Male	88	16.8
1.6	Green	Female	76	4.8
1.5	Blue	Female	56	-15.2
1.8	Red	Male	73	1.8
1.5	Green	Male	77	5.8
1.4	Blue	Female	57	-14.2

GRADIENT BOOSTED TREE

71.2



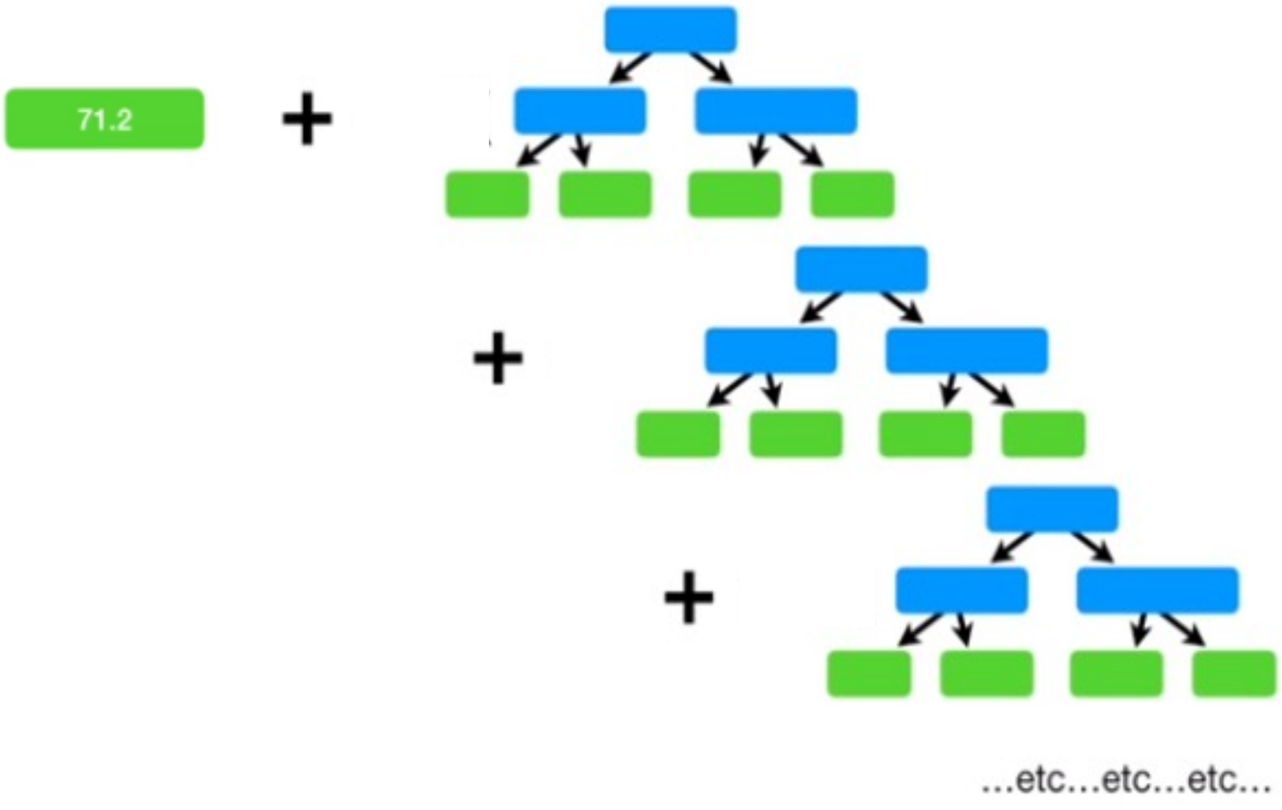
Height (m)	Favorite Color	Gender	Weight (kg)	Residual
1.6	Blue	Male	88	16.8
1.6	Green	Female	76	4.8
1.5	Blue	Female	56	-15.2
1.8	Red	Male	73	1.8
1.5	Green	Male	77	5.8
1.4	Blue	Female	57	-14.2

GRADIENT BOOSTED TREE

Height (m)	Favorite Color	Gender	Weight (kg)	Residual
1.6	Blue	Male	88	16.8
1.6	Green	Female	76	4.8
1.5	Blue	Female	56	-15.2
1.8	Red	Male	73	1.8
1.5	Green	Male	77	5.8
1.4	Blue	Female	57	-14.2

GRADIENT BOOSTED TREE

Height (m)	Favorite Color	Gender	Weight (kg)	Residual
1.6	Blue	Male	88	16.8
1.6	Green	Female	76	4.8
1.5	Blue	Female	56	-15.2
1.8	Red	Male	73	1.8
1.5	Green	Male	77	5.8
1.4	Blue	Female	57	-14.2



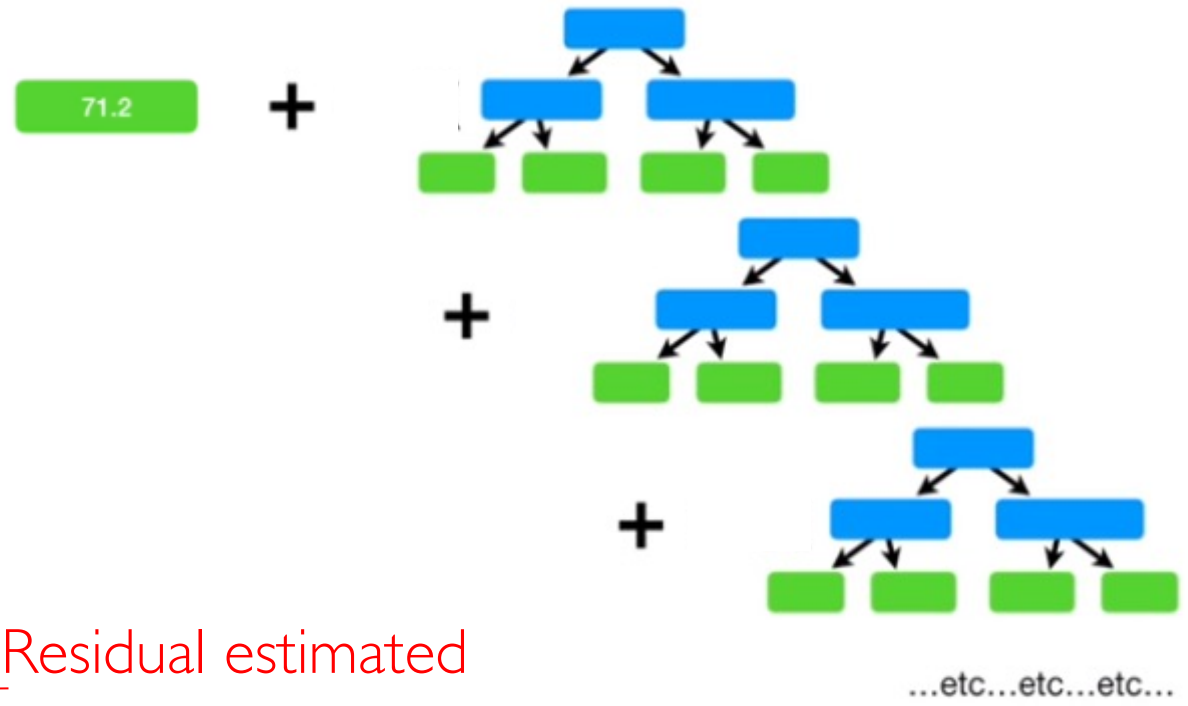
GRADIENT BOOSTING: REGRESSION

- Initialize a constant model $f(x)$ for data $(x_1, y_1), \ldots, (x_n, y_n)$
- Fit a regression tree $h(x)$ to new data, $(x_1, y_1\text{-}f(x_1)), \ldots, (x_n, y_n\text{-}f(x_n))$
- $y_i\text{-}f(x_i)$ are pseudo residuals - parts that existing model f cannot do well
- Update $f(x) + h(x)$ as prediction, if it is not satisfactory, reiterate

How does this relate to gradient descent?

GRADIENT BOOSTING: REGRESSION

- Loss function: $L(\mathbf{y}, f(\mathbf{X})) = \sum_i \frac{1}{2}(y_i - f(\mathbf{x}_i))^2$
- Treat f($\mathbf{x}_i$) as parameters and take derivatives
$$\frac{\partial L}{\partial f(\mathbf{x_i})} = f(\mathbf{x_i}) - y_i$$
- Interpret residuals as negative gradients (learning rate = 1 here)



$$f(\mathbf{x_i}) := f(\mathbf{x_i}) + \boxed{y_i - f(\mathbf{x_i})}$$

Residual estimated by next tree

$$f(\mathbf{x_i}) := f(\mathbf{x_i}) - 1\frac{\partial L}{\partial \mathbf{x}_i}$$

GRADIENT TREE BOOSTING: ALGORITHM

Algorithm 10.3 *Gradient Tree Boosting Algorithm.*

1. Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^N L(y_i, \gamma)$. ← Constant model

2. For $m = 1$ to M:

   (a) For $i = 1, 2, \ldots, N$ compute

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

   ← Residual of i or negative gradient

   (b) Fit a regression tree to the targets r_{im} giving terminal regions $R_{jm},\ j = 1, 2, \ldots, J_m$. ← Jm is the size of the tree

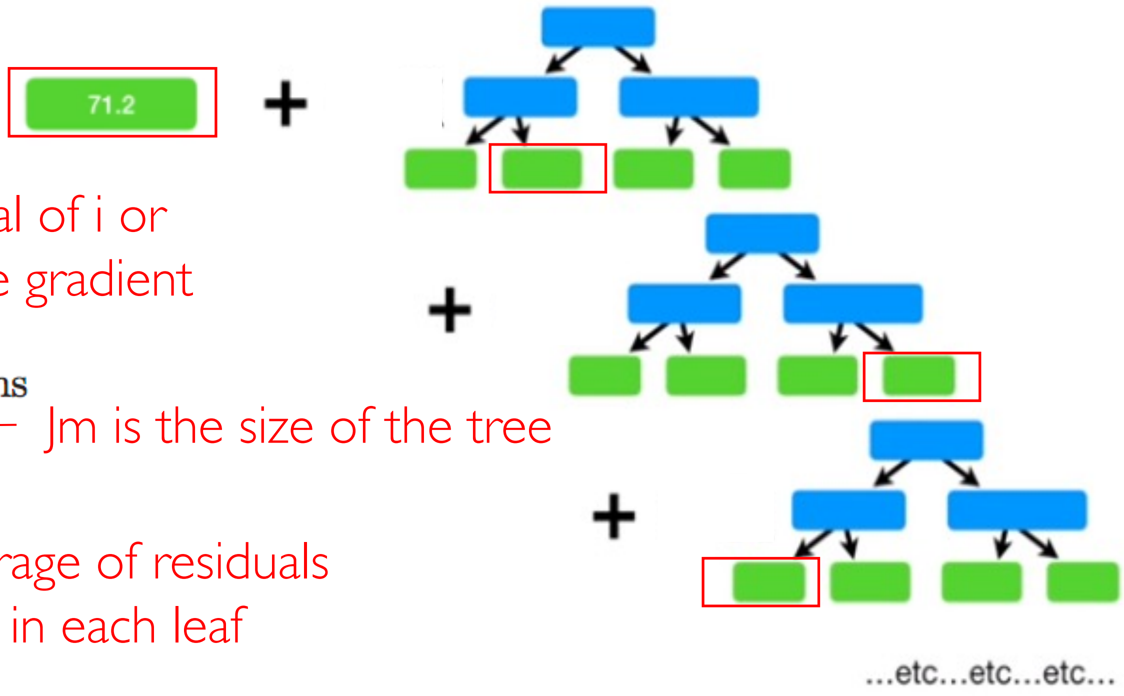
   (c) For $j = 1, 2, \ldots, J_m$ compute

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

   ← Average of residuals in each leaf

   (d) Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$. ← update

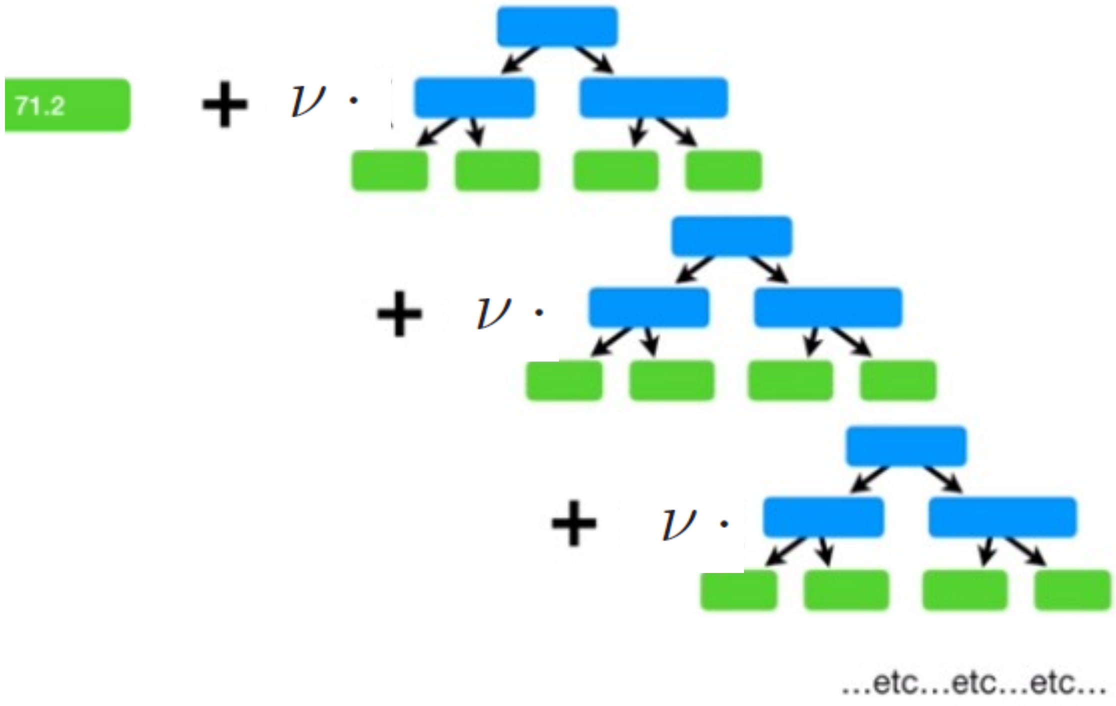
3. Output $\hat{f}(x) = f_M(x)$.



SHRINKAGE

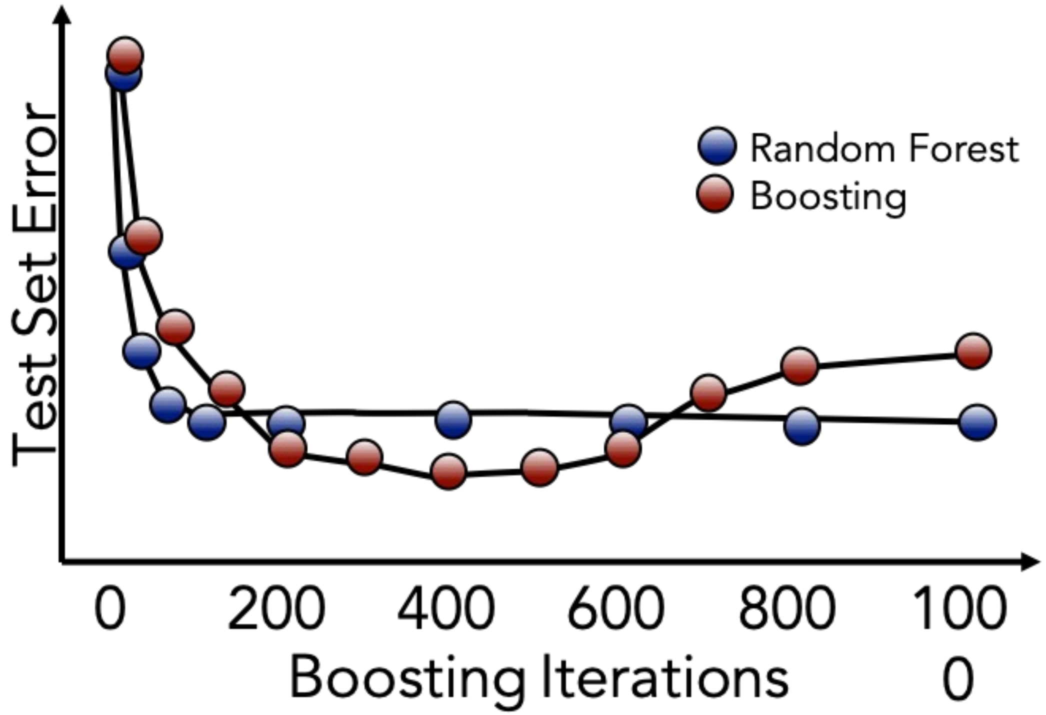
Algorithm 10.3 *Gradient Tree Boosting Algorithm.*

1. Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$.
2. For $m = 1$ to M:
   (a) For $i = 1, 2, \ldots, N$ compute
   $$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}.$$
   (b) Fit a regression tree to the targets r_{im} giving terminal regions $R_{jm},\ j = 1, 2, \ldots, J_m$.
   (c) For $j = 1, 2, \ldots, J_m$ compute
   $$\gamma_{jm} = \arg\min_{\gamma} \sum_{x_i \in R_{jm}} L\left(y_i, f_{m-1}(x_i) + \gamma\right).$$
   (d) Update $f_m(x) = f_{m-1}(x) + \boxed{\nu} \cdot \sum_{j=1}^{J} \gamma_{jm} I(x \in R_{jm})$.
3. Output $\hat{f}(x) = f_M(x)$.



ν: add a shrinkage parameter to control the learning rate of the boosting procedure.

GRADIENT BOOSTING: ITERATIONS

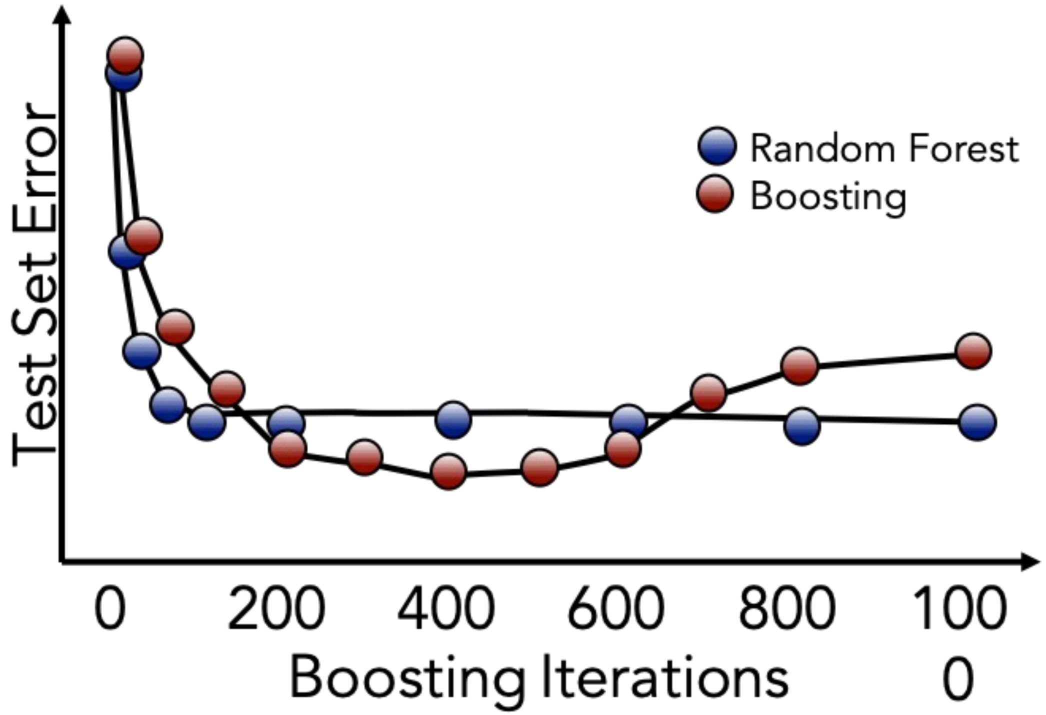


What do we observe here?

GRADIENT BOOSTING: ITERATIONS

- Boosting (bias reduction) vs random forest (variance reduction)
- Boosting is additive, reduces bias, so can overfit (variance can increase)

  (like over studying practice exam)
- Use standard mechanism to tune the parameters



SIZE OF TREES

- J_m: the size of tree for each iteration
- Simplest strategy is to use the same $J_m = J$
- Best method is to grow small trees with no pruning
- Right size will depend on level of interaction between variables
- ~4-8 leaves works well

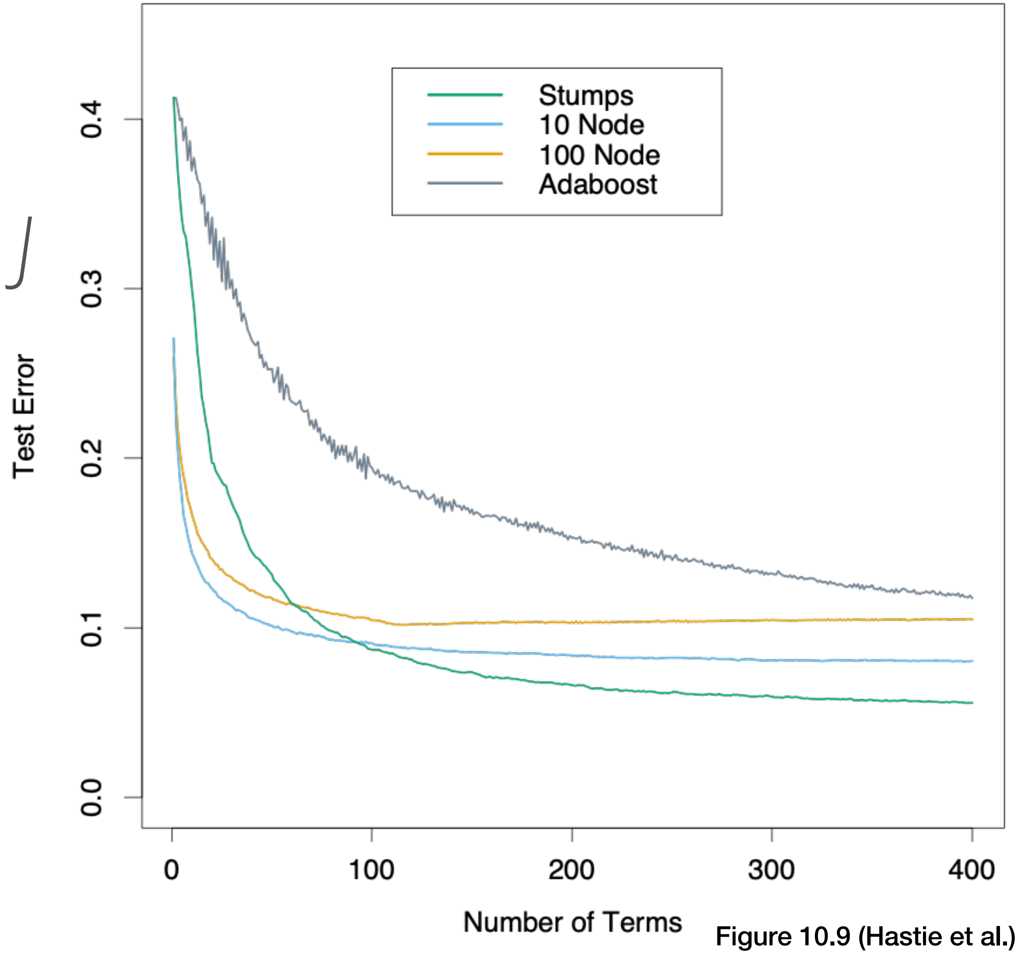
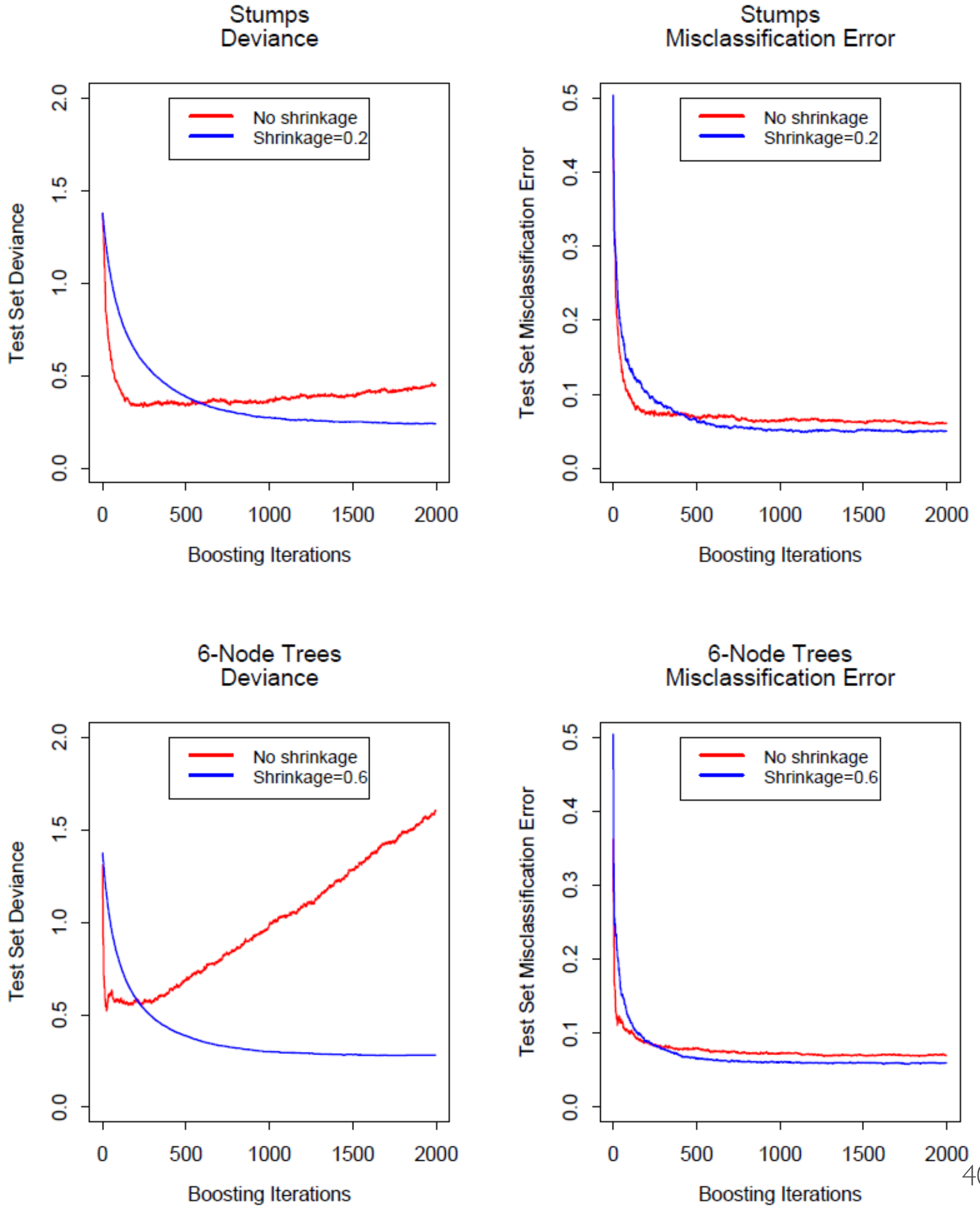


Figure 10.9 (Hastie et al.)

SHRINKAGE

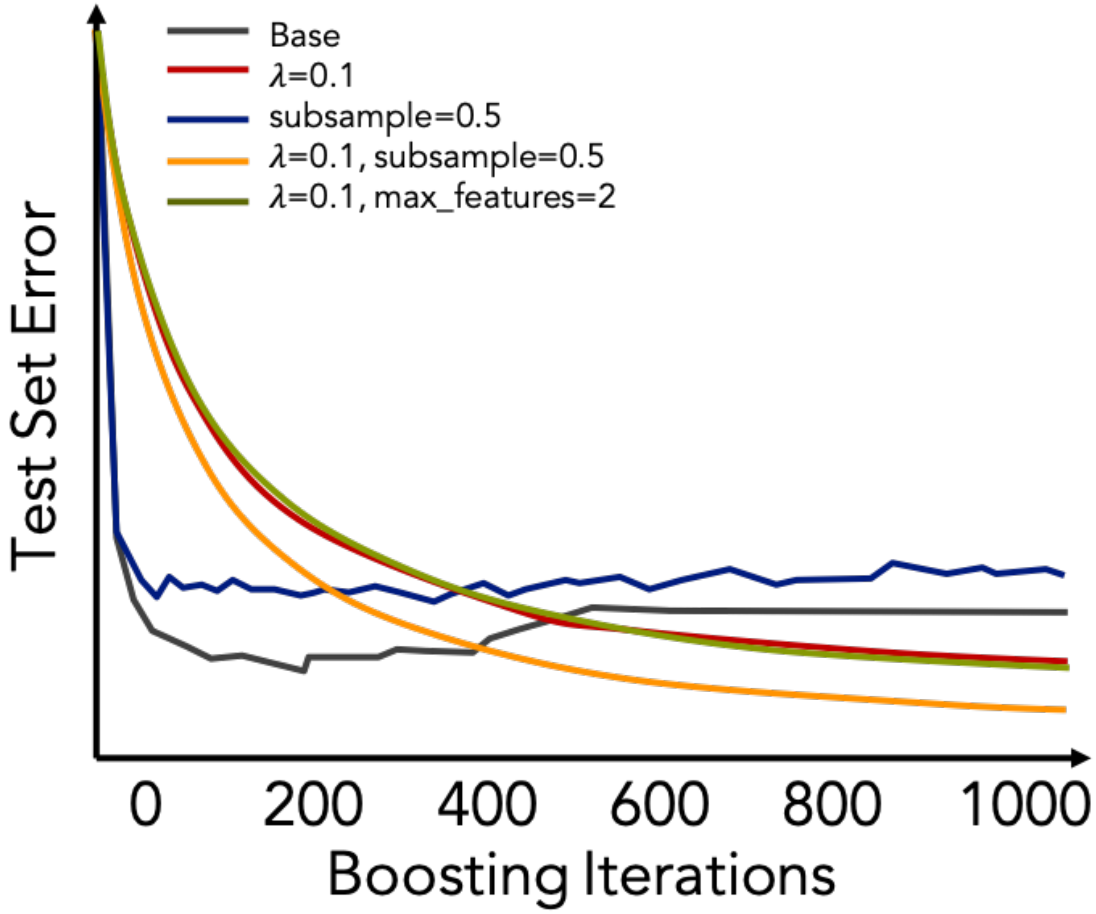
- Typically values are <0.1, use 0.01 or 0.001

- Can choose M for early stopping



STOCHASTIC GRADIENT BOOSTING

- Subsampling: each iteration, a fraction of training observations (without replacements) is used to grow the tree
- Can subsample on record or features
- Not only reduces computing time, can also produce more accurate model (less overfitting)



BAGGING VS BOOSTING

Bagging	Boosting
Bootstrapped samples	Fit entire data set (can have subsamples)
Base trees created independently	Base trees created successively
Only data points considered	Use residuals from previous models
No weighting used	Up-weight misclassified points (explicit weight or as residuals)
Excess trees will not overfit	Beware of overfitting

SKLEARN: GRADIENT BOOSTING TREE

```
>>> from sklearn.datasets import make_hastie_10_2
>>> from sklearn.ensemble import GradientBoostingClassifier
```

```
>>> X, y = make_hastie_10_2(random_state=0)
>>> X_train, X_test = X[:2000], X[2000:]
>>> y_train, y_test = y[:2000], y[2000:]
```

```
>>> clf = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0,
...     max_depth=1, random_state=0).fit(X_train, y_train)
>>> clf.score(X_test, y_test)
0.913...
```

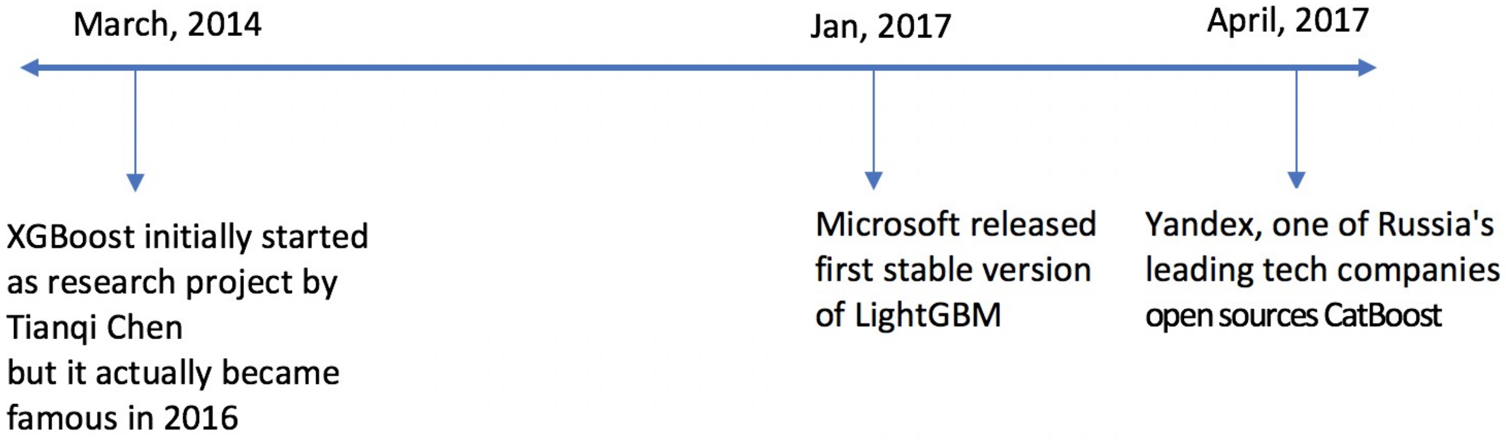
GRADIENT BOOSTING LIBRARIES

- XGBoost (Extreme Gradient Boosting)

  https://xgboost.readthedocs.io

- LightGBM (Light Gradient Boosted Machine)

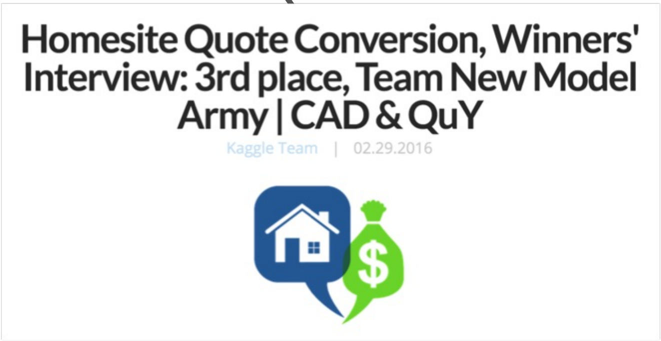
  https://lightgbm.readthedocs.io/



XGBOOST AND LIGHTGBM

- Implements basic idea of GBM with some tweaks
  - Regularization of base tree
  - Approximate split finding
  - Weighted quantile sketch
  - Sparsity-aware split finding
  - Cache-aware block structure for out of core computation

EXTREME GRADIENT BOOSTING (XGBOOST)



What these various data mining competitors have in common: all used XGBoost

ENSEMBLE METHODS

- Ensemble of same classifiers
    - Bagging and random forest
    - Boosting and gradient boosted tree
- Ensemble of different classifiers