

Intro to Machine Learning

Lecture 7

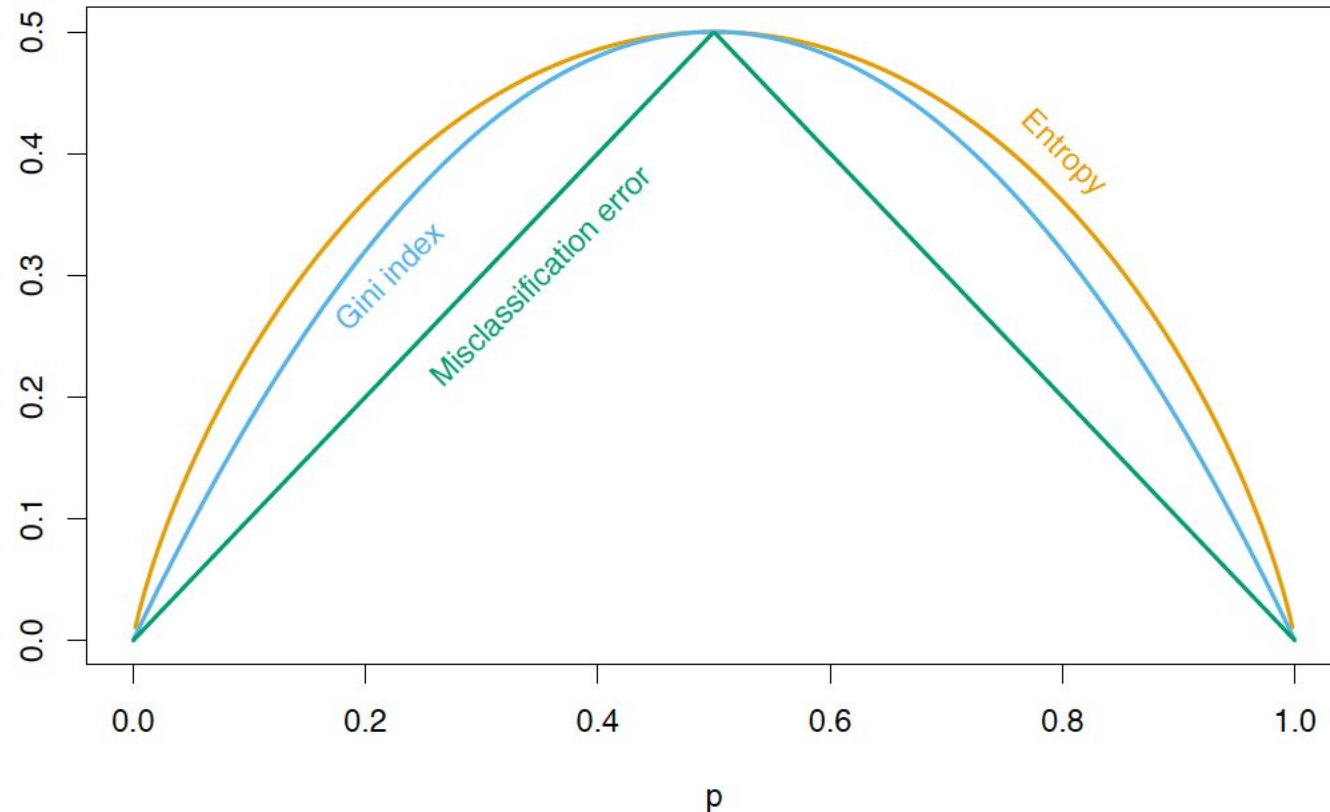
Adil Khan

a.khan@innopolis.ru

Recap

- Decision Trees (in class)
 - for classification
 - Using categorical predictors
 - Using classification error as our metric
- Decision Trees (in lab)
 - For regression
 - Using continuous predictors
 - Using entropy, gini, and information gain

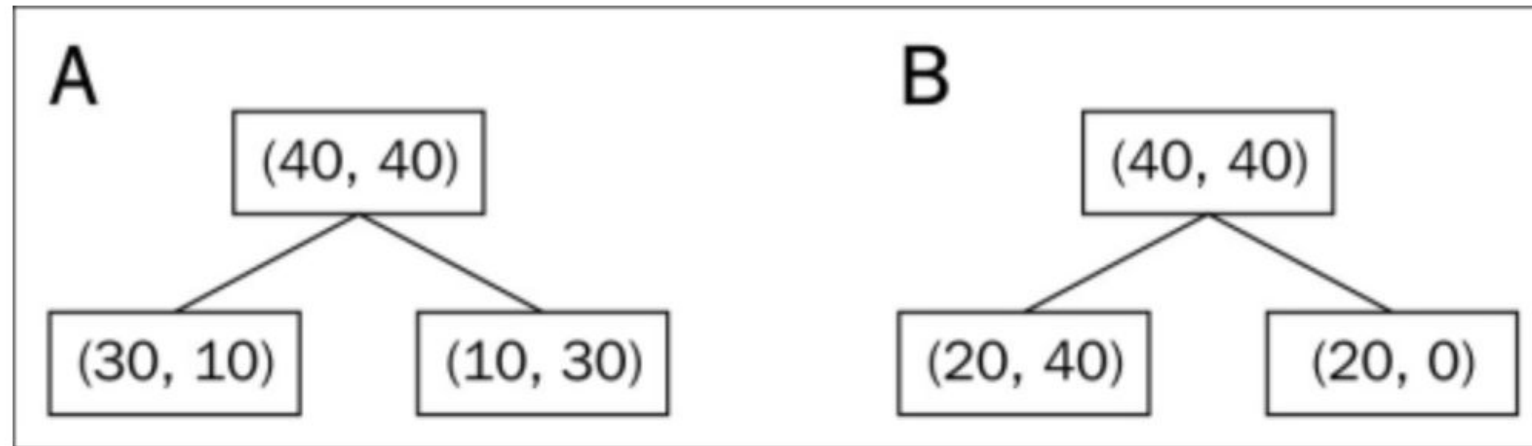
Impurity Measures: Covered in Lab last Week



Node impurity measures for two-class classification, as a function of the proportion p in class 2. Cross-entropy has been scaled to pass through $(0.5, 0.5)$.

$$1 - \max(p, 1 - p), \quad 2p(1 - p) \quad \text{and} \quad -p \log p - (1 - p) \log (1 - p)$$

Practice Yourself



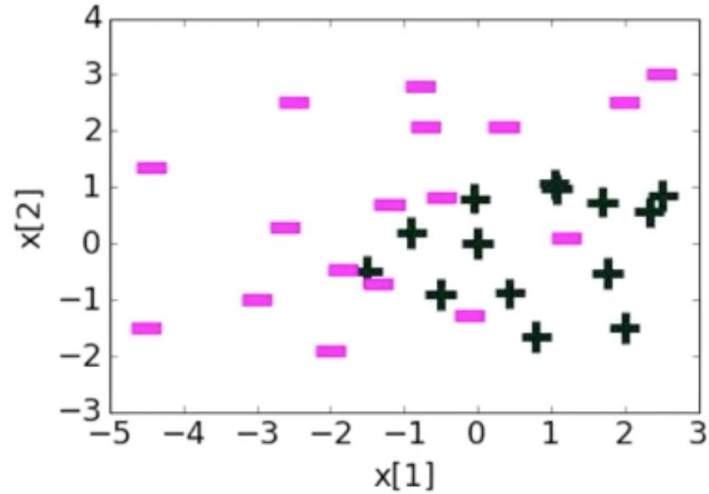
For each criteria, solve to figure out which split will it favor.

Today's Objectives

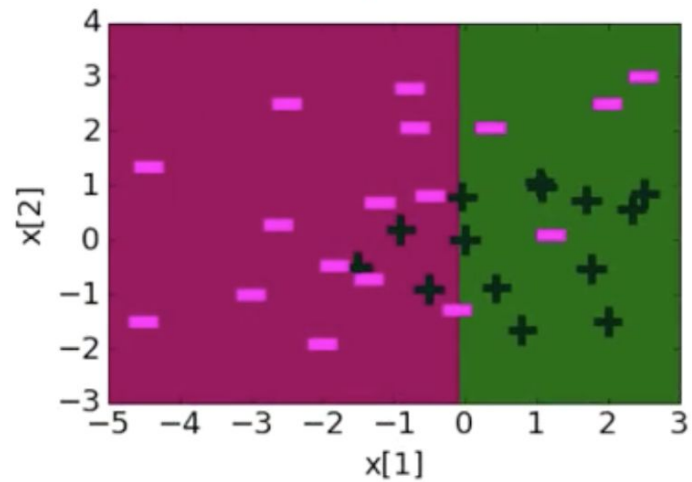
- Overfitting in Decision Trees (Tree Pruning)
- Ensemble Learning (combine the power of multiple models in a single model while overcoming their weaknesses)
 - **Bagging** (overcoming variance)
 - **Boosting** (overcoming bias)

Overfitting in Decision Trees

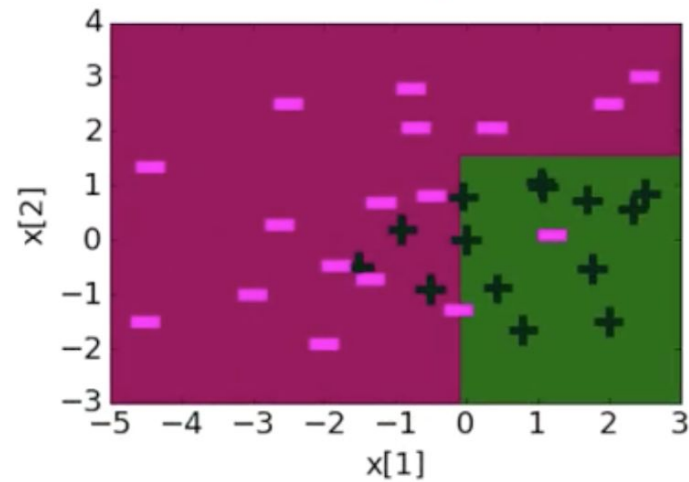
Decision Boundaries at Different Depths



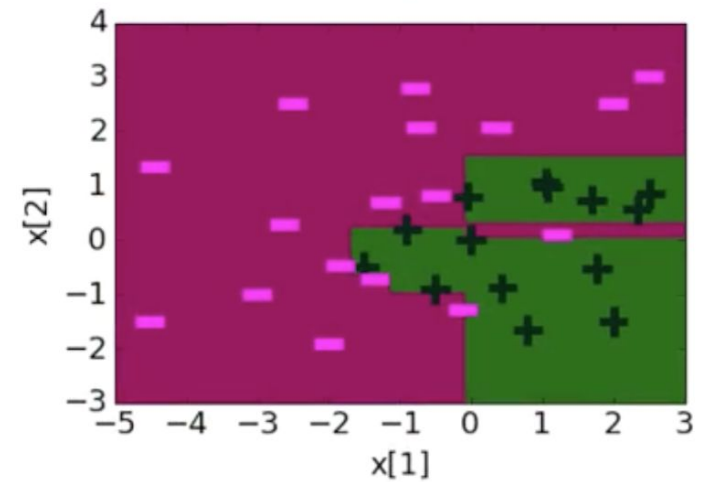
Depth 1



Depth 2



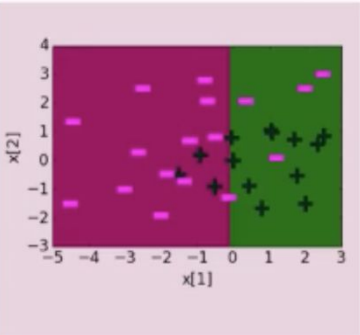
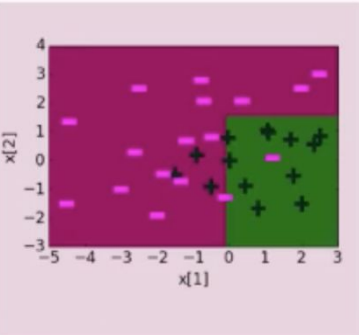
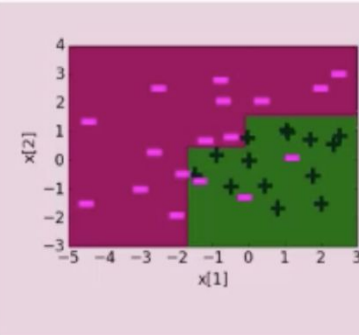
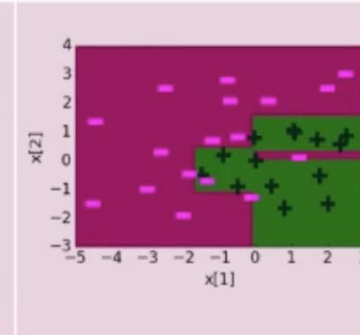
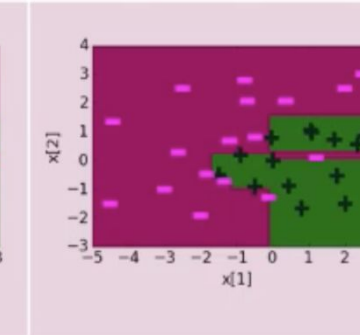
Depth 10



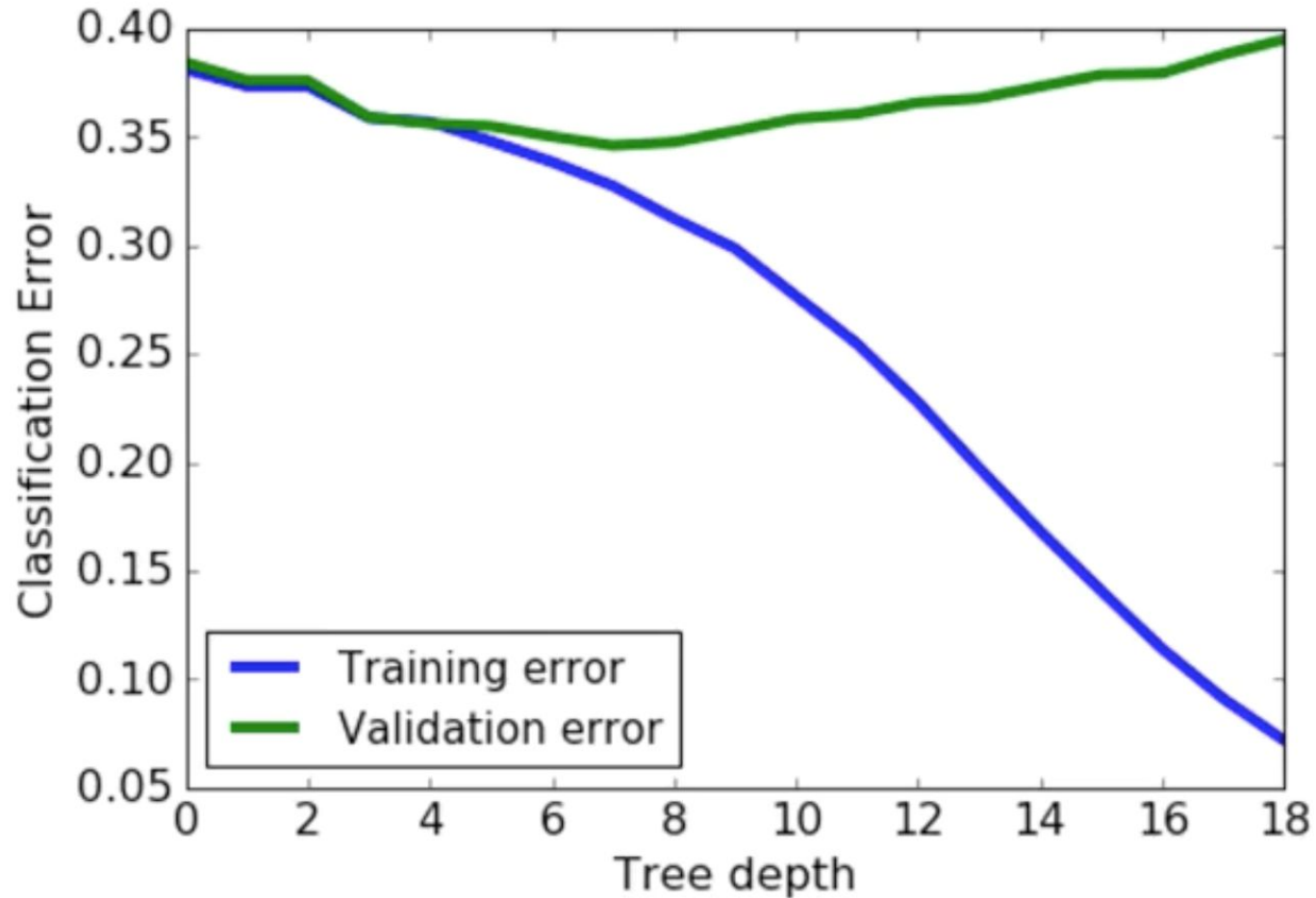
Generally Speaking

Training error reduces with depth



Tree depth	depth = 1	depth = 2	depth = 3	depth = 5	depth = 10
Training error	0.22	0.13	0.10	0.03	0.00
Decision boundary					

Decision Tree Over fitting on Real Data



Simple is Better

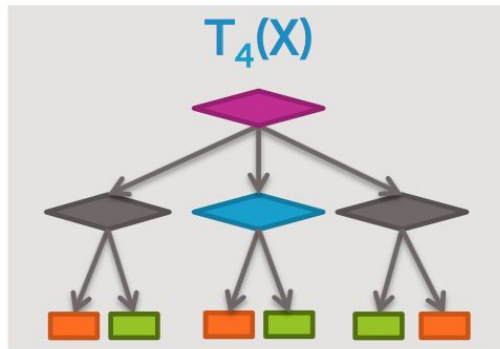
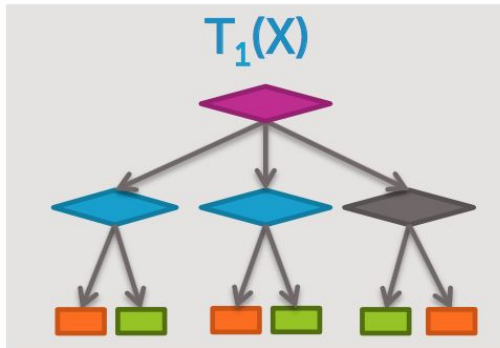
- When two trees have the same classification error on validation set, choose the one that is **simpler**

Complexity	Training Error	Validation Error
Low	0.23	0.24
Moderate	0.12	0.15
Complex	0.7	0.15
Super Complex	0.0	0.18

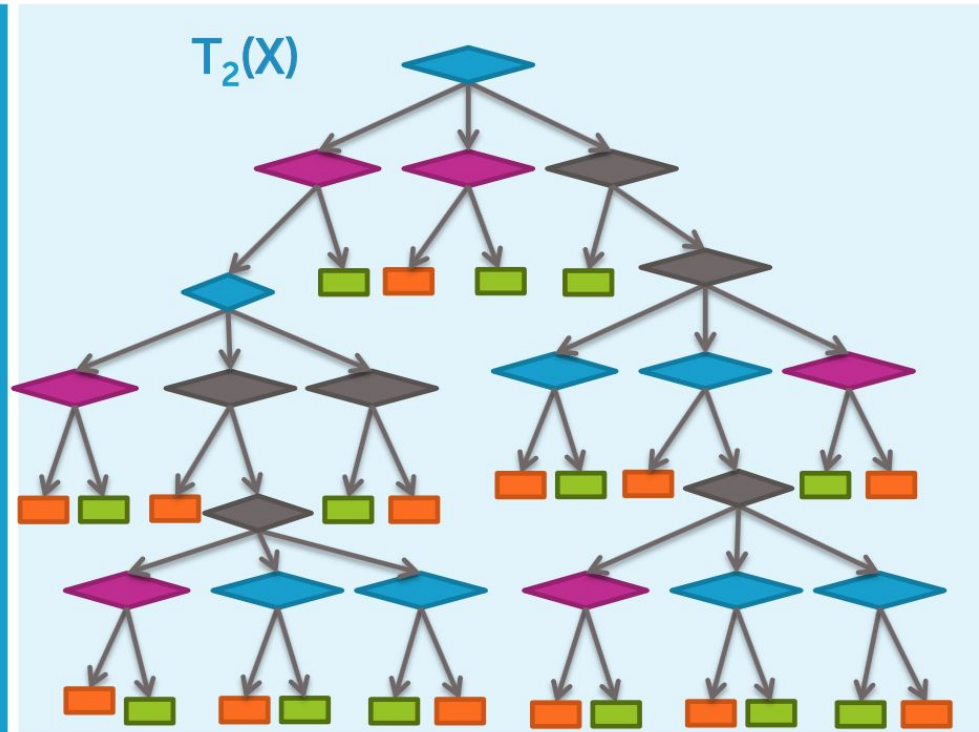
Modified Tree Learning Problem

Find a “**simple**” decision tree with low classification error

Simple trees



Complex trees

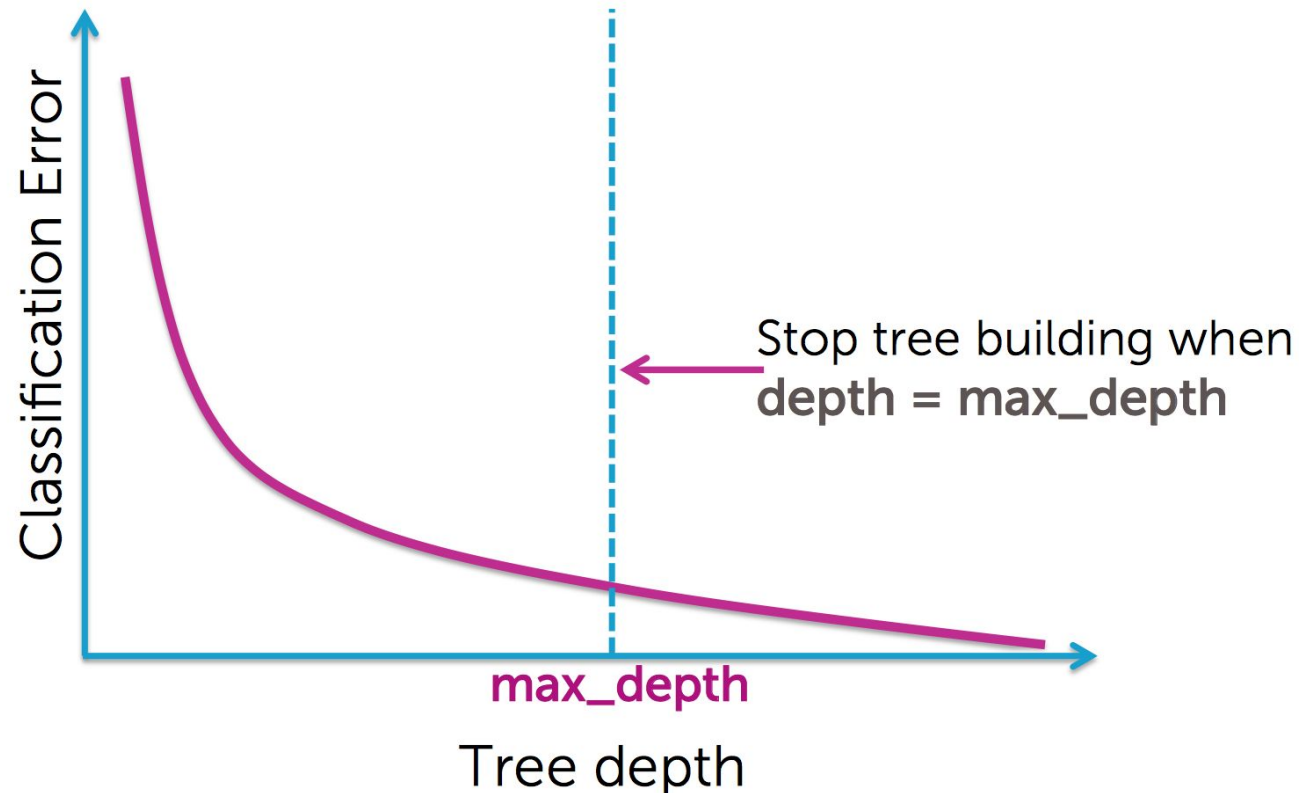


Finding Simple Trees

- **Early Stopping**: Stop learning before the tree becomes too complex
- **Pruning**: Simplify tree after learning algorithm terminates

Criteria 1 for Early Stopping

- **Limit the depth:** stop splitting after `max_depth` is reached

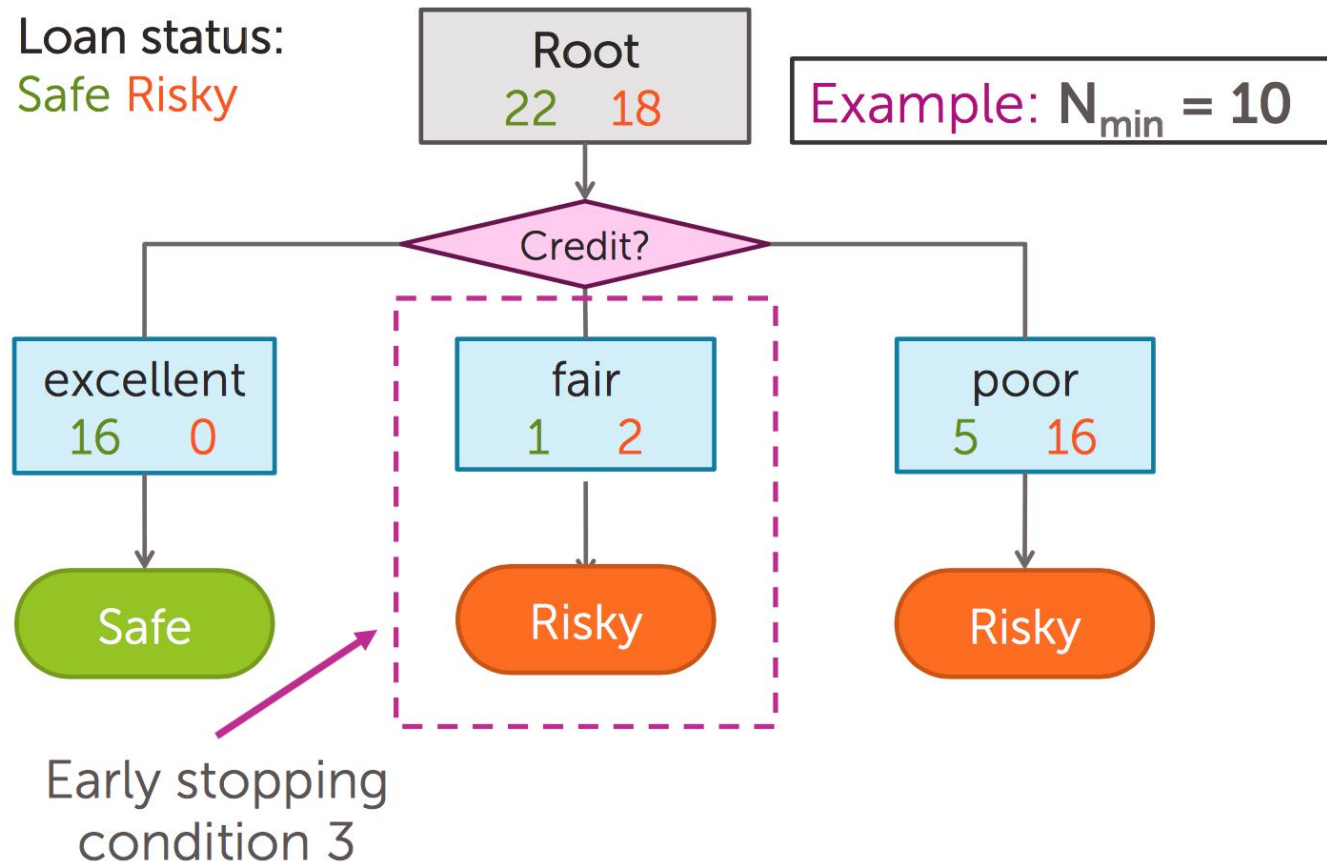


Criteria 2 for Early Stopping

- Use a **threshold for decrease ε** in error with a split
 - Stop if the error does not decrease more than ε
 - Mostly works, but may cause problems in some cases

Criteria 3 for Early Stopping

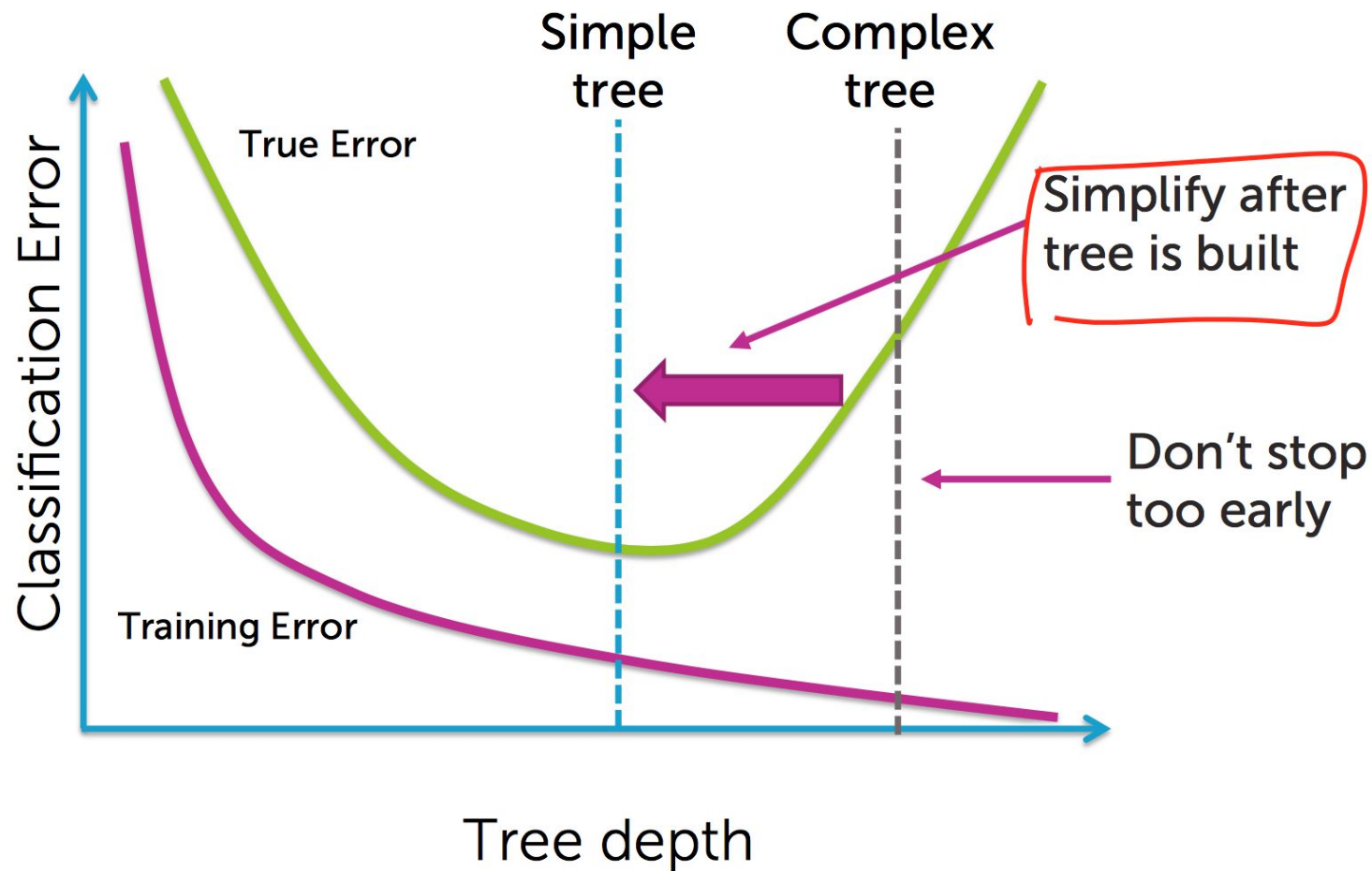
Stop when data points in a node $\leq N_{\min}$



Early Stopping: Summary

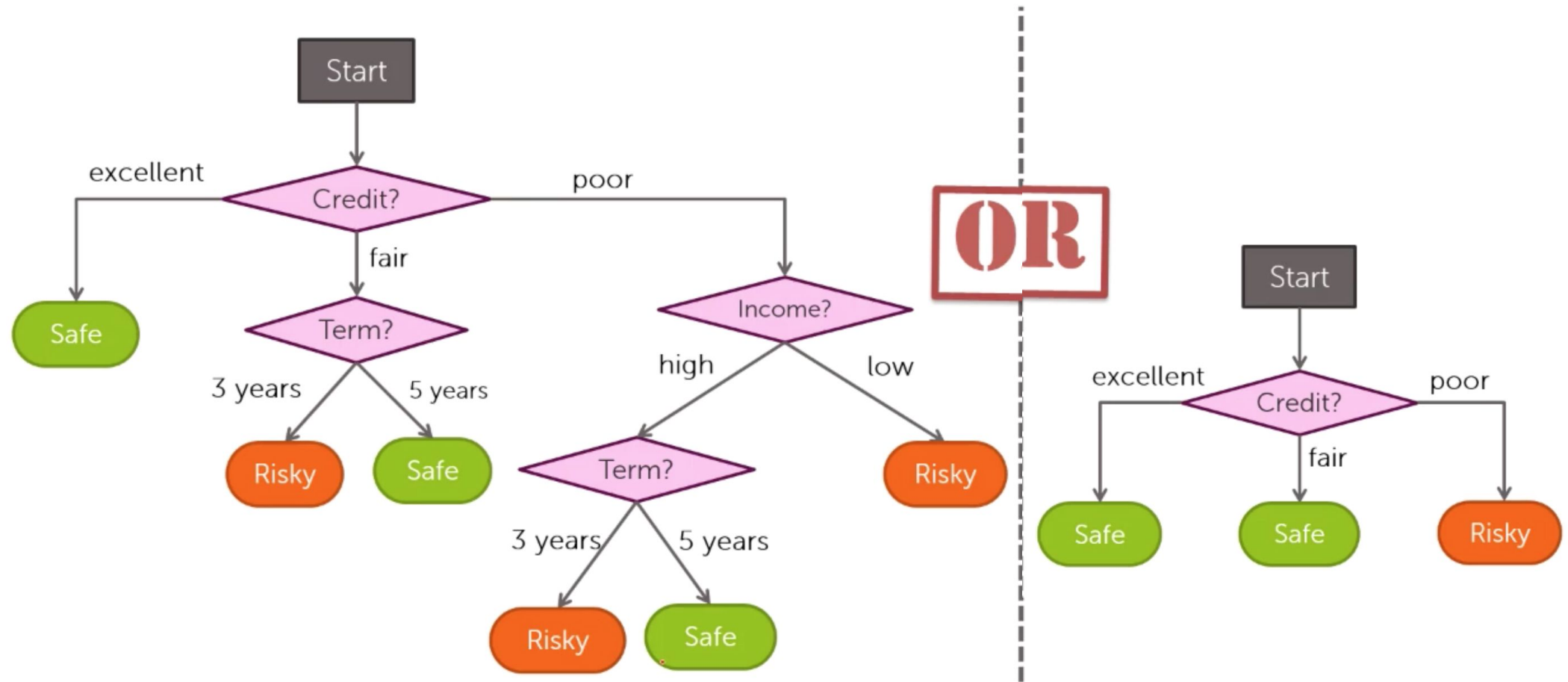
1. **Limit tree depth:** Stop splitting after a certain depth
2. **Classification error:** Do not consider any split that does not cause a sufficient decrease in classification error
3. **Minimum node "size":** Do not split an intermediate node which contains too few data points

Pruning

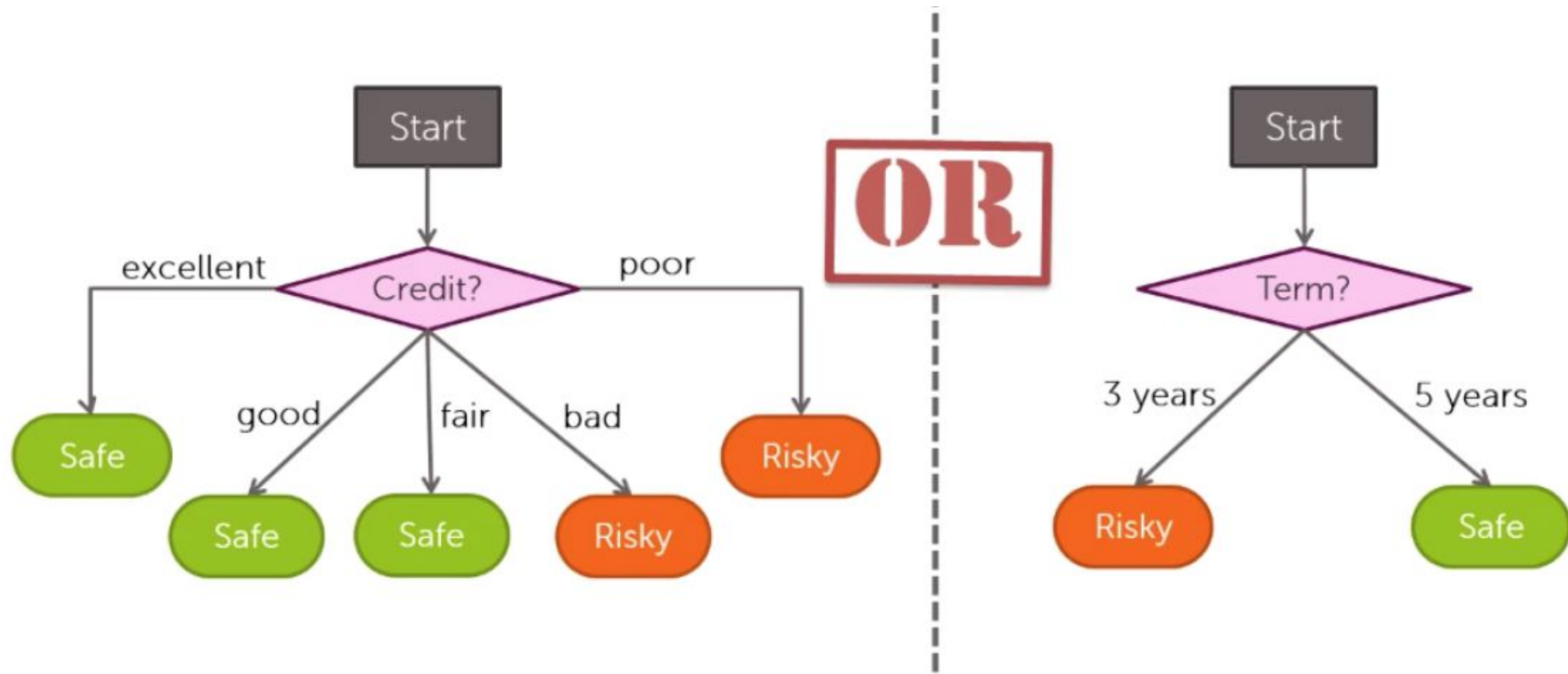


To simplify a tree, we need to define what do we mean by simplicity of the tree

Which Tree is Simpler?

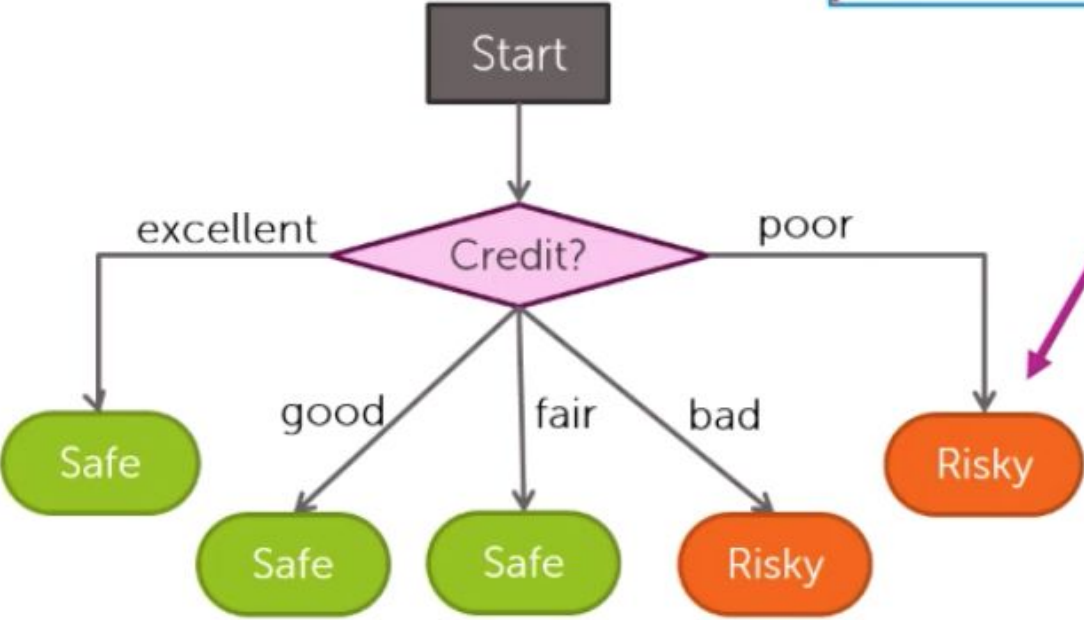


Which Tree is Simpler



Thus, Our Measure of Complexity

$$L(\mathbf{T}) = \# \text{ of leaf nodes}$$



New Optimization Goal

Total Cost = Measure of Fit + Measure of Complexity

Measure of Fit = Classification Error (large means bad fit to the data)

Measure of complexity = Number of Leaves (large means likely to overfit)

$$\text{Total cost } C(\mathbf{T}) = \text{Error}(\mathbf{T}) + \lambda L(\mathbf{T})$$

Tree Pruning Algorithm

- Let T be the final tree
- Start at the bottom of T and traverse up, apply *prune_split* at each decision node M

prune_split

- Prune_split (T , M)
 1. Compute total cost $C(T)$
 2. Let T_{small} be the tree after pruning T at M
 3. Compute $C(T_{small})$
 4. If $C(T_{small}) < C(T)$, prune T to T_{small}

Ensemble Learning

Bias and Variance

- A **complex** model could exhibit **high variance**
- A **simple** model could exhibit **high bias**

We can solve each case with ensemble learning.
Let's first see what is ensemble learning.

Ensemble Classifier in General

- Goal:
 - Predict output y
 - Either +1 or -1
 - From input \mathbf{x}
- Learn ensemble model:
 - Classifiers: $f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_T(\mathbf{x})$
 - Coefficients: $\hat{w}_1, \hat{w}_2, \dots, \hat{w}_T$
- Prediction:

$$\hat{y} = \text{sign} \left(\sum_{t=1}^T \hat{w}_t f_t(\mathbf{x}) \right)$$

Ensemble Classifier in General

- Goal:
 - Predict output y
 - Either +1 or -1
 - From input \mathbf{x}
- Learn ensemble model:
 - Classifiers: $f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_T(\mathbf{x})$
 - Coefficients: $\hat{w}_1, \hat{w}_2, \dots, \hat{w}_T$
- Prediction:

$$\hat{y} = \text{sign} \left(\sum_{t=1}^T \hat{w}_t f_t(\mathbf{x}) \right)$$

Ensemble Classifier in General

- Goal:
 - Predict output y
 - Either +1 or -1
 - From input \mathbf{x}
- Learn ensemble model:
 - Classifiers: $f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_T(\mathbf{x})$
 - Coefficients: $\hat{w}_1, \hat{w}_2, \dots, \hat{w}_T$

- Prediction:

$$\hat{y} = \text{sign} \left(\sum_{t=1}^T \hat{w}_t f_t(\mathbf{x}) \right)$$

Important

- A necessary and sufficient condition for an ensemble of classifiers to be more accurate than any of its individual members is if the members are **accurate** and **diverse** (Hansen & Salamon, 1990)

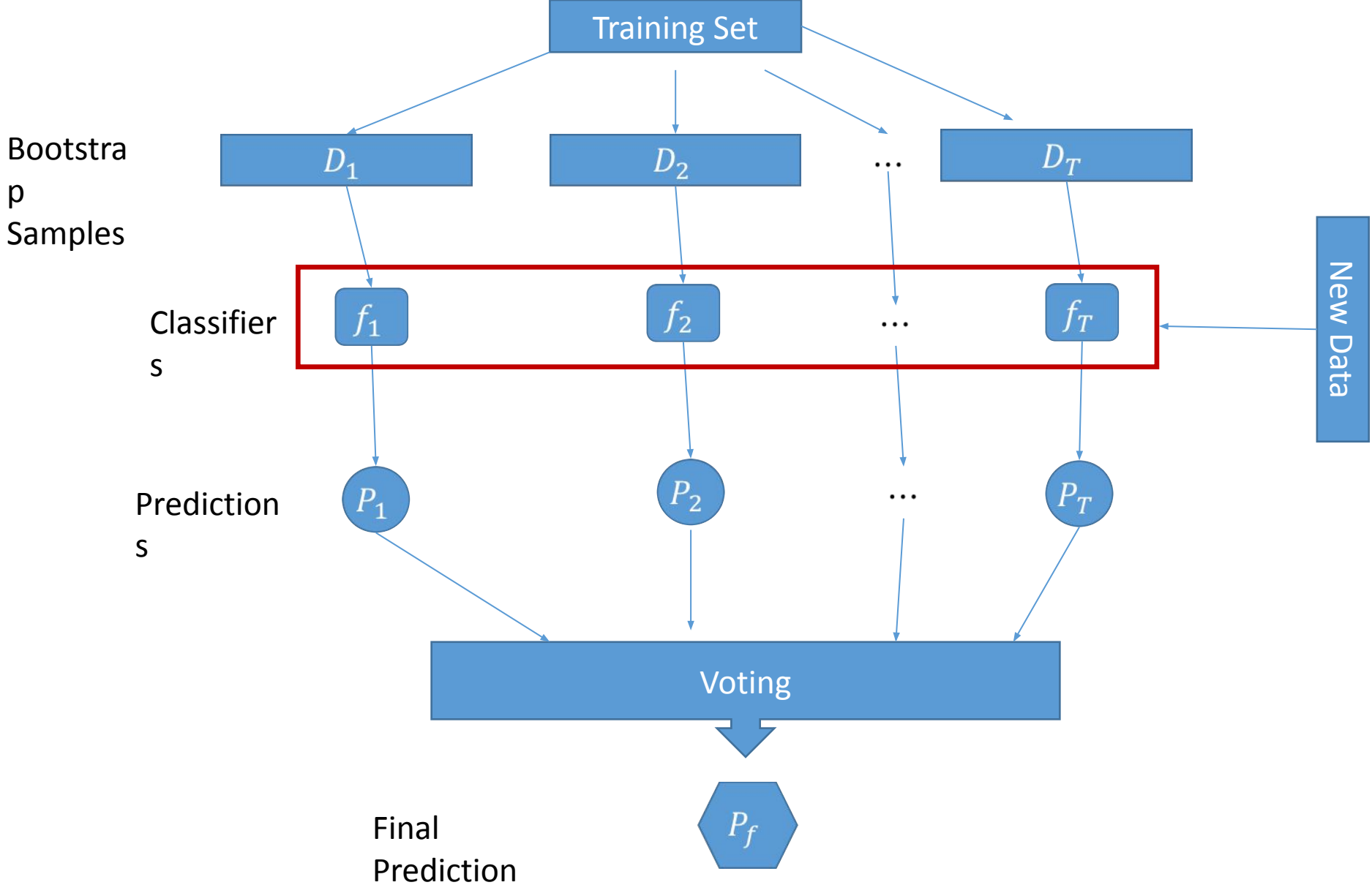
Bagging: Reducing Variance
using An Ensemble of Classifiers
from Bootstrap Samples

Aside: Bootstrapping

Training Data	Bootstrap 1	Bootstrap 2	
1	2	7	
2	2	3	
3	1	2	
4	3	1	
5	7	1	
6	2	7	
7	4	7	

Creating new datasets from the training data *with replacement*

Bagging



Why Bagging Works?

- **Averaging** reduces variance
- Let Z_1, Z_2, \dots, Z_N be i.i.d random variables

$$\text{Var} \left(\frac{1}{N} \sum_i Z_i \right) = \frac{1}{N} \text{Var}(Z_i)$$

Bagging Summary

- Bagging was first proposed by Leo Breiman in a technical report in 1994
- He also showed that bagging can improve the accuracy of unstable models and decrease the degree of overfitting.
- I highly recommend you read about his research in L. Breiman. **Bagging Predictors**. Machine Learning, 24(2):123–140, 1996,

Random Forests – Example of Bagging

1. Draw a random **bootstrap** sample
2. Grow a decision tree from the bootstrap sample. At each node:
 - a) **Randomly** select **d** features without replacement ($d = \sqrt{n}$).
 - b) Split the node using the feature that provides the best split according to the objective function, for instance, by maximizing the information gain.
3. Repeat the steps 1 to 2 **k** times.
4. Aggregate the prediction by each tree to assign the class label by **majority voting**

Making a Prediction

the ensemble of trees $\{T_b\}_1^B$

To make a prediction at a new point x :

Regression: $\hat{f}_{\text{rf}}^B(x) = \frac{1}{B} \sum_{b=1}^B T_b(x)$.

Classification: Let $\hat{C}_b(x)$ be the class prediction of the b th random-forest tree. Then $\hat{C}_{\text{rf}}^B(x) = \text{majority vote } \{\hat{C}_b(x)\}_1^B$.

Boosting: Converting Weak Learners to Strong Learners through Ensemble Learning

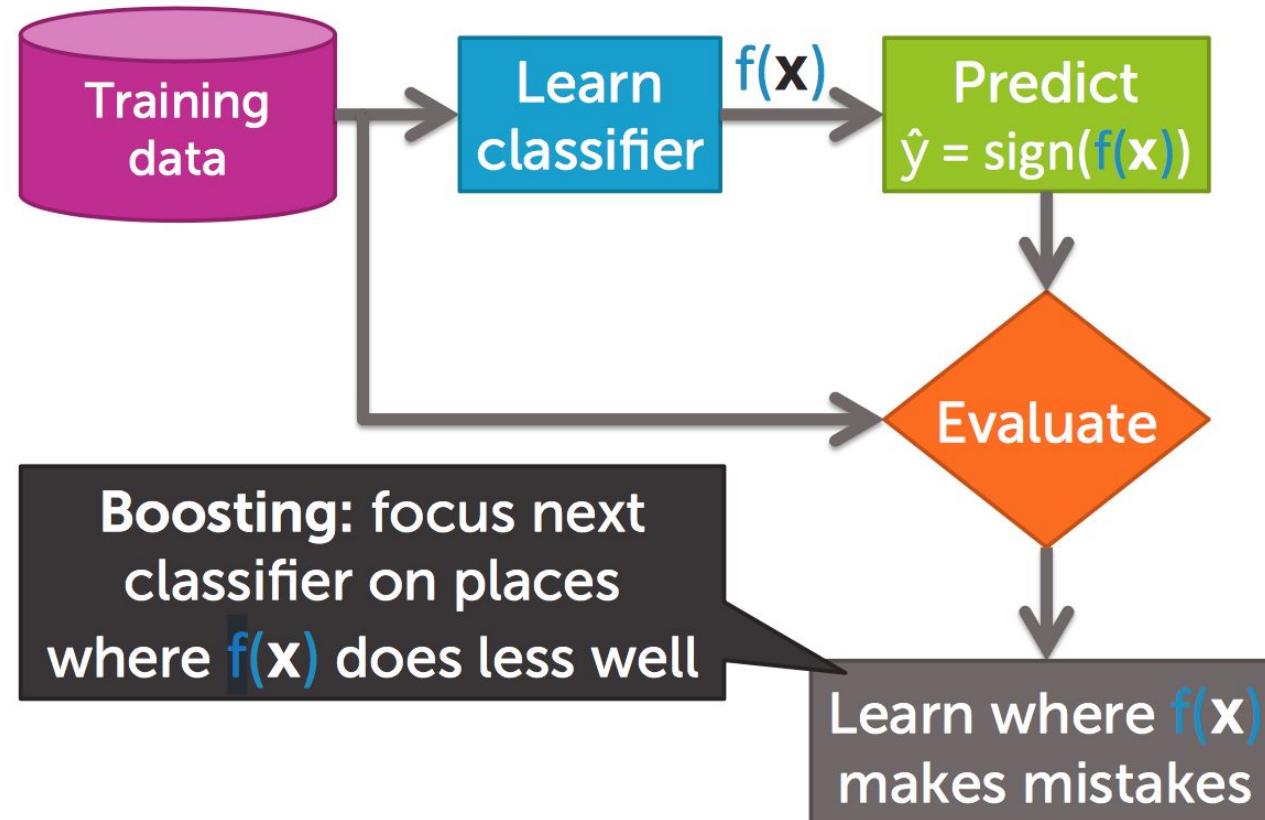
Boosting and Bagging

- Works in a similar way as bagging.
- Except:
 - **Models are built sequentially**: each model is built using information from previously built models.
 - **Boosting does not involve bootstrap sampling**; instead each tree is fit on a modified version of the original data set

Boosting: (1) Train A Classifier



Boosting: (2) Train Next Classifier by Focusing More on the Hard Points

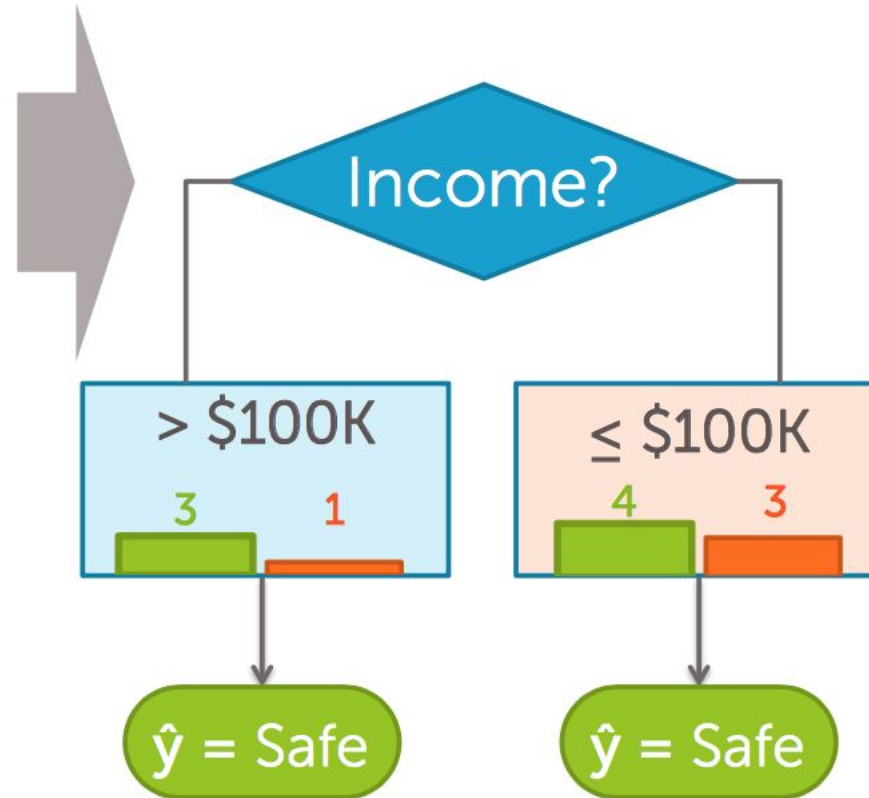


What does it mean to focus more?

- Weighted dataset:
 - Each \mathbf{x}_i, y_i weighted by α_i
 - More important point = higher weight α_i
- Learning:
 - Data point j counts as α_j data points
 - E.g., $\alpha_j = 2 \rightarrow$ count point twice

Example (Unweighted): Learning a Simple Decision Stump

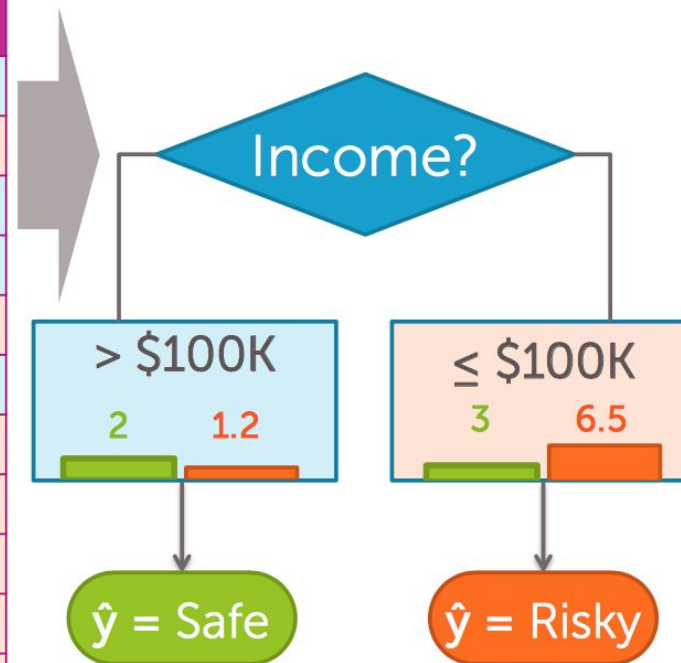
Credit	Income	y
A	\$130K	Safe
B	\$80K	Risky
C	\$110K	Risky
A	\$110K	Safe
A	\$90K	Safe
B	\$120K	Safe
C	\$30K	Risky
C	\$60K	Risky
B	\$95K	Safe
A	\$60K	Safe
A	\$98K	Safe



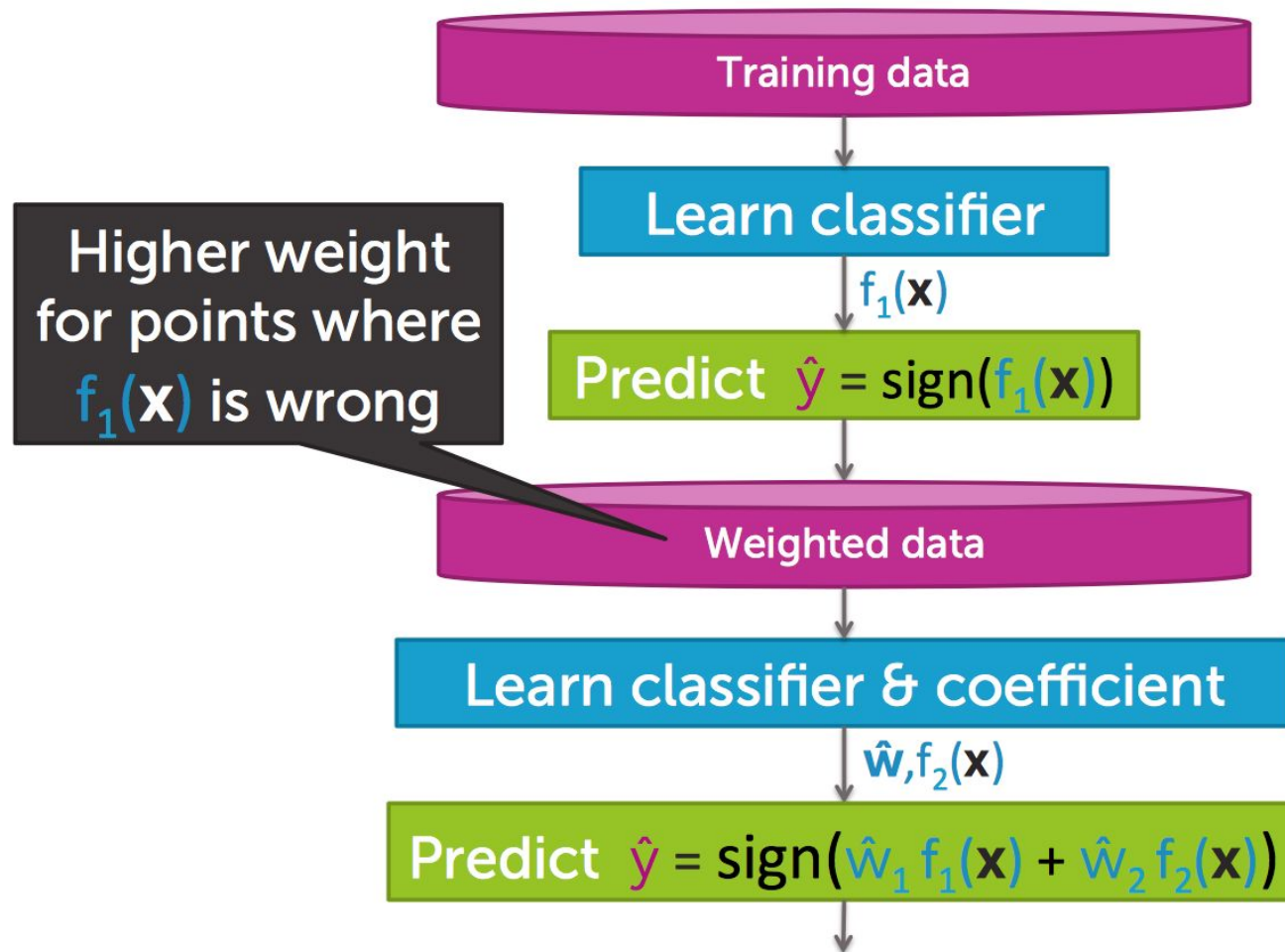
Example (Weighted): Learning a Decision Stump on Weighted Data

Increase weight α of harder/
misclassified points

Credit	Income	y	Weight α
A	\$130K	Safe	0.5
B	\$80K	Risky	1.5
C	\$110K	Risky	1.2
A	\$110K	Safe	0.8
A	\$90K	Safe	0.6
B	\$120K	Safe	0.7
C	\$30K	Risky	3
C	\$60K	Risky	2
B	\$95K	Safe	0.8
A	\$60K	Safe	0.7
A	\$98K	Safe	0.9



Boosting

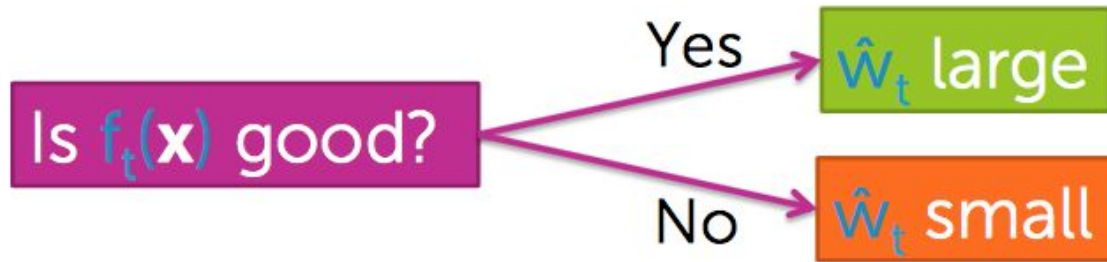


AdaBoost (Example of Boosting)

- Start with the same weights for all points: $\alpha_i = \frac{1}{m}$
- For each $t = 1, \dots, T$
 - Learn $f_t(x)$ with data weights α_i
 - Compute coefficient \hat{w}_t ← *Weight of the model*
 - Recompute weights α_i ← *New weights of the data points*
- Final model predicts as:

$$\hat{y} = \text{sign} \left(\sum_{t=1}^T \hat{w}_t f_t(x) \right)$$

AdaBoost: Computing coefficient \hat{w}_t of classifier $f_t(\mathbf{x})$



- $f_t(\mathbf{x})$ is good \rightarrow f_t has low training error
- Measuring error in weighted data?
 - Just weighted # of misclassified points

Weighted Classification Error

- Total weight of the mistakes:

$$= \sum_{i=1}^m \alpha_i I(\hat{y}_i \neq y_i)$$

- Total weight of all points:

$$= \sum_{i=1}^m \alpha_i$$

- Weighted error measures fraction of weight of mistakes:

$$= \frac{\textit{Total weight of the mistakes}}{\textit{Total weight of all points}}$$

- Best possible values is 0.0

AdaBoost: Computing Classifier's Weights

-

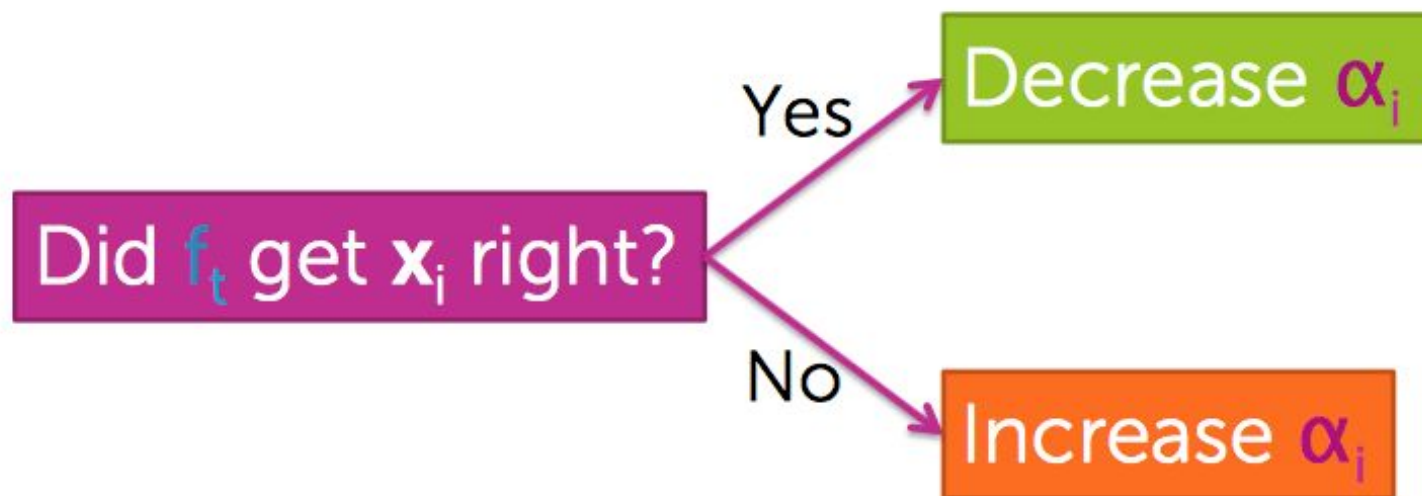
Weighted error on training data		
0.01		
0.5		
0.99		

AdaBoost

-



AdaBoost: Updating weights α_i based on where classifier $f_t(x)$ makes mistakes



AdaBoost: Recomputing A Sample's Weight

$$\alpha_i \leftarrow \begin{cases} \alpha_i e^{-\hat{w}_t}, & \text{if } f_t(\mathbf{x}_i) = y_i \\ \alpha_i e^{\hat{w}_t}, & \text{if } f_t(\mathbf{x}_i) \neq y_i \end{cases}$$

Predicted Label	Classifier weight		Result
Correct	2.3	0.1	?
Correct	0	1	?
Mistake	2.3	9.98	?
Mistake	0	1	?

Increase, Decrease, or Keep the Same

AdaBoost: Recomputing A Sample's Weight

$$\alpha_i \leftarrow \begin{cases} \alpha_i e^{-\hat{w}_t}, & \text{if } f_t(\mathbf{x}_i) = y_i \\ \alpha_i e^{\hat{w}_t}, & \text{if } f_t(\mathbf{x}_i) \neq y_i \end{cases}$$

Predicted Label	Classifier weight		Result
Correct	2.3	0.1	Decrease the importance of this sample
Correct	0	1	Keep the importance the same
Mistake	2.3	9.98	Increase the importance of the sample
Mistake	0	1	Keep the same

AdaBoost

- Start same weight for all points: $\alpha_i = 1/N$
- For $t = 1, \dots, T$

- Learn $f_t(\mathbf{x})$ with data weights α_i

- Compute coefficient \hat{w}_t

- Recompute weights α_i

$$\hat{w}_t = \frac{1}{2} \ln \left(\frac{1 - \text{weighted_error}(f_t)}{\text{weighted_error}(f_t)} \right)$$

- Final model predicts by:

$$\hat{y} = \text{sign} \left(\sum_{t=1}^T \hat{w}_t f_t(\mathbf{x}) \right)$$

$$\alpha_i \leftarrow \begin{cases} \alpha_i e^{-\hat{w}_t}, & \text{if } f_t(\mathbf{x}_i) = y_i \\ \alpha_i e^{\hat{w}_t}, & \text{if } f_t(\mathbf{x}_i) \neq y_i \end{cases}$$

AdaBoost: Normalizing Sample Weights

If x_i often mistake,
weight α_i gets very
large

If x_i often correct,
weight α_i gets very
small

Can cause numerical instability
after many iterations

Normalize weights to
add up to 1 after every iteration

$$\alpha_i \leftarrow \frac{\alpha_i}{\sum_{j=1}^N \alpha_j}$$

AdaBoost

- Start same weight for all points: $\alpha_i = 1/N$

$$\hat{w}_t = \frac{1}{2} \ln \left(\frac{1 - \text{weighted_error}(f_t)}{\text{weighted_error}(f_t)} \right)$$

- For $t = 1, \dots, T$

- Learn $f_t(\mathbf{x})$ with data weights α_i

- Compute coefficient \hat{w}_t

- Recompute weights α_i

- Normalize weights α_i

$$\alpha_i \leftarrow \begin{cases} \alpha_i e^{-\hat{w}_t}, & \text{if } f_t(\mathbf{x}_i) = y_i \\ \alpha_i e^{\hat{w}_t}, & \text{if } f_t(\mathbf{x}_i) \neq y_i \end{cases}$$

- Final model predicts by:

$$\hat{y} = \text{sign} \left(\sum_{t=1}^T \hat{w}_t f_t(\mathbf{x}) \right)$$

$$\alpha_i \leftarrow \frac{\alpha_i}{\sum_{j=1}^N \alpha_j}$$

Self Study

- What is the effect of of:
 - Increasing the number of classifiers in *bagging*
 - vs.
 - Increasing the number of classifiers in *boosting*

Boosting Summary

"Can a set of weak learners be combined to create a stronger learner?" *Kearns and Valiant (1988)*



Yes! *Schapire (1990)*



Boosting



Amazing impact: • simple approach • widely used in industry • wins most Kaggle competitions

Summary

- Decision Tree Pruning
- Ensemble Learning
- Bagging
- Boosting