

Foundation of Data Science

Lecture 5, Module 2

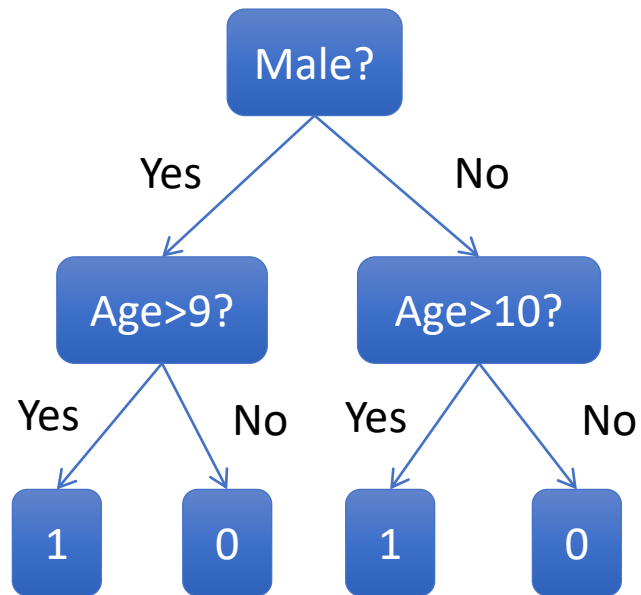
Spring 2022

Rumi Chunara, PhD

Fine Print: these slides are, and always will be a work in progress. The material presented herein is original, inspired, or borrowed from others' work (in this module from Prof. Yisong Yue, Caltech). Where possible, attribution and acknowledgement will be made to content's original source. Do not distribute without the instructor's permission.

Supervised Learning

- **Goal:** learn predictor $h(x)$
 - High accuracy (low error)
 - Using training data $\{(x_1, y_1), \dots, (x_n, y_n)\}$



Person	Age	Male?	Height > 55"	
Alice	14	0	1	✓
Bob	10	1	1	✓
Carol	13	0	1	✓
Dave	8	1	0	✓
Erin	11	0	0	✗
Frank	9	1	1	✗
Gena	8	0	0	✓

$$x = \begin{bmatrix} \text{age} \\ 1_{[\text{gender}=\text{male}]} \end{bmatrix} \quad y = \begin{cases} 1 & \text{height} > 55'' \\ 0 & \text{height} \leq 55'' \end{cases}$$

Different Classifiers

- Performance
 - No classifier is perfect
 - Complementary
 - Examples which are not correctly classified by one classifier may be correctly classified by the other classifiers
- Potential Improvements?
 - Utilize the complementary property

Ensemble Methods

- Currently using *one single classifier* induced from training data as our model, to predict class of test instance
- What if we used *multiple* decision trees?
- *Motivation*: committee of experts working together are likely to better solve a problem than a single expert
 - But no “group think”: each model should make predictions independently of other models in the ensemble
- *In practice*: methods work surprisingly well, usually greatly improve decision tree accuracy

Ensemble Characteristics

1. Build *multiple* models from the same training data by creating each model on a *modified* version of the training data.
2. Make a final, ensemble prediction by aggregating the predictions of the individual models
 - Classification prediction: Let each model have a vote on the correct class prediction. Assign the class with the most votes.
 - Regression prediction: Measure of central tendency (mean or median)

Ensembles of Classifiers

- Idea
 - Combine the classifiers to improve the performance
- Ensembles of Classifiers
 - Combine the classification results from different classifiers to produce the final output
 - Unweighted voting
 - Weighted voting

Ensembles: Rationale

- How can an ensemble method improve a classifier's performance?
 - Assume we have 25 binary classifiers
 - Each has error rate: $\epsilon = 0.35$
1. If all 25 classifiers are identical:
 - They will vote the same way on each test instance
 - Ensemble error rate: $\epsilon = 0.35$


















































Rationale

- How can an ensemble method improve a classifier's performance?
 - Assume we have 25 binary classifiers
 - Each has error rate: $\varepsilon = 0.35$
2. If all 25 classifiers are independent (errors are uncorrelated):
- Ensemble method only makes a wrong prediction if more than half of the base classifiers predict incorrectly.

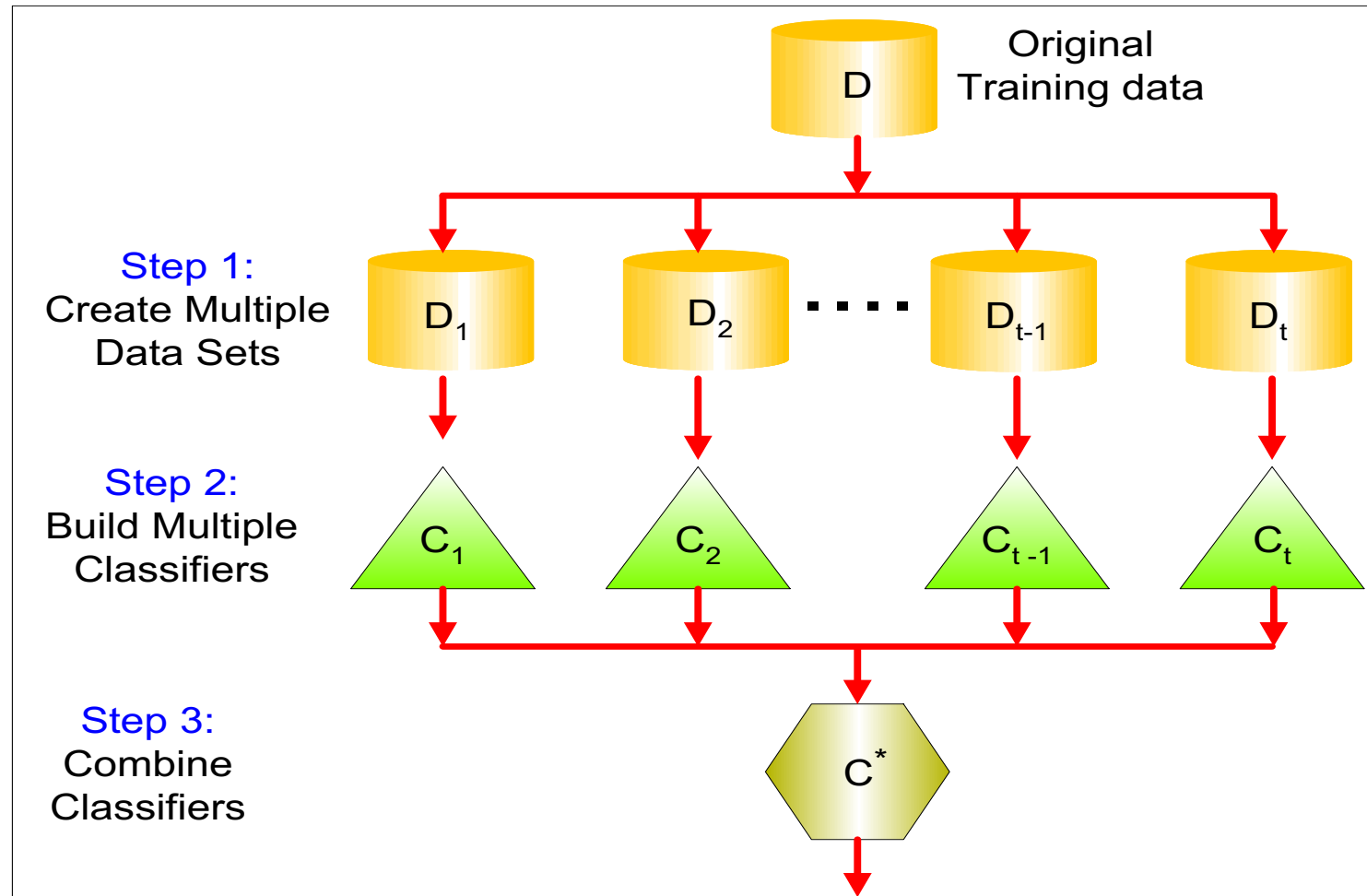
$$e_{\text{ensemble}} = \sum_{i=13}^{25} \binom{25}{i} \varepsilon^i (1 - \varepsilon)^{25-i} = 0.06$$

6% much less than 35%

Example: Weather Forecast

Reality							
1							
2							
3							
4							
5							
Combine							

Methods for Constructing an Ensemble Classifier



Methods for Constructing an Ensemble Classifier

1. By manipulating the training set.
2. By manipulating the input features.
3. By manipulating the class labels.
4. By manipulating the learning algorithm.

Methods for Constructing an Ensemble Classifier

1. **By manipulating the training set.**
2. By manipulating the input features.
3. By manipulating the class labels.
4. By manipulating the learning algorithm.

Methods for Constructing an Ensemble Classifier

1. By manipulating the training set.

- Multiple training sets created by resampling original data
- Resample according to some sampling distribution
 - *Example:* equal probability
 - *Example:* weighted
 - Determines how likely it is that an example will be selected for training.
- Classifier built from each training set.
- Ensemble methods that manipulate the training set:
 1. Bagging
 2. Boosting

Methods for Constructing an Ensemble Classifier

2. By manipulating the input features.

- Subset of input features is chosen at random from overall collection of features
- Each training set has different feature set
- Ensemble method that manipulates input features:
 1. Random Forest
 - Works well with datasets that contain highly redundant features

Methods for Constructing an Ensemble Classifier

3. By manipulating the class labels.

- Used when large number of classes
- Transform into many binary class problems
- Training Approach:
 1. Randomly partition class labels into two disjoint subsets A_0 (class 0) and A_1 (class 1)
 2. Train a base classifier based on this class reassignment.
 3. Repeat multiple times, once for each base classifier.
- Testing Approach:
 1. Each base classifier predicts test instance with its respective binary class subset
 2. All classes in subset receive a vote
 3. Class with highest count wins

Methods for Constructing an Ensemble Classifier

4. By manipulating the learning algorithm.
 - ... so that applying algorithm on same training data may result in different models
 - How to introduce randomness into decision tree induction?
 - Instead of choose best splitting attribute at each node, randomly choose one of top k attributes for splitting

Boosting

- Iteratively creating models and adding them to the ensemble
- Iteration stops when a predefined number of models have been added
- Each new model added to the ensemble is biased to pay more attention to instances that previous models misclassified (weighted dataset).

General Boosting Algorithm

1. Initially instances are assigned weights of $1/N$
 - Each is equally likely to be chosen for sample
2. Sample drawn *with replacement*: D_i
3. Classifier induced on D_i
4. Weights of training examples are updated:
 - Instances classified incorrectly have weights increased
 - Instances classified correctly have weights decreased

General Boosting Algorithm

During each iteration the algorithm:

1. Induces a model and calculates the total error, ϵ , by summing the weights of the training instances for which the predictions made by the model are incorrect.
2. Increases the weights for the instances misclassified
$$\mathbf{w}[i] \leftarrow \mathbf{w}[i] \times \left(\frac{1}{2 \times \epsilon} \right)$$
3. Decreases the weights for the instances correctly classified
$$\mathbf{w}[i] \leftarrow \mathbf{w}[i] \times \left(\frac{1}{2 \times (1 - \epsilon)} \right)$$
4. Calculate a confidence factor α , for the model such that α increases as ϵ decreases
$$\alpha = \frac{1}{2} \times \log_e \left(\frac{1 - \epsilon}{\epsilon} \right)$$

Boosting Example

Instances

Boosting (Round 1)	7	3	2	8	7	9	4	10	6	3
--------------------	---	---	---	---	---	---	---	----	---	---

- Suppose that **Instance #4** is hard to classify.
- Weight for this instance will be increased in future iterations, as it gets misclassified repeatedly.
- Examples not chosen in previous round (e.g. *Instances #1, #5*) also may have better chance of being selected in next round.
 - *Why?* Predictions in previous round are likely to be wrong since they weren't trained on.

Boosting (Round 2)	5	4	9	4	2	5	1	7	4	2
--------------------	---	---	---	---	---	---	---	---	---	---

Boosting (Round 3)	4	4	8	10	4	5	4	6	3	4
--------------------	---	---	---	----	---	---	---	---	---	---

- As boosting rounds proceed, instances that are the hardest to classify become even more prevalent.

Prediction

- Once the set of models have been created the ensemble makes predictions using a weighted aggregate of the predictions made by the individual models.
- The weights used in this aggregation are simply the confidence factors associated with each model.

Boosting Algorithms

- Several different boosting algorithms exist
- Different by:
 1. How weights of training instances are updated after each boosting round
 2. How predictions made by each classifier are combined
 - Each boosting round produces one base classifier

AdaBoost

- AdaBoost is a popular boosting algorithm
- Regarding predictions of final ensemble classifier:
 - Importance of a base classifier depends on its error rate

$$\varepsilon_i = \frac{1}{N} \left[\sum_{j=1}^N \omega_j I(C_i(x_j)) \neq y_j \right]$$

$I(p) = 1$ if predicate p is true, and 0 otherwise

$$\alpha_i = \frac{1}{2} \ln \left(\frac{1 - \varepsilon_i}{\varepsilon_i} \right)$$

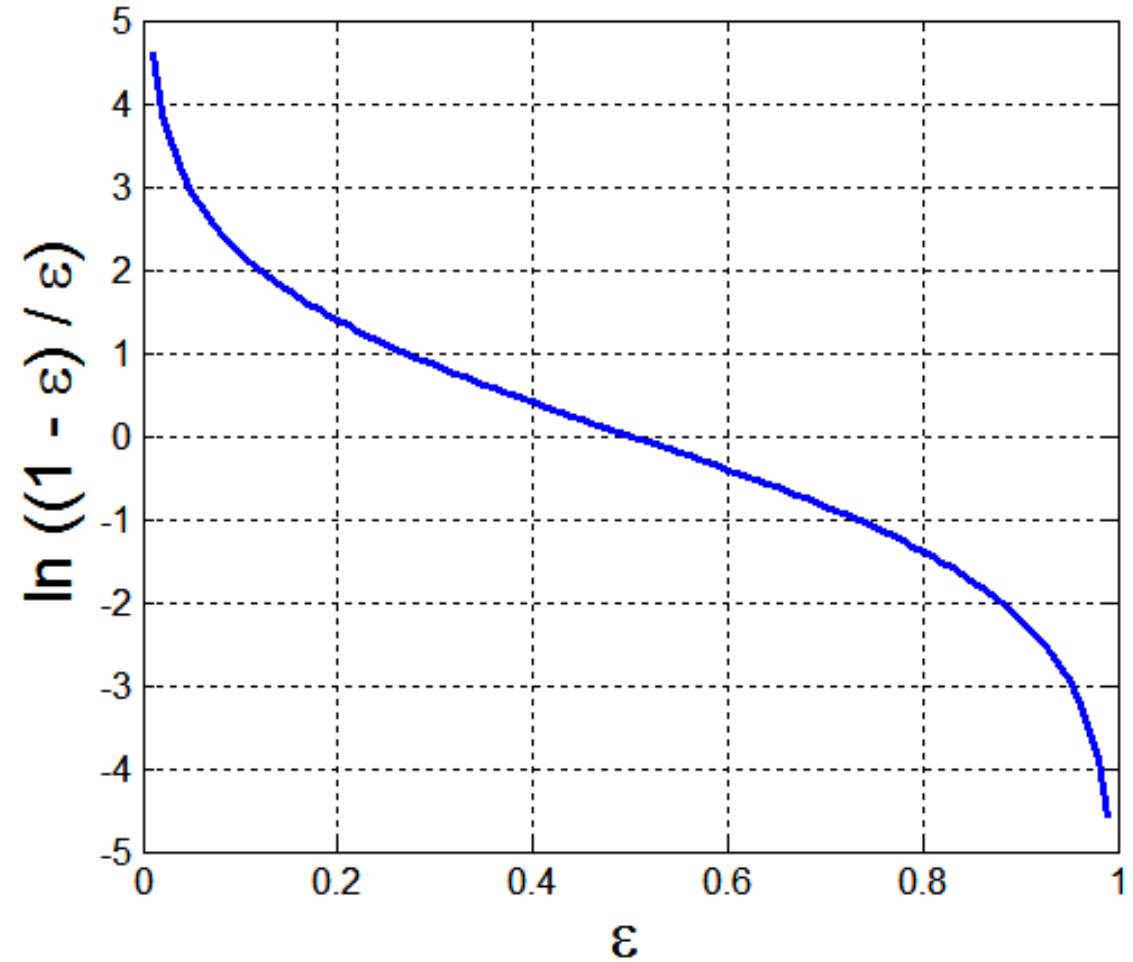
C_i : Base Classifier

ε_i : error rate

α_i : importance of classifier

AdaBoost

- α_i has a large positive value if error rate is close to 0
- α_i has a large negative value if error rate is close to 1



AdaBoost

- α_i also used to update weight of training examples after each boosting round

$$\omega_i^{(j+1)} = \frac{\omega_i^{(j)}}{Z_j} \times \begin{cases} \exp^{-\alpha_j} & \text{if } C_j(x_i) = y_i \\ \exp^{\alpha_j} & \text{if } C_j(x_i) \neq y_i \end{cases}$$

j : current round

$j+1$: next round

Z_j : normalization factor

$$\sum_i \omega_i^{(j+1)} = 1$$

- Increases weights of incorrectly classified instances
- Decreases weights of correctly classified instances

Boosting Algorithm

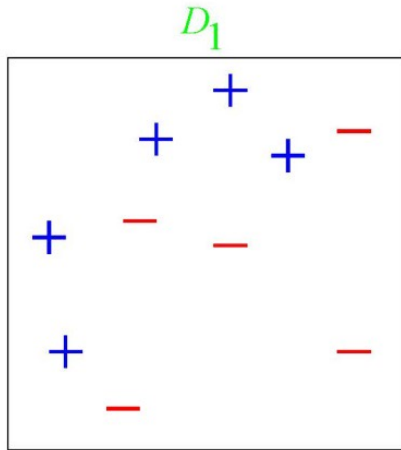
From Tan, Alg. 5.7

Algorithm 5.7 AdaBoost algorithm.

- 1: $\mathbf{w} = \{w_j = 1/N \mid j = 1, 2, \dots, N\}$. {Initialize the weights for all N examples.}
 - 2: Let k be the number of boosting rounds.
 - 3: **for** $i = 1$ to k **do**
 - 4: Create training set D_i by sampling (with replacement) from D according to \mathbf{w} .
 - 5: Train a base classifier C_i on D_i .
 - 6: Apply C_i to all examples in the original training set, D .
 - 7: $\epsilon_i = \frac{1}{N} [\sum_j w_j \delta(C_i(x_j) \neq y_j)]$ {Calculate the weighted error.}
 - 8: **if** $\epsilon_i > 0.5$ **then**
 - 9: $\mathbf{w} = \{w_j = 1/N \mid j = 1, 2, \dots, N\}$. {Reset the weights for all N examples.}
 - 10: Go back to Step 4.
 - 11: **end if**
 - 12: $\alpha_i = \frac{1}{2} \ln \frac{1-\epsilon_i}{\epsilon_i}$.
 - 13: Update the weight of each example according to Equation 5.69.
 - 14: **end for**
 - 15: $C^*(\mathbf{x}) = \operatorname{argmax}_y \sum_{j=1}^T \alpha_j \delta(C_j(\mathbf{x}) = y)$.
-

AdaBoost

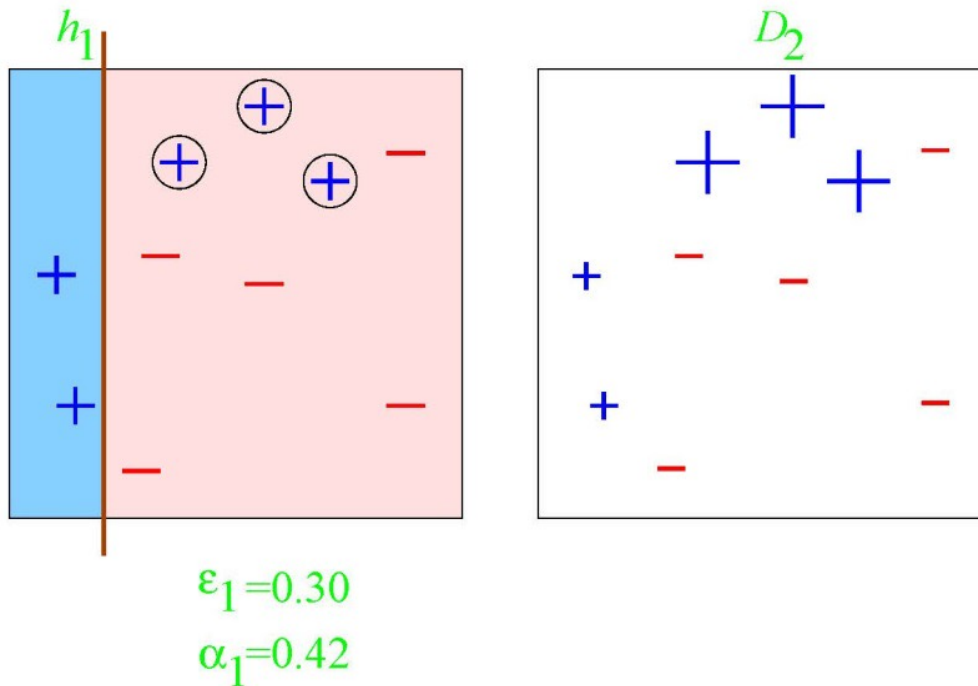
Toy Example



weak classifiers = vertical or horizontal half-planes

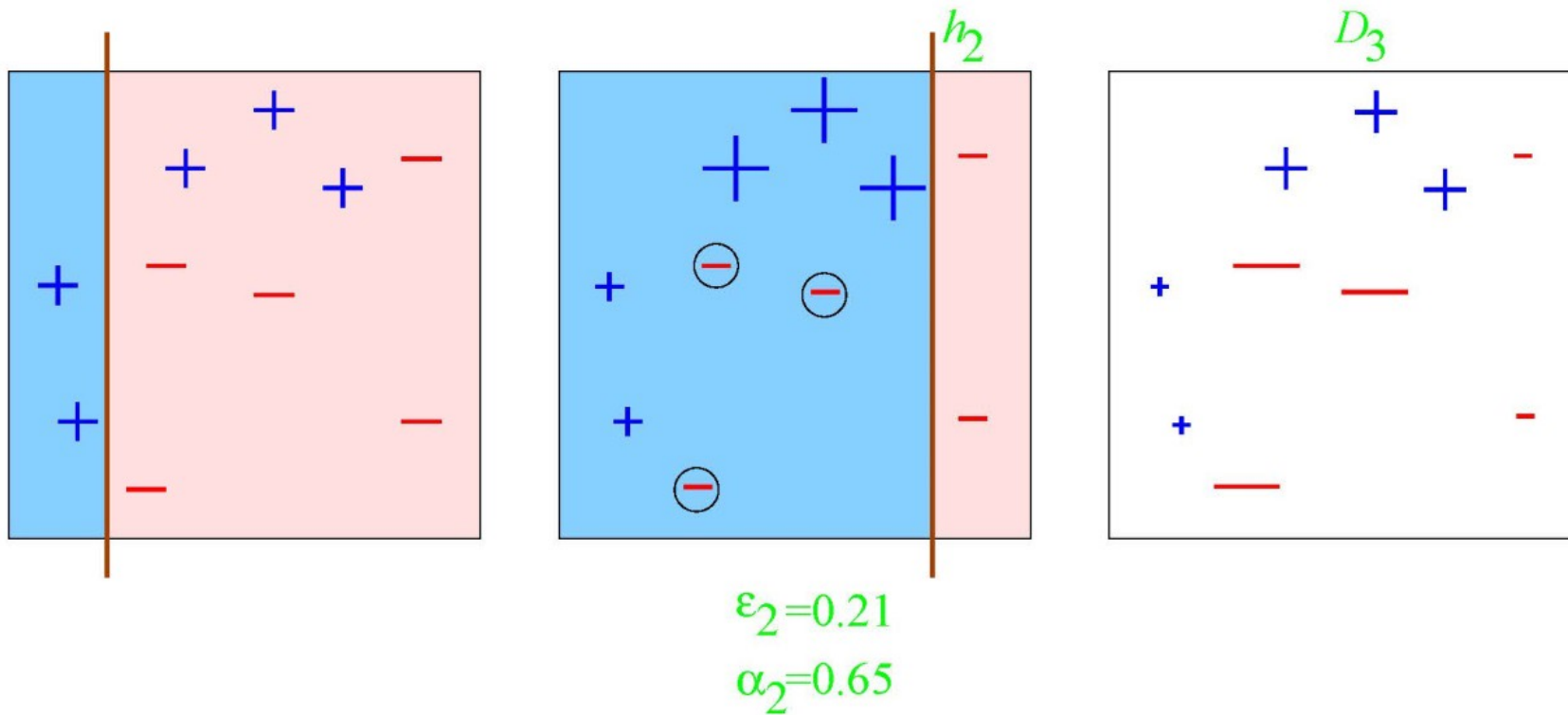
AdaBoost: visual example

Round 1



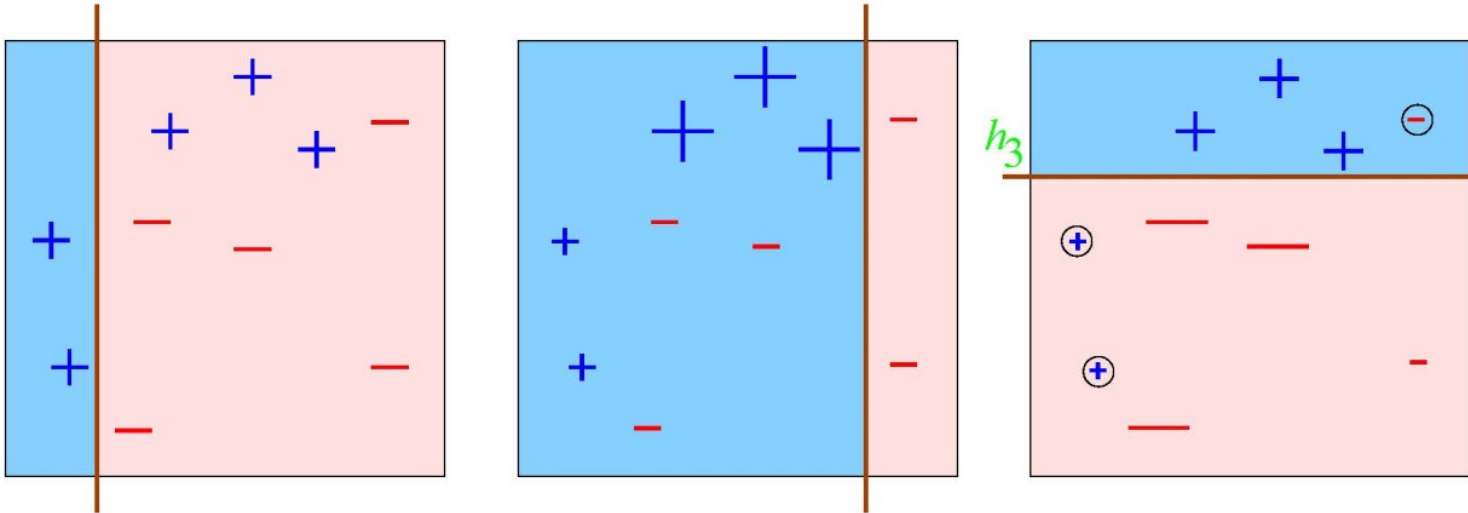
AdaBoost: visual example

Round 2



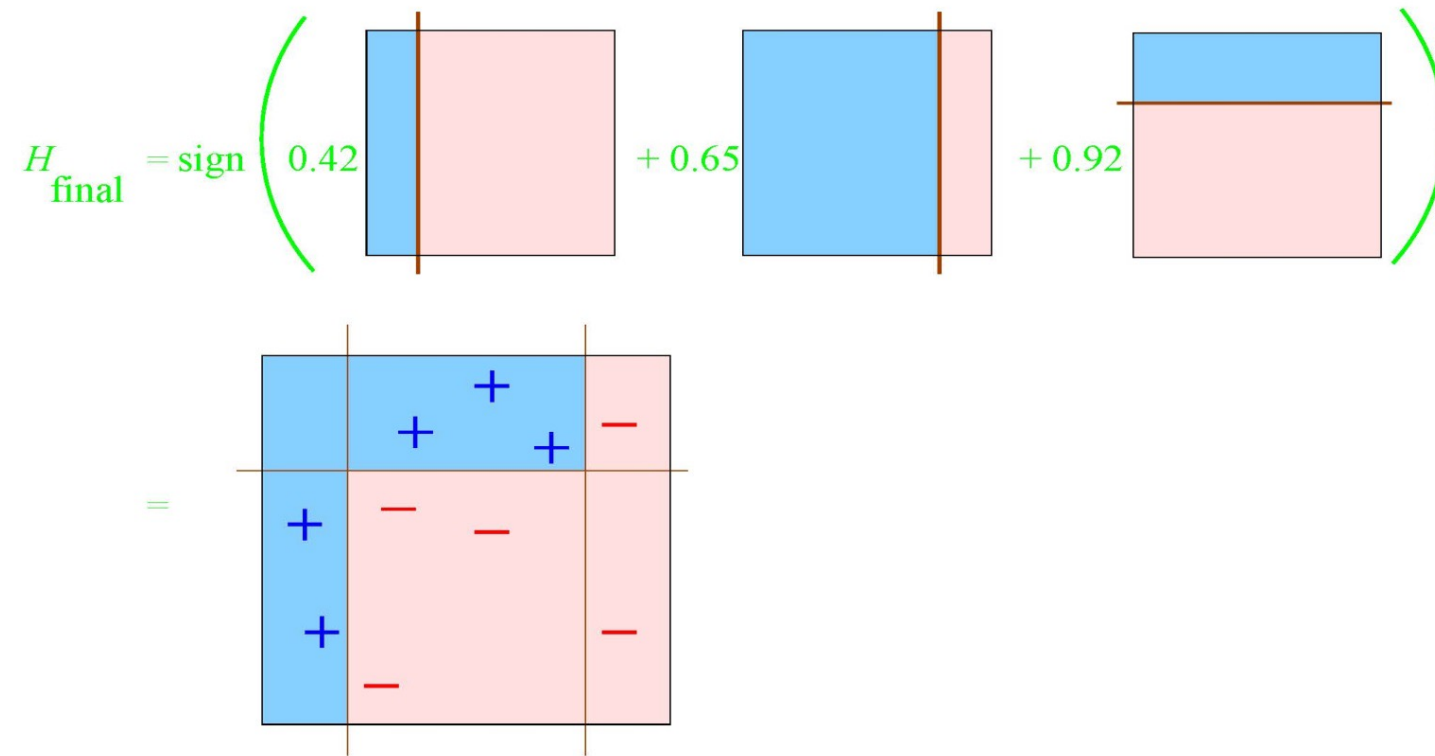
AdaBoost: visual example

Round 3



AdaBoost: visual example

Final Classifier

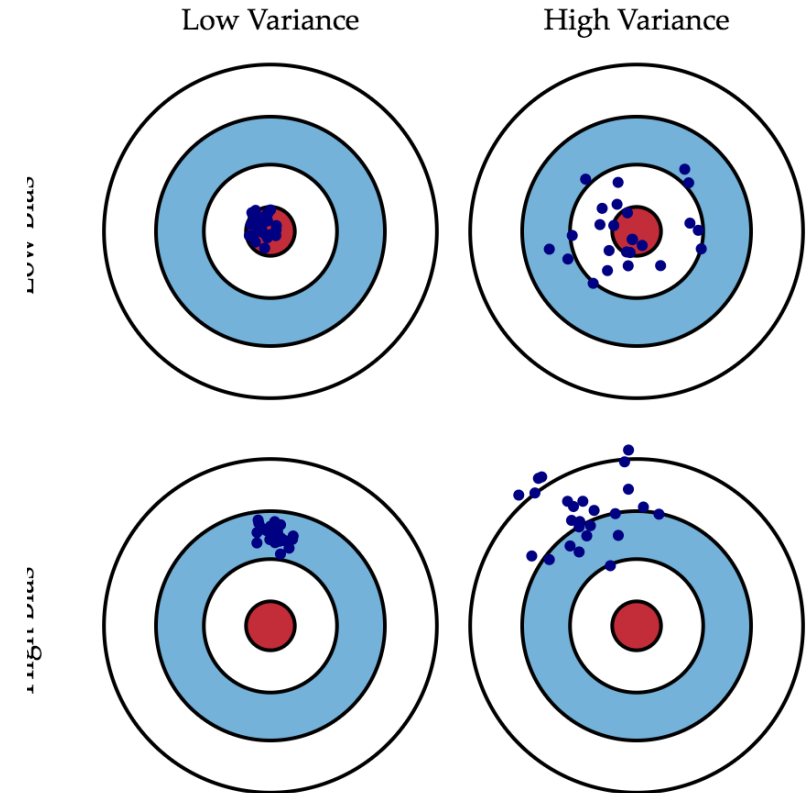


AdaBoost vs. XGBoost

- Adaboost was the original implementation of boosting, with a single cost function. Not easy to adapt to different link functions, and slow.
- Gradient boosting faster (speeds up gradient descent, etc.), can use multiple base learners (trees, linear terms, splines, etc.), and cost and link functions are modifiable.

Boosting and error

- Boosting is especially useful in models that exhibit underfitting. These models are highly biased and have low variance.
- However, a boosting based model aims to reduce both. It is a weighted average of the predictions from multiple weak models and if these weak models are independent then it reduces the prediction error and thus both bias and variance.
- Because the weak models are often low bias (and low variance), boosting does not generally add to bias



General Bagging Algorithm

1. Randomly sample training data
2. Determine classifier C_i on sampled data
3. Go to step 1 and repeat m times
4. For final classifier output the majority vote
5. Popular example: random forest
6. Similar to tree bagging
 - Compute decision trees on bootstrapped datasets
 - Return majority vote

Bagging

On average, each D_i will contain 63% of original training data.
Probability of sample being selected for D_i : $1 - (1 - (1/N))^N$
• Converges to: $1 - 1/e = 0.632$

- Ensemble method that “manipulates the training set/features”
- *Action*: repeatedly sample with replacement according to uniform probability distribution
 - Every instance has equal chance of being picked
 - Some instances may be picked multiple times; others may not be chosen
- *Sample Size*: same as training set
- D_i : each bootstrap sample
- *Footnote*: also called bootstrap aggregating

Consequently, every bootstrap sample will be missing some of the instances from the dataset so each bootstrap sample will be different and this means that models trained on different bootstrap samples will also be different

Bagging Algorithm

Model Generation:

- Let n be the number of instances in the training data.
- For each of t iterations:
 - Sample n instances with replacement from training data.
 - Apply the learning algorithm to the sample.
 - Store the resulting model.

Classification:

- For each of the t models:
 - Predict class of instance using model.
- Return class that has been predicted most often.

Bagging Example

Now apply bagging and create many decision stump base classifiers.

Dataset: 10 instances

Predictor Variable: x

Target Variable: y

X	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
Y	1	1	1	-1	-1	-1	-1	1	1	1

- *Decision Stump*: one-level binary decision tree
- What's the best performance of a decision stump on this data?
- Splitting condition will be $x \leq k$, where k is the split point
 - Best splits: $x \leq 0.35$ or $x \leq 0.75$
 - Best accuracy: 60%

Bagging Example

- First choose how many “bagging rounds” to perform
 - Chosen by analyst
- We’ll do 10 bagging rounds in this example:

In each round, create D_i by sampling with replacement

Bagging Example

Round 1:

X	0.1	0.2	0.2	0.3	0.4	0.4	0.5	0.6	0.9	0.9
Y	1	1	1	1	-1	-1	-1	-1	1	1

Learn decision stump.

What stump will be learned?

If $x \leq 0.35$ then $y = 1$

If $x > 0.35$ then $y = -1$

X	0.1	0.2	0.2	0.3	0.4	0.4	0.5	0.6	0.9	0.9
Y	1	1	1	1	-1	-1	-1	-1	1	1
X	0.1	0.2	0.3	0.4	0.5	0.8	0.9	1	1	1
Y	1	1	1	-1	-1	1	1	1	1	1
X	0.1	0.2	0.3	0.4	0.4	0.5	0.7	0.7	0.8	0.9
Y	1	1	1	-1	-1	-1	-1	-1	1	1
X	0.1	0.1	0.2	0.4	0.4	0.5	0.5	0.7	0.8	0.9
Y	1	1	1	-1	-1	-1	-1	-1	1	1
X	0.1	0.1	0.2	0.5	0.6	0.6	0.6	1	1	1
Y	1	1	1	-1	-1	-1	-1	1	1	1
X	0.2	0.4	0.5	0.6	0.7	0.7	0.7	0.8	0.9	1
Y	1	-1	-1	-1	-1	-1	-1	1	1	1
X	0.1	0.4	0.4	0.6	0.7	0.8	0.9	0.9	0.9	1
Y	1	-1	-1	-1	-1	1	1	1	1	1
X	0.1	0.2	0.5	0.5	0.5	0.7	0.7	0.8	0.9	1
Y	1	1	-1	-1	-1	-1	-1	1	1	1
X	0.1	0.3	0.4	0.4	0.6	0.7	0.7	0.8	1	1
Y	1	1	-1	-1	-1	-1	-1	1	1	1
X	0.1	0.1	0.1	0.1	0.3	0.3	0.8	0.8	0.9	0.9
Y	1	1	1	1	1	1	1	1	1	1

If x <= 0.35 then y = 1
If x > 0.35 then y = -1
If x <= 0.65 then y = 1
If x > 0.65 then y = -1
If x <= 0.35 then y = 1
If x > 0.35 then y = -1
If x <= 0.3 then y = 1
If x > 0.3 then y = -1
If x <= 0.35 then y = 1
If x > 0.35 then y = -1
If x <= 0.75 then y = -1
If x > 0.75 then y = 1
If x <= 0.75 then y = -1
If x > 0.75 then y = 1
If x <= 0.75 then y = -1
If x > 0.75 then y = 1
If x <= 0.75 then y = -1
If x > 0.75 then y = 1
If x <= 0.05 then y = 1
If x > 0.05 then y = -1

10 bagging rounds. 10 D_i 's. 10 learned models.

Classify test instance by using each base classifier and taking majority vote.

Bagging Example

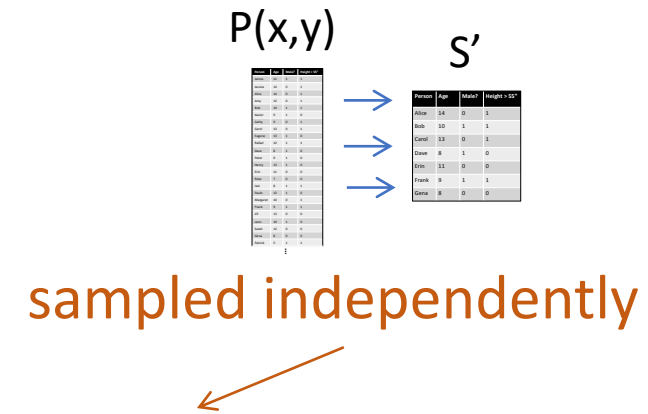
10 test instances. Let's see how each classifier votes:

100% overall ensemble method accuracy (*improvement from 60%*)

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	X=1.0
1	1	1	1	-1	-1	-1	-1	-1	-1	-1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
4	1	1	1	-1	-1	-1	-1	-1	-1	-1
5	1	1	1	-1	-1	-1	-1	-1	-1	-1
6	-1	-1	-1	-1	-1	-1	-1	1	1	1
7	-1	-1	-1	-1	-1	-1	-1	1	1	1
8	-1	-1	-1	-1	-1	-1	-1	1	1	1
9	-1	-1	-1	-1	-1	-1	-1	1	1	1
10	1	1	1	1	1	1	1	1	1	1
Sum	2	2	2	-6	-6	-6	-6	2	2	2
Sign	1	1	1	-1	-1	-1	-1	1	1	1
True	1	1	1	-1	-1	-1	-1	1	1	1

Bagging

- **Goal:** reduce variance
- **Ideal setting:** many training sets S'
 - Train model using each S'
 - Average predictions



Variance reduces linearly
Bias unchanged

$$E(\hat{y} - y)^2 = E[(\hat{y} - E[\hat{y}])^2] + E[(E[\hat{y}] - y)^2]$$

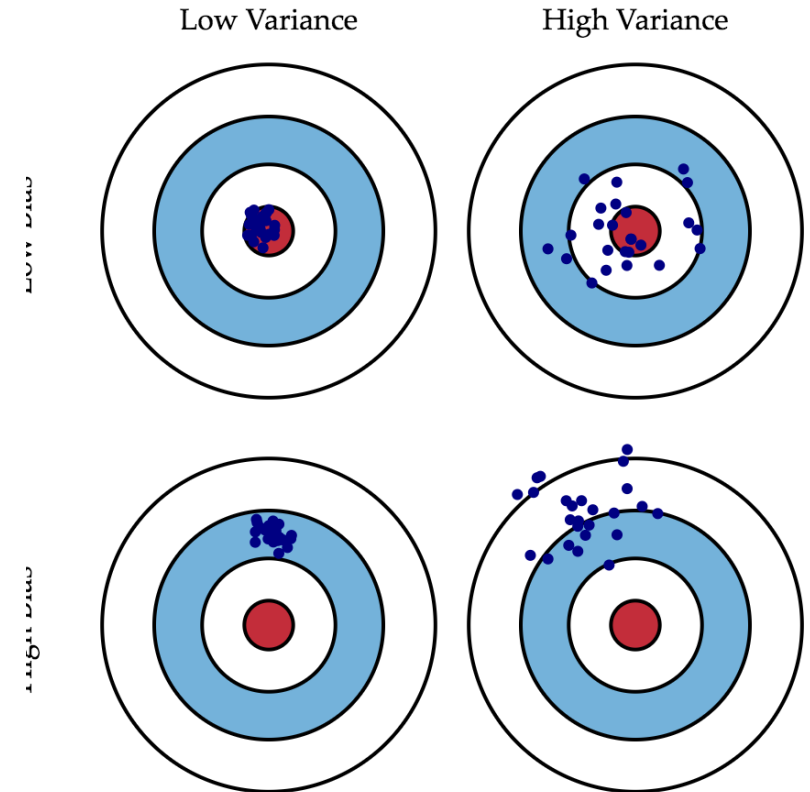
Expected Error Variance Bias

Bagging Summary

- *In Previous Example:* even though base classifier was decision stump ($depth=1$), bagging aggregated the classifiers to effectively learn a decision tree of $depth=2$
- Bagging helps to reduce variance
- Bagging does not focus on any particular instance of training data
 - less susceptible to model overfitting with noisy data
 - robust to minor perturbations in training set
- If base classifiers are stable (not much variance), bagging can degrade performance
 - Training set is $\sim 37\%$ smaller than original data

Bagging and error

- In bagging, we are often aggregating many low bias, high variance predictors (e.g. decision trees), therefore we can reduce the variance while retaining the low bias.



Bagging vs. Boosting

- Boosting: Start with simple model with low variance and high bias, add new models to reduce bias
 - *Approach*: focus on instances that are harder to classify
 - *How*: give harder instances more weight in future rounds
- Bagging: Construct many trees to reduce variance, with minimal effects on bias
 - *Approach*: use the data at hand (aka bootstrap aggregating)
 - *How*: Each instance has an equal chance of being selected

Random Forests

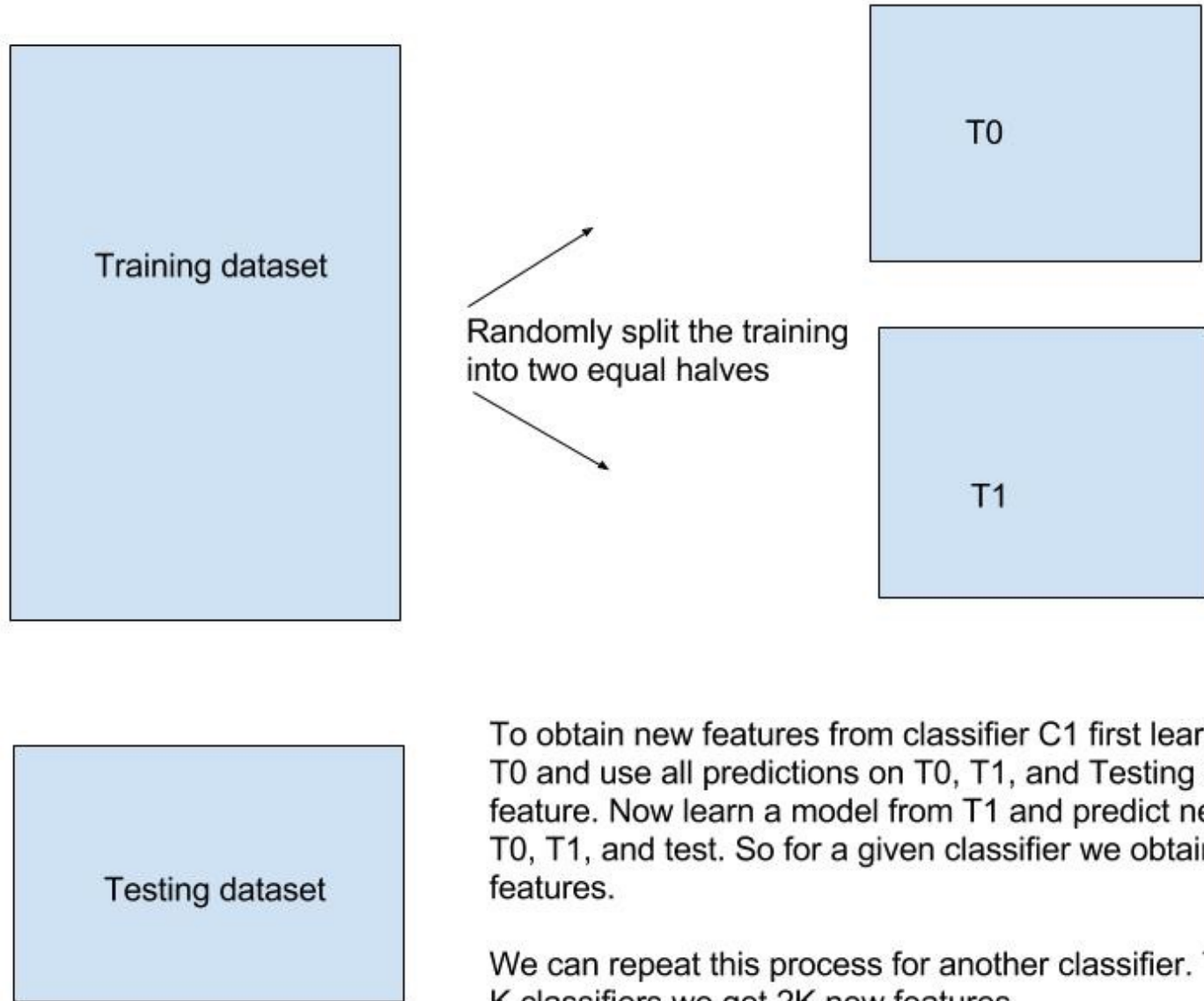
- **Goal:** reduce variance
 - Bagging can only do so much
 - Resampling training data asymptotes
- **Random Forests:** sample data & features!
 - Sample S'
 - Train Decision Tree
 - At each node, sample features (\sqrt{p})
 - Average predictions

Further de-correlates trees



Stacking

Starts with heterogeneous weak learners, learns them in parallel and combines them by training a meta-model to output a prediction based on the different weak model predictions



Algorithms

- Ensemble methods that minimize variance
 - Bagging
 - Random Forests
- Ensemble methods that minimize bias
 - Functional Gradient Descent: *Gradient descent except minimize over the space of possible functions.*
 - Boosting
 - Ensemble Selection: *Automatically select a subset of ensemble models just-in-time when making a prediction.*

Which to use?

- Which approach should we use? Bagging is simpler to implement and parallelize than boosting and, so, may be better with respect to ease of use and training time.
- Empirical results indicate:
 - boosted decision tree ensembles were the best performing model of those tested for datasets containing up to 4,000 descriptive features.
 - random forest ensembles (based on bagging) performed better for datasets containing more than 4,000 features.

Ensemble Methods Summary

- Advantages:
 - Astonishingly good performance
 - Modeling human behavior: making judgment based on many trusted advisors, each with their own specialty
- Disadvantage:
 - Combined models rather hard to analyze
 - Tens or hundreds of individual models
 - Current research: making models more interpretable

Ensemble Methods Summary

Method	Minimize Bias?	Minimize Variance?	Other Comments
Bagging	Complex model class. (Deep DTs)	Bootstrap aggregation (resampling training data)	Does not work for simple models.
Random Forests	Complex model class. (Deep DTs)	Bootstrap aggregation + bootstrapping features	Only for decision trees.
Gradient Boosting (AdaBoost)	Optimize training performance.	Simple model class. (Shallow DTs)	Determines which model to add at run- time.
Ensemble Selection	Optimize validation performance	Optimize validation performance	Pre-specified dictionary of models learned on training set.

...and many other ensemble methods as well.

- State-of-the-art prediction performance
 - Won Netflix Challenge
 - Won numerous KDD Cups
 - Industry standard

Extra Slides

Comparison of Decision Tree Algorithms

- C4.5
 - Proprietary extension of ID3 Algorithm to handle continuous and categorical features, and missing values
 - Sorts continuous attributes at each node
 - Uses post-pruning to mitigate overfitting
 - Uses information gain, gain ratio to select optimal split
 - J48: open source implementation of the C4.5 algorithm
- CART (Classification and Regression Trees)
 - Restricted to binary splits
 - Less bushy trees
 - Uses Gini

Performing evaluation experiments using different models and variants helps determine which will work best for a specific problem.