#4 Decision Trees

AI Training

Decision Forest

Decision forests are an alternative to neural networks.

Introduction

A family of **supervised learning models** built from many **decision trees** working together (e.g., random forests, gradient boosted trees).

Why Use Decision Forests?

- **Easy to Configure**
  - Fewer hyperparameters than neural networks
  - Comes with **good defaults** — less trial and error
- **Minimal Preprocessing**
  - Natively supports **numeric**, **categorical**, and **missing** values
  - Reduces preprocessing code and risk of errors
- **Strong Performance Out of the Box**
  - Robust to **noisy data**
  - Offers **interpretable models** (like tree structures)
- **Efficient on Smaller Datasets**
  - Trains **much faster** than neural networks on datasets **< 1 million examples**

Introduction

Proven in the Real World

- Widely used in industry and **machine learning competitions**
- Known for being:
  - **Practical**
  - **Efficient**
  - **Interpretable**

What Can Decision Forests Do?

Decision forests are versatile — they support a wide range of **supervised learning tasks**:

- **Classification** – Predict discrete class labels
- **Regression** – Predict continuous values
- **Ranking** – Order items based on relevance or score
- **Uplift Modeling** – Estimate the impact of treatment vs. control (e.g., in marketing or medicine)

Appropriate data

Decision forests are most effective when you have a tabular dataset (data you might represent in a spreadsheet, csv file, or database table). Tabular data is one of the most common data formats, and decision forests should be your "go-to" solution for modeling it.

Table 1. An example of a tabular dataset.

Number of legs	Number of eyes	Weight (lbs)	Species (label)
2	2	12	Penguin
8	6	0.1	Spider
4	2	44	Dog
…	…	…	…

Appropriate data

Ideal for Tabular Data

- Decision forests are designed to work **natively with structured, tabular data**
- No need for:
    - Feature normalization
    - One-hot encoding
    - Manual imputation of missing values

This simplifies development and reduces preprocessing errors.

Not Ideal for Unstructured Data

- **Unstructured data** (e.g., images, text) is **not well suited** for decision forests
- While workarounds exist, they are often **inefficient or suboptimal**
- Neural networks are typically better for tasks involving:
    - Images
    - Natural language
    - Audio

Performance

Fast Inference

- Decision forests typically infer faster than comparable neural networks
- Example:
    - A medium-sized forest can make predictions in just a few microseconds on a modern CPU

Decision Trees

Decision trees

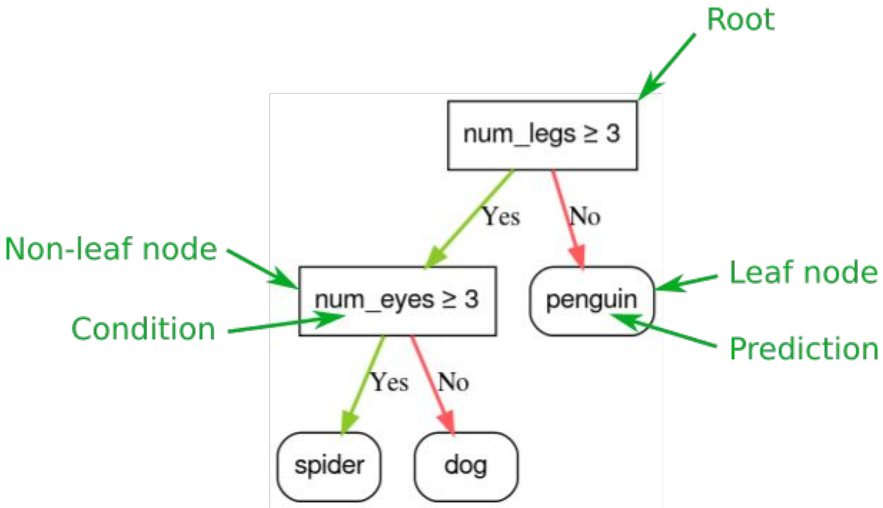
Decision forest models are composed of decision trees. Decision forest learning algorithms (like random forests) rely, at least in part, on the learning of decision trees.

In this lecture, you will study a small example dataset, and learn how a single decision tree is trained. Later, you will learn how decision trees are combined to train decision forests.

The model

A **decision tree** is a model composed of a collection of "questions" organized hierarchically in the shape of a tree. The questions are usually called a **condition**, a **split**, or a **test**. We will use the term "condition" in this class. Each non-leaf node contains a condition, and each leaf node contains a prediction.

Note: Botanical trees generally grow with the root at the bottom; however, decision trees are usually represented with the **root** (the first node) at the top.



The model

Inference of a decision tree model is computed by routing an example from the root (at the top) to one of the leaf nodes (at the bottom) according to the conditions. The value of the reached leaf is the decision tree's prediction. The set of visited nodes is called the **inference path**.

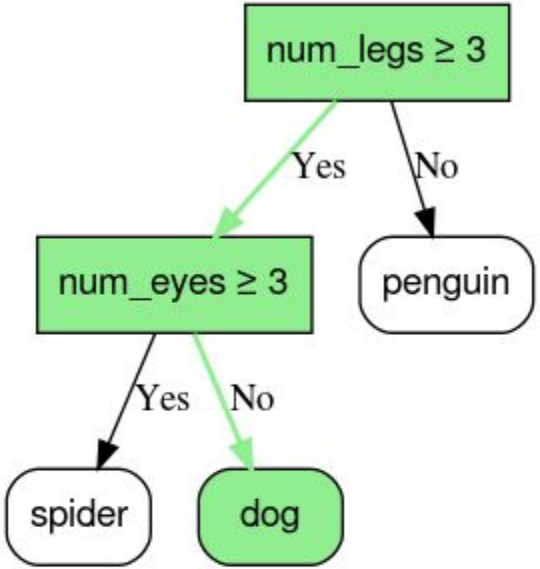
The model

For example, consider the following feature values:

num_legs	num_eyes
4	2

The prediction would be dog. The inference path would be:

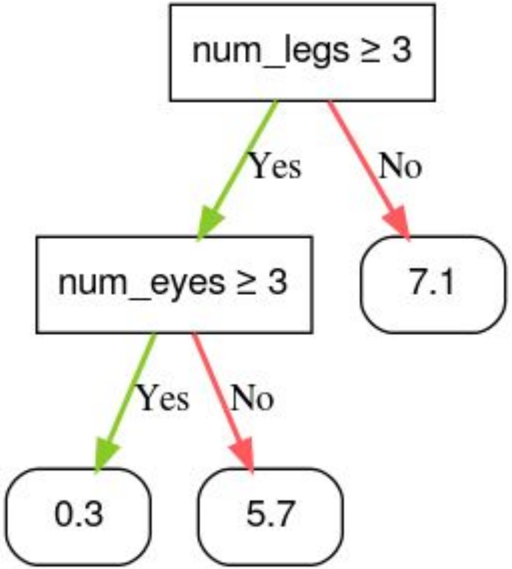
```
1.  num_legs ≥ 3 → Yes
2.  num_eyes ≥ 3 → No
```



The model

In the previous example, the leaves of the decision tree contain classification predictions; that is, each leaf contains an animal species among a set of possible species.

Similarly, decision trees can predict numerical values by labeling leaves with regressive predictions (numerical values). For example, the following decision tree predicts a numerical cuteness score of an animal between 0 and 10.



Conditions

This section focuses on different types of **conditions** used to build decision trees.

Axis-aligned vs. oblique conditions

An **axis-aligned condition** involves only a single feature. An **oblique condition** involves multiple features. For example, the following is an axis-aligned condition:

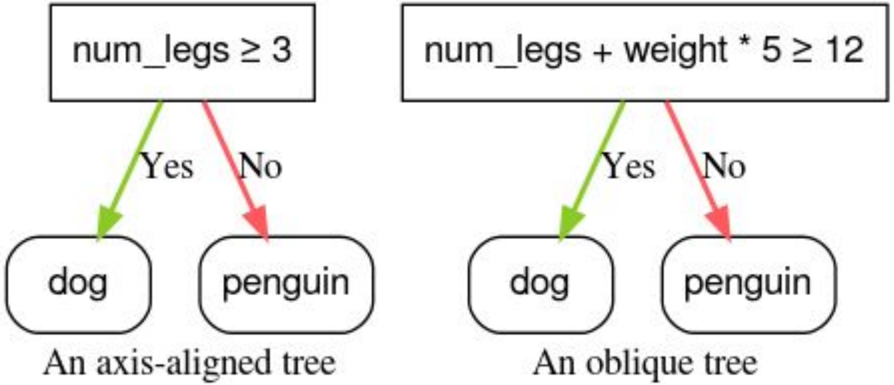
```
num_legs ≥ 2
```

While the following is an oblique condition:

```
num_legs ≥ num_fingers
```

Axis-aligned vs. oblique conditions

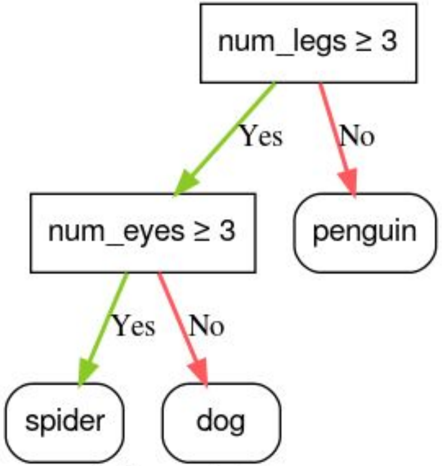
Often, decision trees are trained with only axis-aligned conditions. However, oblique splits are more powerful because they can express more complex patterns. Oblique splits sometime produce better results at the expense of higher training and inference costs.



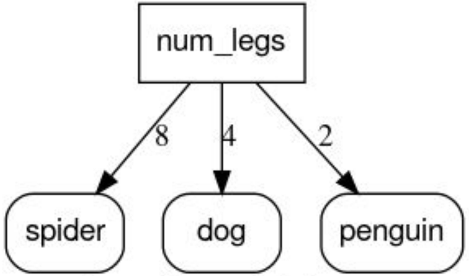
Binary vs. non-binary conditions

Conditions with two possible outcomes (for example, true or false) are called **binary conditions**. Decision trees containing only binary conditions are called **binary decision trees**.

Non-binary conditions have more than two possible outcomes. Therefore, non-binary conditions have more discriminative power than binary conditions. Decisions containing one or more non-binary conditions are called **non-binary decision trees**.



A binary decision tree



A non-binary decision tree

Binary vs. non-binary conditions

The most common type of condition is the **threshold condition** expressed as:

```
feature ≥ threshold
```

For example:

```
num_legs ≥ 2
```

Common types of binary conditions.

Name	Condition	Example
threshold condition	$\mathrm{feature}_i \geq \mathrm{threshold}$	num_legs ≥ 2
equality condition	$\mathrm{feature}_i = \mathrm{value}$	species = "cat"
in-set condition	$\mathrm{feature}_i \in \mathrm{collection}$	species ∈ {"cat", "dog", "bird"}
oblique condition	$\sum_i \mathrm{weight}_i \mathrm{feature}_i \geq \mathrm{threshold}$	5*num_legs + 2*num_eyes ≥ 10
feature is missing	$\mathrm{feature}_i \mathrm{isMissing}$	num_legsisMissing

Check Your Understanding

CYU: Decision trees

The inference of a decision tree runs by routing an example…

A. from the leaf to the root.
B. from the root to the leaf.
C. from one leaf to another.

CYU: Decision trees

The inference of a decision tree runs by routing an example…

A. from the leaf to the root.
B. from the root to the leaf.
C. from one leaf to another.

CYU: Decision trees

Do all types of conditionals involve only a single feature?

A. Yes.
B. No.

CYU: Decision trees

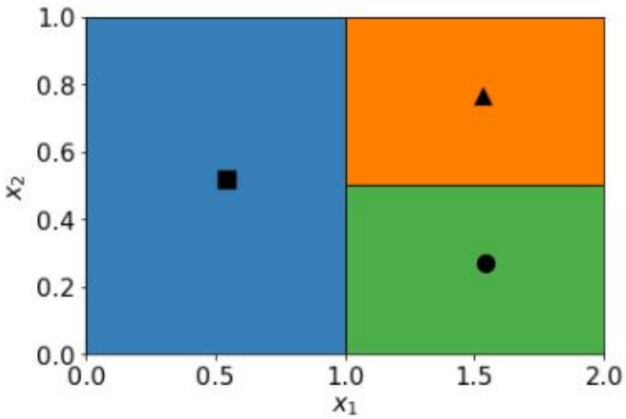
Do all types of conditionals involve only a single feature?

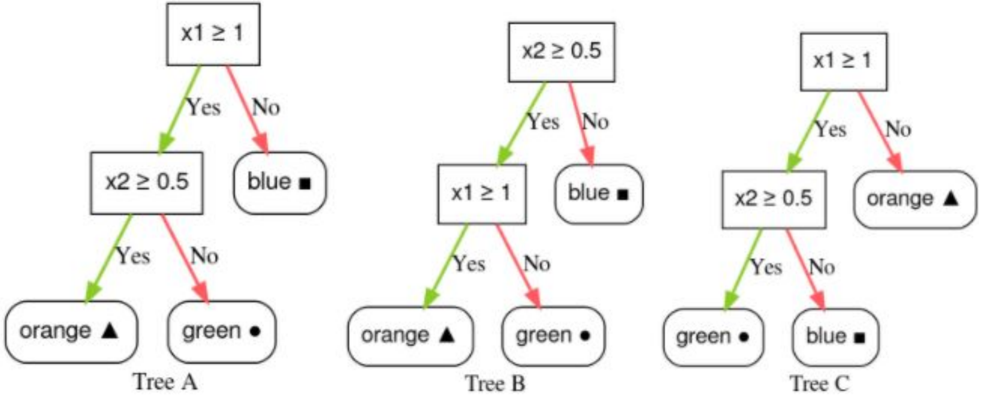
A. Yes.
B. No.

CYU: Decision trees

Consider the following prediction map on two features x1 and x2:

Which of the following decision trees match the prediction map?



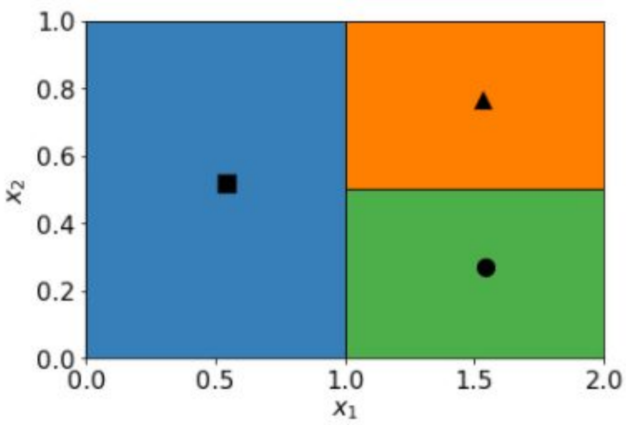


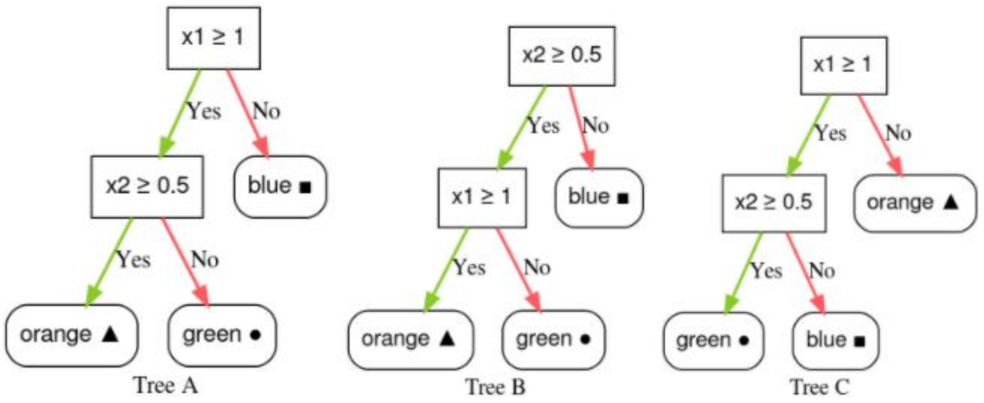
- Decision Tree C.
- Decision Tree A.
- Decision Tree B.

CYU: Decision trees

Consider the following prediction map on two features x1 and x2:

Which of the following decision trees match the prediction map?





- Decision Tree C.
- Decision Tree A.
- Decision Tree B.

Train Decision Trees

Growing decision trees

Growing Decision trees

Most algorithms used to train decision trees work with a greedy **divide and conquer** strategy. The algorithm starts by

- creating a single node (the root), and
- recursively and greedily grows the decision tree.

At each node, **all the possible conditions are evaluated and scored**.

The algorithm selects the "best" condition, that is, the condition with the highest score. For now, just know that the score is a metric that correlates with the task, and conditions are selected to maximize that metric.

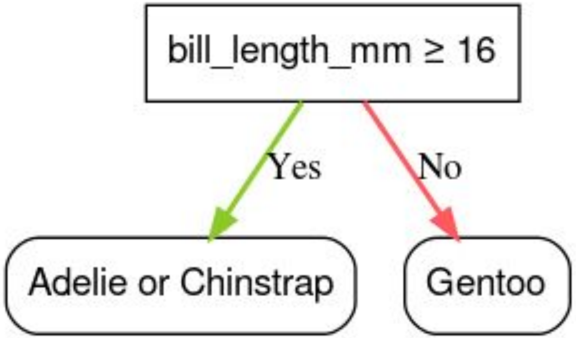
Growing Decision trees

For example, in the **Palmer Penguins** dataset (used for code examples later in this course),

- most Adelie and Chinstrap penguins have a bill's length greater than 16mm,
- while most of the Gentoo penguins have smaller bills.

`bill_length_mm ≥ 16`

✅ Perfect for Gentoo



What about Adelies vs. Chinstraps?

The algorithm then repeats recursively and independently on both children nodes. When no satisfying conditions are found, the node becomes a leaf. The leaf prediction is determined as the most representative label value in the examples.

Growing Decision trees: algorithm

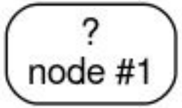
The algorithm is as follows:

```
def train_decision_tree(training_examples):
  root = create_root() # Create a decision tree with a single empty root.
  grow_tree(root, training_examples) # Grow the root node.
  return root

def grow_tree(node, examples):
  condition = find_best_condition(examples) # Find the best condition.

  if condition is None:
      # No satisfying conditions were found, therefore the grow of the branch stops.
      set_leaf_prediction(node, examples)
      return

  # Create two childrens for the node.
  positive_child, negative_child = split_node(node, condition)

  # List the training examples used by each children.
  negative_examples = [example for example in examples if not condition(example)]
  positive_examples = [example for example in examples if condition(example)]

  # Continue the growth of the children.
  grow_tree(negative_child, negative_examples)
  grow_tree(positive_child, positive_examples)
```

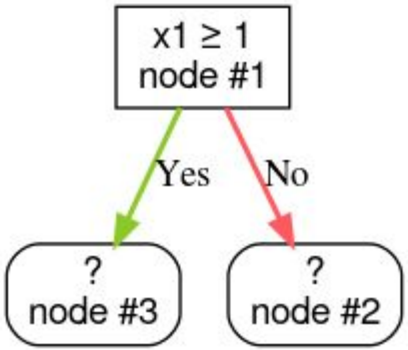
Growing Decision trees: algorithm

Let's go through the steps of training a particular decision tree in more detail.

Step 1: Create a root:

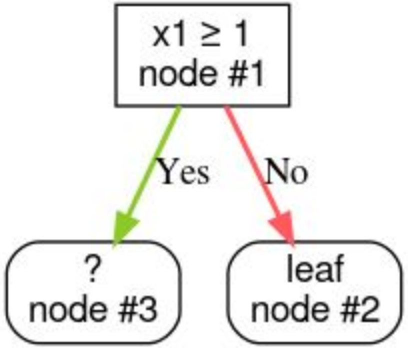


Step 2: Grow node #1. The condition "x1 ≥ 1" was found. Two child nodes are created:



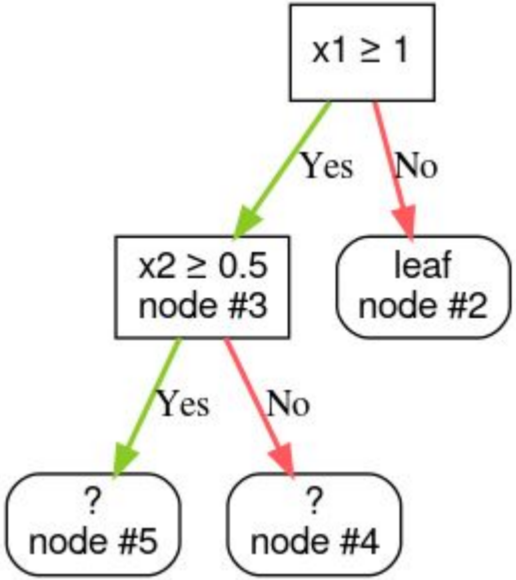
Growing Decision trees: algorithm

Step 3: Grow node #2. No satisfying conditions were found. So, make the node into a leaf:



Growing Decision trees: algorithm

Step 4: Grow node #3. The condition "x2 ≥ 0.5" was found. Two child nodes are created.



Growing Decision trees: Splitter

Depending on the number and type of input features, the number of possible conditions for a given node can be huge, generally infinite. For example, given a threshold condition $\mathtt{feature}_{\mathtt{i}} \geq t$, the combination of all the possible threshold values for $t \in \mathbb{R}$ is infinite.

The routine responsible for finding the best condition is called the **splitter**. Because it needs to test a lot of possible conditions, splitters are the bottleneck when training a decision tree.

The score maximized by the splitter depends on the task. For example:

- **Information gain** and **Gini** (both covered later) are commonly used for classification.
- Mean squared error is commonly used for regression.

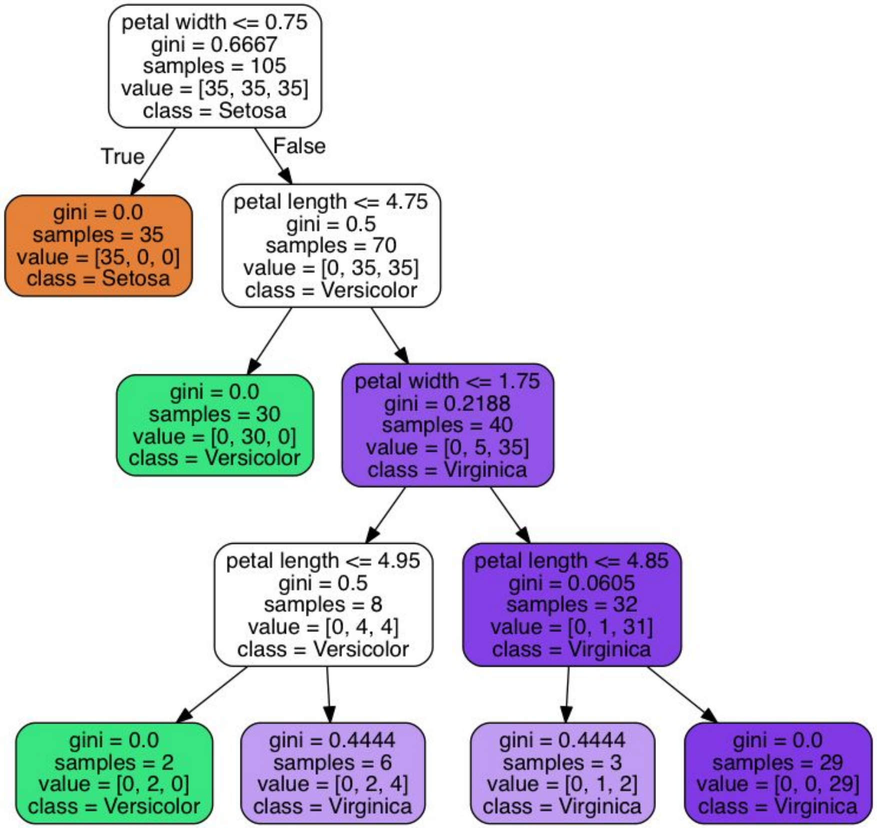
Growing Decision trees: Splitter

There are many splitter algorithms, each with varying support for:

- The type of **features**; for example, numerical, categorical, text
- The **task**; for example, binary classification, multi-class classification, regression
- The type of **condition**; for example, threshold condition, in-set condition, oblique condition
- The **regularization** criteria; for example, exact or approximated splitters for threshold conditions

In addition, there are equivalent splitter variants with different trade-offs regarding memory usage, CPU usage, computation distribution, and so on.

Growing Decision trees: Splitter

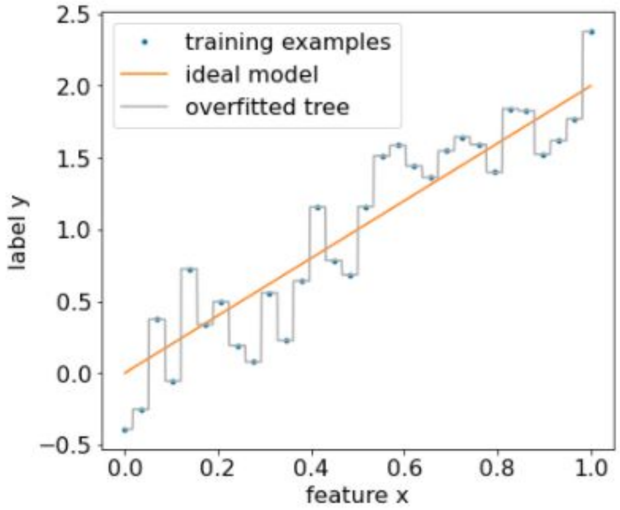


Overfitting and pruning

Overfitting and pruning

We can train a decision tree that will perfectly classify training examples, assuming the examples are separable. However, if the dataset contains noise, this tree will overfit to the data and show poor test accuracy.

This model (below) correctly predicts all the training examples. However, on a new dataset containing the same linear pattern and a different noise instance, the model would perform poorly.

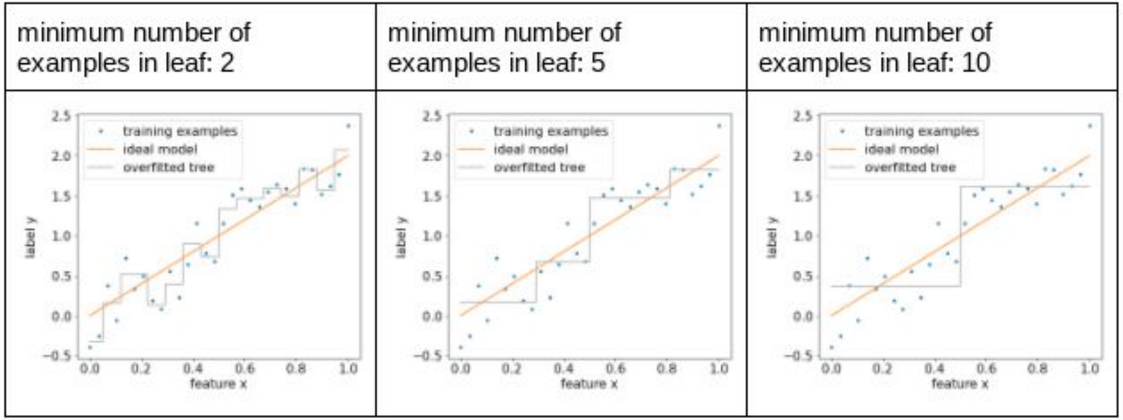


Overfitting and pruning

To limit overfitting a decision tree, apply one or both of the following regularization criteria while training the decision tree:

- **Set a maximum depth**: Prevent decision trees from growing past a maximum depth, such as 10.
- **Set a minimum number of examples in leaf**: A leaf with less than a certain number of examples will not be considered for splitting.

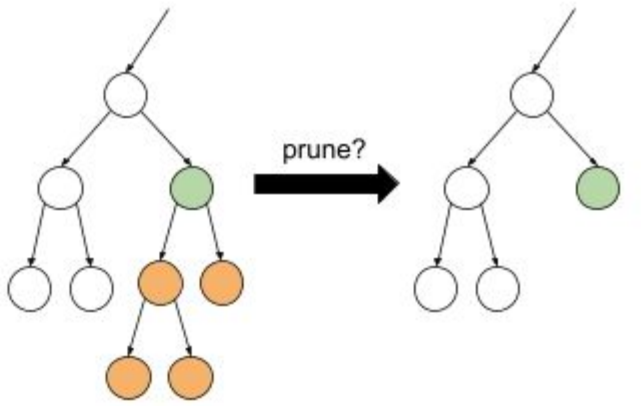
The effect of differing minimum number of examples per leaf. The model captures less of the noise.

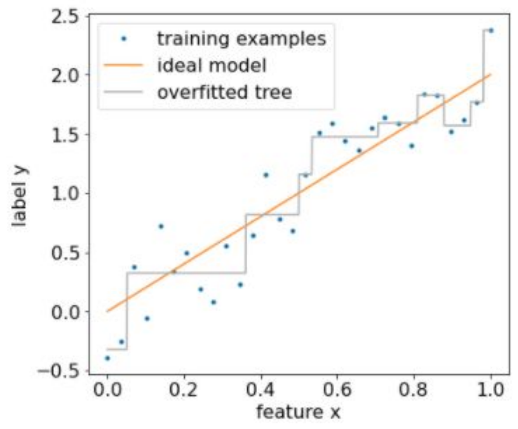


Pruning

You can also regularize after training by selectively removing (**pruning**) certain branches, that is, by converting certain non-leaf nodes to leaves.

A common solution to select the branches to remove is to use a validation dataset. That is, if removing a branch improves the quality of the model on the validation dataset, then the branch is removed.





Using 20% of the dataset to prune the decision tree

Pruning

Note that using a validation dataset reduces the number of examples available for the initial training of the decision tree.

Many AI engineers apply multiple criteria. For example, you could do all of the following:

- Apply a minimum number of examples per leaf.
- Apply a maximum depth to limit the growth of the decision tree.
- Prune the decision tree.

Decision Forests

Decision Forests

A **decision forest** is a generic term to describe models made of multiple decision trees.

- The prediction of a decision forest is the aggregation of the predictions of its decision trees.
- The implementation of this aggregation depends on the algorithm used to train the decision forest.

For example,

- In a multi-class classification random forest (a type of decision forest), each tree votes for a single class, and the random forest prediction is the most represented class.

- In a binary classification gradient boosted Tree (GBT) (another type of decision forest), each tree outputs a logit (a floating point value), and the gradient boosted tree prediction is the sum of those values followed by an activation function (e.g. sigmoid).

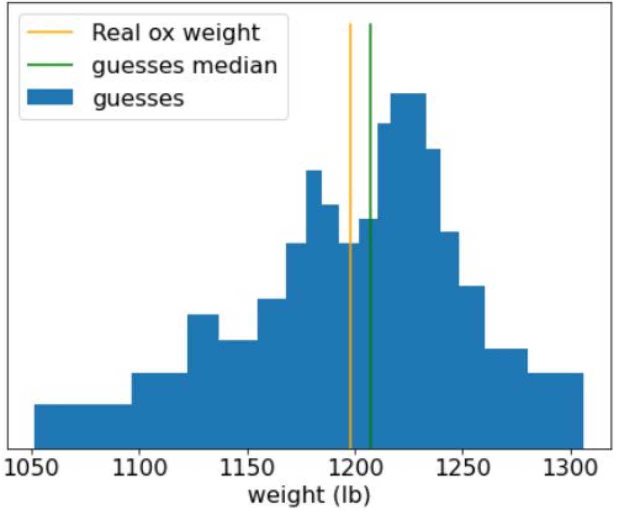
Random Forests

Introduction

This is an Ox.

Introduction

In 1906, a weight judging competition was held in England. 787 participants guessed the weight of an ox. The median error of individual guesses was 37 lb (an error of 3.1%). However, the overall median of the guesses was only 9 lb away from the real weight of the ox (1198 lb), which was an error of only 0.7%.



This story illustrates the **Wisdom of the crowd**: *In certain situations, collective opinion provides very good judgment.*

Ensemble

Definition

- An ensemble is a **collection of models**
- Their predictions are **combined** (e.g., by averaging or voting) to produce a final prediction

Why Use Ensembles?

- Well-constructed ensembles often perform **better than any individual model**
- This is because they can:
  - Reduce **variance**
  - Correct for **individual model errors**
  - Improve **generalization**

Ensemble: Voting Classifiers

Suppose You Have a Few Classifiers...Each achieves around **80% accuracy** individually:

- Logistic Regression
- Support Vector Machine (SVM)
- Random Forest
- K-Nearest Neighbors (K-NN)
- And maybe a few more

Combine Them Using a **Voting Classifier**

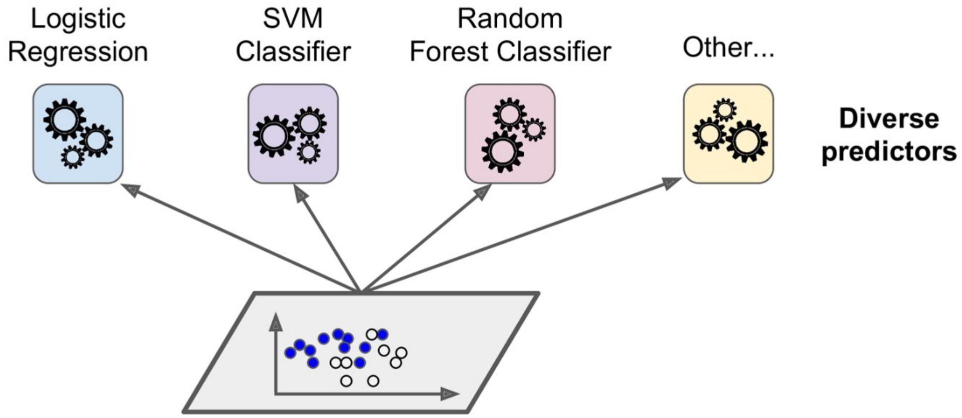


Figure 3.4: Training diverse classifiers

Ensemble: Hard Voting Classifier

How It Works

- Combine multiple classifiers
- Each one **votes** for a class label
- The final prediction is the class with the **majority vote**
- Often outperforms the best individual model in the group
- Even if individual classifiers are only slightly better than guessing (weak learners)...

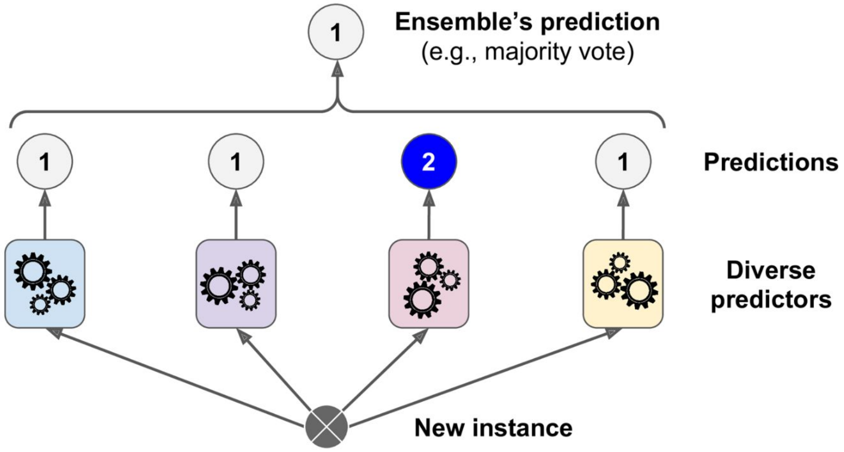


Figure 3.5: Hard voting classifier predictions

Ensemble: Hard Voting Classifier

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

log_clf = LogisticRegression()
rnd_clf = RandomForestClassifier()
svm_clf = SVC()

voting_clf = VotingClassifier(
    estimators=[('lr', log_clf), ('rf', rnd_clf), ('svc', svm_clf)],
    voting='hard')
voting_clf.fit(X_train, y_train)
```

Let's look at each classifier's accuracy on the test set:

```python
>>> from sklearn.metrics import accuracy_score
>>> for clf in (log_clf, rnd_clf, svm_clf, voting_clf):
...     clf.fit(X_train, y_train)
...     y_pred = clf.predict(X_test)
...     print(clf.__class__.__name__, accuracy_score(y_test, y_pred))
...
LogisticRegression 0.864
RandomForestClassifier 0.896
SVC 0.888
VotingClassifier 0.904
```

Hard voting classifier predictions

Random forests

What it is

- A **random forest (RF)** is an ensemble of **decision trees**.
- Each decision tree is trained with a specific random noise.

Random forests are the most popular form of decision tree ensemble.

Random forests: Bagging

What Is Bagging?

- Stands for **B**ootstrap **Agg**regat**ing**
- Each model (e.g., a decision tree) is trained on a **random subset** of the training data
- The subset is the **same size** as the original dataset, but selected **with replacement**

What "With Replacement" Means

- Some examples are used more than once
- Some examples are left out entirely

On average, each bootstrap sample contains about **63% unique examples**

The rest (~37%) are not sampled - **out-of-bag (oob) instances**

Why It Matters

- Leads to **diverse models** that make **different errors**
- Reduces **overfitting**

Random forests: Bagging

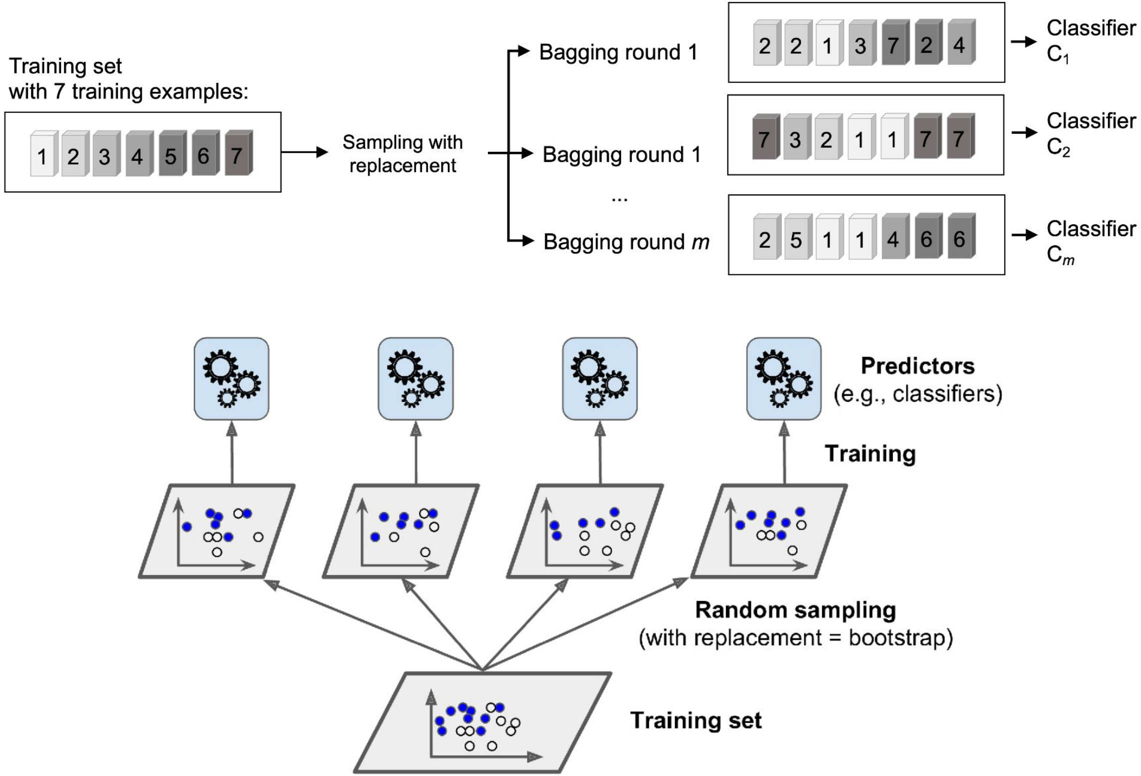


Figure 3.6: An example of bagging

Random forests: Bagging

For example, the table below shows how bagging could distribute six examples across three decision trees. Notice the following:

- Each decision tree trains on a total of six examples.
- Each decision tree trains on a different set of examples.
- Each decision tree reuses certain examples. For example, example #4 is used twice in training decision tree 1;

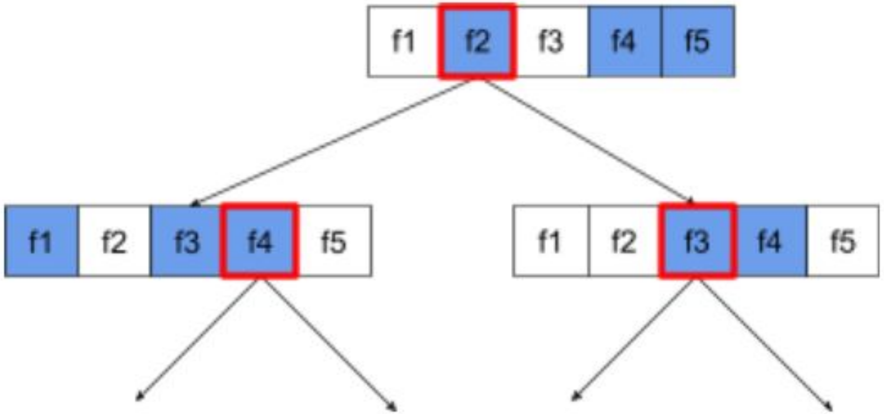
training examples						
	#1	**#2**	**#3**	**#4**	**#5**	**#6**
original dataset	1	1	1	1	1	1
decision tree 1	1	1	0	2	1	1
decision tree 2	3	0	1	0	2	0
decision tree 3	0	1	3	1	0	1

RF: Attribute sampling

Attribute sampling = Randomly selecting a subset of features to consider when training decision trees.

There are two main types:

1. Per-Split Attribute Sampling (most common):
   - At **each node** of each tree, a random subset of features is chosen to evaluate for the split
2. Per-Tree Attribute Sampling (less common):
   - **Each tree** is trained using only a random subset of features



- The blue nodes represent the tested features while the white ones are not tested.
- The condition is built from the best tested features (represented with a red outline).

RF: Disabling DT regularization

The decision trees in a random forest are trained without pruning. The lack of pruning significantly:

- **increases the variance** and significantly
- **reduces the bias** of the individual decision tree learning.

In other words, the *individual decision trees overfit*, **but the random forest is not**. 🤔

The two sources of randomness (bagging and attribute sampling) ensure the relative independence between the decision trees. This independence corrects the overfitting of the individual decision trees. Consequently, the ensemble is not overfitted.

RF: Out-of-bag evaluation

Why OOB?

- No need for a separate validation or test set
- Evaluates model quality using only the training data
- Works like **built-in cross-validation**

How It Works

- Each decision tree is trained on a **bootstrap sample** (~63% of training data)
- So, for every tree, ~37% of examples are **left out** (not seen during training)

OOB Evaluation Strategy

- For each training example:
  - Identify all the trees that **didn't see it**
  - Aggregate their predictions (e.g., majority vote)
  - Compare to the **true label**

This gives a **reliable estimate of generalization performance** — like cross-validation, but **for free**

RF: Out-of-bag evaluation

The following table illustrates OOB evaluation of a random forest with 3 decision trees trained on 6 examples. The table shows which decision tree is used with which example during OOB evaluation.

	Training examples						Examples for OOB Evaluation
	#1	**#2**	**#3**	**#4**	**#5**	**#6**	
original dataset	1	1	1	1	1	1	
decision tree 1	1	1	0	2	1	1	#3
decision tree 2	3	0	1	0	2	0	#2, #4, and #6
decision tree 3	0	1	3	1	0	1	#1 and #5

In the example shown in the table, the OOB predictions for training example #1 will be computed with decision tree 3 (since decision trees 1 and 2 used this example for training).

Gradient Boosted Decision Trees

GB Decision trees

In your next homework/classwork!

Example,

- XG Boost
- CatBoost
- AdaBoost (can even do face classification)
- Light GB
- etc.

Wondering why they work so well 🤔 ?

Sorry, NOT TODAY!

Practice

Classification with Logistic Regression and Decision Trees

Predicting diabetes likelihood in individuals.

Diabetes Classification Challenge

Any questions?

THANKS