2695 Introduction to Machine Learning
Masters Program in Economics, Finance and Management

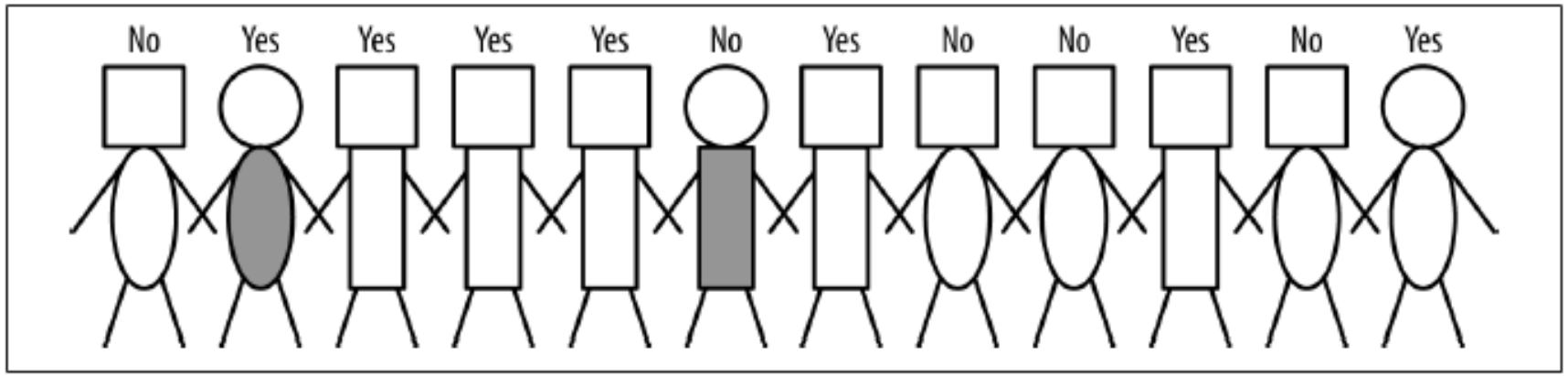
NOVA SCHOOL OF BUSINESS & ECONOMICS

5a Video Lecture

DECISION TREE

Selecting informative features

- Which of the attributes would be best to segment these people into groups, in a way that will distinguish those with label YES from those with label NO?



- Attributes:
  - — head-shape: square, circular
  - — body-shape: rectangular, oval
  - — body-color: gray, white

Target variable:

— write-off: Yes, No

Examples in this section are from: Data Science for Business: What You Need to Know about Data Mining and Data-Analytic Thinking, F. Provost, T. Fawcett

Splitting into less impure groups

- Goal: after split, resulting groups as *pure* as possible
  homogeneous with respect to the target variable.
  - Pure group: every member of a group has the same value for the target.
  - Impure group: there is at least one member of the group that has a different value for the target variable than the rest of the group.

- Objective: Based on customer features, partition the customers into subgroups that are less impure – with respect to the class.

Entropy

- Entropy: measure of disorder
- Disorder: how mixed (impure) the segment is with respect to these properties of interest.

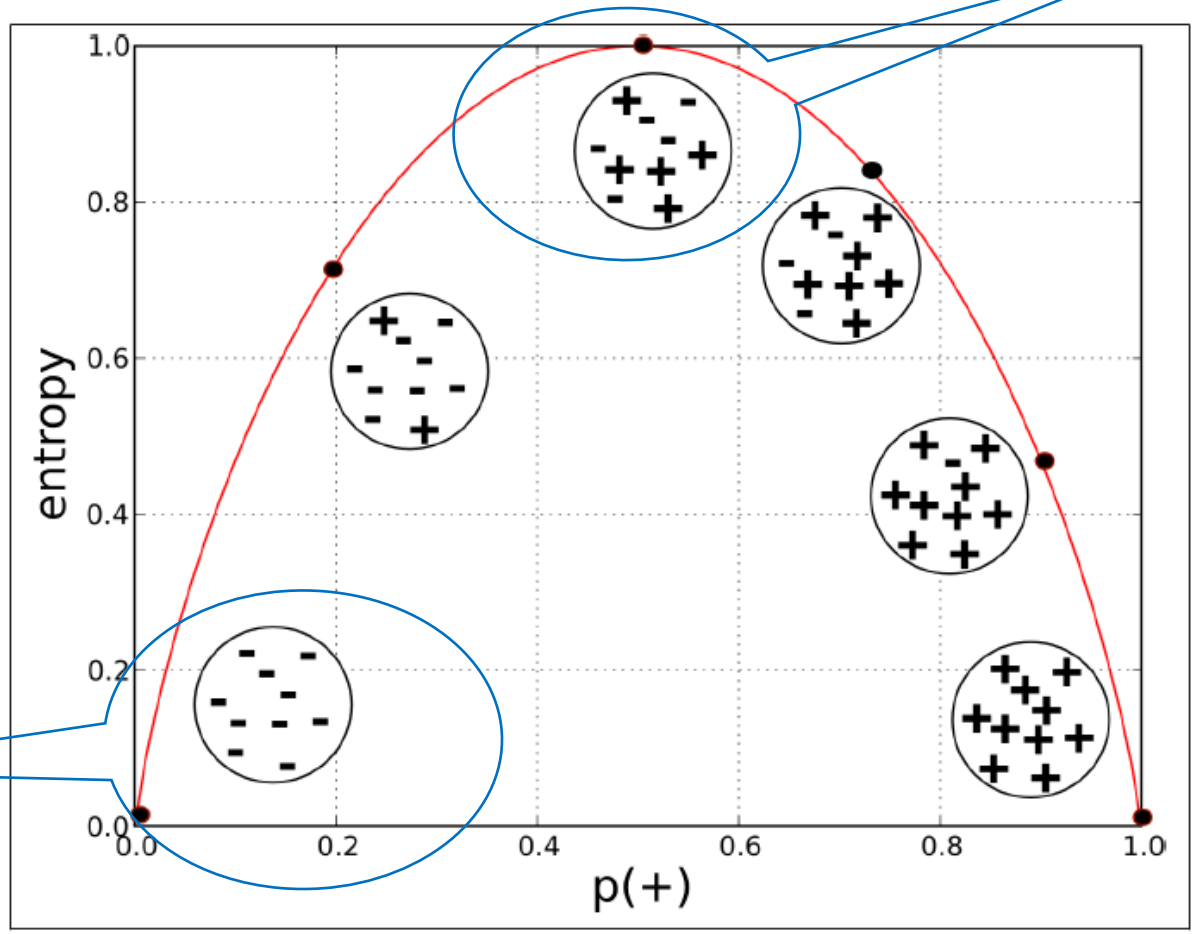
$$entropy = - p_1 \log(p_1) - p_2 \log(p_2) - \cdots$$

- Each p_i is the probability (the relative percentage) of property *i* within the set:
  p_i = 0 when no members of the set have property *i*
  p_i = 1 when all members of the set have property *i*
- Note: Log base 2

Entropy: example

$$entropy = - p_1 \log(p_1) - p_2 \log(p_2) - \cdots$$

Highest Entropy:
$p_-=p_+=5/10=0.5$
Entropy= $-0.5\log_2 0.5 - 0.5 \log_2 0.5 = 0.5 + 0.5 = 1$



Minimal Entropy:
$p_-=10/10=1$, $p_+=0/10=0$
Entropy= $- 1 \log_2 1 - 0 \log_2 0 = 0$
(convention: $0 \log 0 = 0$)

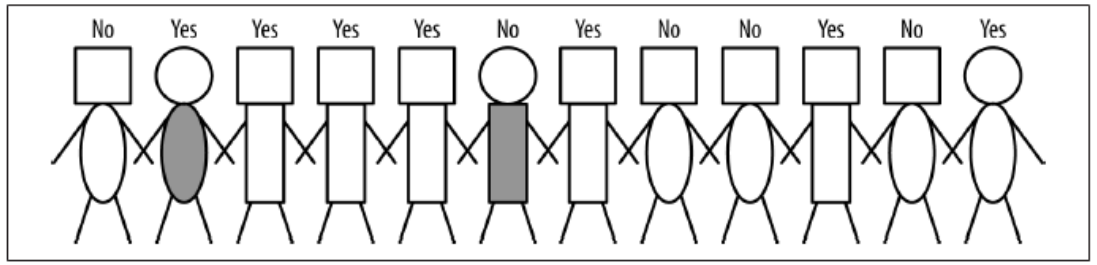
Entropy of a set containing 10 instances of two classes

- probability (the relative percentage) of a positive class : p_+
- probability (the relative percentage) of a negative class $p_- := 1 - p_+$

Information gain

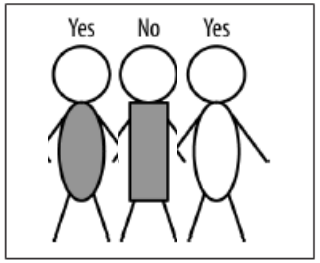
- **Information gain (IG)**: change in entropy due to any amount of new information being added.
  How much an attribute improves (decreases) entropy over the whole segmentation it creates.

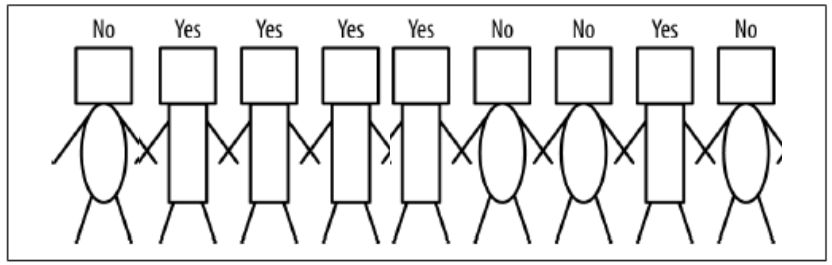
- Feature we split on has *K* different values.
  - Parent set: original set
  - K Children sets: result of splitting on the attribute values



parent set

children set 1



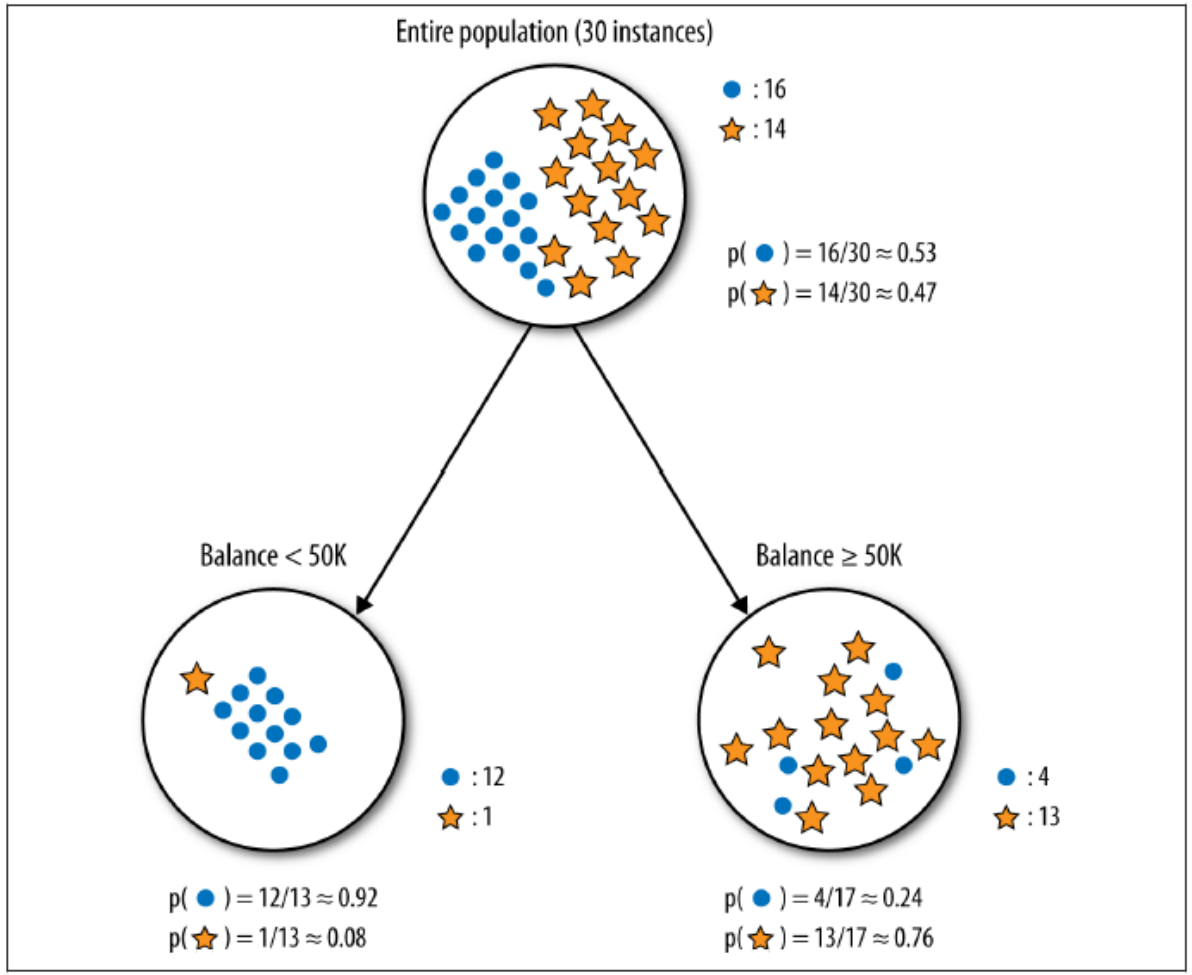


children set 2

$$IG(parent, children) = entropy(parent) - [p(c_1) \times entropy(c_1) + p(c_2) \times entropy(c_2) + \cdots]$$

Calculating information gain

$$IG(parent, children) = entropy(parent) - [p(c_1) \times entropy(c_1) + p(c_2) \times entropy(c_2) + \cdots]$$



$$\begin{aligned} entropy(parent) &= -[p(\bullet) \times \log_2 p(\bullet) + p(\star) \times \log_2 p(\star)] \\ &\approx -[0.53 \times -0.9 + 0.47 \times -1.1] \\ &\approx 0.99 \quad \text{(very impure)} \end{aligned}$$

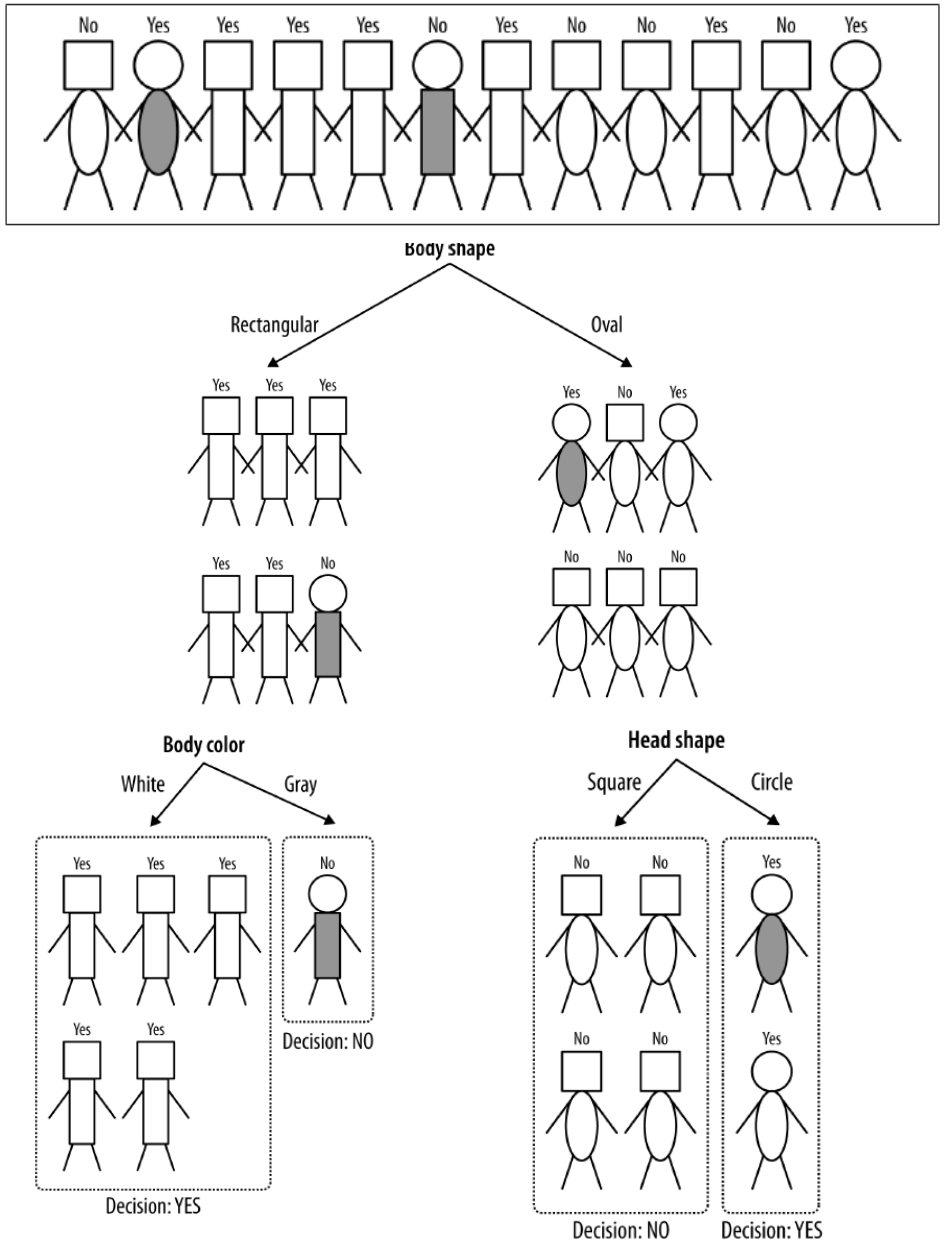
$$\begin{aligned} entropy(Balance < 50K) &= -[p(\bullet) \times \log_2 p(\bullet) + p(\star) \times \log_2 p(\star)] \\ &\approx -[0.92 \times (-0.12) + 0.08 \times (-3.7)] \\ &\approx 0.39 \end{aligned}$$

$$\begin{aligned} entropy(Balance \geq 50K) &= -[p(\bullet) \times \log_2 p(\bullet) + p(\star) \times \log_2 p(\star)] \\ &\approx -[0.24 \times (-2.1) + 0.76 \times (-0.39)] \\ &\approx 0.79 \end{aligned}$$

$$\begin{aligned} IG &= entropy(parent) - [p(\text{Balance} < 50\text{K}) \times entropy(\text{Balance} < 50\text{K}) \\ &\quad + p(\text{Balance} \geq 50\text{K}) \times entropy(\text{Balance} \geq 50\text{K})] \\ &\approx 0.99 - [0.43 \times 0.39 + 0.57 \times 0.79] \\ &\approx 0.37 \end{aligned}$$

Splitting the “loan write-off” sample into two segments, by splitting the Balance attribute (account balance) at 50K

Constructing trees



1. Select the most informative feature
   - Create branch for each possible feature value

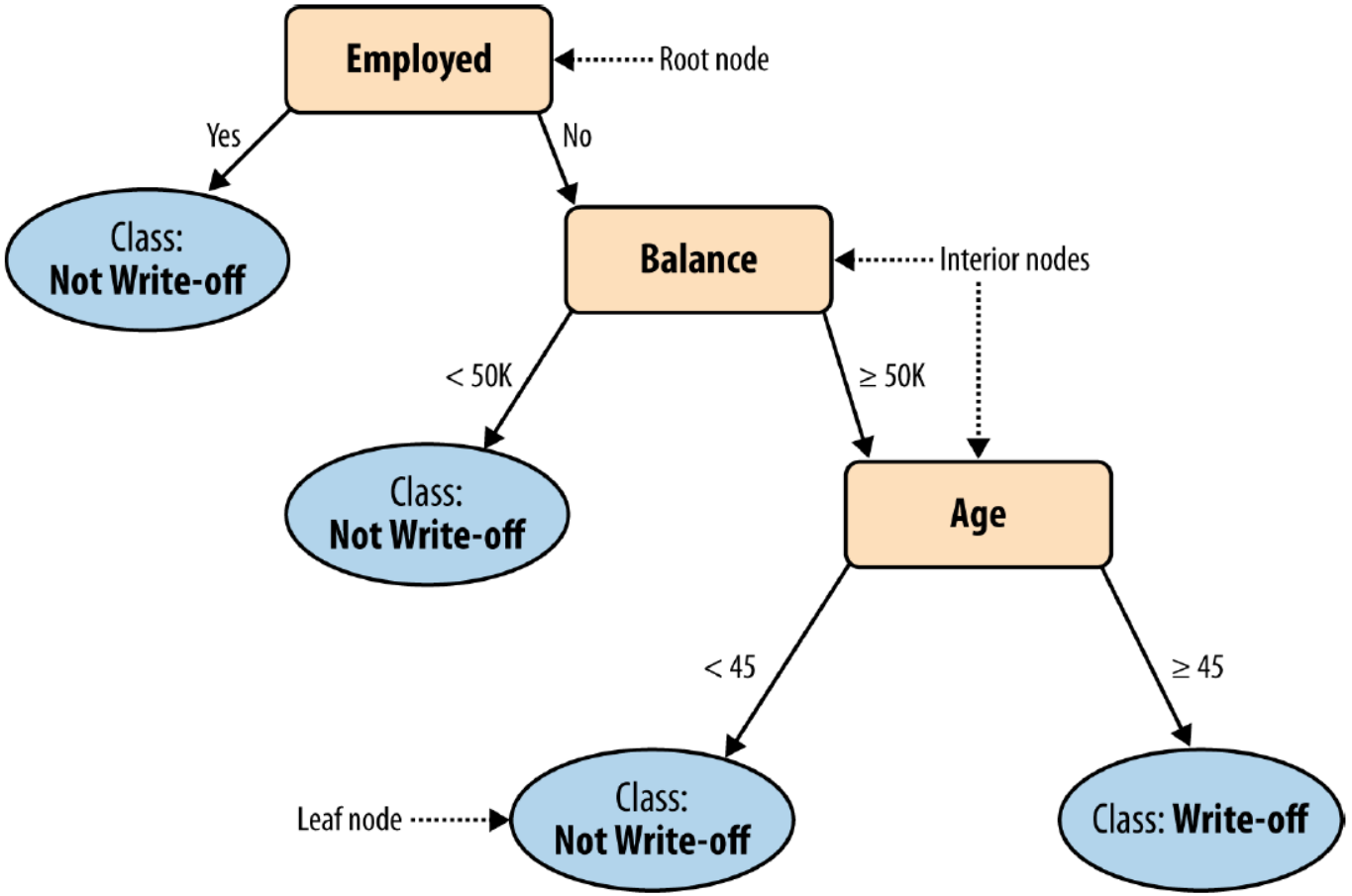
2. Split instances into child sets
   - One for each branch extending from the node

3. Repeat recursively for each branch, using only instances that reach the branch

When to stop?

- all instances have the same class
- there are no more features for splitting
- other stopping criteria are met

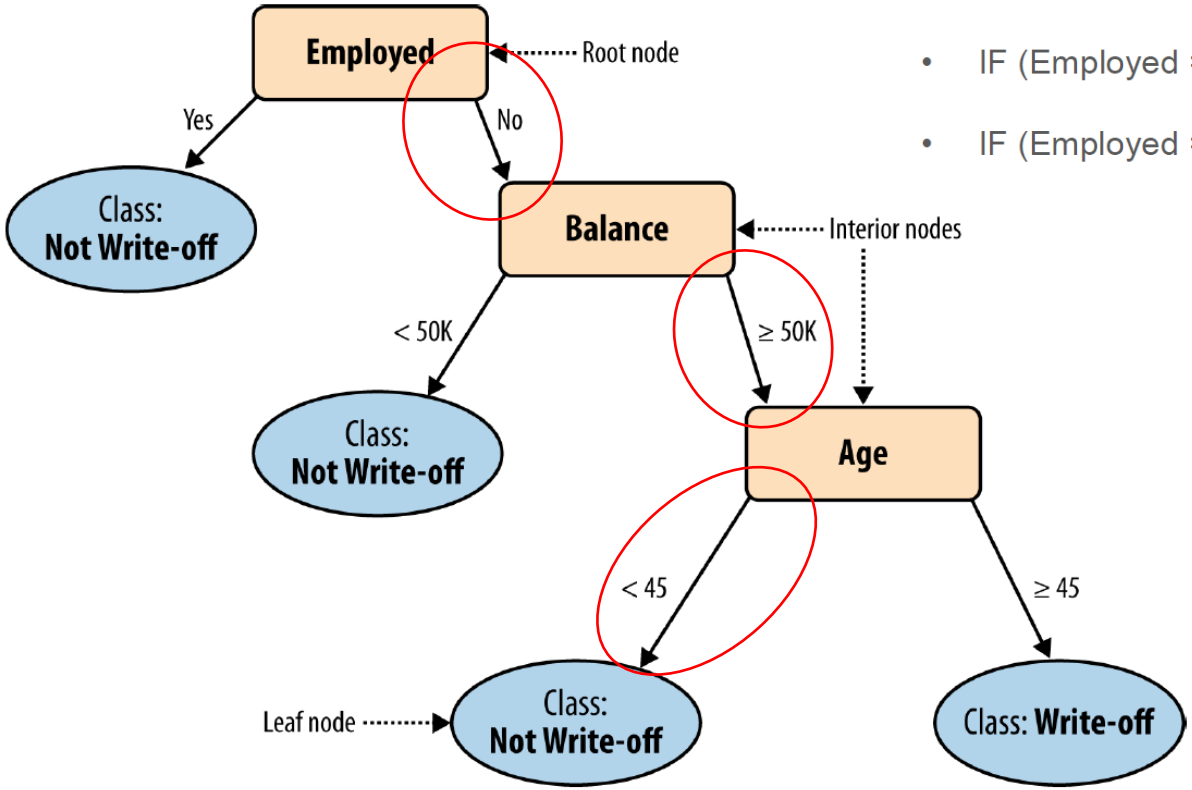
Tree structure



- Each leaf contains a subset of training data points.
- The class with the most occurrences in that leaf is assigned as the predicted class.

Classifying using trees

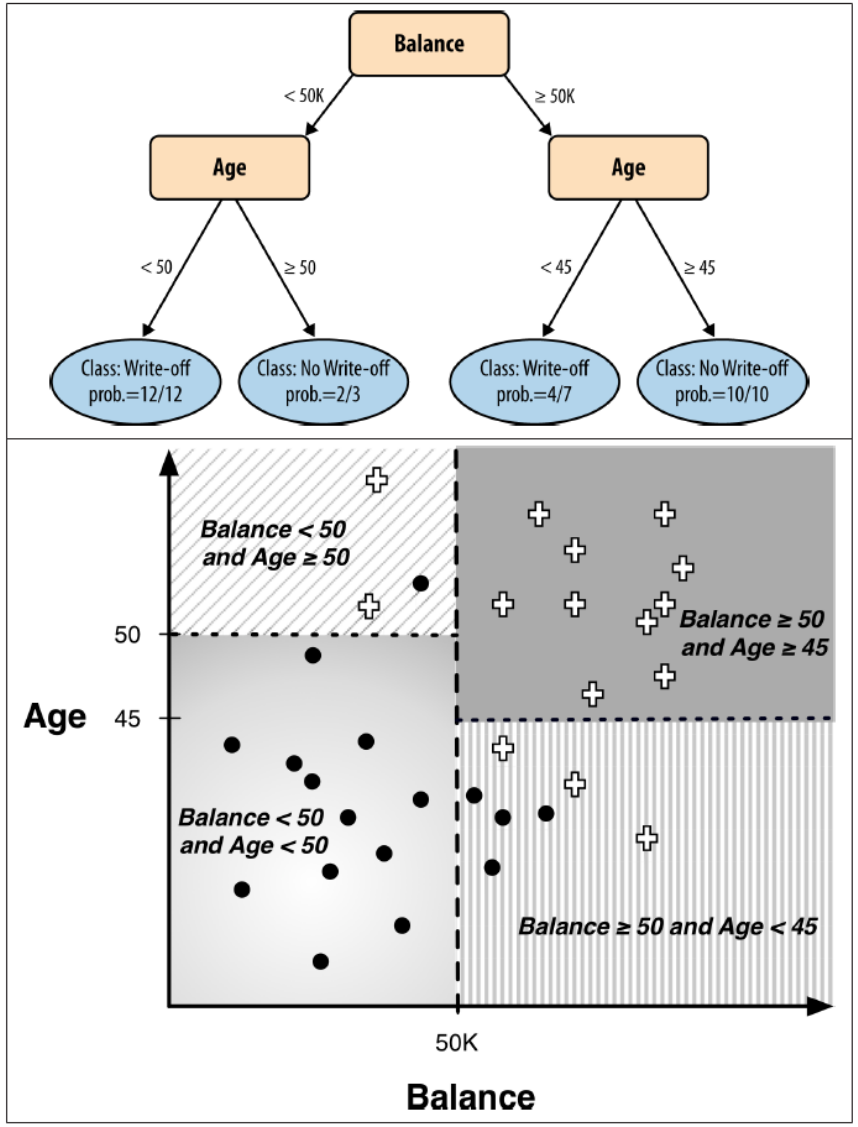
- Classify ‘John Doe’
  - Balance=115K, Employed=No, and Age=40



- IF (Employed = Yes) THEN Class=No Write-off
- IF (Employed = No) AND (Balance < 50k) THEN Class=No Write-off
- IF (Employed = No) AND (Balance ≥ 50k) AND (Age < 45) THEN Class=No Write-off
- IF (Employed = No) AND (Balance ≥ 50k) AND (Age ≥ 45) THEN Class=Write-off

John Doe classified as NOT loan write-off

Visualizing decision boundary



Each node of a classification tree tests a single feature against a fixed value so the decision boundary corresponding to it will always be perpendicular to the axis representing this feature.

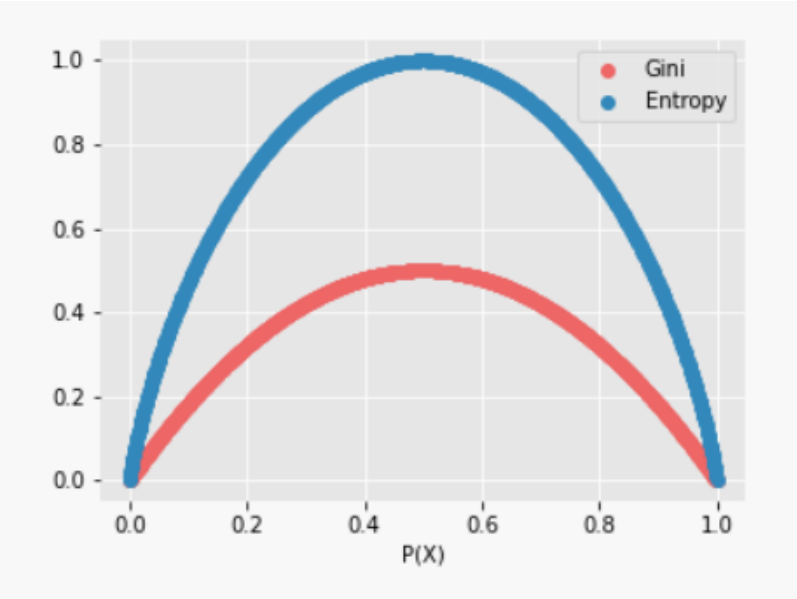
The black dots correspond to instances of the class Write-off, the plus signs correspond to instances of class non-Write-off.

Gini index: another splitting criterion

- Gini index or Gini impurity is a measure of how often a randomly chosen element from the set would be incorrectly labeled if it was randomly labeled.

$$Gini = 1 - \sum_{i=1}^{C} (p_i)^2$$

where p_i is the proportion of the samples that belong to a particular class i, and C is the number of classes.



C1	0
C2	6

P(C1) = 0/6 = 0, **P(C2)** = 6/6=1

GINI = 1 - P(C1)² - P(C2)² = 1 - 0 - 1 = 0

Minimum impurity: 0

C1	2
C2	4

P(C1) = 2/6 = 1/3, **P(C2)** = 4/6=2/3

GINI = 1 - (1/3)² - (2/3)² = 0.444

C1	3
C2	3

P(C1) = 3/6 = 1/2, **P(C2)** = 3/6 = 1/2

GINI = 1 - (1/2)² - (1/2)² = 1 - 1/4 - 1/4 = 0.5

Maximum impurity: 0.5 (binary classification)

Using Gini index for finding the best split

- When a node is split into k children sets, the quality of the split is calculated as:

$$GINI_{split} = \sum_{i=1}^{k} \frac{n_i}{n} GINI(i)$$

where n_i is the number of instances at child node *i*, and *n* is the number of instances of the parent node.

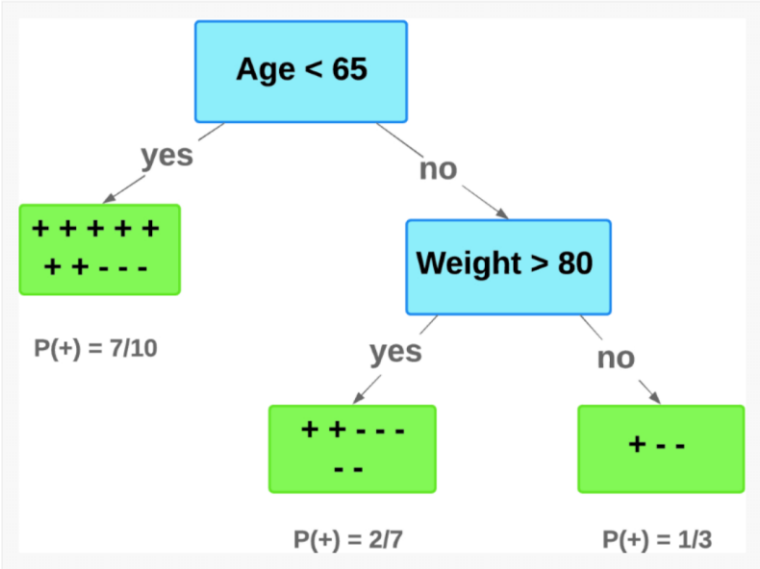
- At each split, we choose a feature with the lowest Gini Index.

- **GINI(Hot)** $= 1 - (3/4)^2 - (1/4)^2 = \mathbf{0.375}$
- **GINI(Mid)** $= 1 - (3/5)^2 - (2/5)^2 = \mathbf{0.48}$
- **GINI(Cold)** $= 1 - (3/5)^2 - (2/5)^2 = \mathbf{0.48}$
- **GINI(Children)** $= 4/14 * 0.375 + 5/14 * 0.48 + 5/14 * 0.48 = \mathbf{0.45}$

Temp.?	Hot	Med	Cold
Play	3	3	3
Stay	1	2	2
Gini	0.375	0.48	0.48
Gini(Children) = 0.45			

From classification trees to probability estimation trees

- Each leaf of a tree model: an estimate of the probability of membership to the different classes.
  - n positive instances
  - m negative instances
  - probability of any new instance being positive: $n/(n+m)$
- Laplace correction $p(c) = \frac{n+1}{n+m+2}$



Regression tree

- Instead of trying to split the training set in a way that minimizes impurity, it now tries to split the training set in a way that **minimizes the mean square error** (MSE).

- Recursive splits: Model begins with the entire data set and searches every distinct value of every input variable to find the variable and the value of split that partitions the data into regions.

- Value of the target variable is calculated as the mean of the target values of all the training instances associated with the leaf node.

Summary: Decision tree

Advantages

- Simple to understand and easy to interpret
- Require little data preparation
- Decision trees can handle both continuous and categorical variables
- Implicitly perform feature selection
- Top used classifiers for structured data are some versions of decision trees
  (structured: tabular data, unstructured: image, sound, text)

Disadvantages

- Prone to over-fitting
- Sensitive to small variations in the training data, small variations can result in a completely different tree
- Greedy approach does not guarantee the best solution

5b Video Lecture

ENSEMBLE LEARNING

Basics of ensemble learning

- Combines multiple models to improve performance
- Aggregates predictions from different models for more robust results
- Often outperforms individual models

Common types of Ensemble Learning

- Voting/Averaging
- Boosting

Voting and averaging-based ensemble methods

Actual (Target)						
Model 1						
Model 2						
Model 3						
Model 4						
Model 5						
Majority Vote						

Voting is used for classification and averaging is used for regression.

- **hard voting:** predicts the class with the most votes
- **soft voting**: averages the predicted class probabilities

Why averaging works: example

- Three binary classifiers, each independently correct with probability 0.80.
- With simple averaging, ensemble is correct if we have “at least 2 right”:
  - P(all 3 right) = 0.8^3= 0.512.

  - P(2 rights, 1 wrong) = $3*0.8^2 * (1-0.8)$ = 0.384.

- Therefore, ensemble is right with probability 0.896 (which is 0.512+0.384).

Notes on why averaging works

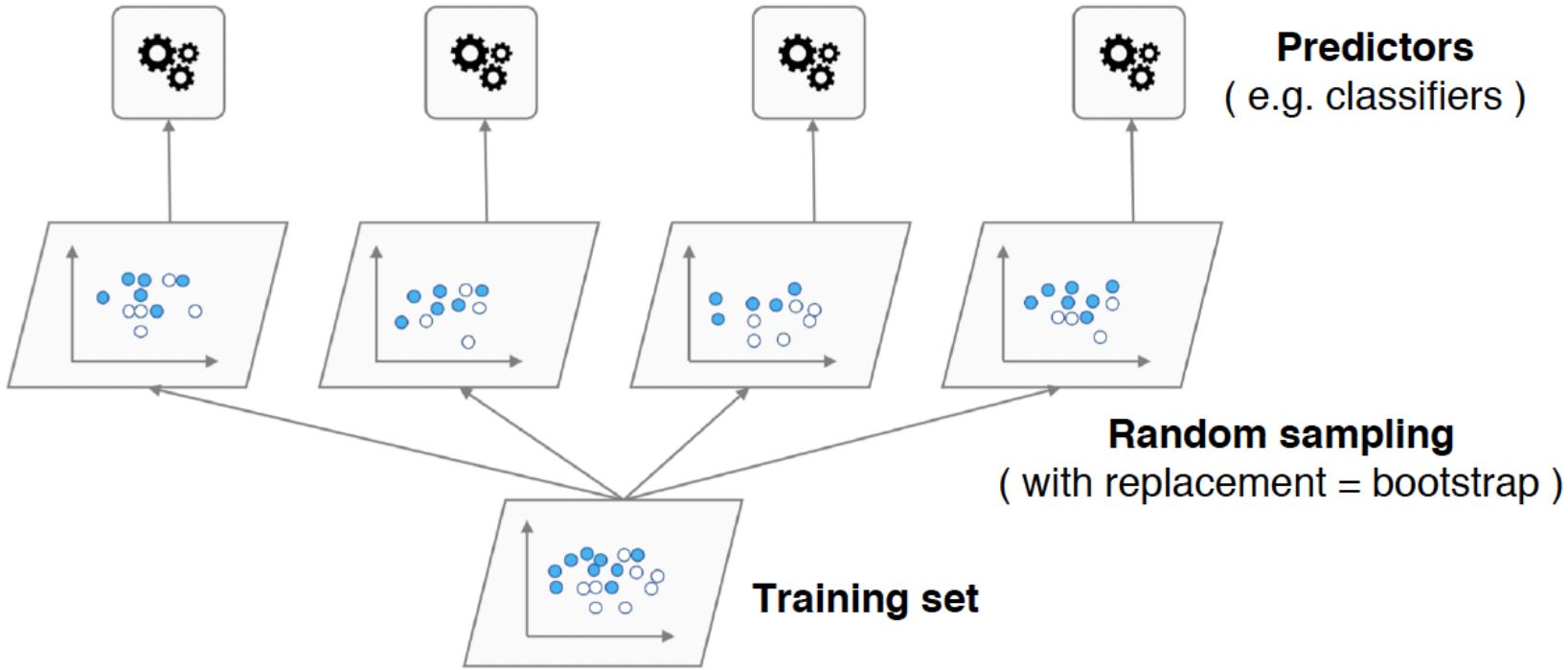
- For averaging to work:
  - Classifiers need to be at least somewhat independent.
  - Their probability of being right to be > 0.5, otherwise it will do much worse.
  - Probabilities also shouldn't be too different (otherwise, it might be better to take the most accurate).

- Classifiers that overfit (like deep decision trees):
  - If they all overfit in exactly the same way, averaging brings no benefit.
- But if they make independent errors:
  - Less attention to specific overfitting of each classifier.
  - Probability that "average" is wrong can be lower than for each classifier.

Bagging

Bagging (short for *bootstrap aggregating*): using bootstrap samples for ensemble learning.

- Generate several bootstrap samples of the examples.
- Train a classifier on each bootstrap sample.
- Average the predictions.



Bagging:

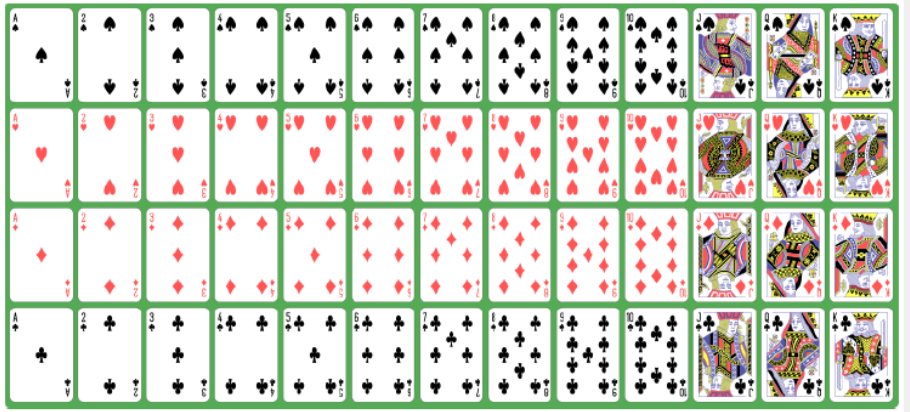
- has little effect on bias
- reduces variance

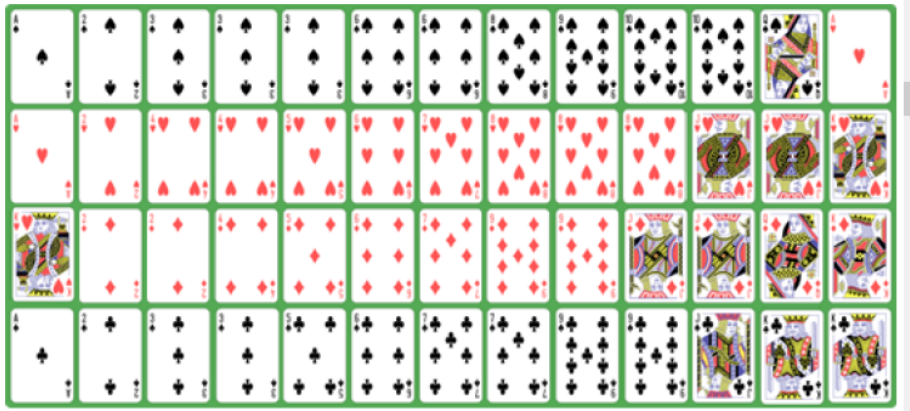
Bootstrap Sampling: example

- Start with a standard deck of 52 cards:
  1. Sample a random card: (put it back and re-shuffle) 
  2. Sample a random card: (put it back and re-shuffle) 
  3. Sample a random card: (put it back and re-shuffle) 

  – …

  52. Sample a random card: (which may be a repeat) 
- Make a new deck of the 52 samples:





https://commons.wikimedia.org/wiki/File:English_pattern_playing_cards_deck.svg

Slide credit: M. Schmidt

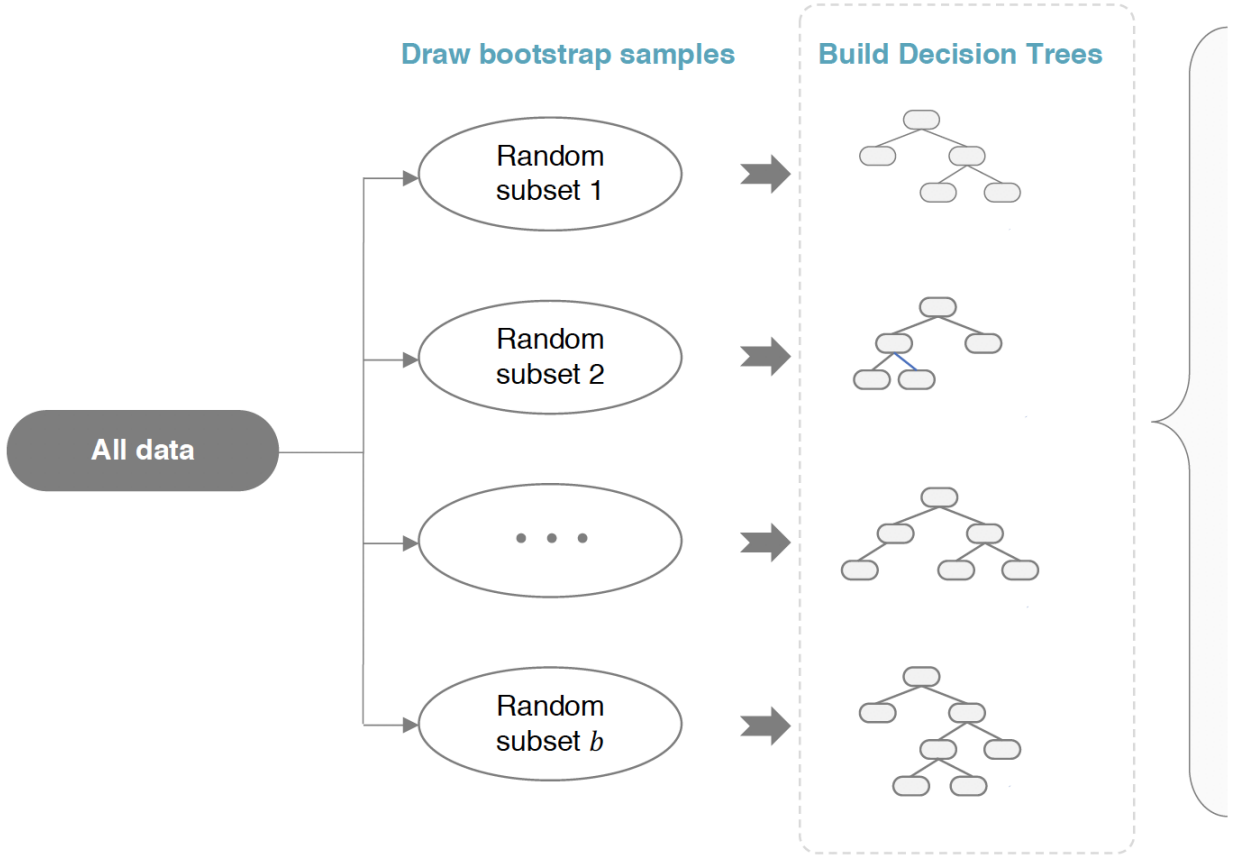
Bootstrap Sampling properties

- Same number of cards 52, but some cards will be missing, and some cards will be duplicated.
- Calculations on the bootstrap sample will give different results than on the original data.
- However, the bootstrap sample roughly maintains trends:
  - Roughly 25% of the cards will be diamonds.
  - There will be roughly four “10” cards.
- In general, bootstrapping gives a new dataset of *n* examples, with some instances duplicated and some missing.
  - For large *n*, approximately 63% of original examples are included.

Random Forest = Bagging + Decision trees

- Random forests averages a set of deep decision trees.
  - Tend to be one of the best “out of the box” classifiers.
  - Often close to the best performance of any method on the first run.
  - And predictions are very fast.
- Do deep decision trees make independent errors?
  - No: with the same training data you’ll get the same decision tree.
- Two key ingredients in random forests:
  - **Bootstrapping**.
  - Different trees.

Random Forest: Generate different trees



Fit each random-forest tree to different bootstrap samples

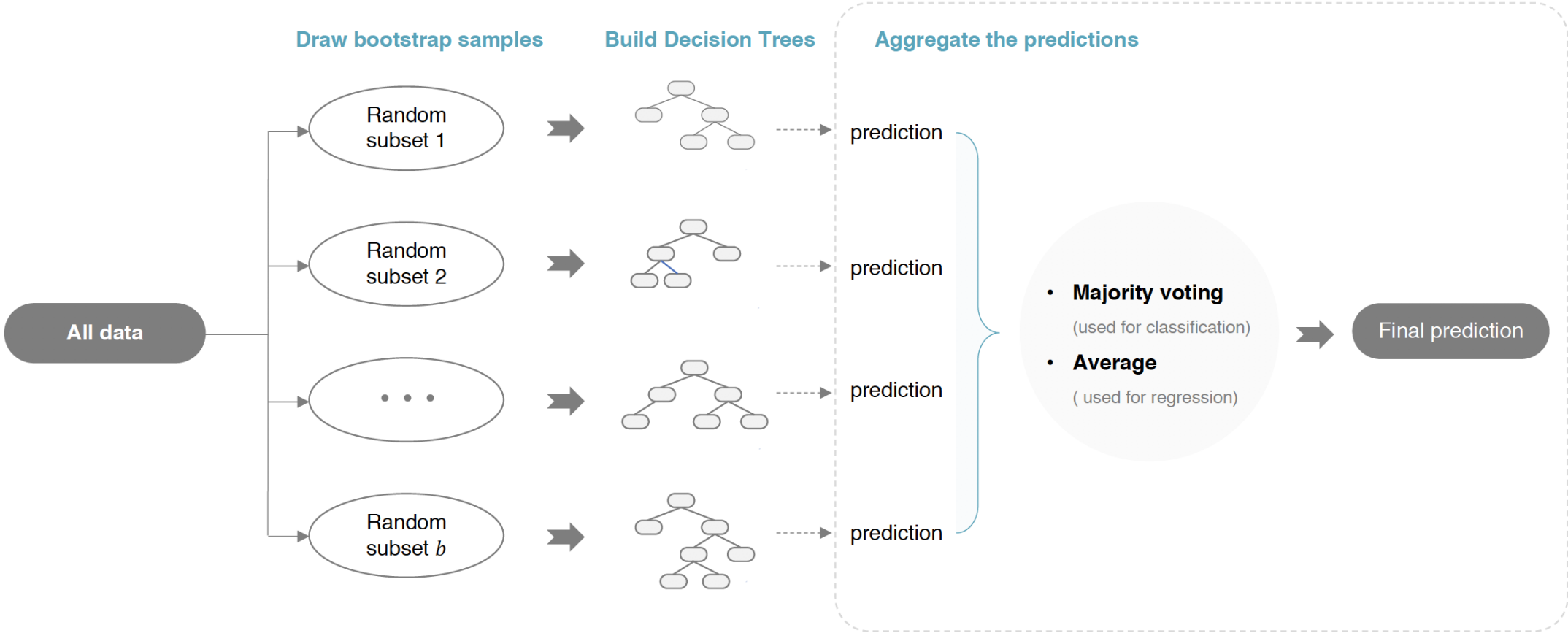
Grow a decision tree, by recursively repeating:

- **Select m variables at random from the p variables** (m<p)
- Calculate the best split based on these m variables in the training set.

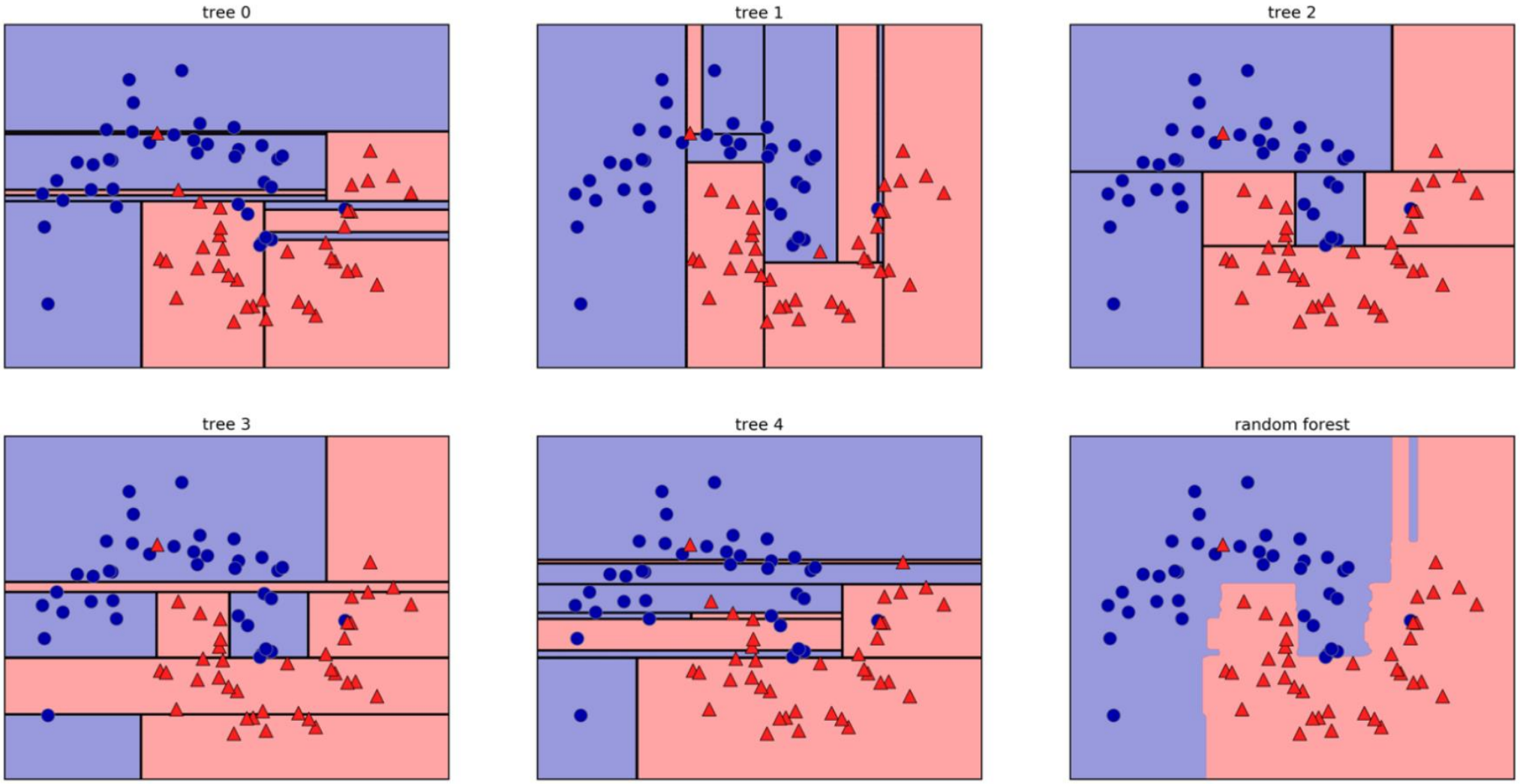
Each tree is grown to the largest extent possible

Random Forest: Aggregate predictions

Use “majority votes” for classification, and “average” for regression problem.



Random forest example: Decision boundaries



Random Forest error rate

Random forest error rate depends on two things:

- The **correlation** between any two trees in the forest
  Increasing the correlation: higher forest error rate.
- The **strength** of each individual tree in the forest
  Higher strength: lower forest error rate.

Number of variables used to build each tree (m):

- lower number: lower correlation and the strength.
- higher number: higher correlation and the strength.

Somewhere in between is an "optimal" range of m:

- m = p/3 for regression problems (p total number of features)
- m = sqrt(p) for classification problems.
- hyperparameter to be tuned

Out-of-Bag (OOB) error

- Each tree is constructed using a different bootstrap sample from the original data.
- About one-third of the instances are left out of the bootstrap sample and are not used in the construction of the tree.
- We can use each instances left out in the construction of a particular tree evaluate that tree.
- Each instance ends up being a test set in about one-third of the trees.
- We can predict the target of each instance using all the trees where this data instances was not used to build the tree, and this gives us the **oob error estimate.**

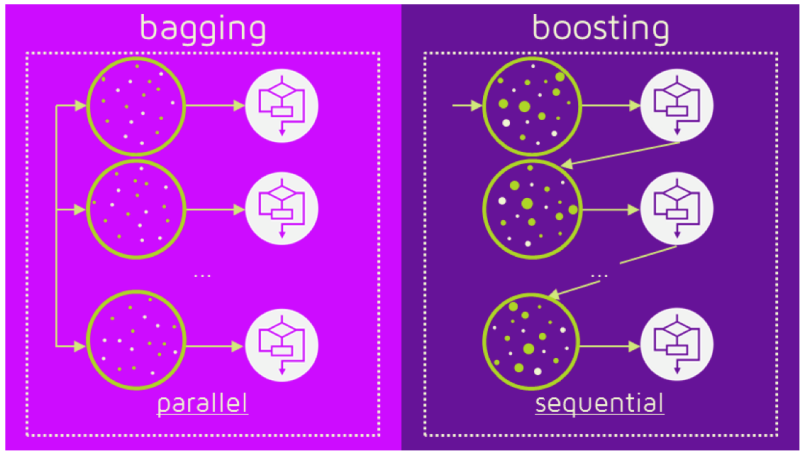
Feature importance with out of bag samples

- The prediction accuracy on the out-of-bag sample is measured.
- The values of one feature in the out-of-bag-sample are randomly shuffled, keeping all other features the same.
- The prediction accuracy is obtained using this changed dataset.
- Intuitively, if the accuracy does not decrease with the random shuffling, it means that the feature has no predictive power.
- If a feature has very little predictive power, shuffling may even lead to a slight increase in accuracy due to random noise.

Boosting

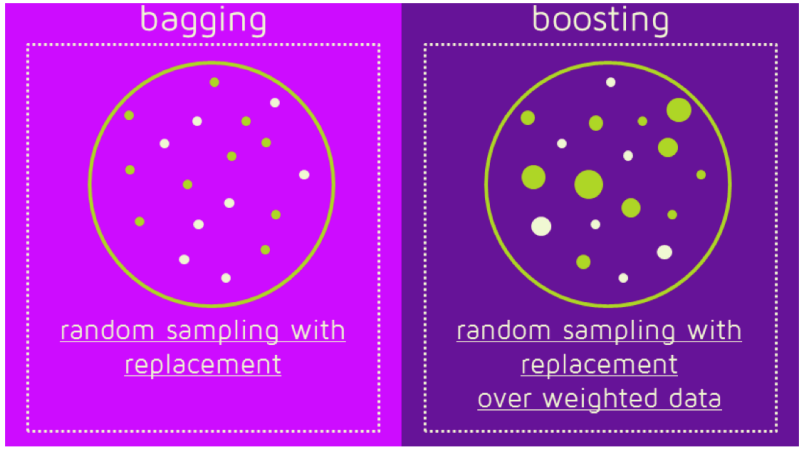
- Widely used in practice
- Used by most winners of machine learning competitions (Kaggle, KDD cup,…)

Bagging reduces variance, boosting reduces bias.



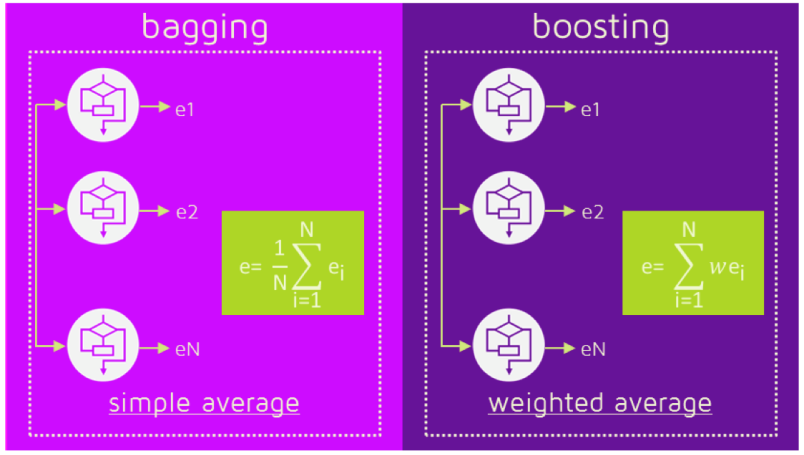
Bagging
Base models run in parallel.

Boosting
Boosting trains models sequentially, where each new model focuses on correcting the errors made by the previous ones



Bagging
N new training data sets are produced by random sampling with replacement from the original set.

Boosting
Incorrectly predicted observations are weighted more heavily and appear in the new datasets more often.



Bagging
The result is obtained by averaging the responses of the N learners (or majority vote).

Boosting
The final boosting ensemble uses a weighted average, more weight to those with better performance on training data.

Gradient Boosting

Regression

- Start with a simple model (**weak learner:** high bias and low complexity)
- Calculate the **residuals** (difference between actual values and predicted values)
- Train a new model to correct the errors
  - train a new decision tree **to predict the residuals**
- Update the predictions
  - add the new model's predictions to the original model's predictions
  - use a **learning rate** to control how much of the correction is applied
- Repeat the process
  - keep adding more small trees, each trained on the remaining errors
  - get a strong model that makes very accurate predictions after many iterations

Prediction: **sum the initial prediction and all the tree corrections**, scaled by the **learning rate**

Classification

- uses regression trees, but to the predict **class probabilities**

XGBoost: Extreme Gradient Boosting

- Optimized and advanced version of Gradient Boosting.
  - Adds regularization (L1 & L2), which helps in preventing overfitting.
  - Implements **Stochastic GBM** with column (features) and row (instances) sampling for better generalization.
  - Parallelizes finding the best split for each tree to speed up training.
  - Implements **early stopping**: automatically stopping training when performance on a validation set stops improving, helps XGBoost automatically select the optimal number of trees

LightGBM

- Faster and more memory-efficient, optimized for massive datasets
- Implements Leaf-Wise Growth: expands the leaf node that reduces loss the most instead of evenly expanding all nodes
- Handles categorical features natively, eliminating the need for encoding.

Catboost

- Best support for categorical features
- Symmetric Tree Growth: both left and right child nodes are split simultaneously at each depth: Faster inference speed, as all trees have the same structure.
- Performs well even on small datasets
- Less need for extensive hyperparameter tuning

Hyperparameters of gradient boosting algorithms

- Tree Structure and Model Complexity
  - maximum tree depth
  - number of leaves per tree
- Learning Process
  - number of boosting rounds
  - percentage of rows sampled per tree
  - fraction of features used per tree
  - learning rate
- Regularization
  - L1 regularization
  - L2 regularization