INFSCI 2915: Machine Learning

Decision Trees and Ensemble Methods

Mai Abdelhakim

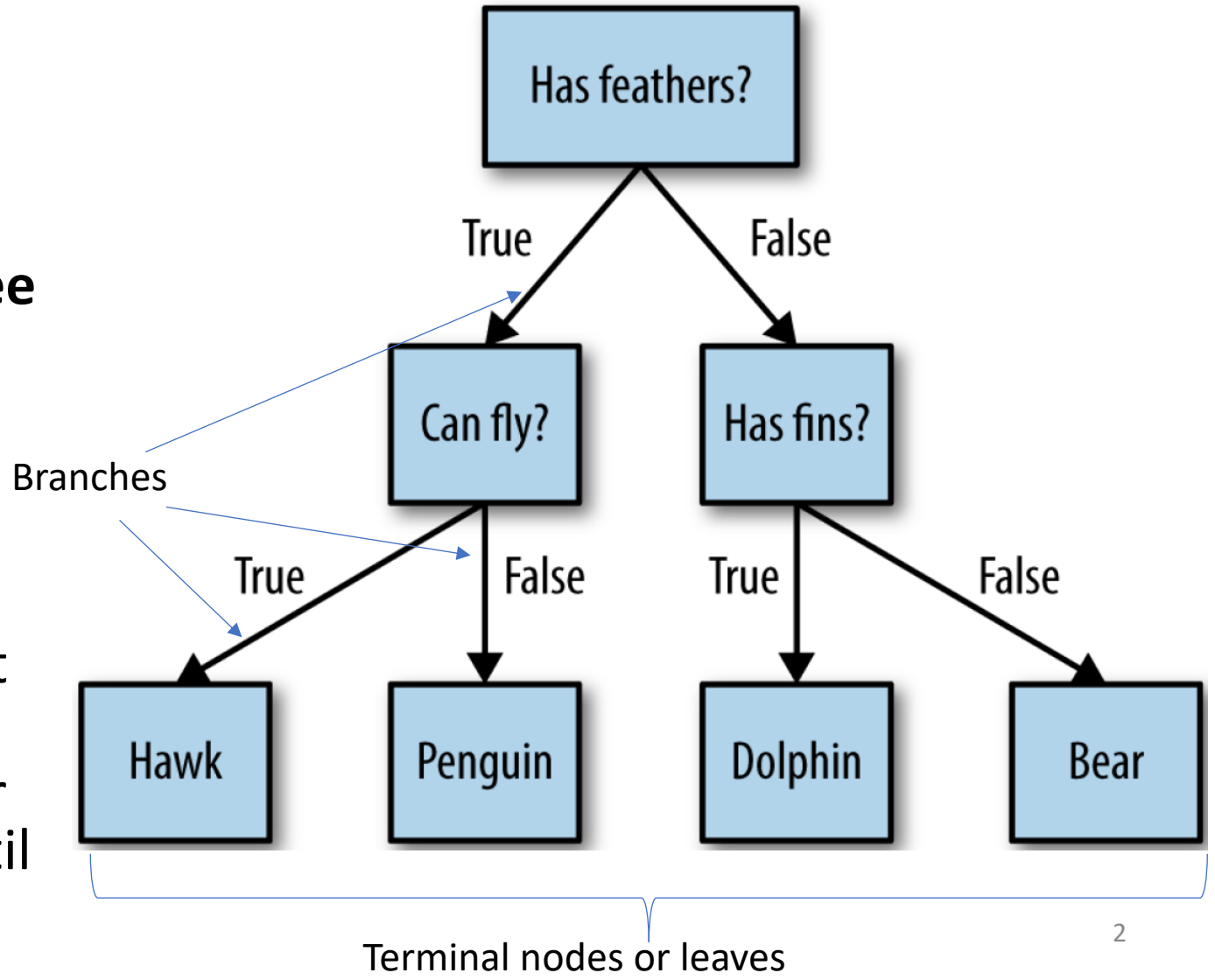
School of Computing and Information

610 IS Building

Spring 2018

Decision Trees

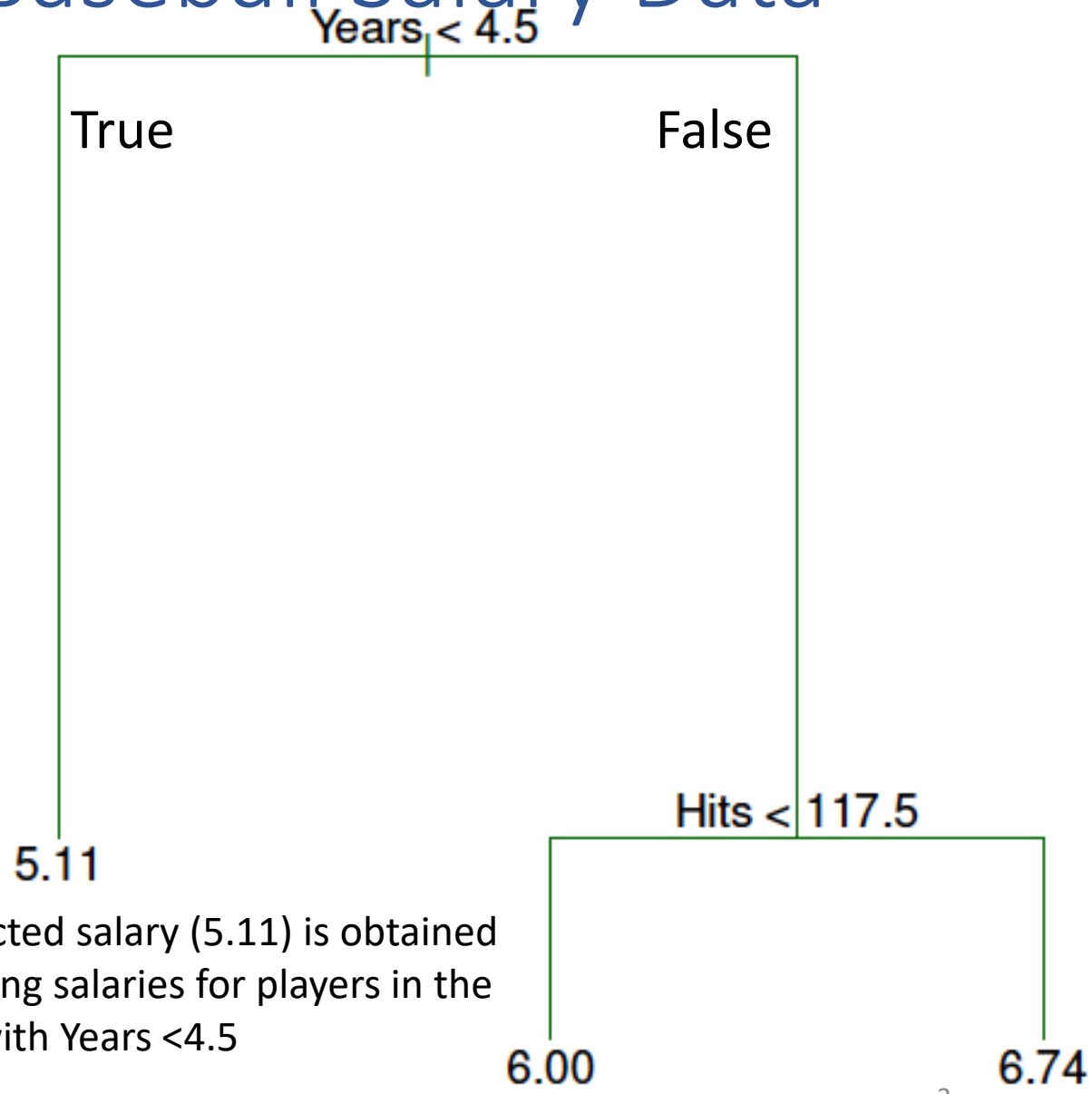
- Used for regression and classification problems
- Rely on **segmenting the feature space** into number of **regions** using set of **splitting rules**
- These rules are represented in a **tree**
  - Nodes: **Internal** or **terminal**
    - **Terminal** nodes represent **regions** in feature space
    - **Internal** nodes are points of **splits**
  - Branches: segment that connect nodes
  - **Tree depth** is maximum number of branches (from tree root) until a terminal node



Regression Trees Example: Baseball Salary Data

Example: predict a baseball player salary's (Salary) using two features

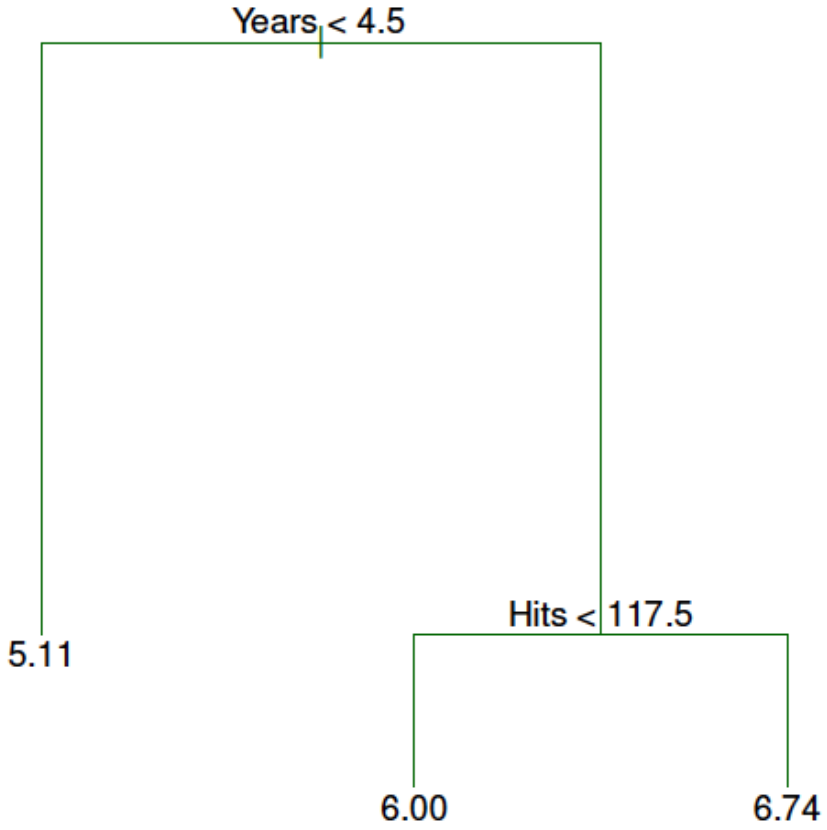
- Features:
  - **Years**: number of years the player is in a major league
  - **Hits**: number of hits the player made in the previous year
- Tree consist of series of spitting rules
  - Start with checking if Years < 4.5
  - If Years<4.5, the tree checks Hits

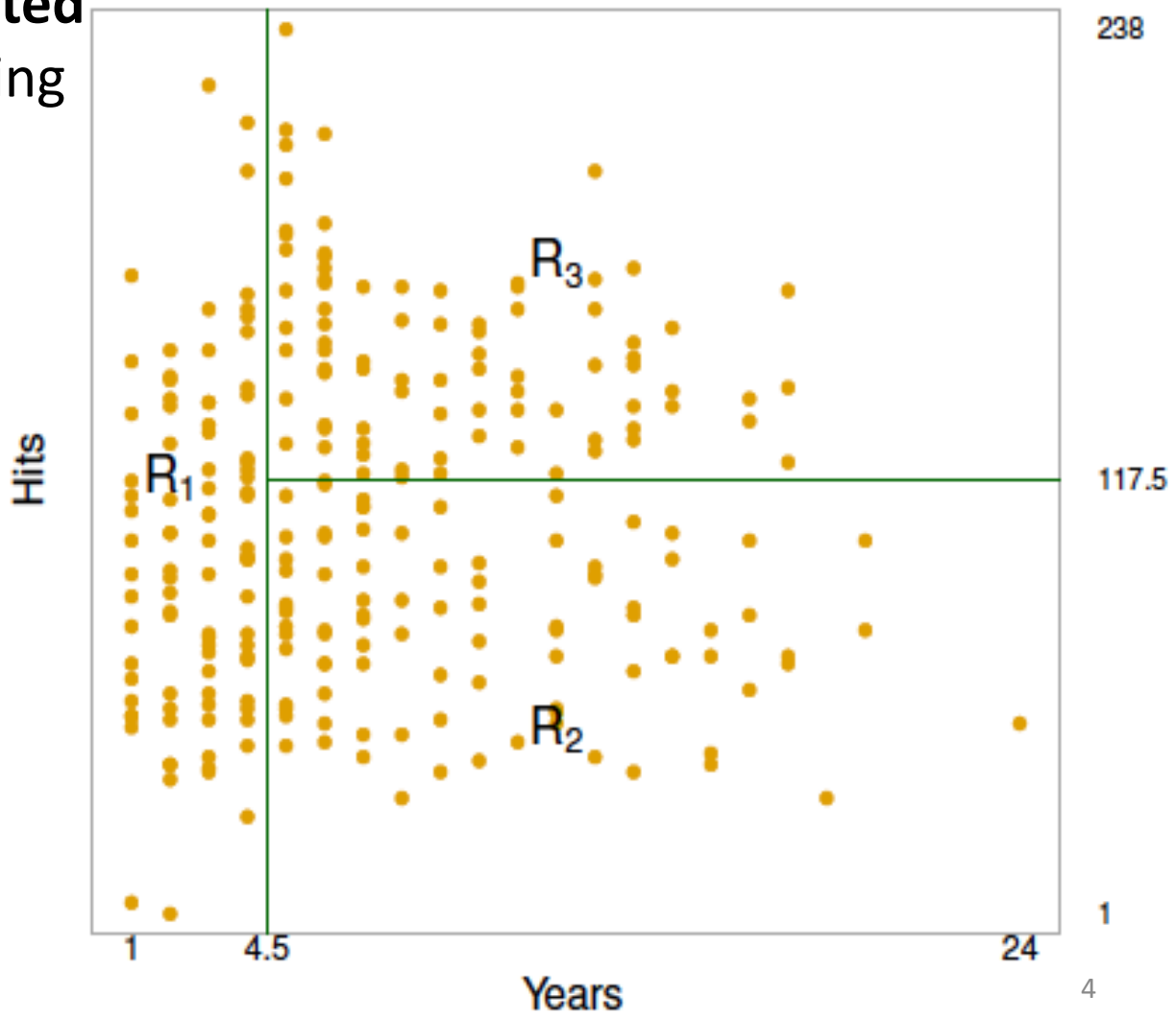


The predicted salary (5.11) is obtained by averaging salaries for players in the data set with Years <4.5

Regression Trees Example: Baseball Salary Data

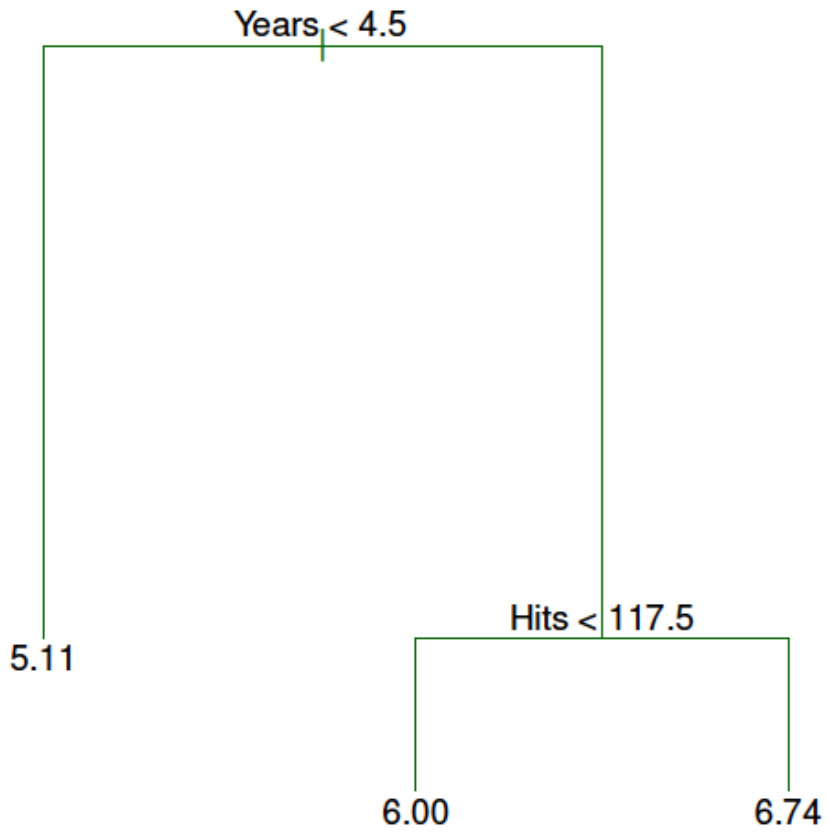
- The tree divides the feature space into regions
- If observation lies in region R_i, then **predicted response is the average response** of training observation in **that region**



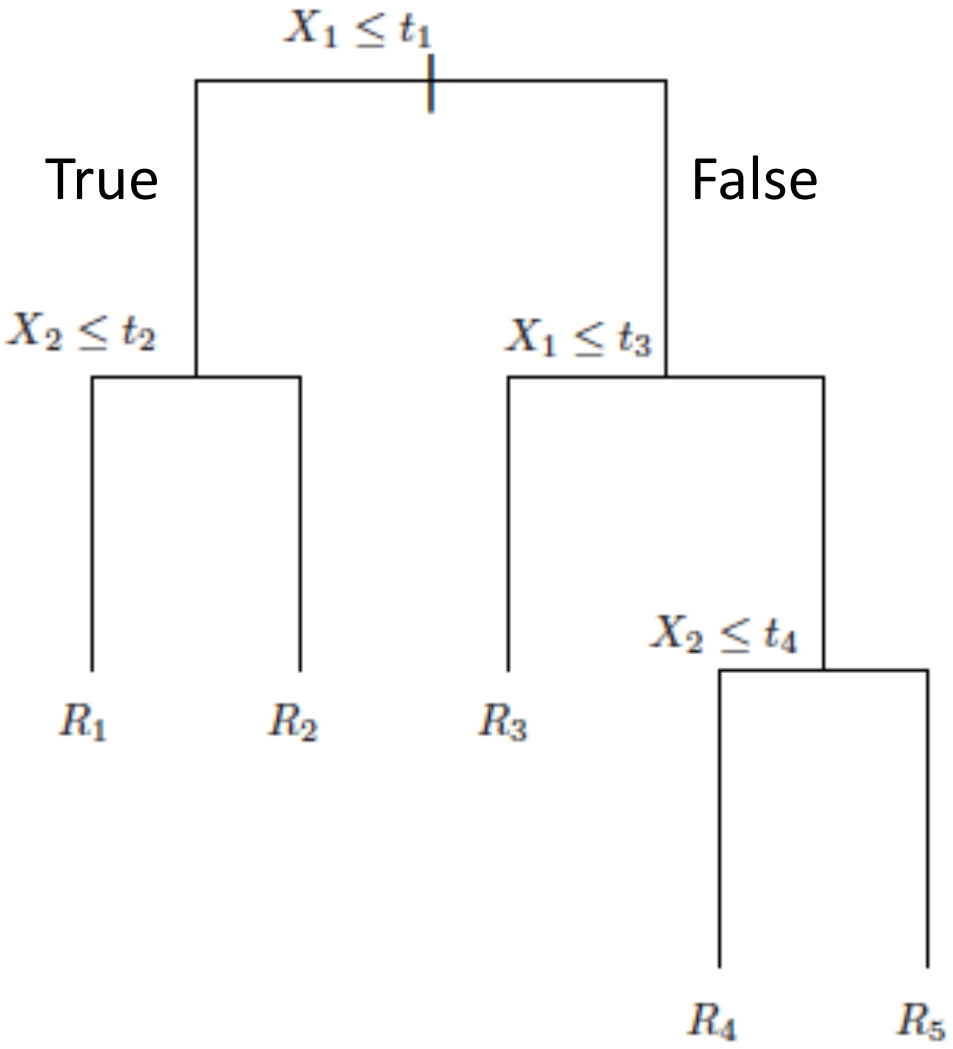


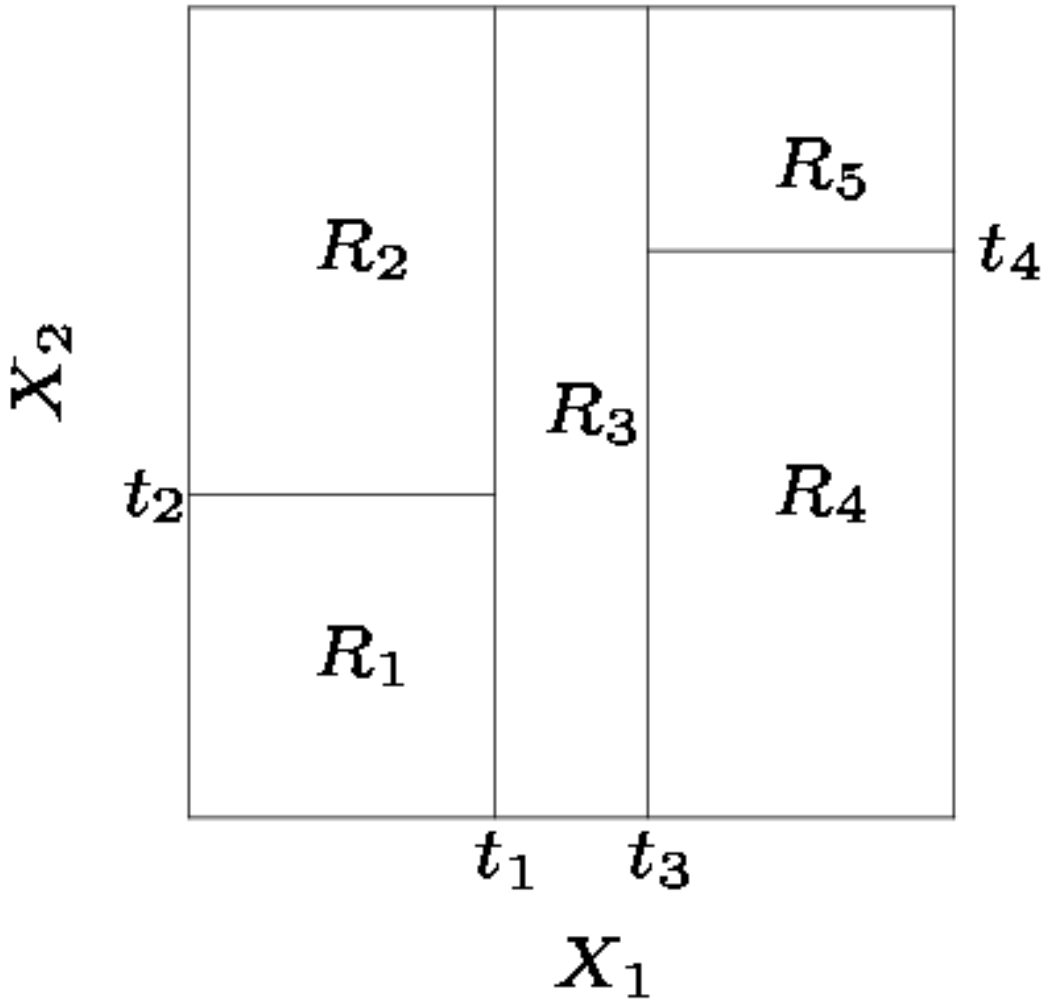
Tree Interpretation

- In this tree example, the following interpretations can be made
  - Years is the most important feature -- Appears at root of the tree
    - Player with less experience, gets lower salary
  - If a player is less experienced, Hits will not be a significant feature in determining the salary
  - If player is experienced, then "Hits" plays a key role in determining the salary
    - More hits, higher salary



Tree Divides the Feature Space into Rectangular Regions





How to Build a Regression Tree ?

- The goal is to divide the feature space into ***J* regions** $(R_1, R_2 \ldots . R_J)$, and find these region that minimizes the RSS
- The RSS is given by:

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$$

  - $\hat{y}_{R_j}$ is the mean response of training observation in region j
  - y_i is the response of the ith training example
- **Greedy approach**: try every possible set of partitions ➔ **Infeasible**
- **Instead,** we use **recursive binary splitting**
  - **Find one split at a time**

Recessive Binary Splitting

- Start with entire space, and iterate:

1. Each iteration select **predictor X_j and cutpoint s** such that splitting the space into regions with s leads to the **largest reduction in the RSS**

Region 1 : $X_j < s$ Region 2: $X_j \geq s$

$$R_1(j,s) = \{X | X_j < s\} \qquad R_2(j,s) = \{X | X_j \geq s\}$$

- Focus on **one split at a time** (instead of finding all splits at the same time like the greedy approach). Then find a best feature along with its splitting rule that minimizes RSS.

- Find feature X_j and s that minimize

$$\sum_{i:\, x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:\, x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2$$

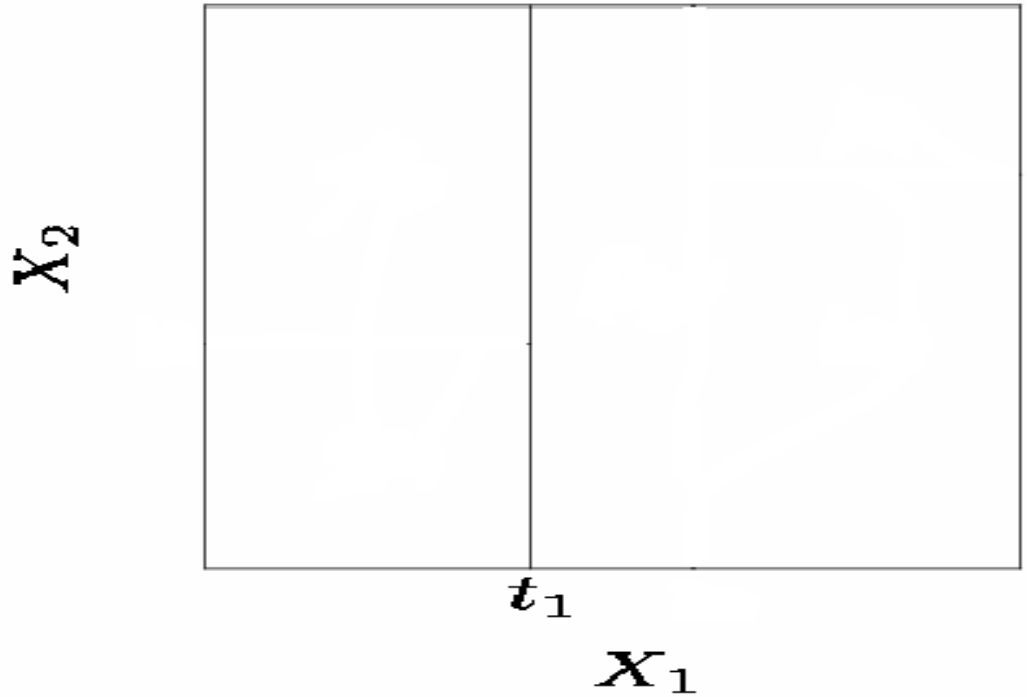
2. **Repeat** the process with redefined regions: find the features and cutpoint to split the data further for one of the regions already obtained (not entire space)

- We choose the one with largest reduction in RSS

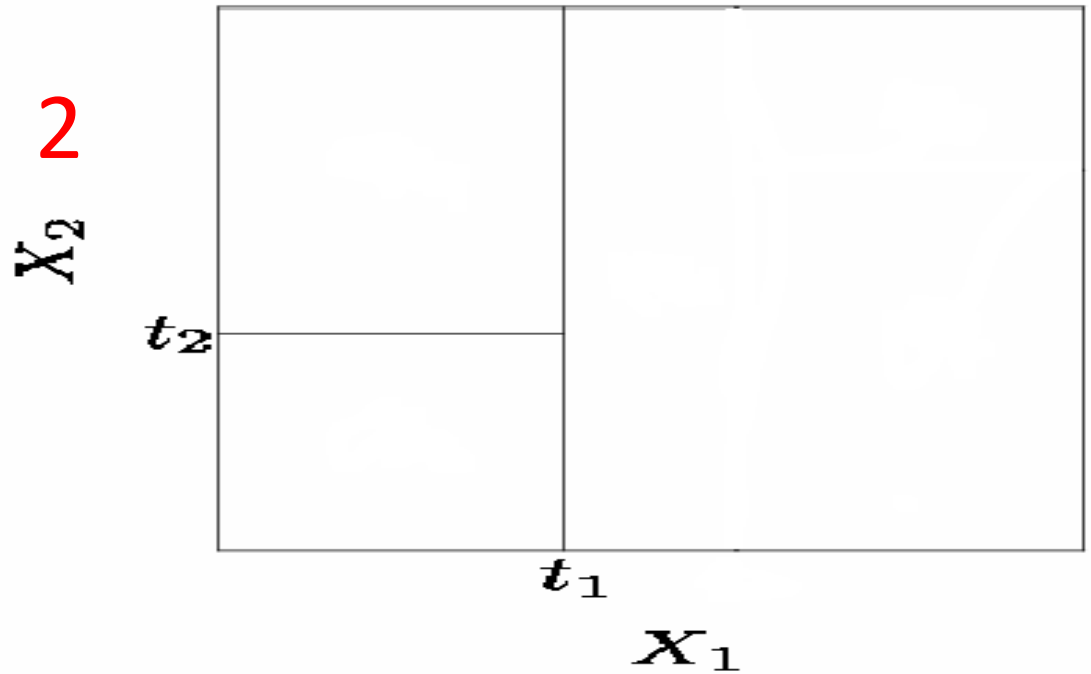
- In other words: in each iteration, we **split a region into two region**. In the next iteration, we focus on one of the regions then split it into two, and so on

- We repeat until a stopping criterion

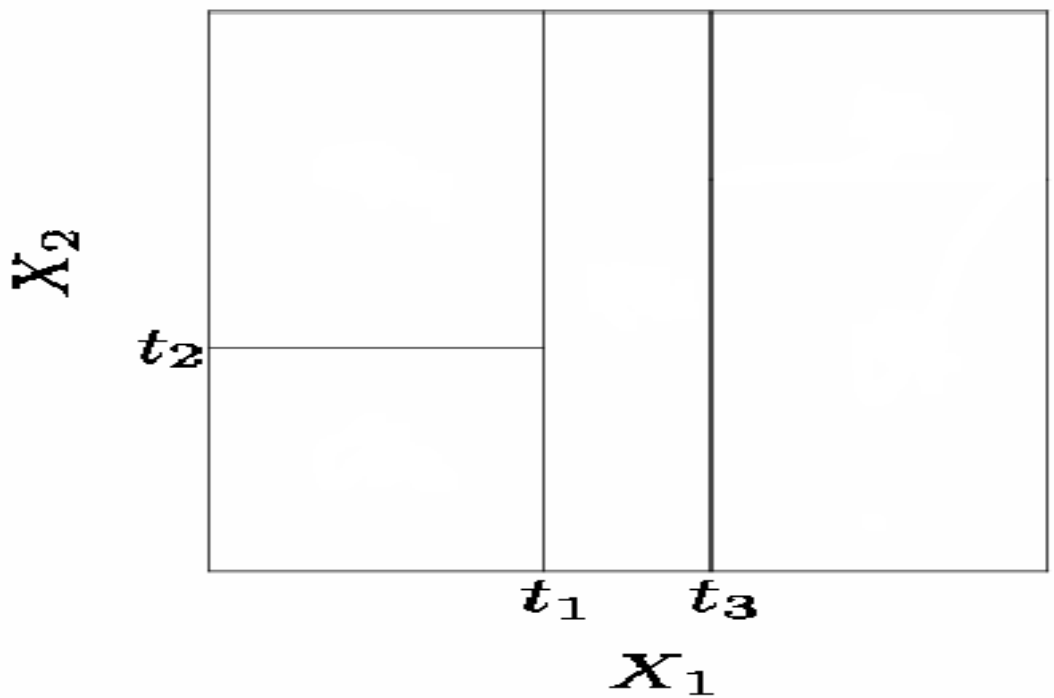
1



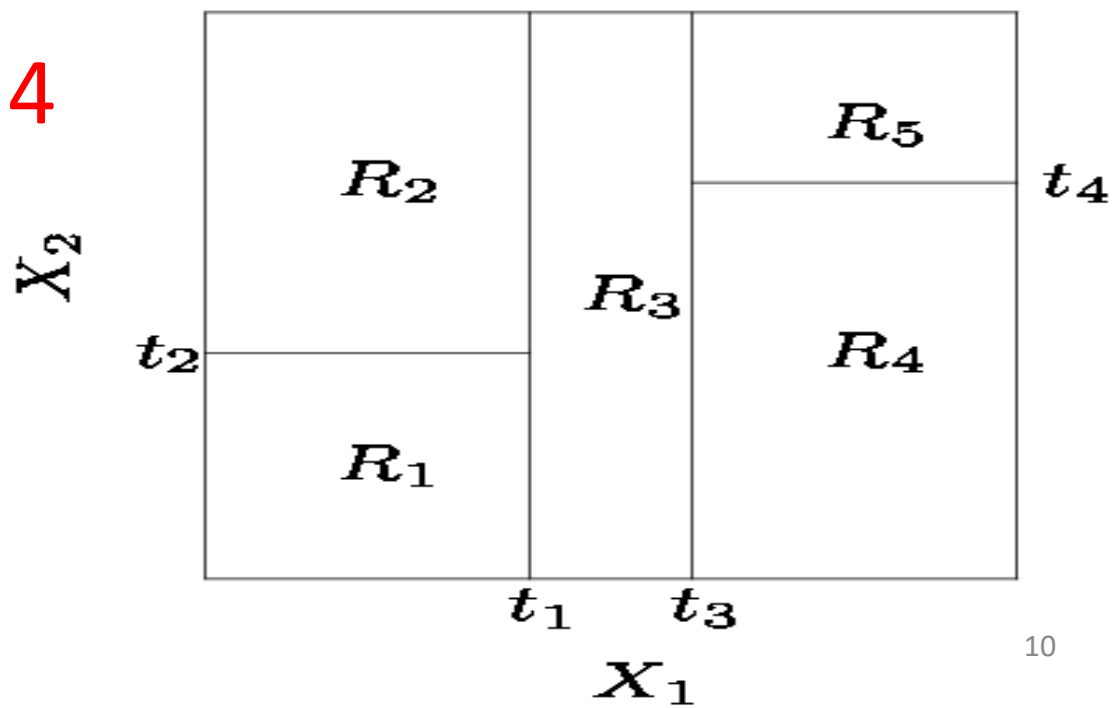
2



3



4



Avoid Overfitting

- We can have **very long trees** ➔ Complex tress would **overfit** the data
  - Tree can grow until there is one sample (one one label) per region
- Short trees (fewer splits/regions) are easy to interpret ➔ have low variance, but may have high bias
- We want to achieve a good bias variance trade-off
- **Pre-pruning: Set a stopping criteria to prevent overfitting.**
  - Like **limit** on **depth** of tree, or number of **observations per region**, number of leaves (regions), threshold on reduction of RSS
- **Post-pruning (or just pruning)**: Grow a very long tree using training data, then **prune** it to obtain a subtree
  - In each step, check if you remove a node, how well the pruned tree work on the validation set
  - Get the tree that minimizes average cross validation error

Classification Trees

- Classification trees are used to predict a quantitative response
- Prediction: assign the observation to the **most commonly occurring class** in the region
- Similar to regression trees, **recursive binary splitting** is used to grow the tree
  - However, instead of using the RSS in regression we use other metrics

Metrics for Growing a Tree

- Let $\hat{p}_{mk}$ be the proportion of training observations in the ***m*th region** that are from the ***k*th class**

- **Classification error rate:**
  - **Training observations in the region that do not belong to the most common class**
    - **Error in region m is E: fraction of observation that does not belong to the common class**

$$E = 1 - \max_k(\hat{p}_{mk})$$

- Other metrics are more common than error rate for growing a tree:
  - These metrics are: **Gini index** and **Entropy (Cross-entropy)**

Metrics for Growing a Tree – Gini Index

- **Gini index (*G*):** is a measure of a **node purity**
  - Nodes are pure when they contain observations that belongs to a single class

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

  - **When nodes are pure, the Gini index value is zero**
    - Example: consider K=2 (two classes), and all observations in region *m* belong to class 1. Then $\hat{p}_{m1} = 1$ and $\hat{p}_{m2} = 0$ ➔ In this case the Gini index G=0
  - **Gini index is maximal if classes are mixed**
    - Example: consider K=2 (two classes), and half observations in region *m* belong to class 1 and the other half belongs to class 2. Then $\hat{p}_{m1} = 0.5$ and $\hat{p}_{m2} = 0.5$➔ In this case the Gini index G=0.5 (maximum)

Metrics for Growing a Tree – Cross-Entropy

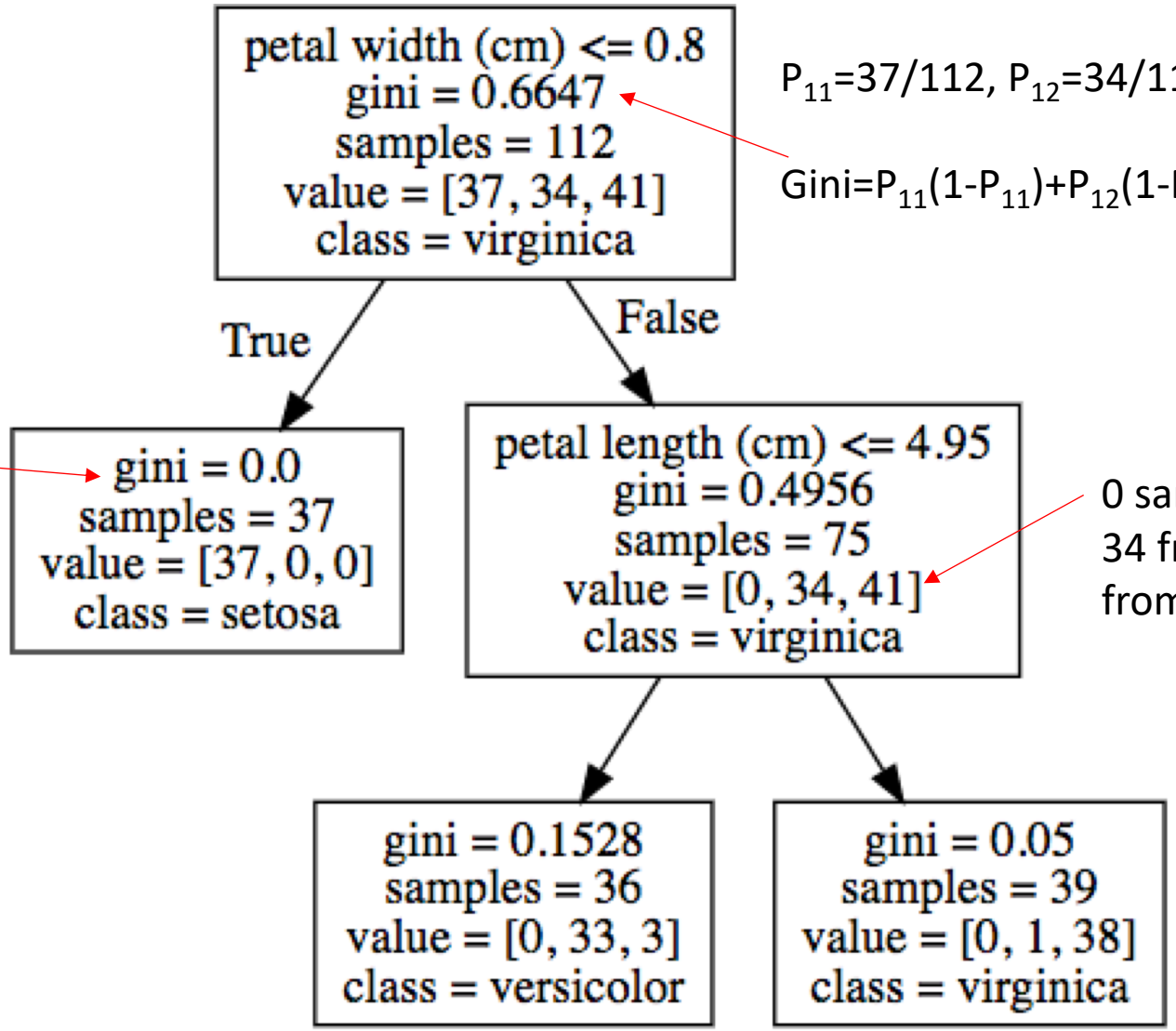
- Entropy is a measure of uncertainty
- Similar to the Gini metric, it measures node purity
- Cross entropy is defined as:

(note the log here is base 2)
$Log_2(p)=ln(P)/ln(2)$

$$D = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}$$

  - D is close to **zero** when nodes are almost **pure**
  - D is maximum if classes are mixed.
- We search for splits the minimizes the metric (Gini or cross-entropy) in a recursive way as we did before

Example: Iris Dataset



P_{11}=37/112, P_{12}=34/112, P_{13}=41/112

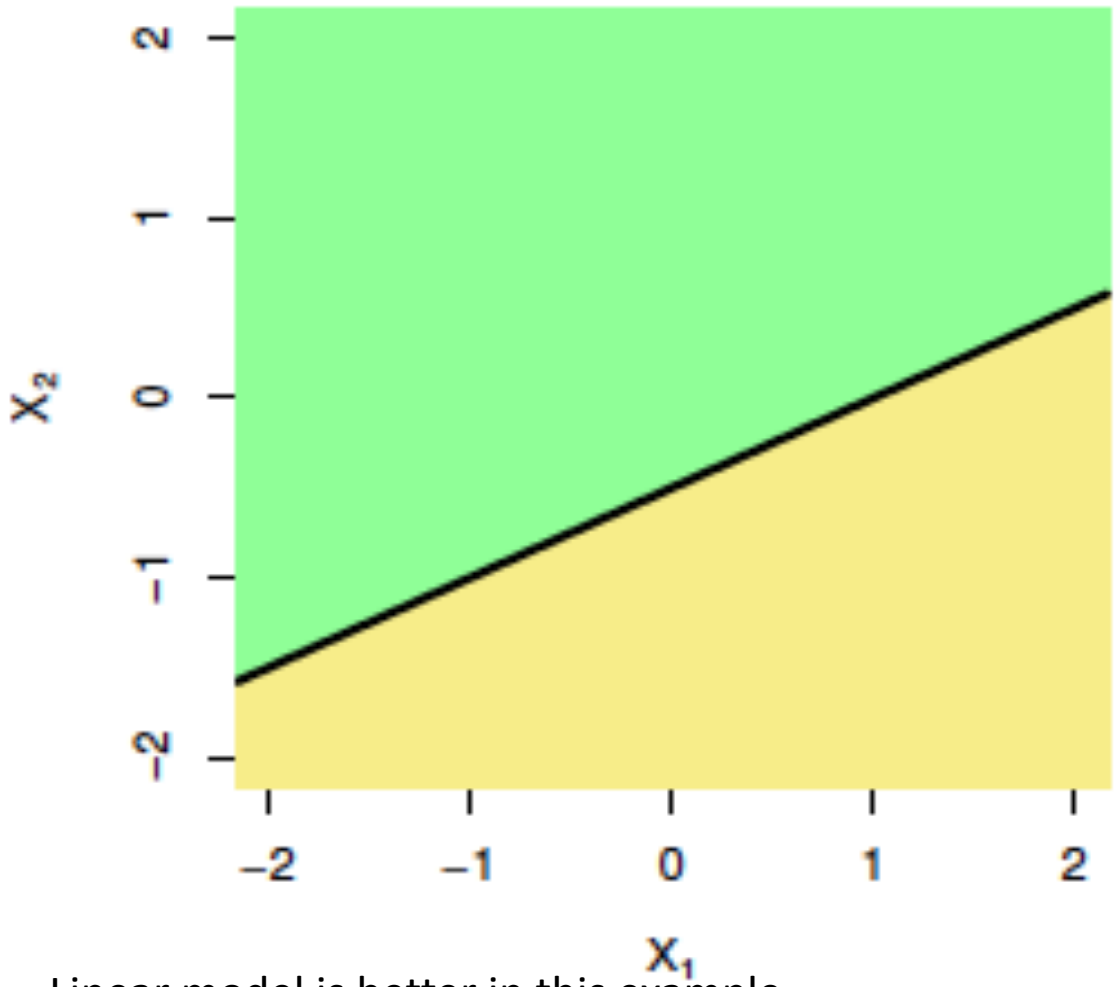
Gini=$P_{11}(1-P_{11})+P_{12}(1-P_{12})+P_{13}(1-P_{13})$=0.6647

Gini index is 0 since all samples belong to a in this region belongs to a single class (setosa)

0 samples from class 1 (Setosa) 34 from class 2 (Versicolor), and 41 from class 3 (Virginica)

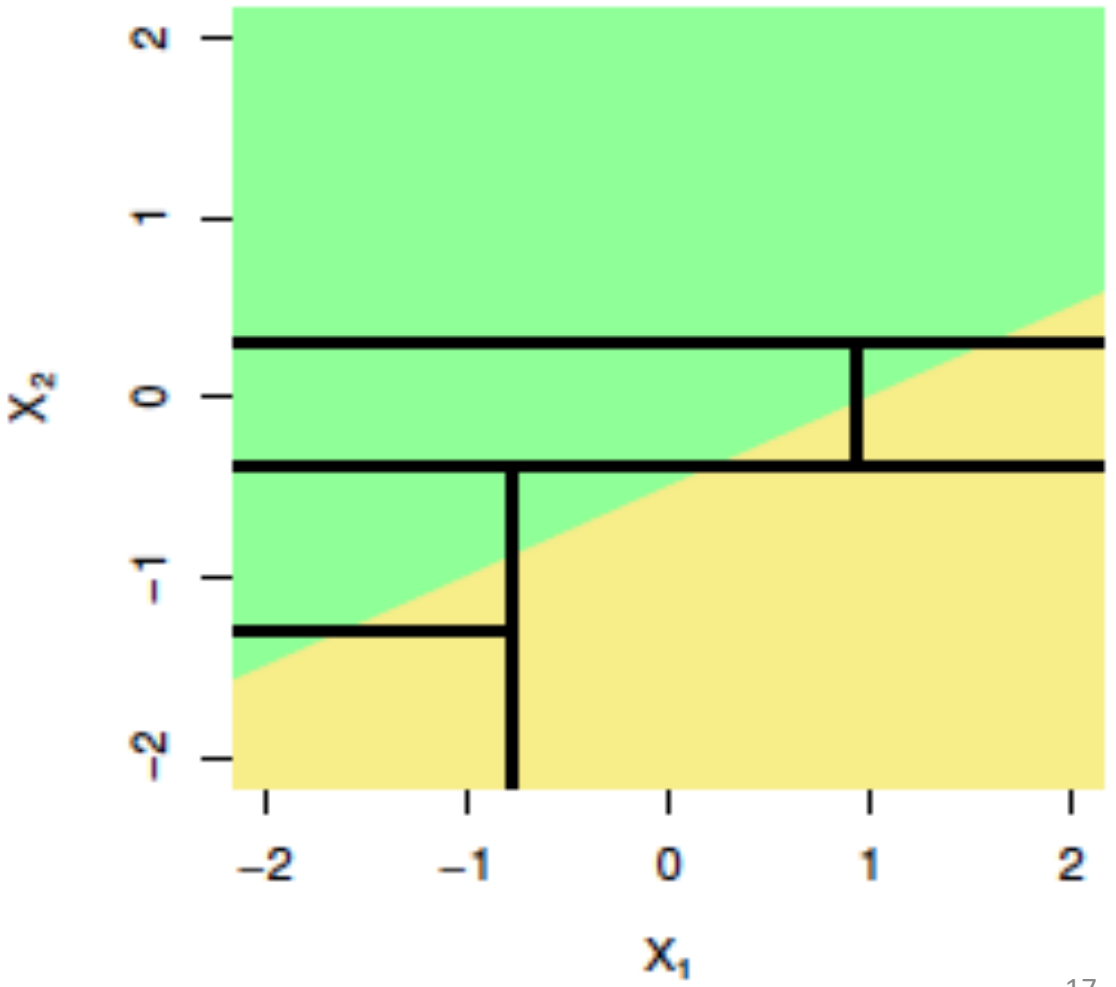
Example Tree vs Linear Model Boundaries

Linear model



Linear model is better in this example

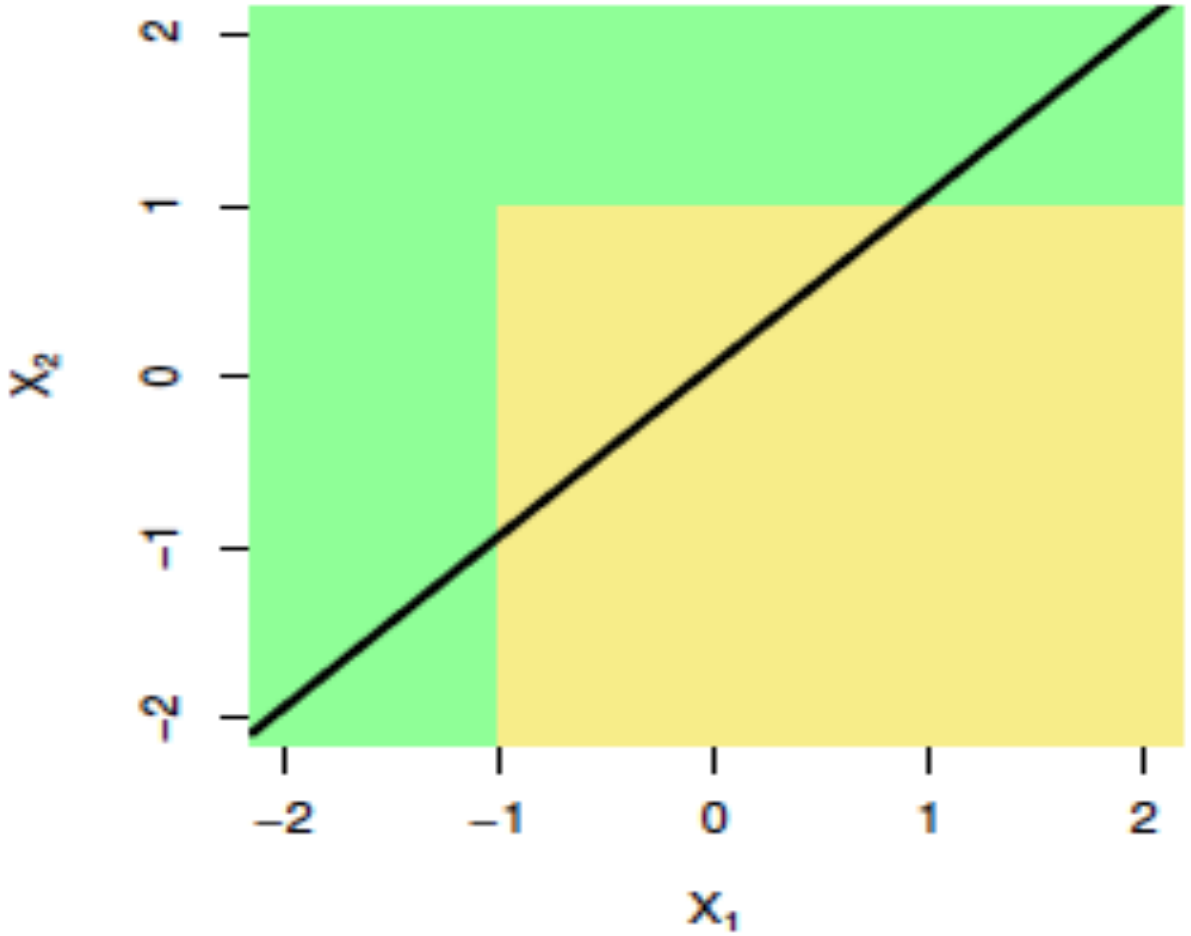
Tree model



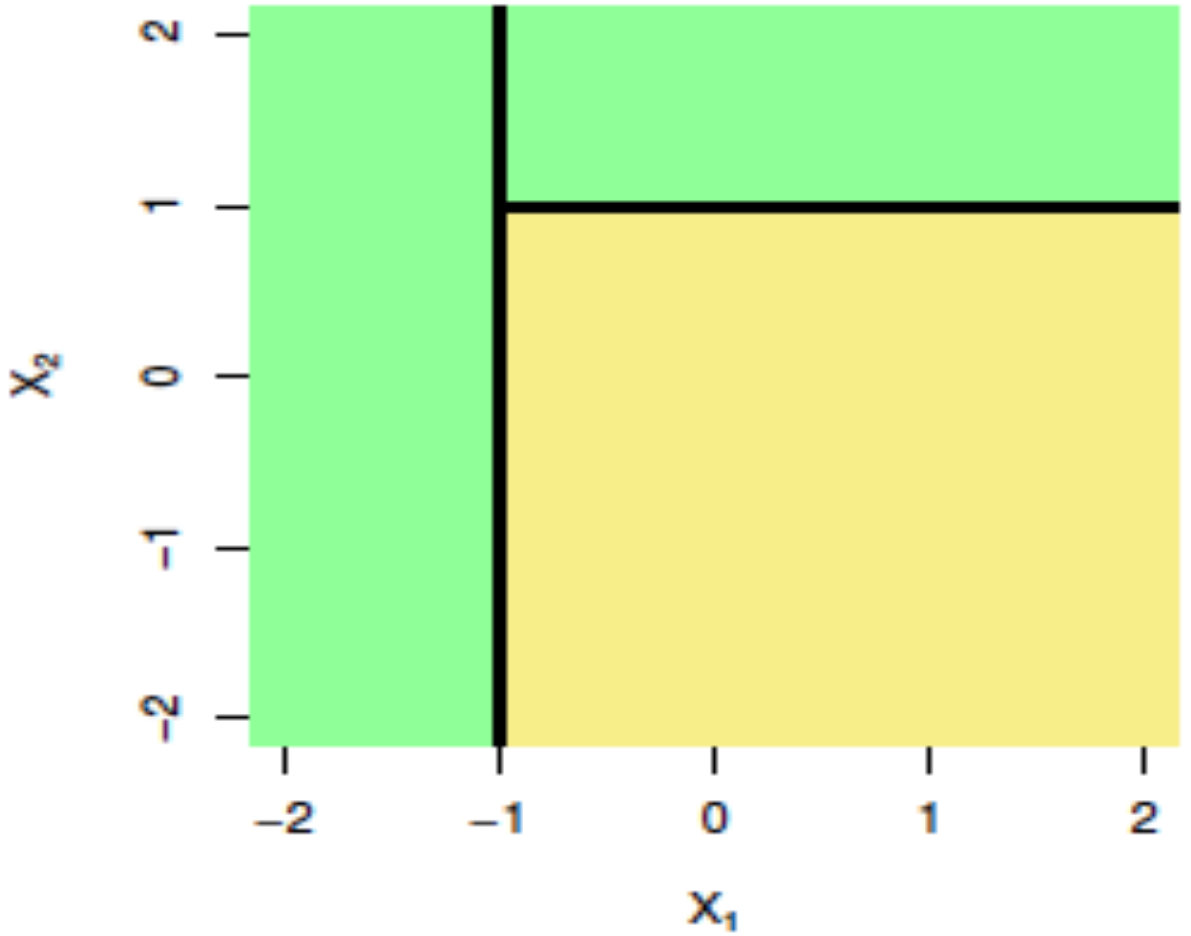
Tree splits the space into rectangular regions

Example Tree vs Linear Model Boundaries

Linear model



Tree model



Tree model is better in this example

Decision Trees: Advantages and Disadvantages

- **Very useful for interpretation,** but generally they **do not have the same level of accuracy** compared to other approaches

- Low accuracy would be from overfitting the training data
  - If you have other training samples, maybe splitting rules are different

- Can be very accurate when we combine multiple trees together:
  - Helps in avoiding overfiting
  - Combining trees improve the **accuracy**, but are not **interpretable**
  - Examples of approaches that do combining that are **random forest**, **boosting**

Combining Trees

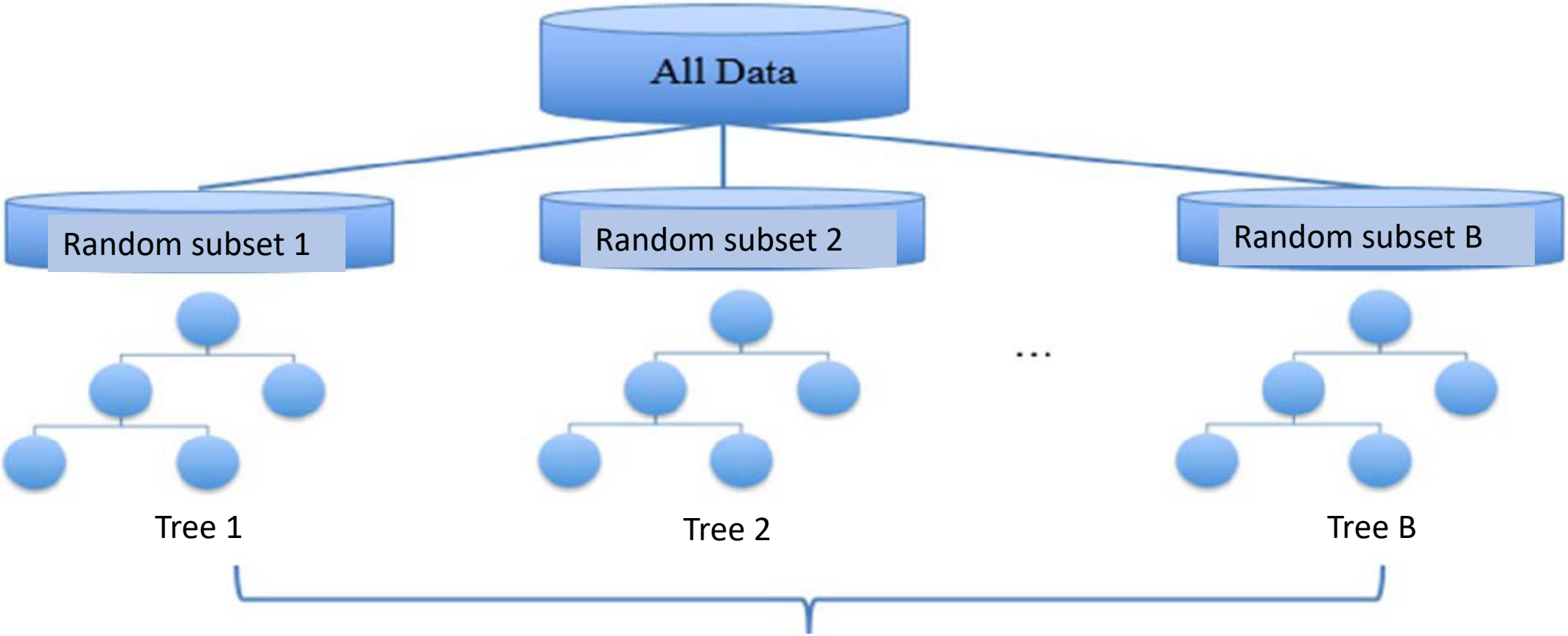
Combining methods are also called **ensemble methods**

- Bagging (not specific to decision trees)
- Random Forests
- Boosting (not specific to decision trees)

Bagging (Bootstrap)

- **General procedure for reducing the variance of a statistical learning method** (not specific to decision trees)
- Idea:
  1. **Create multiple training sets** from the the original training set – Say we now have **B training subsets**
     - **Repeated sampling with replacement** (meaning that an observation can be found in more than one of these subsets)
  2. **Train different models** (B models) each is trained with a different training subset
  3. **Get overall response using B models for presiction**
     - For regression: predicted response is **average** of predicted responses by different models
     - For classification: predicted response is obtained by **majority vote** from different models

Bagging



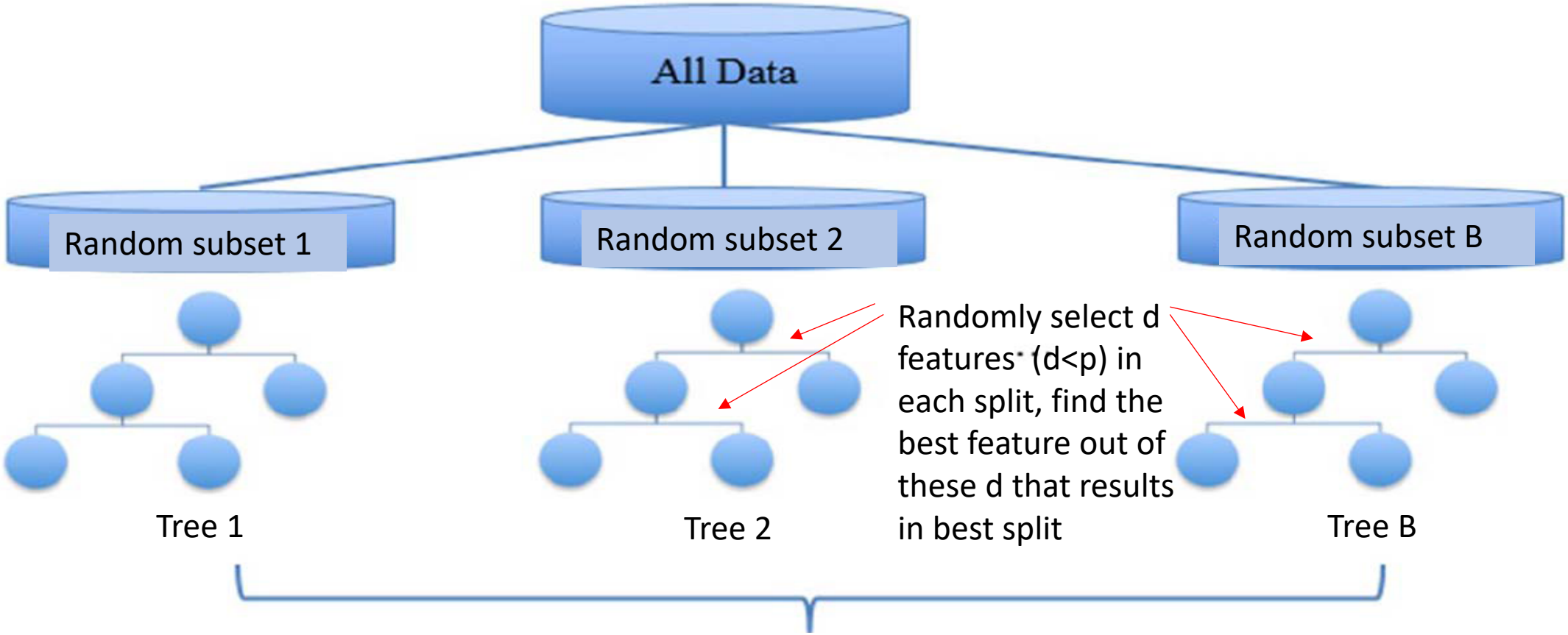
Use voting for classification, or averaging for regression to get final response

Ref.: Kun Xie et al, ASCE 2017

Random Forests

- Have better classification performance compared to bagging
- In Radom forest we build a number of decision trees on bootstrapped training samples
- Similar to Bagging, however, in each split in a tree, we randomly select subset of features (d features) and find the best one out of these d features for splitting
  - This is instead of considering all features and finding the best one
  - We can select $d = \sqrt{p}, p/2 \ldots$
- Aggregate the results through voting (classification) or averaging (regression)

Random Forests



Use voting for classification, or averaging for regression to get final response

Boosting

- **General approach that can be applied to statistical learning methods** (not specific to decision trees)

- **Purpose**: **Combine output of multiple weak learners to produce a powerful learner**

- In bagging, the models created are independent of each other

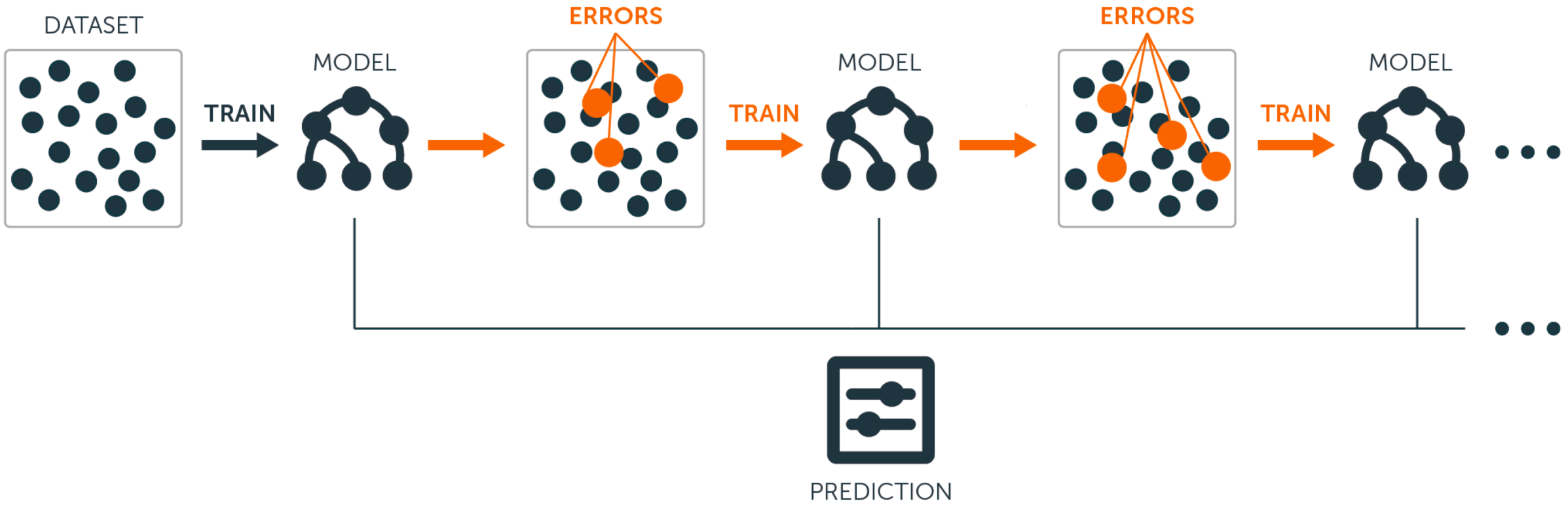
- In boosting, models (e.g. decision trees) are gown sequentially, and each model uses information from previous model

Adaptive Boosting (AdaBoost)

- AdaBoost (1997) is a popular boosting algorithm

- When applied with classification decision trees:
  - Build trees in sequence
  - Each tree can be very small (only few nodes)
  - Each tree **focuses on areas where previous trees do not perform well**
    - Made through assigning a weight for each data point, and the weight gets higher when the point is misclassified
  - Combine results using
    - **Weighted** majority vote used for classification
    - **Weighted** average can be used for regression

AdaBoost: Models Trained in Sequence



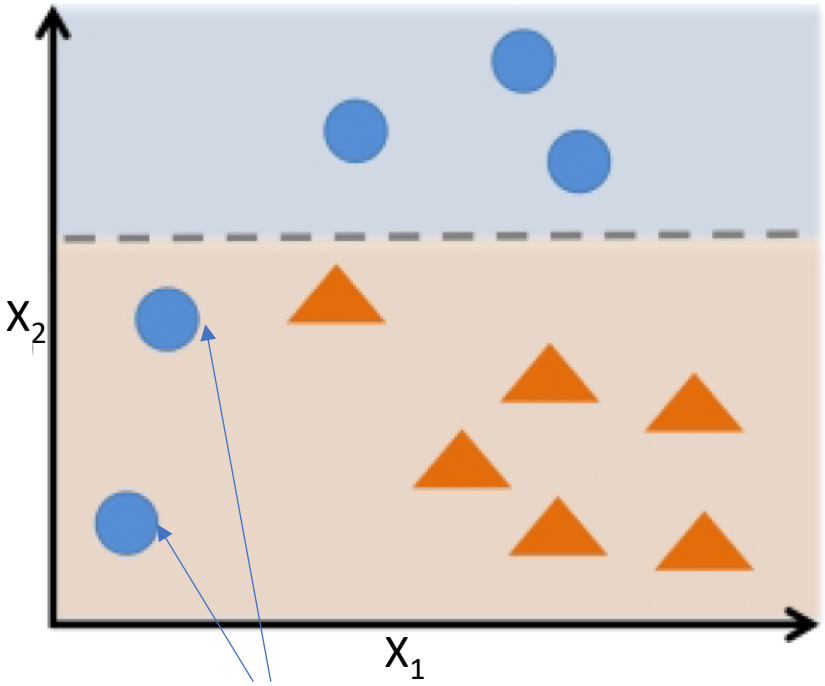
https://blog.bigml.com/2017/03/14/introduction-to-boosted-trees/

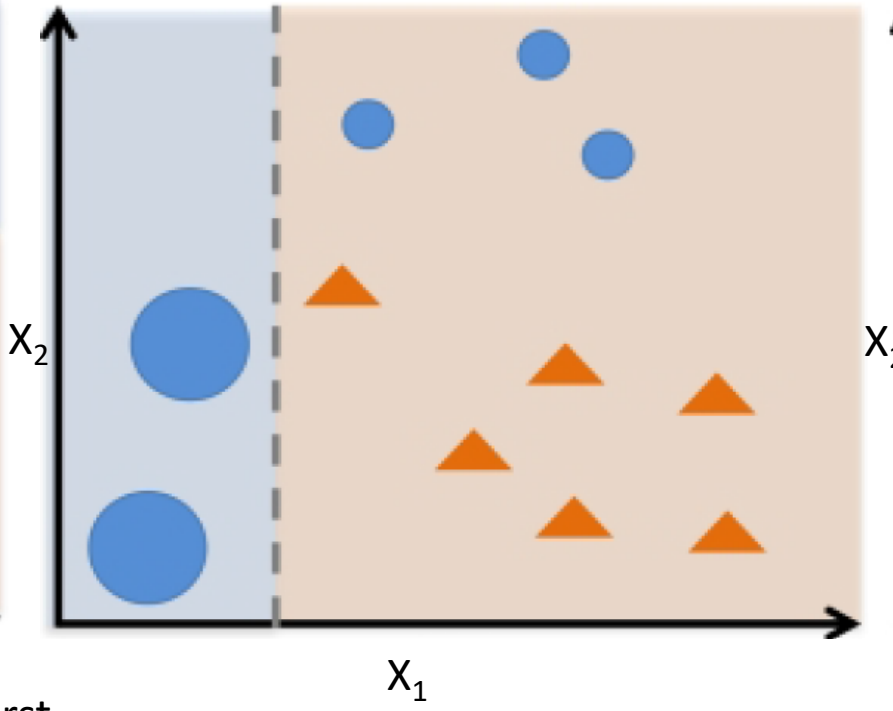
Example: AdaBoost with 3 Learners (2 features)

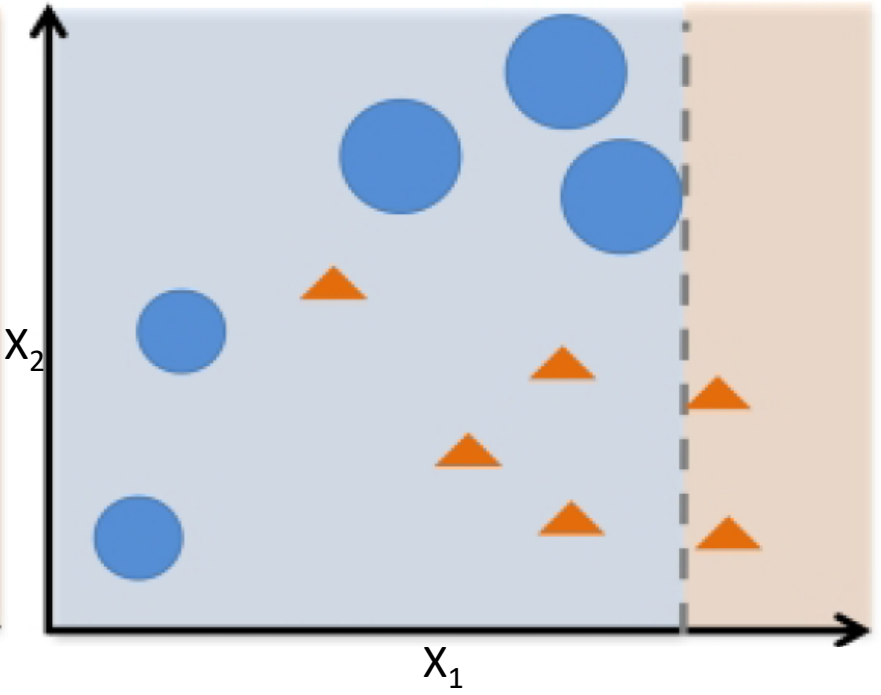
First tree (weak learner) result in this splitting of feature space

Second tree focuses on misclassified samples from previous tree (have higher weight) and results in this splitting of feature space

Third tree focuses on misclassified samples from previous and results in this splitting of feature space





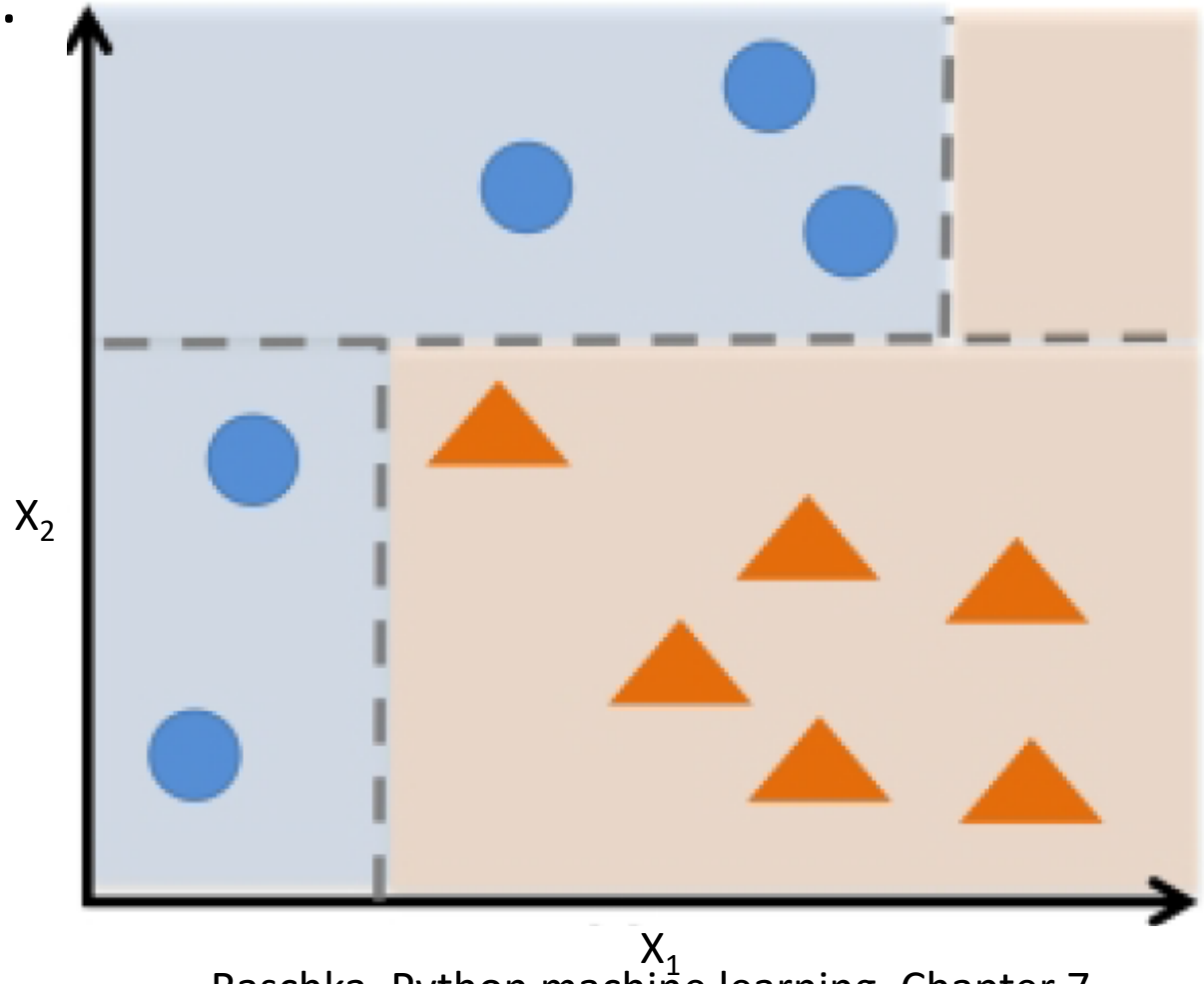


These samples were misclassified by first tree, will get higher weight

Raschka, Python machine learning, Chapter 7

Example: AdaBoost with 3 Learners ...Cont.

- When combining the results of the previous three learners using **weighted majority vote** we get:



Raschka, Python machine learning, Chapter 7

AdaBoost

- Consider two classes, with response coded as Y=-1 or Y=1, and feature x
- We have M classifiers:
  - The prediction of classifier m is $G_m(x)$, m=1,2,…M
    - $G_m(x)$ is either -1 or 1 (predicted class label)
- The **final prediction** from all of classifiers are combined through **weighted majority vote**

$$G(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m G_m(x)\right)$$

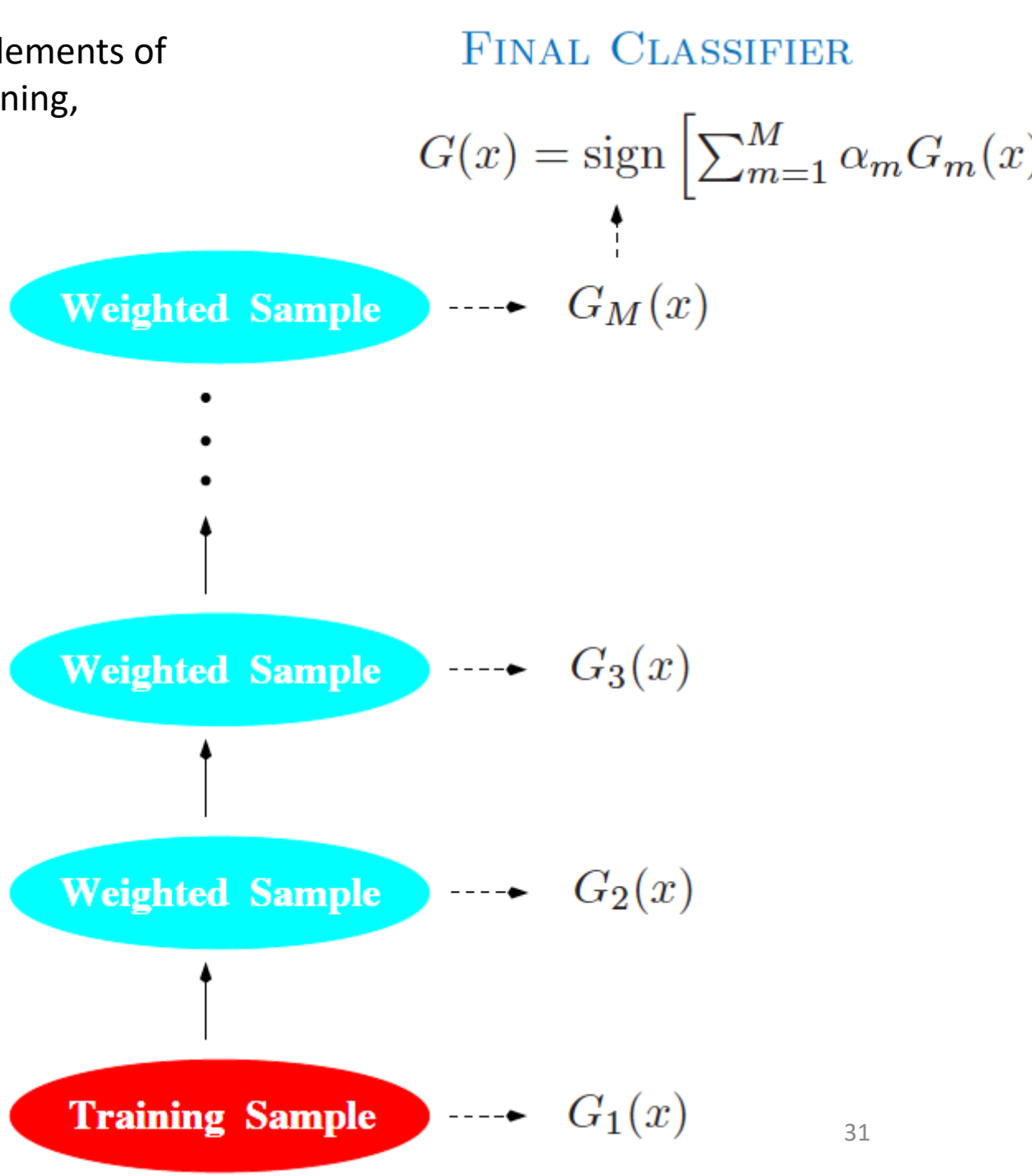
Note:
sign(F) = -1, if F is negative
sign(F) = 1, if F is positive

- α_m are the weights computed by the boosting algorithm

AdaBoost

Hastie et al., Elements of Statistical Learning, Chapter 10

- Every boosting step we apply **weights to each training observation**
  - Initially weights are set to 1/n for all observations
  - For successive iterations, the weights are modified
    - Observations that are **misclassified** have their **weights increased**
    - Observations that were **correctly classified** have their **weights decreased**
- Each successive classifier is hence forced to concentrate on training observations that are missed by previous ones



- Note: if predicted labels of 5 classifiers are {1,1,1,-1,-1}
  - Typical majority gets final prediction to be 1
  - Weighted majority with weights {0.1,0.1,0.1, 0.6,0.1) results in -1
    - 0.1x1 +0.1x1 +0.1x1 +0.6x(-1) +0.1 x(-1) = - 0.4 < 0 ➔Final prediction is -1

AdaBoost Algorithm

Raschka, Python machine learning, Chapter 7

1. Initialize equal weights of all observation (w_i=1/n)
2. For m from 1 to M, (loop over classifiers in sequence)
   1. train classifier m: $G_m(x)$
   2. Compute **weighted error**: $err_m = \sum_{i=1}^{n} w_i \, I\left(y_i \neq G_m(x_i)\right)$
   3. Compute weights of classifier m: $\alpha_m = 0.5 \log_e \left(\frac{1-err_m}{err_m}\right)$
      - α_m is higher when errors are lower
   4. Update weights of each observations
      - $\boldsymbol{w_i \leftarrow w_i \exp(-\alpha_m y_i \hat{y}_i)}$
        - The exponent $(-\alpha_m y_i \hat{y}_i)$ is negative if $y_i = \hat{y}_i =$ ➔ ***weight*** **decrease if samples are classified correctly**
          - Otherwise, exponent is positive ➔ weight increase misclassified samples
      - Normalize the final weights by dividing by $\sum_{i=1}^{n} w_i$

$$Note: I\left(y_i \neq G_m(x_i)\right) = \begin{cases} 0 & If\ y_i = G_m(x_i) \\ 1 & If\ y_i \neq G_m(x_i) \end{cases}$$

Example:

In this example we decide y=1 if $x \leq 3$

Sample indices	x	y	Weights	$\hat{y}$(x <= 3.0)?	Correct?	Updated weights
1	1.0	1	0.1	1	Yes	0.072
2	2.0	1	0.1	1	Yes	0.072
3	3.0	1	0.1	1	Yes	0.072
4	4.0	-1	0.1	-1	Yes	0.072
5	5.0	-1	0.1	-1	Yes	0.072
6	6.0	-1	0.1	-1	Yes	0.072
7	7.0	1	0.1	-1	No	0.167
8	8.0	1	0.1	-1	No	0.167
9	9.0	1	0.1	-1	No	0.167
10	10.0	-1	0.1	-1	Yes	0.072

err_1 =0.3, α_1=0.5*ln((1-0.3)/0.3)=0.4 ➔ get updated weights in last column, the normalize to get values in last column

Calculation details in Raschka, Python machine learning, Chapter 7

Python

- Decision trees: For classification
- from sklearn.tree import **DecisionTreeClassifier**
  treeModel==DecisionTreeClassifier(max_depth=m, criterion='gini')
  - max_depth is depth of the tree
  - Default criterion is 'gini' (so no need to write it)

  - http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

- Decision trees: For regression, **DecisionTreeRegressor** is used
  - http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html#sklearn.tree.DecisionTreeRegressor

Python

- Random Forest
  from sklearn.ensemble import RandomForestClassifier
  forestModel= RandomForestClassifier (n_estimators=M, max_features=d, max_depth=m, random_state=0)
  - n_estimators: number of trees to grow in the forest
  - max_features: maximum number of features considered to find best split
  - max_depth is depth of the tree

- AdaBoost:
  from sklearn.ensemble import AdaBoostClassifier
  BoostModel= AdaBoostClassifier(n_estimators=M)
  - Take base_estimator parameter which is DecisionTreeClassifier by default
  - n_estimators is the number of models