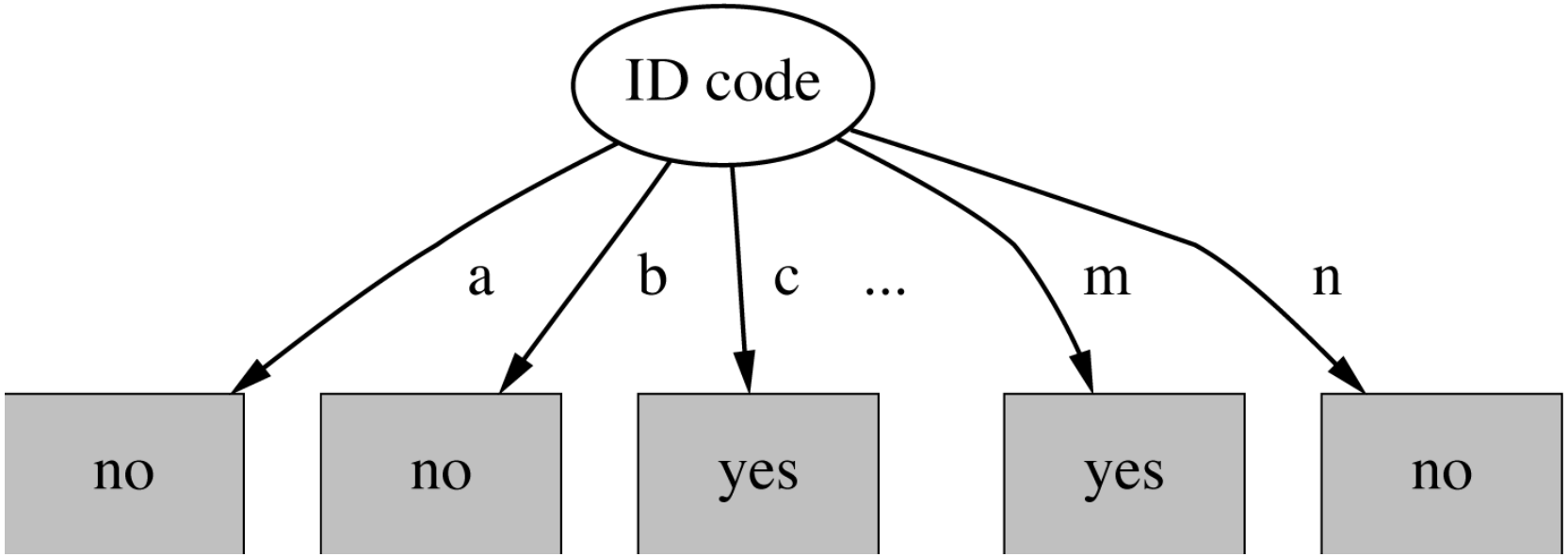
DATA MINING

/CISC 873 - Steven Ding
/Week #4/Lecture 1
Tree cont’d

Information Gain

- Assume that the attributes below that have the **SAME** information gain
  - A: age
  - B: has_1000_driving_tickets
  - C: student_id
  - Which one would you pick?

Split for ID Code Attribute



Entropy of split = 0 (since each leaf node is “pure”, having only one case.

Information gain is maximal for ID code

Gain ratio

- *Gain ratio*: a modification of the information gain that reduces its bias towards high-branch attributes

- Gain ratio should encourage
  - Even distribution (of instances/samples)
  - Low distinct values

- How?
  - Low intrinsic information => the entropy of the attribute itself

Gain Ratio and Intrinsic Info.

- Intrinsic information: entropy of distribution of ***instances*** into branches

$$Intrinsic(A) = -\sum_{A}^{i} \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}$$

$$GainRatio(A) = \frac{Gain(A)}{Intrinsic\ (A)}$$

$$= \frac{Info(D) - Info_A(D)}{Intrinsic\ (A)}$$

More on the gain ratio

- Problem with gain ratio: it may overcompensate
  - May choose an attribute just because its intrinsic information is very low
  - Standard fix:
    - First, only consider attributes with greater than average information gain
    - Then, compare them on gain ratio

Gini Index - CART

Number of unique class labels

$$gini(D) = 1 - \sum_{j=1}^{n} p_j^2$$

p_j is the relative frequency of class j in D

Gini Index - CART

Unique values for attribute A

$$gini_A(D) = \sum_{j}^{v_A} \frac{|D_j|}{|D|} gini(D_j)$$

Ratio of instance for A=j

gini on a subset of D where A=j

Gini Index - CART

- Algorithm for top-down induction of decision trees ("**ID3**") was developed by Ross Quinlan
  - **C4.5** (gain ratio), which can deal with numeric attributes, missing values, and noisy data

- Similar approach: **CART**
- There are many other attribute selection criteria!

Numeric attributes

- Standard method: discretization
  - E.g. temp < 45
- Unlike nominal attributes,
  every attribute has many possible split points
- Solution is straightforward extension:
  - Evaluate info gain (or other measure) for every possible split point of attribute
  - Choose "best" split point
  - Info gain for best split point is info gain for attribute
- Computationally more demanding

Prepruning

- Based on statistical significance test
  - Stop growing the tree when there is ***no statistically significant*** association between any attribute and the class at a particular node
- Most popular test: ***chi-squared test***
- ID3 used chi-squared test in addition to information gain
  - Only statistically significant attributes were allowed to be selected by information gain procedure

Early stopping

	a	b	class
1	0	0	0
2	0	1	1
3	1	0	1
4	1	1	0

- Pre-pruning may stop the growth process prematurely: *early stopping*

- Classic example: XOR/Parity-problem
  - No *individual* attribute exhibits any significant association to the class
  - Structure is only visible in fully expanded tree
  - Pre-pruning won't expand the root node

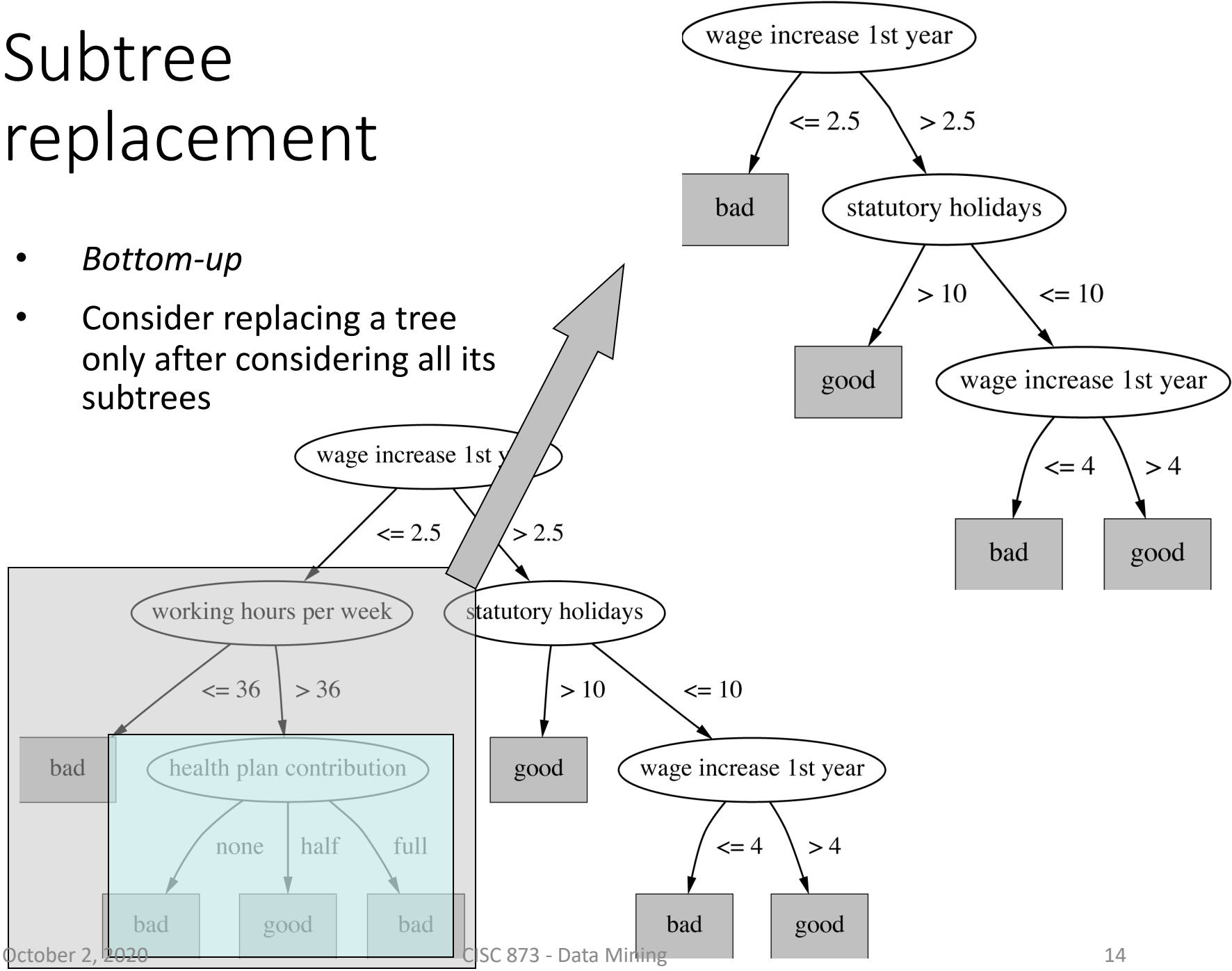
- But: XOR-type problems rare in practice
- And: pre-pruning faster than post-pruning

Post-pruning

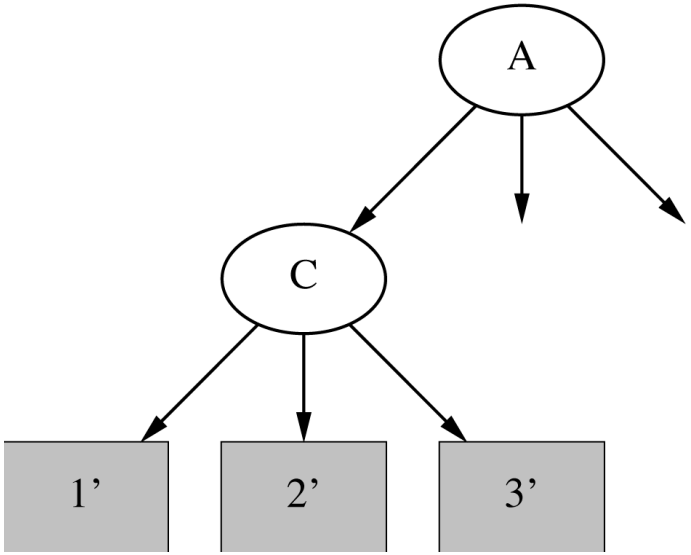
- First, build full tree
- Then, prune it
  - Fully-grown tree shows all attribute interactions
- Problem: some subtrees might be due to chance effects
- Two pruning operations:
  1. *Subtree replacement*
  2. *Subtree raising*
- Possible strategies:
  - error estimation
  - significance testing
  - Minimum Description Length principle

Subtree replacement

- *Bottom-up*
- Consider replacing a tree only after considering all its subtrees

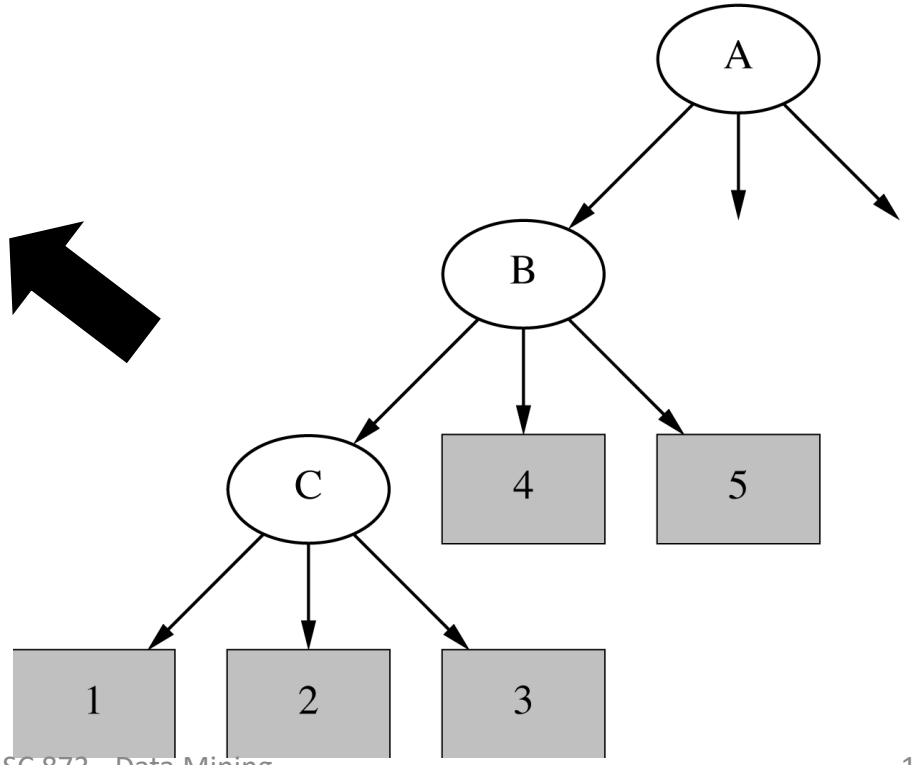


Subtree raising

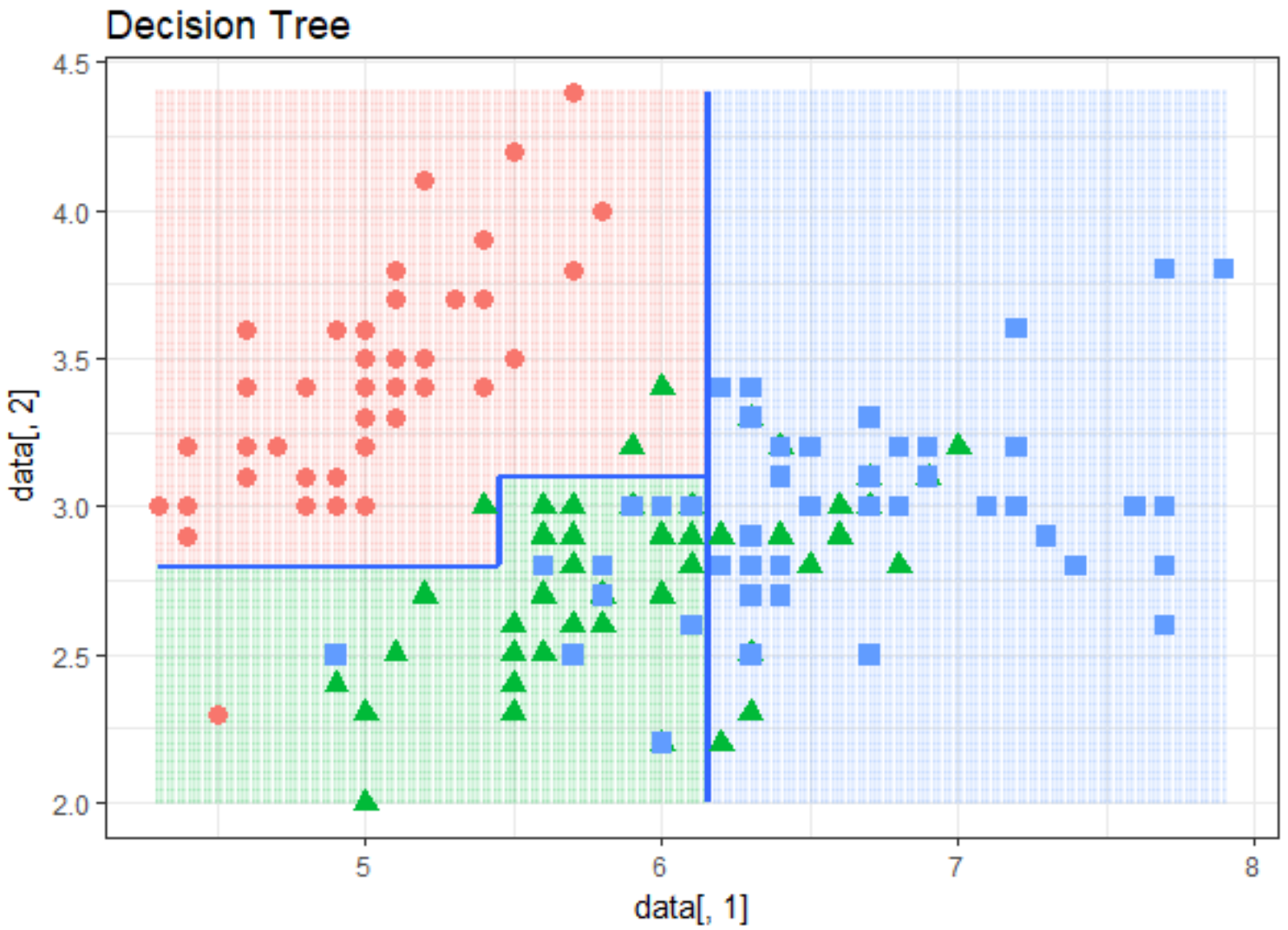


- Delete node
- Redistribute instances
- Slower than subtree replacement

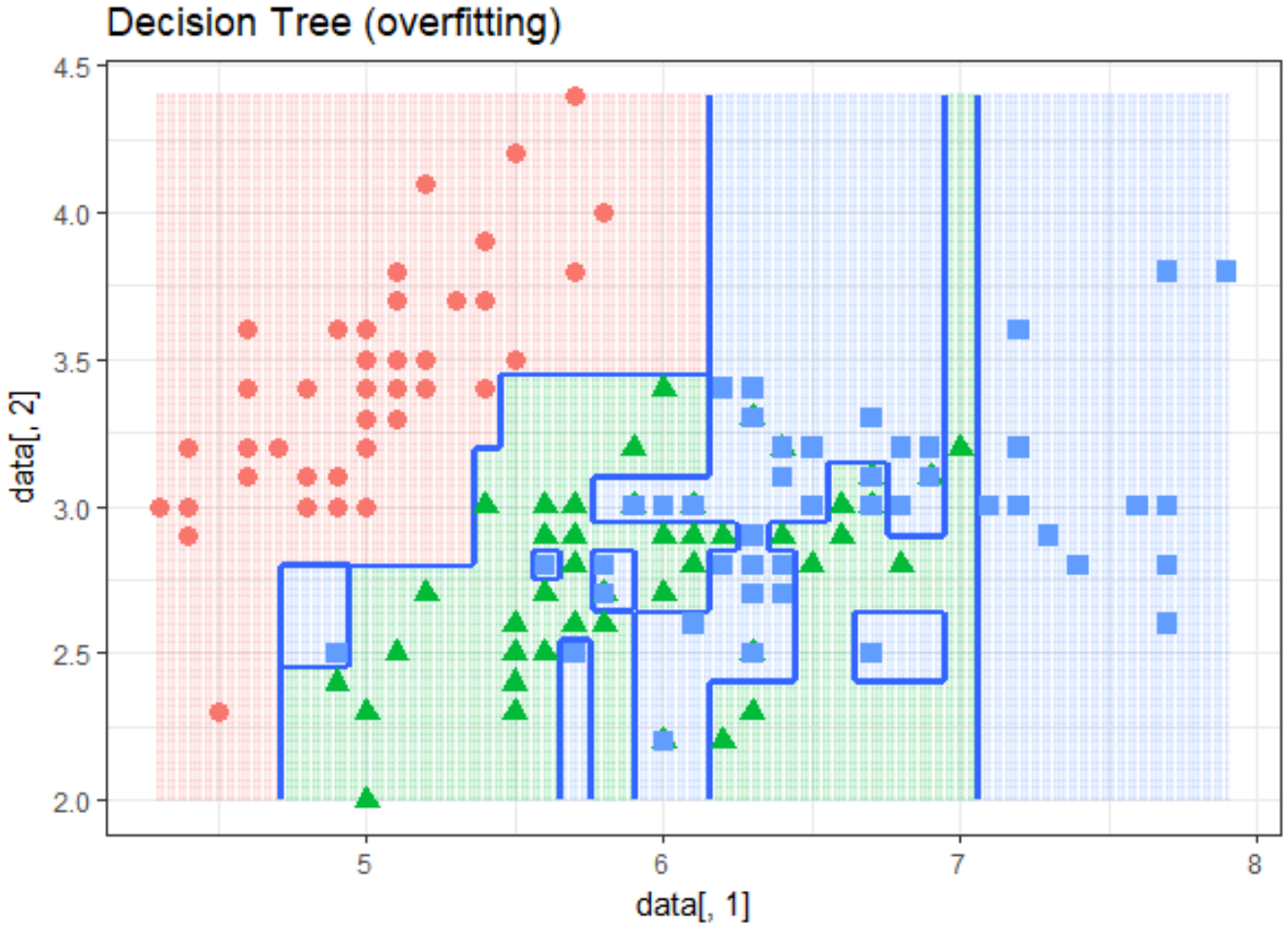
  (Worthwhile?)



Decision Boundary



Decision Boundary – overfit (no pruning)



Feature Selection (is difficult)

- Why?
  - Given m features: high training complexity often m^2
- What (to remove)?
  - Values are not correlated with the target label
    - Correlation – no transitive property.
    - A, B can both correlate to D but A & B may/may not be correlated (so we can't just remove A/B)
    - A correlated to B => we can't just remove A
    - A,B correlates to D, and A,B correlates to each other => redundant
      - But keep A or B?
    - Theoretically we need to test every possible pair of m.
  - Low predictive power
    - A set of feature combined >= sum of each individual
    - Need to evaluate every possible subset of m.

Feature Selection (is difficult)

- Why?
  - Given m features: high training complexity often m^2
- What (to remove)?
  - Values are not correlated with the target label
    - Correlation – no transitive property.
    - A, B can both correlate to D but A & B may/may not be correlated (so we can't just remove A/B)
    - A correlated to B => we can't just remove A
    - A,B correlates to D, and A,B correlates to each other => redundant
      - But keep A or B?
    - Theoretically we need to test every possible pair of m.
  - Low predictive power
    - A set of feature combined >= sum of each individual
    - Need to evaluate every possible subset of m.

Feature Selection

Random Forests

The name comes from the fact that this was originally an extension of decision trees, but the idea applies to any predictor.

The basic idea: build a large number of decision trees, each using only some of the attributes of the dataset. Deploy the forest using voting (classification) or averaging (regression).

Recall: Bootstrapping

Recall: a bootstrap sample from n objects means draw n objects with replacement.

The sample will contain about 2n/3 unique objects, leaving n/3 objects that were never selected.

These unselected objects can be used as a test set with nice properties.

Growing one tree of the forest:

1. Select a bootstrap sample of the object**s**

2. If there are m attributes, choose some k < m (usually k is much smaller than m, maybe √m)

3. Grow a decision tree by randomly selecting k of the m attributes at each level, and choosing the best split based on these k attributes (in whatever standard way)

4. Grow the decision tree to full size (i.e. no pruning)

5. Use the out-of-bag set as a test sample and measure the test error for this tree (ongoing estimate of how well we're doing)

6. Run the entire dataset through the tree. Whenever two objects end up at the same leaf increase the proximity score for the pair.

Growing the forest:

1. Grow some number of individual trees – this is quite cheap so we might grow 100s or 1000s.

2. For each object, consider how many times it appeared in a test set, and count

$$\frac{\text{number of times prediction was wrong}}{\text{number of times it appeared in the test set}}$$

Average this to get an overall test error

3. Divide the proximities by the total number of trees

Advantages:

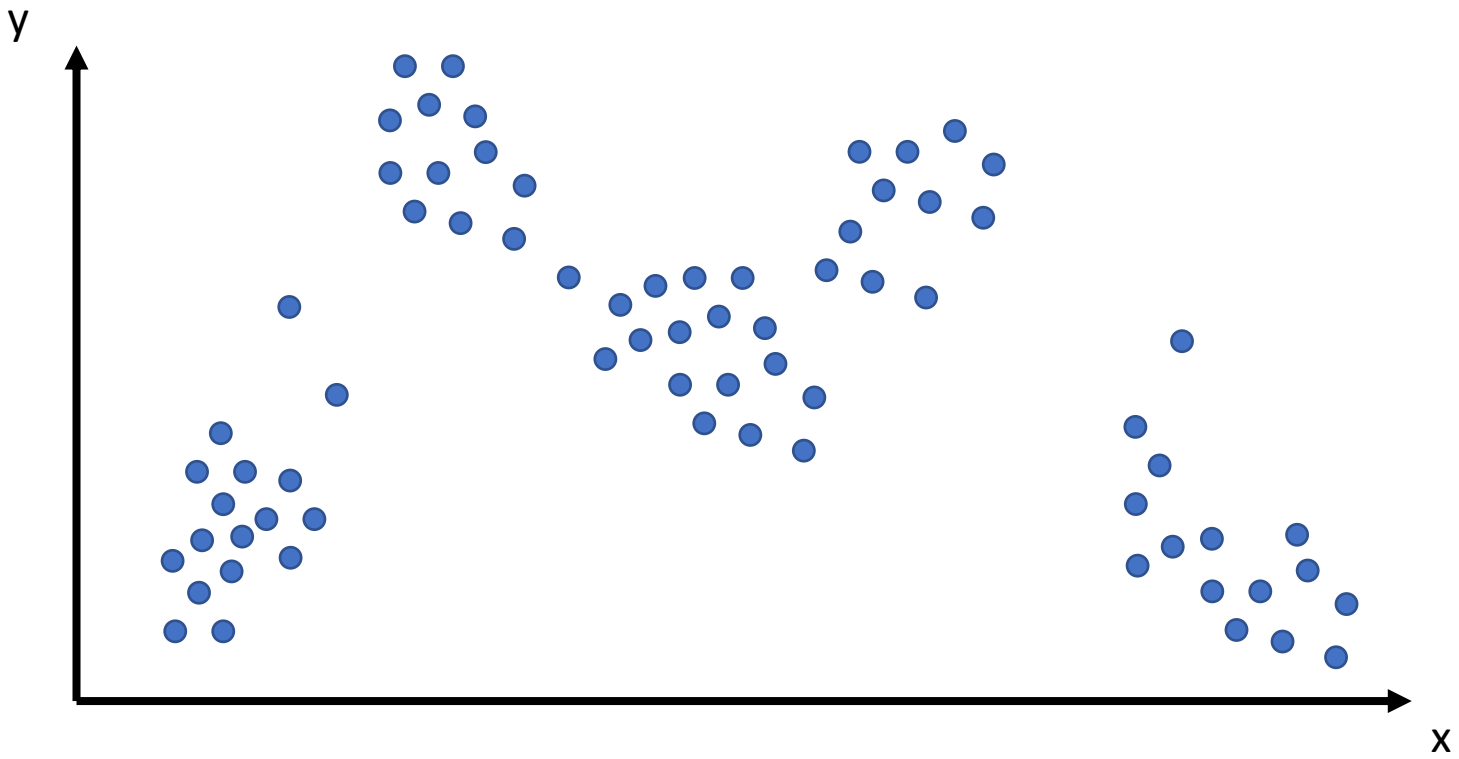
- handles multiple classes
- fast (worse than 1 decision tree, better than a neural network)
- low variance, low bias
- no separate test procedure needed (no cross-validation)
- high accuracy
- effective for large numbers of attributes
- helpful for attribute selection
- does not overfit no matter how many trees

Disadvantages:

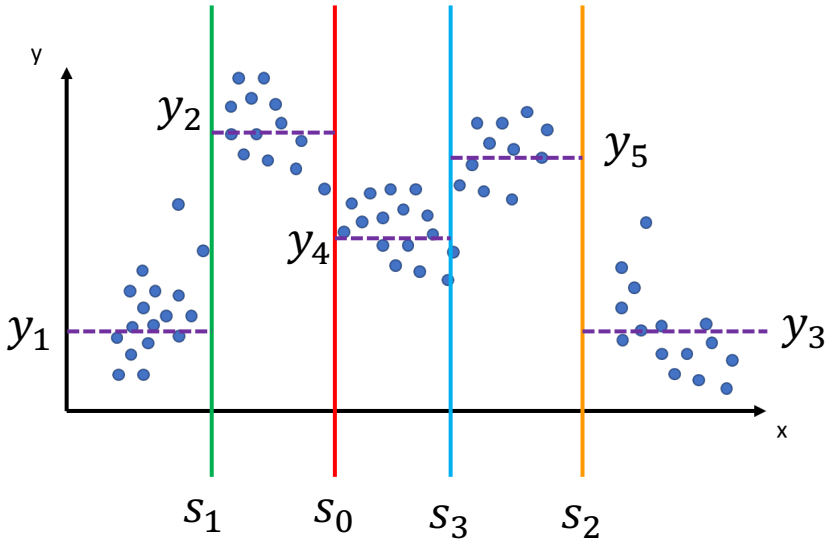
- opaque predictor
- expensive to deploy

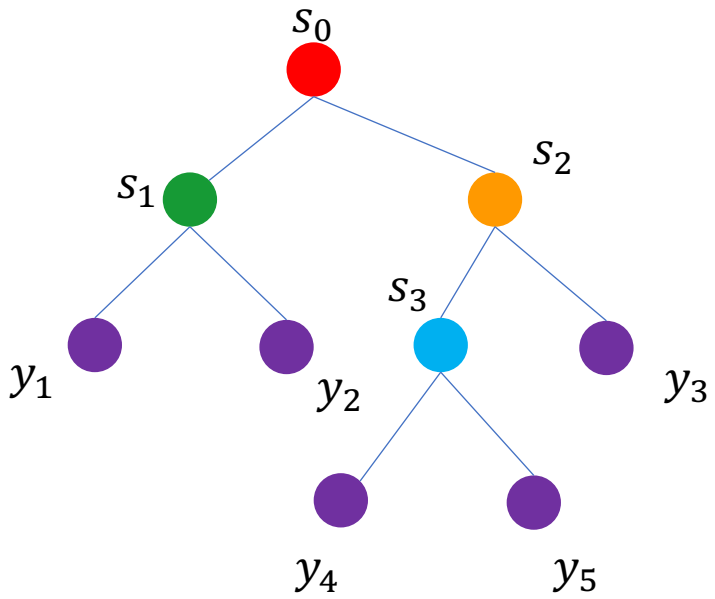
Regression Tree (with CART)

- When your class label and attributes are numeric data

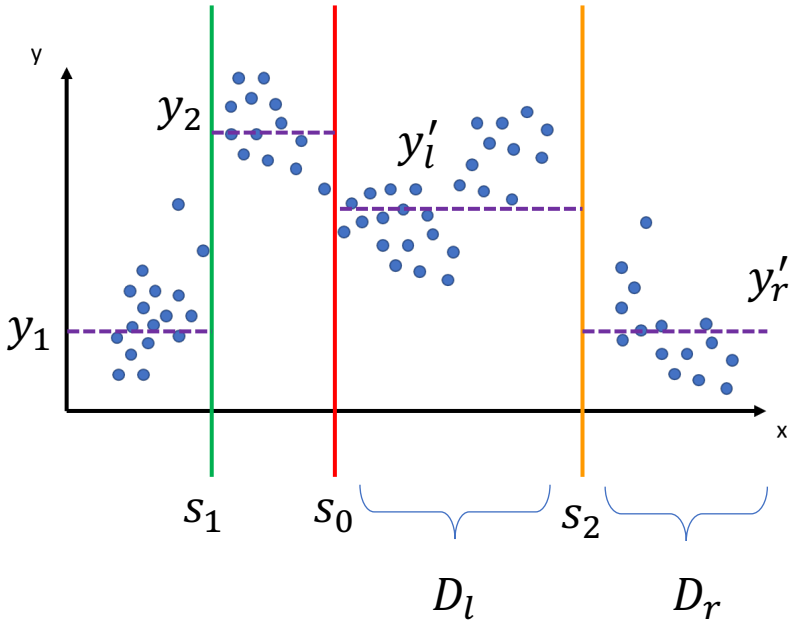


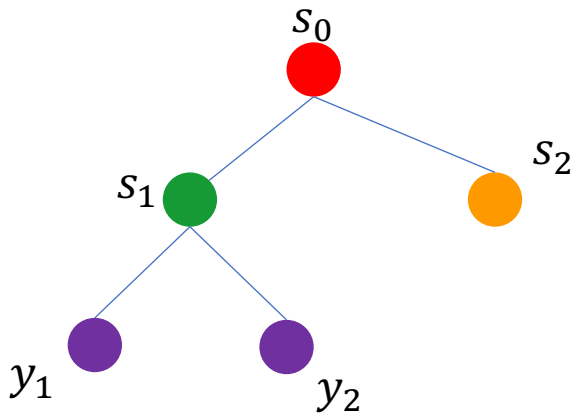
CART – Binary Tree





Regression Tree (with CART)





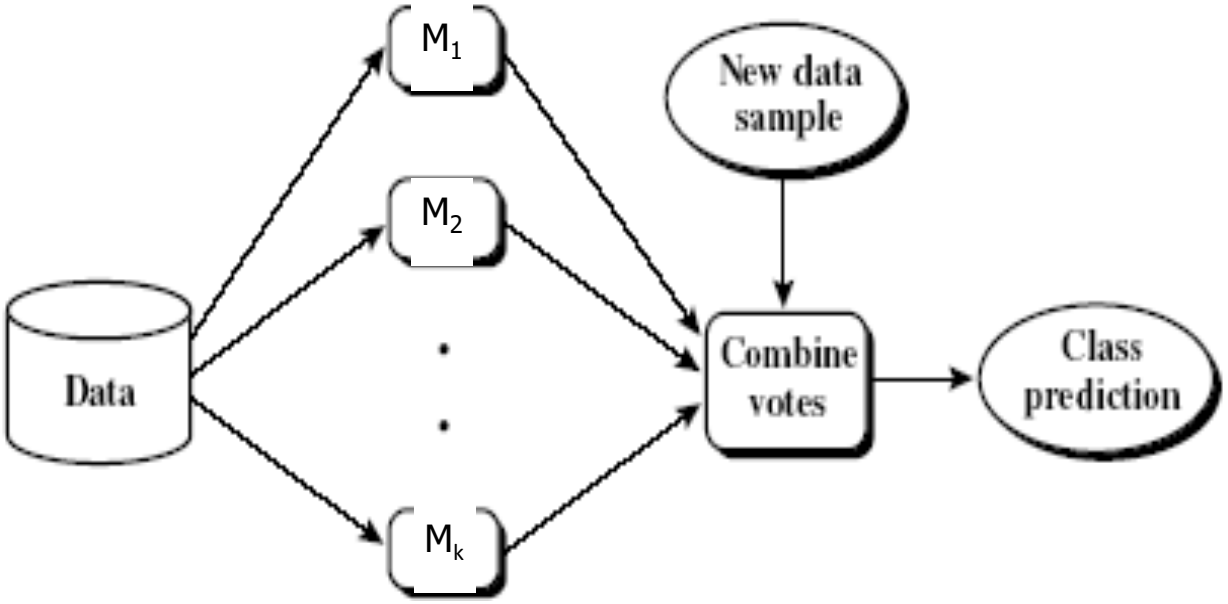
$$SSE = \sum_{i \in D_l} (y_i - y_l')^2 + \sum_{i \in D_r} (y_i - y_r')^2$$

Breiman, Leo, et al. Classification and regression trees. CRC press, 1984.

DATA MINING

/CISC 873 - Steven Ding
/Week #4/Lecture 1
Ensembles and Tree Ensembles

Ensemble methods



Use a combination of models to increase accuracy
Combine a series of k learned models, M_1, M_2, …, M_k, with the aim of creating an improved model M*

Bagging: Boostrap Aggregation

- Analogy: Diagnosis based on multiple doctors' majority vote

Bagging: Boostrap Aggregation

- Training
  - Given a set D of *d* records, at each iteration *i*, a training set $D_i \subset D$ tuples is **sampled with replacement** from D
    - Like we did for bootstrap, but this is only for training
  - A classifier model M_i is learned for each training set D_i

- Classification: classify an unknown sample **X**
  - Each classifier M_i returns its class prediction
  - The bagged classifier M* counts the votes and assigns the class with the majority votes to **X**

- Prediction: can be applied to the prediction of continuous values by taking the average value of each prediction for a given test tuple

Bagging: Boostrap Aggregation

As long as the model being used is **complex *enough***, each of the individual predictors will have small biases.

The effect of the voting is ***to cancel out the errors due to variance***.

So overall the errors will tend to be small.

- Accuracy
  - Often significantly better than a single classifier derived from D
  - For noisy data: not considerably worse, more **robust**
  - Proved improved accuracy in prediction

Bagging: Boostrap Aggregation

- Required: sampling **with replacement**
  - Samples have the right variance properties.

In practice, many variations seem to give decent results.
N.B. especially a partition instead of a set of samples, which is required for a parallel algorithm.

This is probably because real datasets tend to be repetitive.

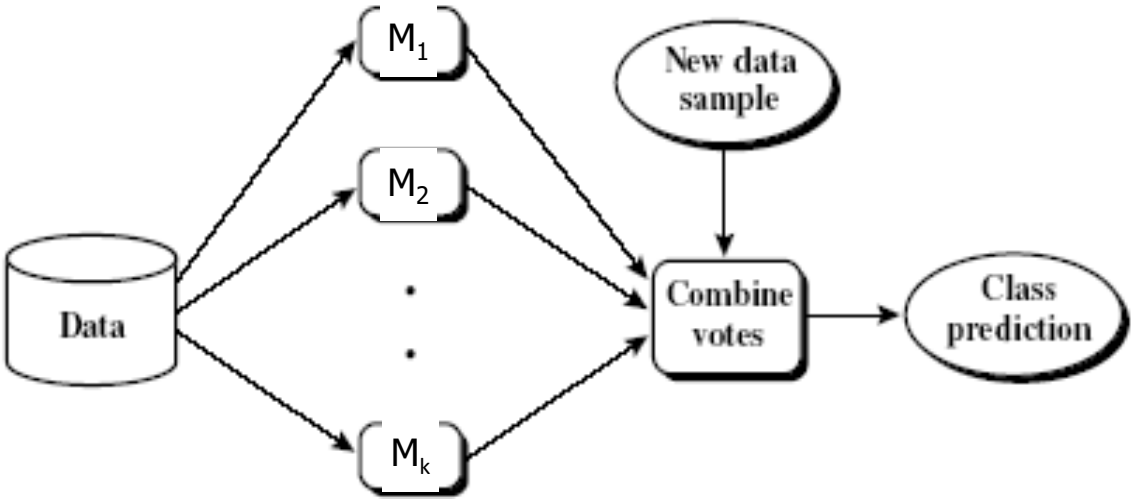
Boosting

In bagging, all samples and so all predictors are created **equal**.

However, as a predictor is built from the first sample, we learn something about **which objects are the hardest to classify**.

Use this information to select a better sample to learn from the second time, the third time,

Boosting



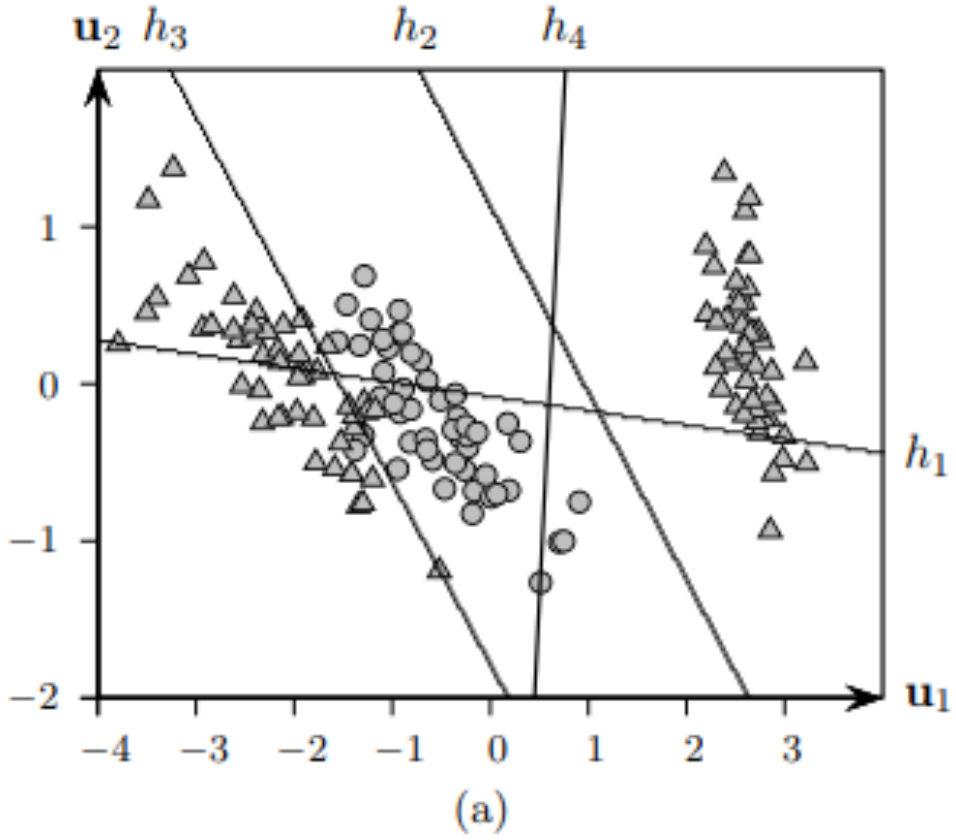
- How boosting works?
  - **Equal Weights** are assigned to each training tuple
  - A series of **k** classifiers is iteratively learned
    - Sample data based on the **weights**
    - Build classifier $\mathbf{M_i}$
    - Weights updated for the whole dataset, $\mathbf{M_{i+1}}$,
    - Until built k classifiers
  - The final M* combines the votes of each individual classifier, where the **weight** of each classifier's vote is a function of its **accuracy**

Boosting

Weight	Rec ID	Attribs.	Class	Correct?
1	100	…	Yes	✓
1	101	…	Yes	✓
1	102	…	Yes	✗
1	103	…	No	✓
1	104	…	No	✓

Weight	Rec ID	Attribs.	Class	Correct?
1	100	…	Yes	✓
1	101	…	Yes	✗
1.2	102	…	Yes	✓
1	103	…	No	✓
1	104	…	No	✓

Boosting – SVM (Linear kernel)



M_t	h_1	h_2	h_3	h_4
e_t	0.280	0.305	0.174	0.282

combined model	$\mathbf{M}^1$	$\mathbf{M}^2$	$\mathbf{M}^3$	$\mathbf{M}^4$
training error rate	0.280	0.253	0.073	0.047

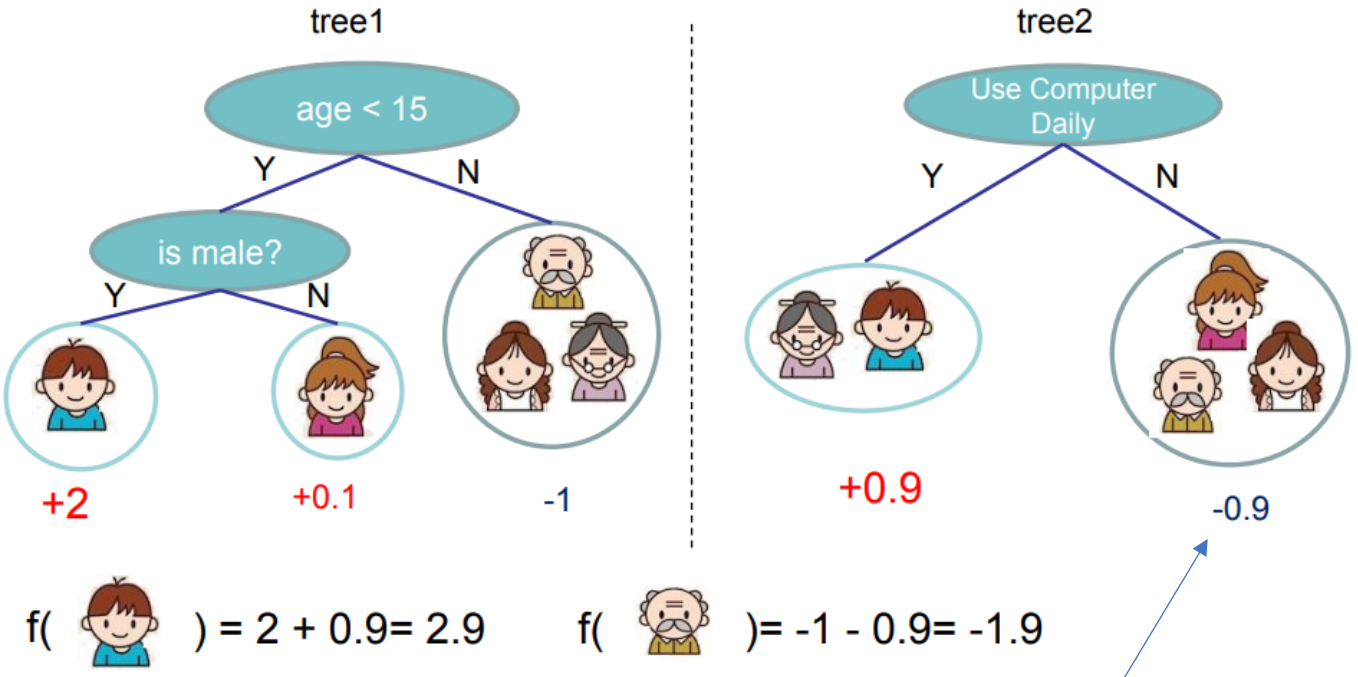
Boosting

In bagging, all of the predictors are equal.

In boosting, the predictors form a sequence, with the later predictors `specializing' in difficult objects.

- Boosting algorithm can be extended for numeric prediction
- Comparing with bagging: boosting tends to achieve **greater accuracy**, but it also **risks overfitting** the model to misclassified data

Tree Ensemble (Regression)



$$\hat{y}_i = \phi(\mathbf{x}_i) = \sum_{k=1}^{K} f_k(\mathbf{x}_i), \quad f_k \in \mathcal{F}, \quad \mathcal{F} = \{f(\mathbf{x}) = w_{q(\mathbf{x})}\}(q : \mathbb{R}^m \rightarrow T, w \in \mathbb{R}^T)$$

Chen, T., & Guestrin, C. (2016, August). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 785-794).

Tree Boosting

- How should we learn the trees?
- Supervised learning => optimization

$$\text{obj}(\theta) = \sum_{i}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

Training loss

Regularization

Chen, T., & Guestrin, C. (2016, August). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 785-794).

Tree Boosting

- Parameters?

$$\hat{y}_i = \phi(\mathbf{x}_i) = \sum_{k=1}^{K} f_k(\mathbf{x}_i), \quad f_k \in \mathcal{F}, \quad \mathcal{F} = \{f(\mathbf{x}) = w_{q(\mathbf{x})}\}(q : \mathbb{R}^m \rightarrow T, w \in \mathbb{R}^T)$$

It is intractable to learn all the trees at once.

Boosting: additive strategy

Chen, T., & Guestrin, C. (2016, August). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 785-794).

Additive learning

- Let t denotes the time (round)

$$\hat{y}_i^{(0)} = 0$$
$$\hat{y}_i^{(1)} = f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i)$$
$$\hat{y}_i^{(2)} = f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i)$$
$$\dots$$
$$\hat{y}_i^{(t)} = \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)$$

- Reduce the problem of: optimizing all the tree
- To: which tree do we want at each step?

Chen, T., & Guestrin, C. (2016, August). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 785-794).

Boosting: additive strategy

- Objective at time step t (with MSE):

$$\text{obj}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^{t} \Omega(f_i)$$

$$= \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + \text{constant}$$

$$\text{obj}^{(t)} = \sum_{i=1}^{n} (y_i - (\hat{y}_i^{(t-1)} + f_t(x_i)))^2 + \sum_{i=1}^{t} \Omega(f_i)$$

$$= \sum_{i=1}^{n} [2(\hat{y}_i^{(t-1)} - y_i) f_t(x_i) + f_t(x_i)^2] + \Omega(f_t) + \text{constant}$$

Chen, T., & Guestrin, C. (2016, August). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 785-794).

Boosting: additive strategy

- Objective at time step t (general case):

$$\text{obj}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^{t} \Omega(f_i)$$

$$\text{obj}^{(t)} = \sum_{i=1}^{n} [l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) + \text{constant}$$

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

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

In terms of MSE:

$$g_i = \partial_{\hat{y}^{(t-1)}} (\hat{y}^{(t-1)} - y_i)^2 = 2(\hat{y}^{(t-1)} - y_i) \quad h_i = \partial^2_{\hat{y}^{(t-1)}} (y_i - \hat{y}^{(t-1)})^2 = 2$$

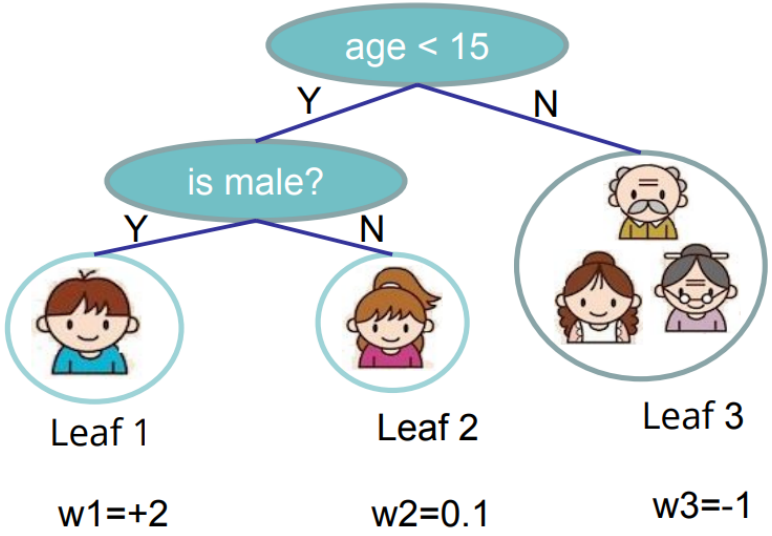
Chen, T., & Guestrin, C. (2016, August). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 785-794).

Tree Complexity

$$\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

Number of leaves

L2 norm of leaf scores



$$\Omega = \gamma 3 + \frac{1}{2}\lambda(4 + 0.01 + 1)$$

Chen, T., & Guestrin, C. (2016, August). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 785-794).

Structure Score

$$\text{obj}^{(t)} = \sum_{i=1}^{n} [l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) + \text{constant}$$

$$\text{obj}^{(t)} \approx \sum_{i=1}^{n} [g_i w_{q(x_i)} + \frac{1}{2} h_i w_{q(x_i)}^2] + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^{T} w_j^2$$

$$= \sum_{j=1}^{T} [(\sum_{i \in I_j} g_i) w_j + \frac{1}{2} (\sum_{i \in I_j} h_i + \lambda) w_j^2] + \gamma T$$

$I_j = \{i | q(x_i) = j\}$ is the set of indices of data points assigned to the j-th leaf.

Chen, T., & Guestrin, C. (2016, August). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 785-794).

Structure Score

$$\text{obj}^{(t)} \approx \sum_{i=1}^{n}[g_i w_{q(x_i)} + \frac{1}{2}h_i w_{q(x_i)}^2] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

$$= \sum_{j=1}^{T}[(\sum_{i\in I_j} g_i)w_j + \frac{1}{2}(\sum_{i\in I_j} h_i + \lambda)w_j^2] + \gamma T$$

In terms of MSE:

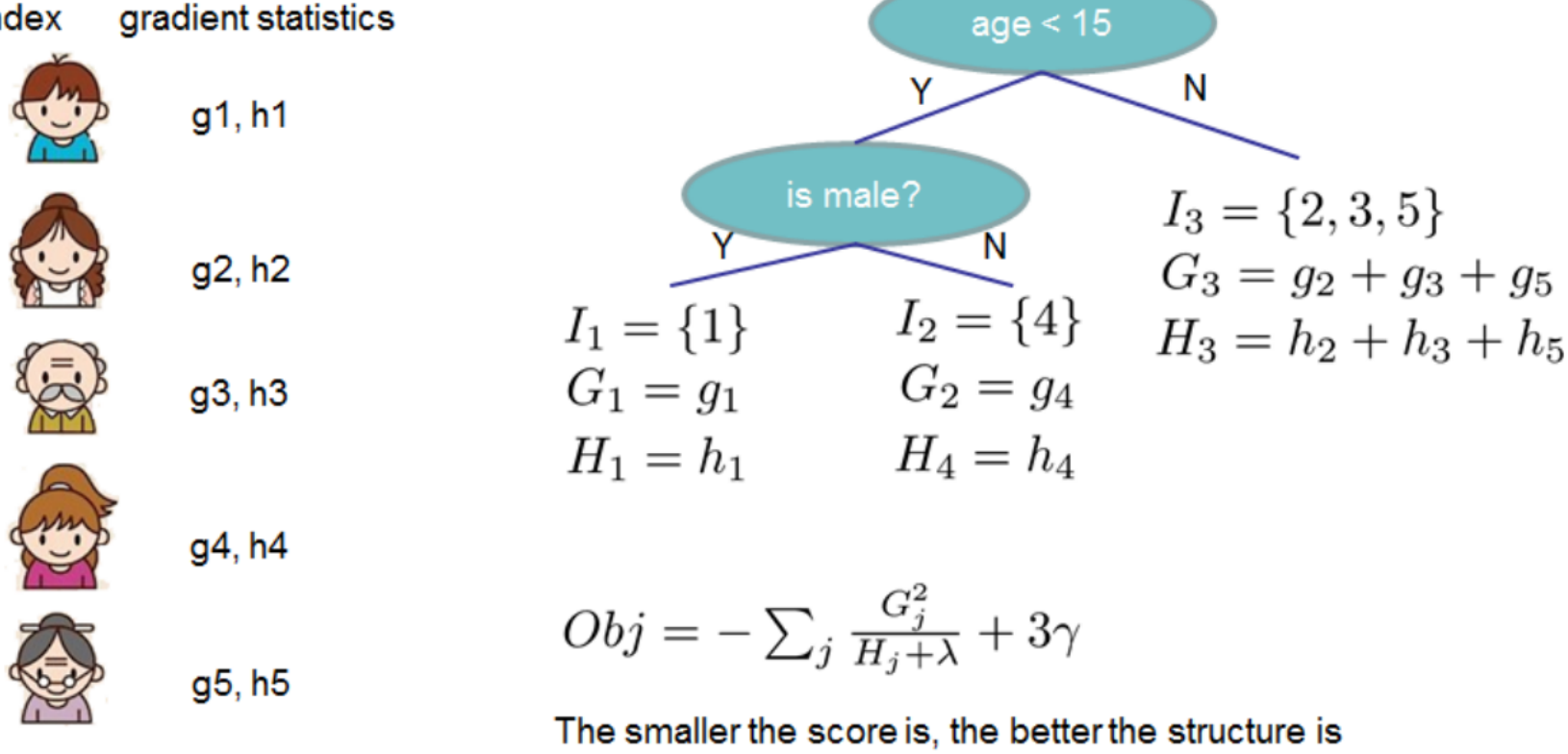
$$g_i = \partial_{\hat{y}^{(t-1)}}(\hat{y}^{(t-1)} - y_i)^2 = 2(\hat{y}^{(t-1)} - y_i) \quad h_i = \partial^2_{\hat{y}^{(t-1)}}(y_i - \hat{y}^{(t-1)})^2 = 2$$

$$G_j = \sum_{i\in I_j} g_i \qquad H_j = \sum_{i\in I_j} h_i$$

$$\text{obj}^* = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j + \lambda} + \gamma T$$

Chen, T., & Guestrin, C. (2016, August). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 785-794).

Instance index	gradient statistics
1	g1, h1
2	g2, h2
3	g3, h3
4	g4, h4
5	g5, h5



$$Obj = -\sum_j \frac{G_j^2}{H_j + \lambda} + 3\gamma$$

The smaller the score is, the better the structure is

In terms of MSE:

$$g_i = \partial_{\hat{y}^{(t-1)}} (\hat{y}^{(t-1)} - y_i)^2 = 2(\hat{y}^{(t-1)} - y_i) \quad h_i = \partial^2_{\hat{y}^{(t-1)}} (y_i - \hat{y}^{(t-1)})^2 = 2$$

Chen, T., & Guestrin, C. (2016, August). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 785-794).

Tree?

- Naïve approach:
  - Enumerate all possible trees
  - Calculate the score
  - Find the best Tree
  - Intractable

- Optimizing the Tree Structure Itself:

$$Gain = \frac{1}{2}\left[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}\right] - \gamma$$

Chen, T., & Guestrin, C. (2016, August). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 785-794).

Tree?

- Naïve approach:
  - Enumerate all possible trees
  - Calculate the score
  - Find the best Tree
  - Intractable

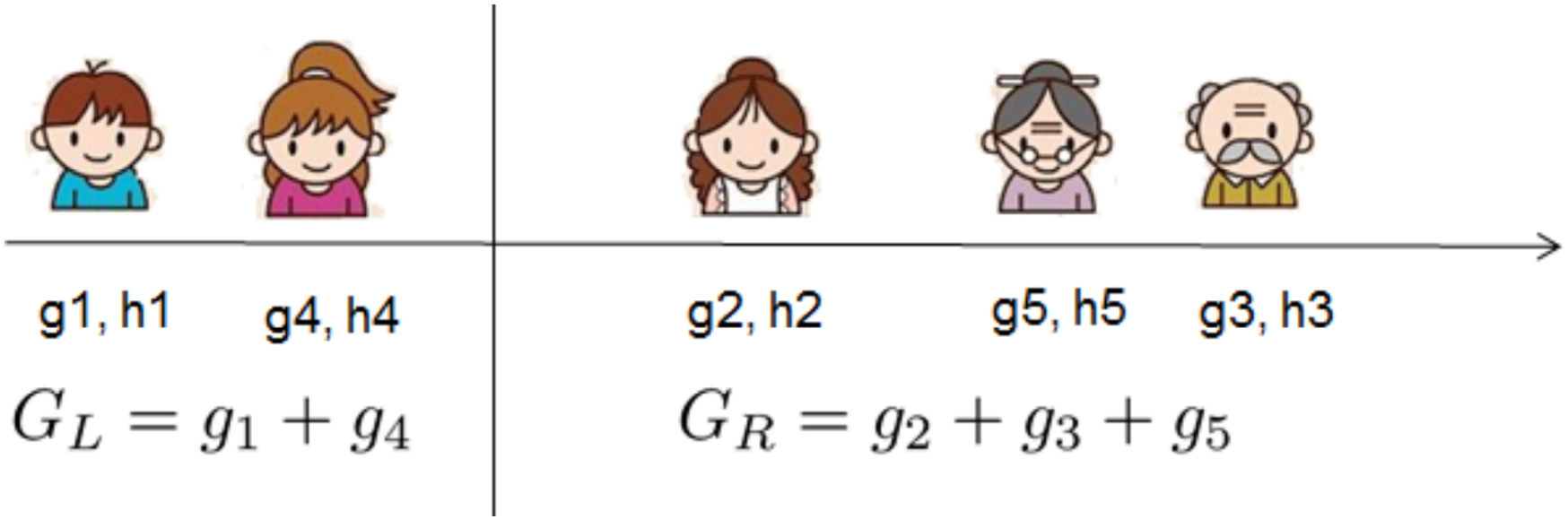
- Optimizing the Tree Structure Itself:

$$Gain = \frac{1}{2}\left[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}\right] - \gamma$$

Chen, T., & Guestrin, C. (2016, August). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 785-794).

Search for Optimal Split

$$Gain = \frac{1}{2}\left[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}\right] - \gamma$$



Chen, T., & Guestrin, C. (2016, August). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 785-794).

DATA MINING

/CISC 873 - Steven Ding
/Week #4/Lecture 1
Tuning Methods

Tuning Hyperparameters

- Using the dataset for tuning hyperparameters
  - Recall – The data science workflow
  - With a training set and a testing set
    - Cross-validation on training set for tuning
    - .623 bootstrapping for tuning
  - With a training set, a validation set, and a testing set
    - Training set is for training
    - Validation used for error estimation
    - Based on the estimated error, adjust hyperparameters
    - Testing set used for final testing (like the leaderboard)

Hypermeter Search Algorithm

- The space of hyperparameter is large:
  - Number of trees: (can be any positive number)
  - Regularization weight: (can be any number)
  - Different configurations: (kernels options, discrete choice)
  - Preprocessing options
- Educated guess
  - Guess the value to start with
  - Or guess the range of the values to start with
- Automated search

Hypermeter Search Algorithm

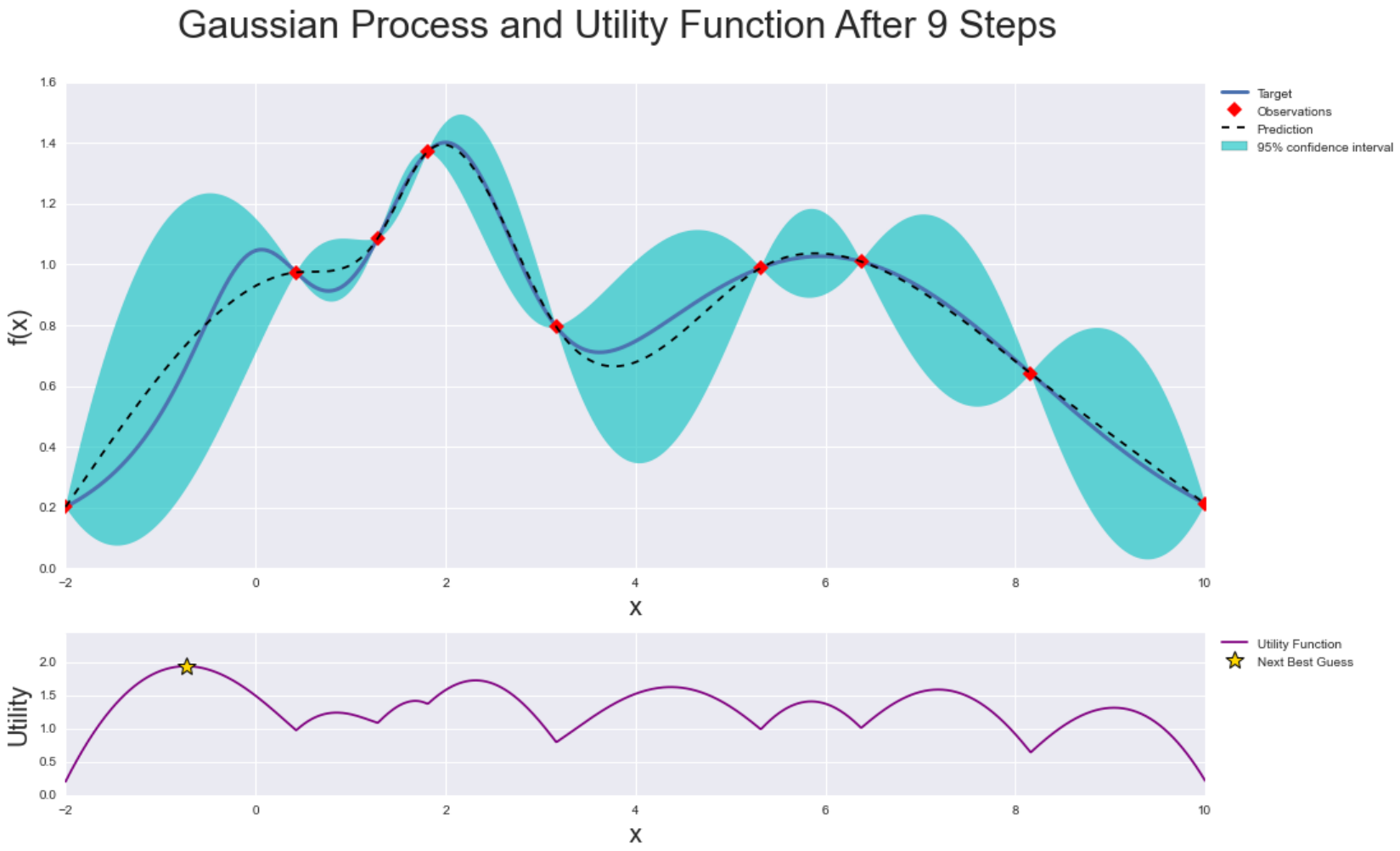
- Automated search
  - Grid search
    - Try out every combination of the parameters:
    - Computationally expensive
    - Global optimal (within the given range)
    - Sklearn: *model_selection.GridSearchCV*
  - Random search
    - Try out a random subset
    - `good enough`
    - Local optimal (within the given range)
    - Efficient (less trials)
    - Sklearn: *model_selection.RandomizedSearchCV*
  - Bayesian Optimization
    - As an optimization problem
    - Trial -> estimated error -> Bayesian model estimates the next parameter to try -> trial -> repeat..
    - *pip install bayesian-optimization*

Bayesian Optimization

- Loss is also a function of hyper-params
  - But *non-differentiable* so cannot be directly optimized.

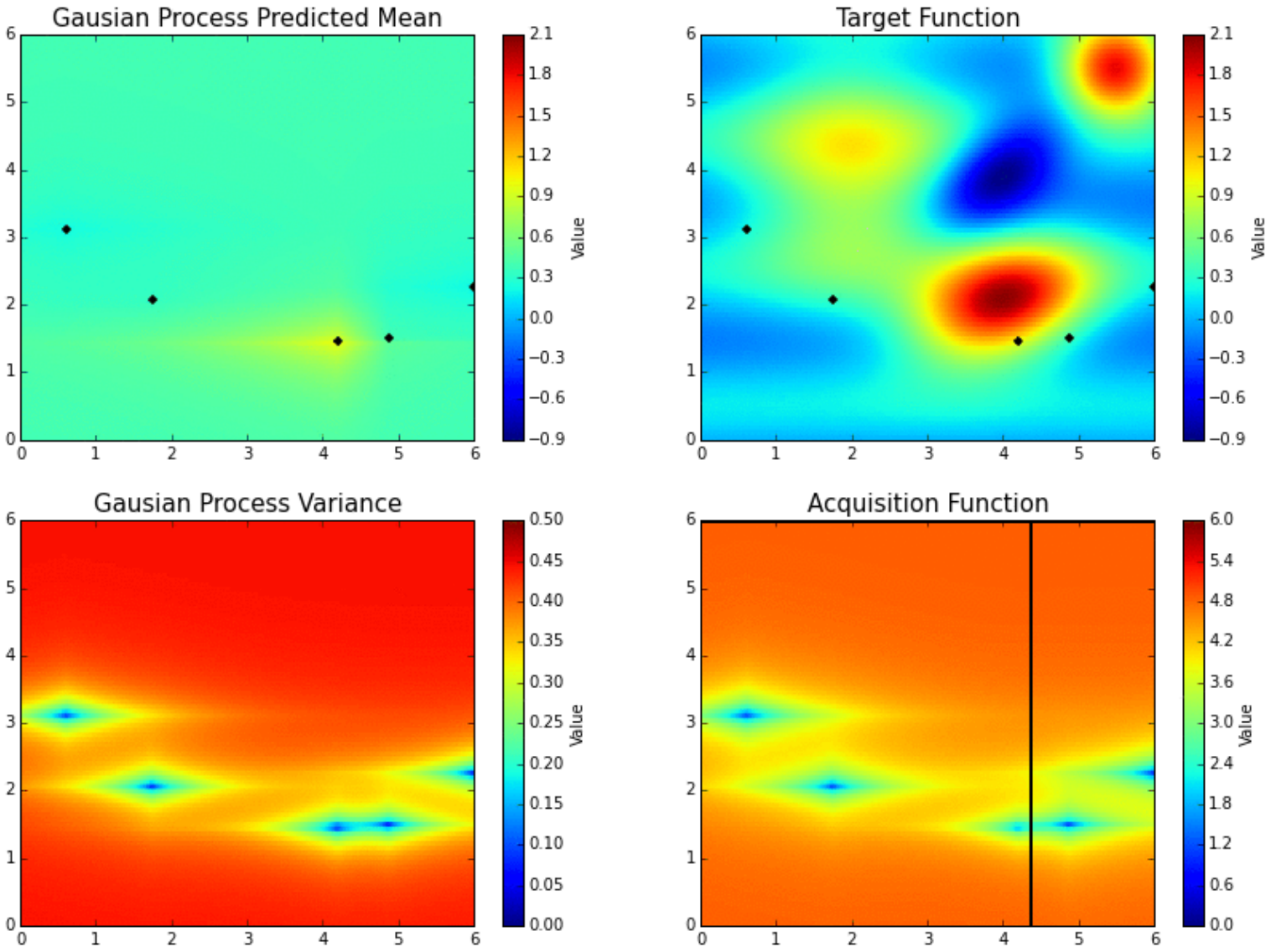
- Posterior distribution of functions (gaussian process) that best describes the loss function
- More observation -> more accurate “description”
- Given current “description”, best guess of the region to explore (get more observations)

Bayesian Optimization



https://github.com/fmfn/BayesianOptimization

Bayesian Optimization in Action



https://github.com/fmfn/BayesianOptimization

DATA MINING

/CISC 873 - Steven Ding
/Week #4/Lecture 1
Max-Margin, SVM, and Kernels