THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

Applied Machine Intelligence and Reinforcement Learning

Professor Hamza F. Alsarhan

SEAS 8505 – DA2

Lecture 2

June 22, 2024

Welcome to SEAS Online at George Washington University

Class will begin shortly

Audio: To eliminate background noise, please be sure your audio is muted. To speak, please click the hand icon at the bottom of your screen (Raise Hand). When instructor calls on you, click microphone icon to unmute. When you've finished speaking, *be sure to mute yourself again*.

Chat: Please type your questions in Chat.

Recordings: As part of the educational support for students, we provide downloadable recordings of each class session to be used exclusively by registered students in that particular class for their own private use. Releasing these recordings is strictly prohibited.

Agenda

- Model Training
- ML Engineering – Data Collection and Preparation Setup
- Model Selection
- Setup – Before you Start
- Hyperparameters and Cross-Validation
- Decision Trees
- Homework Overview

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

Model Training

Professor Hamza F. Alsarhan
SEAS 8505

Context and Motivation

- Having a good understanding of how things work can help you quickly hone in on the appropriate model, the right training algorithm to use, and a good set of hyperparameters for your task.

- Understanding what's under the hood will also help you debug issues and perform error analysis more efficiently.

Linear Regression

Equation 4-1. Linear Regression model prediction

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

In this equation:

- $\hat{y}$ is the predicted value.
- n is the number of features.
- x_i is the i^{th} feature value.
- θ_j is the j^{th} model parameter (including the bias term θ_0 and the feature weights $\theta_1, \theta_2, \cdots, \theta_n$).

- A linear model makes a prediction by simply computing a weighted sum of the input features, plus a constant called the bias term (also called the intercept term)

How Do We Train a Linear Regression Model?

- Training a model means setting its parameters so that the model best fits the training set
- Need a measure of how well (or poorly) the model fits the training data (e.g. root mean square error RMSE)
- To train a Linear Regression model, we need to find the value of θ that minimizes the RMSE or MSE

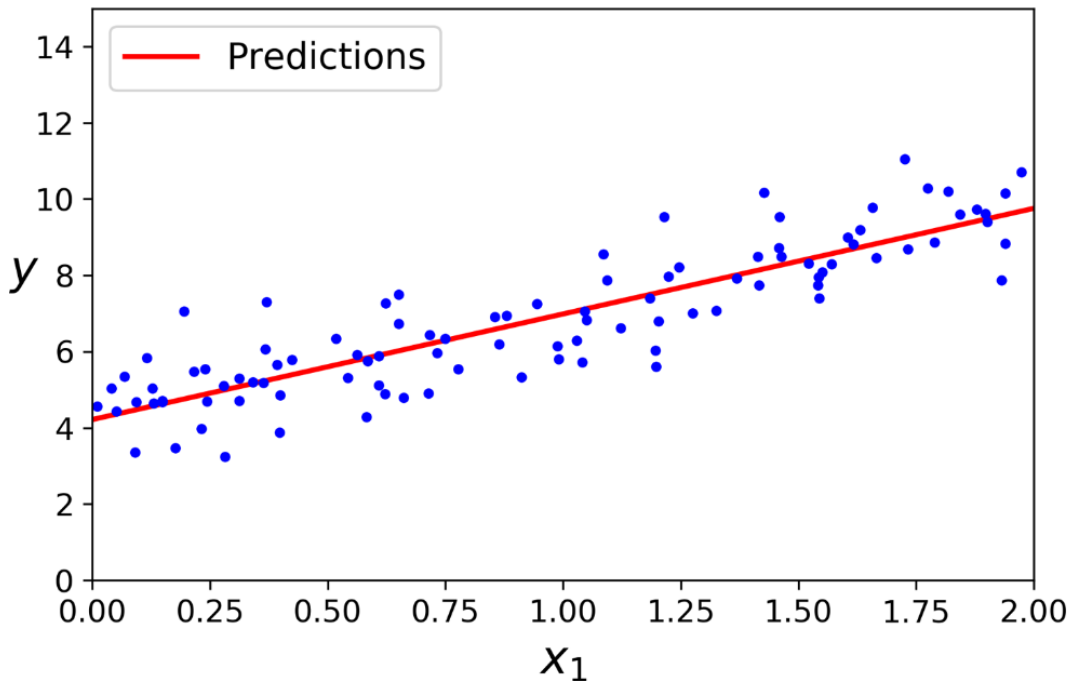


Figure 4-2. Linear Regression model predictions

Searching the Model Parameter Space

- Training a model means searching for a combination of model parameters that minimizes a cost function (over the training set).
- The more parameters a model has, the more dimensions this space has, and the harder the search is.

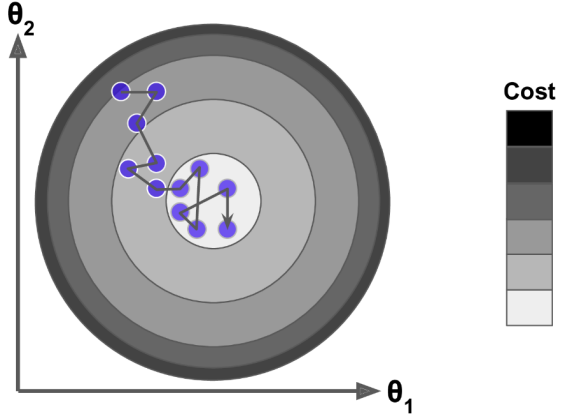


Figure 4-9. With Stochastic Gradient Descent, each training step is much faster but also much more stochastic than when using Batch Gradient Descent

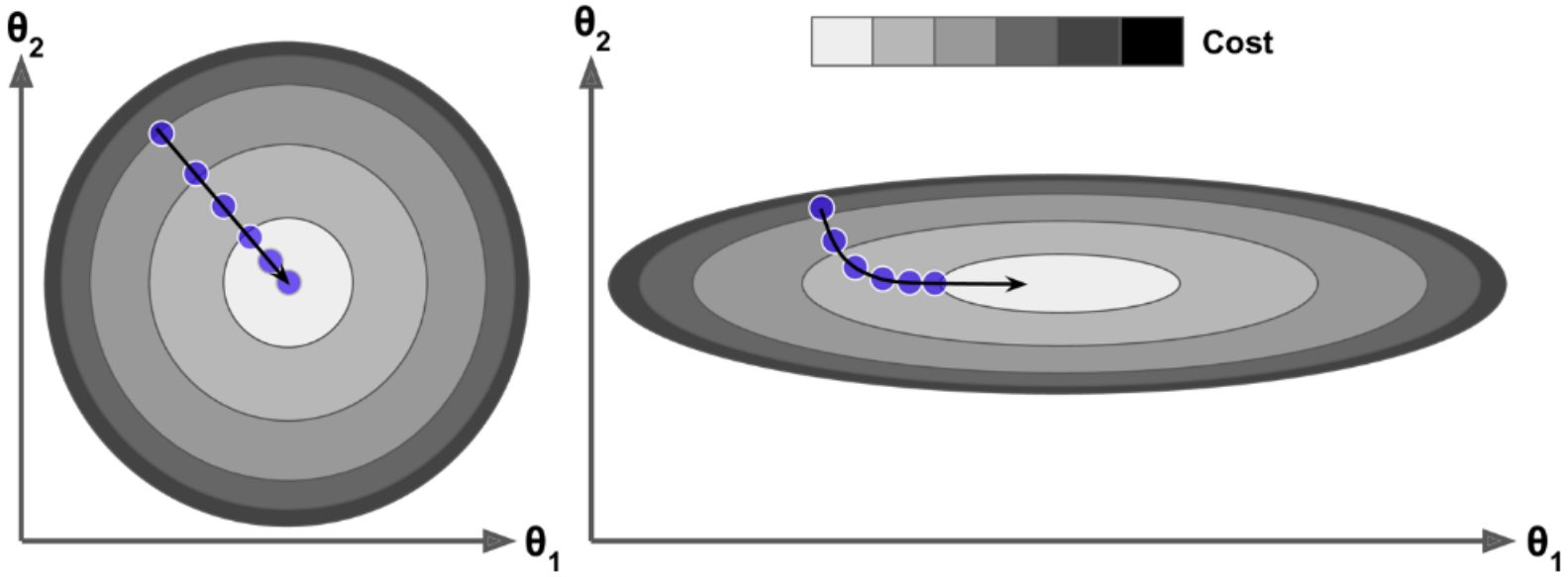


Figure 4-7. Gradient Descent with (left) and without (right) feature scaling

Gradient Descent

- Gradient Descent is a generic optimization algorithm capable of finding optimal solutions to a wide range of problems.
- The general idea of Gradient Descent is to tweak parameters iteratively in order to minimize a cost function.

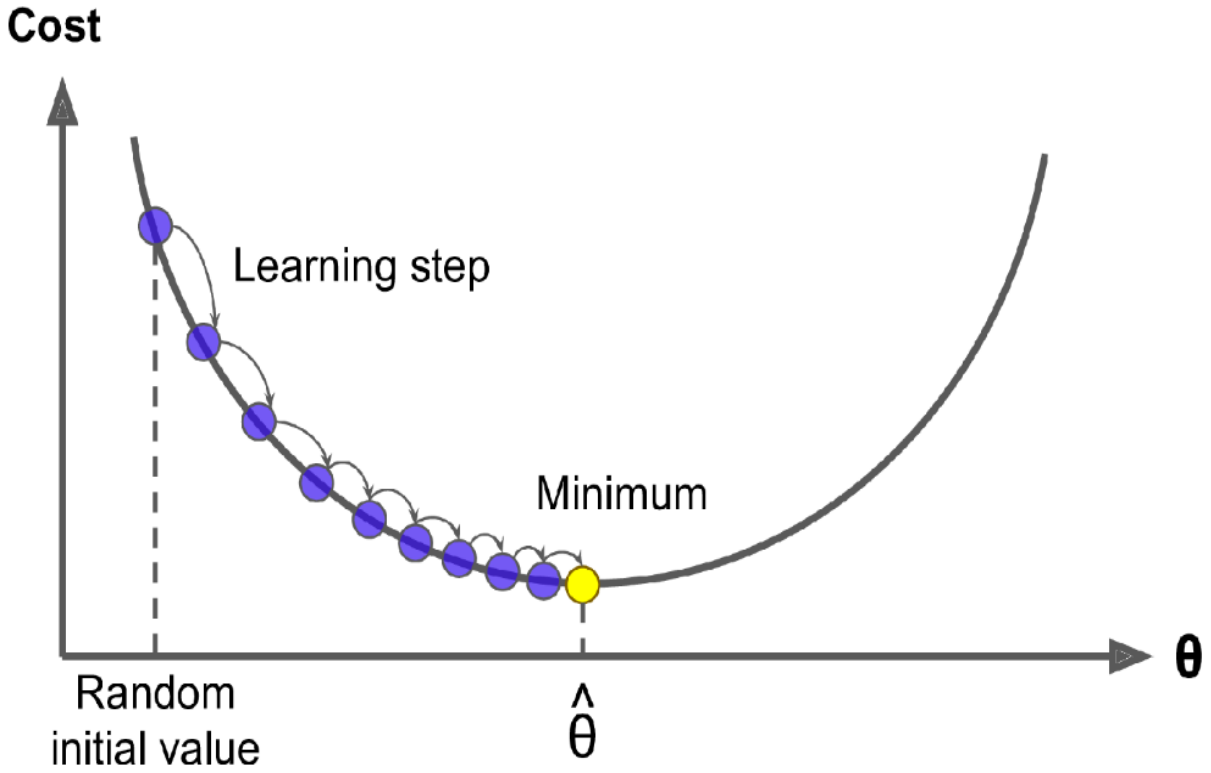


Figure 4-3. In this depiction of Gradient Descent, the model parameters are initialized randomly and get tweaked repeatedly to minimize the cost function; the learning step size is proportional to the slope of the cost function, so the steps gradually get smaller as the parameters approach the minimum

Learning Rate

- An important parameter in Gradient Descent is the size of the steps, determined by the learning rate hyperparameter. If the learning rate is **too small**, then the algorithm will have to go through many iterations to converge, which will take a long time.
- If the learning rate is **too high**, you might jump across the valley and end up on the other side, possibly even higher up than you were before. This might make the algorithm diverge, with larger and larger values, failing to find a good solution.

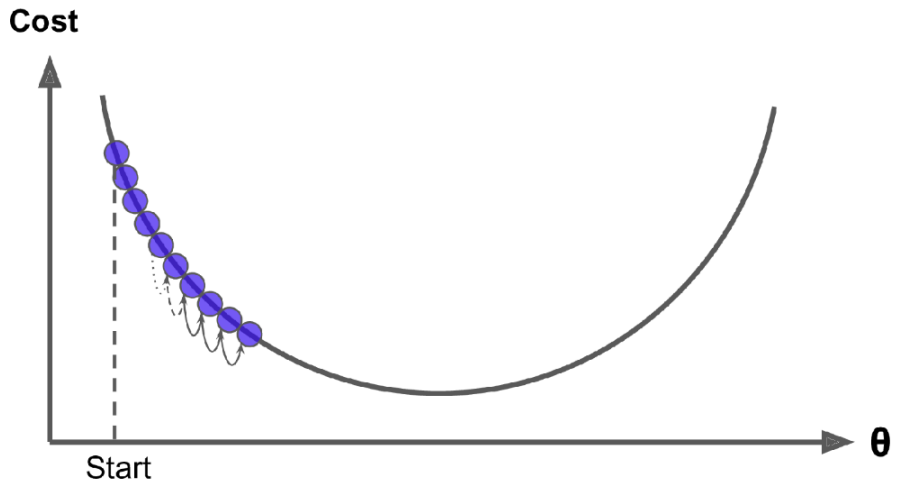


Figure 4-4. The learning rate is too small

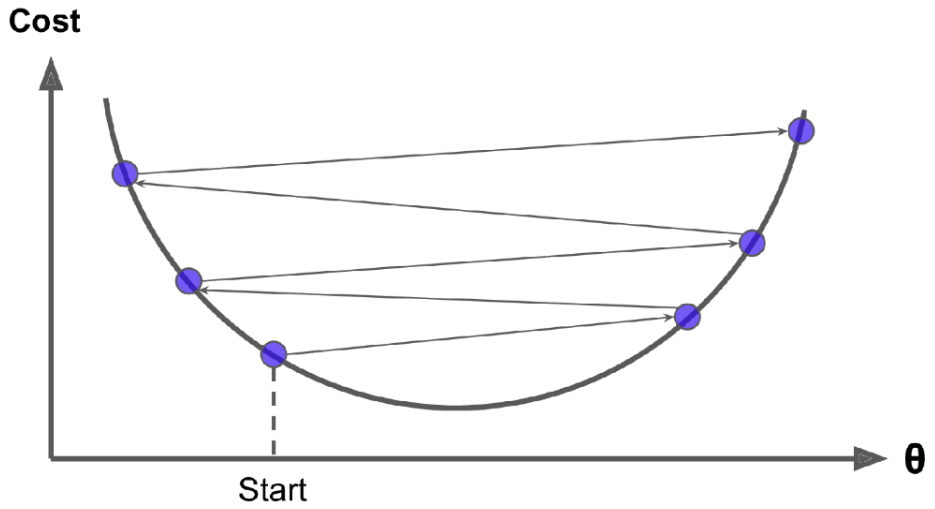


Figure 4-5. The learning rate is too large

Which Minimum is the Least (i.e. the best)?

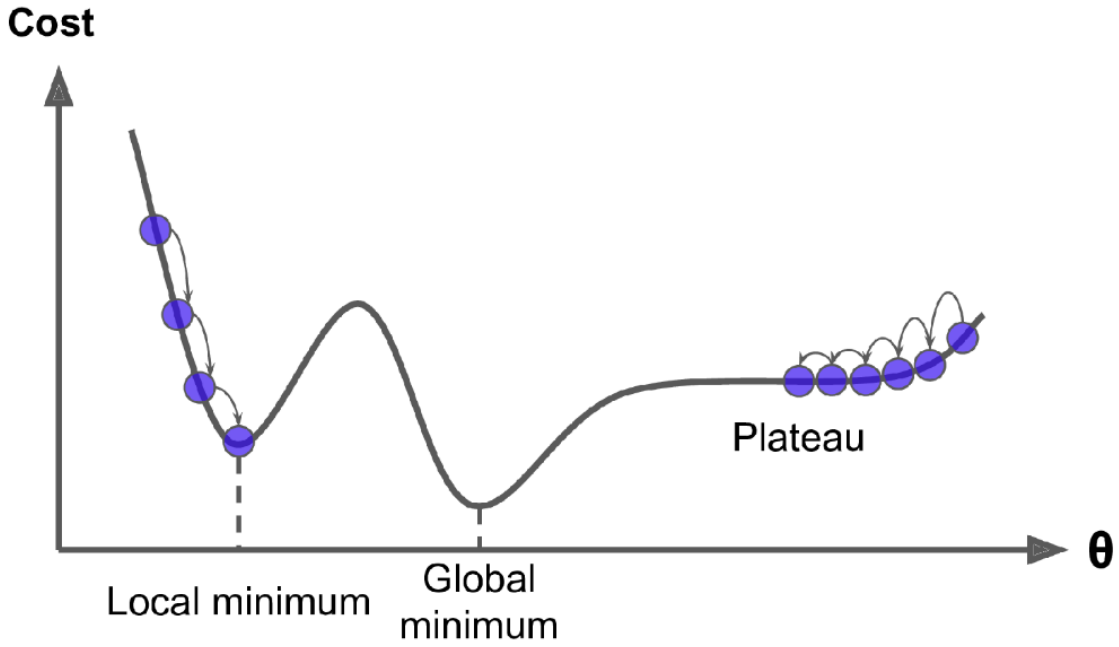


Figure 4-6. Gradient Descent pitfalls

- Not all cost functions look like nice, regular bowls. There may be holes, ridges, plateaus, and all sorts of irregular terrains, making convergence to the minimum difficult.

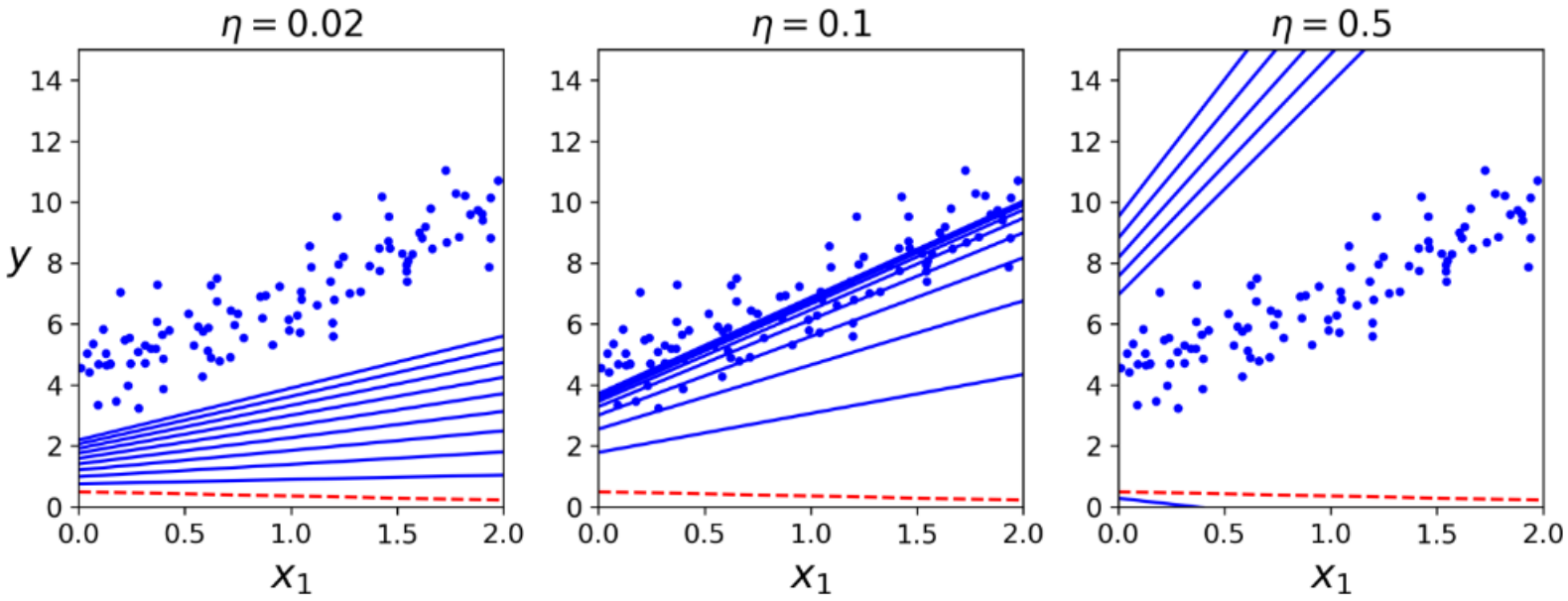


Figure 4-8. Gradient Descent with various learning rates

- On the **left**, the learning rate is too low: the algorithm will eventually reach the solution, but it will take a long time.
- In the **middle**, the learning rate looks pretty good: in just a few iterations, it has already converged to the solution.
- On the **right**, the learning rate is too high: the algorithm diverges, jumping all over the place and actually getting further and further away from the solution at every step.

How to Find a Good Learning Rate?

- To find a good learning rate, you can use grid search (Chapter 2). However, you may want to limit the number of epochs so that grid search can eliminate models that take too long to converge.
  - Scikit-Learn's GridSearchCV class can search for you. All you need to do is tell it which hyperparameters you want it to experiment with and what values to try out, and it will use cross-validation to evaluate all the possible combinations of hyperparameter values.
- You may wonder how to set the number of epochs.
  - If it is too low, you will still be far away from the optimal solution when the algorithm stops.
  - If it is too high, you will waste time while the model parameters do not change anymore.
- A simple solution is to set a very large number of epochs but to interrupt the algorithm when the gradient vector becomes very small

How Can You Tell that Your Model is Overfitting or Underfitting the Data?

- This high-degree Polynomial Regression model is severely overfitting the training data, while the linear model is underfitting it.
  - The model that will generalize best in this case is the quadratic model, which makes sense because the data was generated using a quadratic model.
- Use cross-validation to get an estimate of a model's generalization performance.
  - If a model performs well on the training data but generalizes poorly according to the cross-validation metrics, then your model is overfitting.
  - If it performs poorly on both, then it is underfitting. This is one way to tell when a model is too simple.

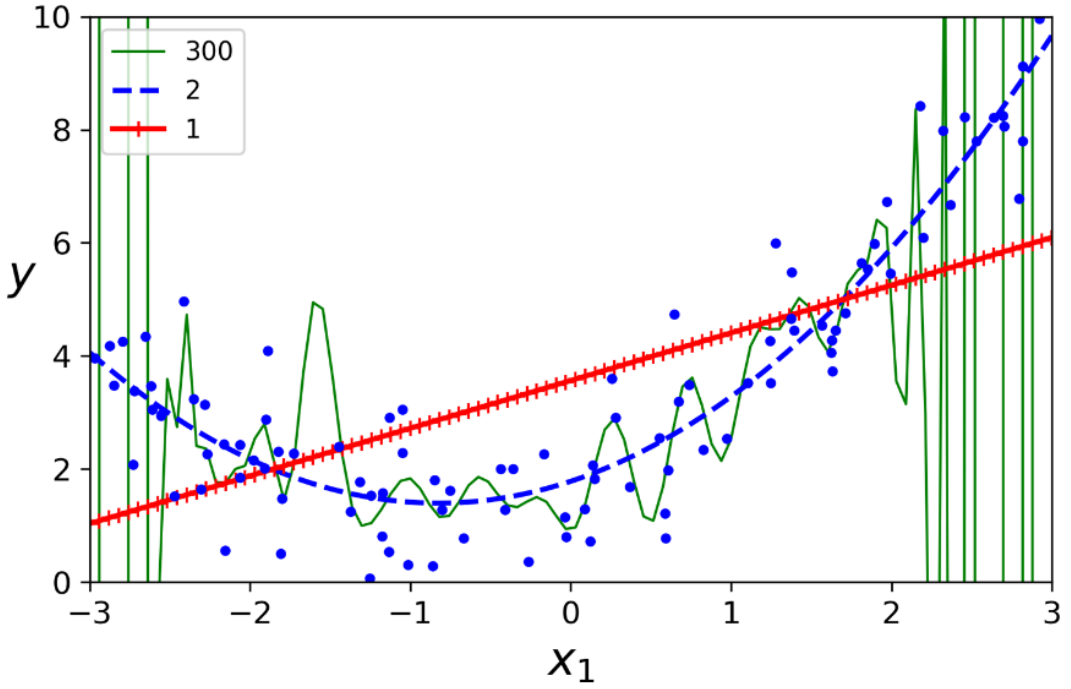


Figure 4-14. High-degree Polynomial Regression

Ridge Regression

- Ridge Regression (also called Tikhonov regularization) is a regularized version of Linear Regression: (L2 regularization) a regularization term equal to half of the square sum of the coefficients is added to the cost function.
- This forces the learning algorithm to not only fit the data but also keep the model weights as small as possible. Note that the regularization term should only be added to the cost function during training. Once the model is trained, you want to use the unregularized performance measure to evaluate the model's performance.

Equation 4-8. Ridge Regression cost function

$$J(\boldsymbol{\theta}) = \text{MSE}(\boldsymbol{\theta}) + \alpha \frac{1}{2} \sum_{i=1}^{n} {\theta_i}^2$$

- The hyperparameter α controls how much you want to regularize the model. If α = 0, then Ridge Regression is just Linear Regression. If α is very large, then all weights end up very close to zero and the result is a flat line going through the data's mean.
- It is recommended to scale the features before using Ridge Regression.

Lasso Regression

- Least Absolute Shrinkage and Selection Operator Regression (usually simply called Lasso Regression) is another regularized version of Linear Regression:
- Just like Ridge Regression, it adds a regularization term to the cost function, but it uses the L1 norm of the weight vector instead of half the square of the L2 norm in Ridge.

Equation 4-10. Lasso Regression cost function

$$J(\boldsymbol{\theta}) = \text{MSE}(\boldsymbol{\theta}) + \alpha \sum_{i=1}^{n} |\theta_i|$$

- An important characteristic of Lasso Regression is that it tends to eliminate the weights of the least important features (i.e., set them to zero).
- Lasso Regression automatically performs feature selection and outputs a sparse model (i.e., with few nonzero feature weights).

Early Stopping

- A way to regularize iterative learning algorithms such as Gradient Descent is to stop training as soon as the validation error reaches a minimum.
- As the epochs go by the algorithm learns, and its prediction error (e.g. RMSE) on the training set goes down, along with its prediction error on the validation set.
- After a while though, the validation error stops decreasing and starts to go back up.
- This indicates that the model has started to overfit the training data.
- With early stopping, you just stop training as soon as the validation error reaches the minimum.

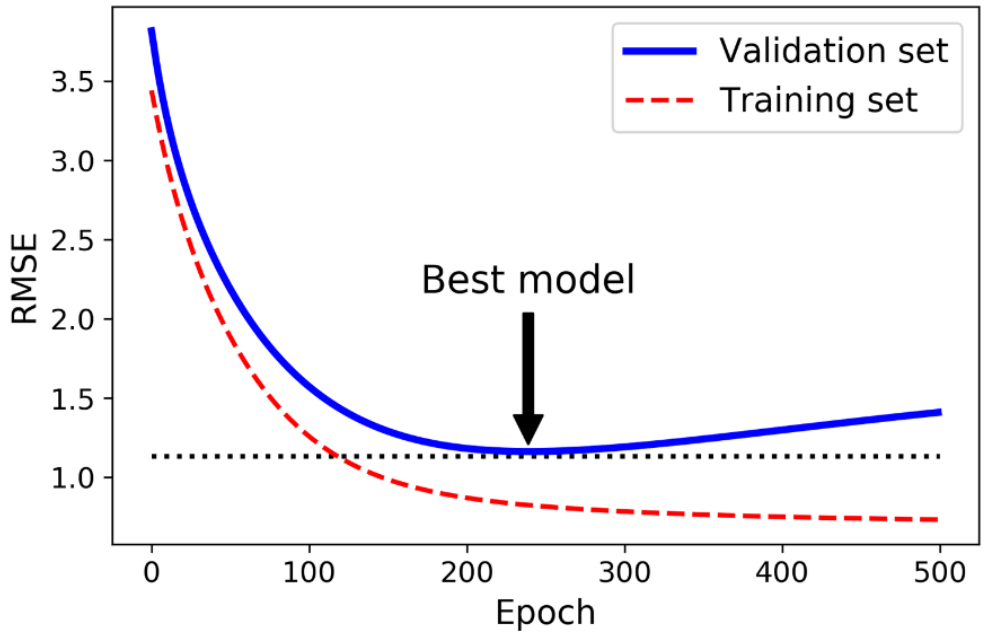


Figure 4-20. Early stopping regularization

Decision Boundaries

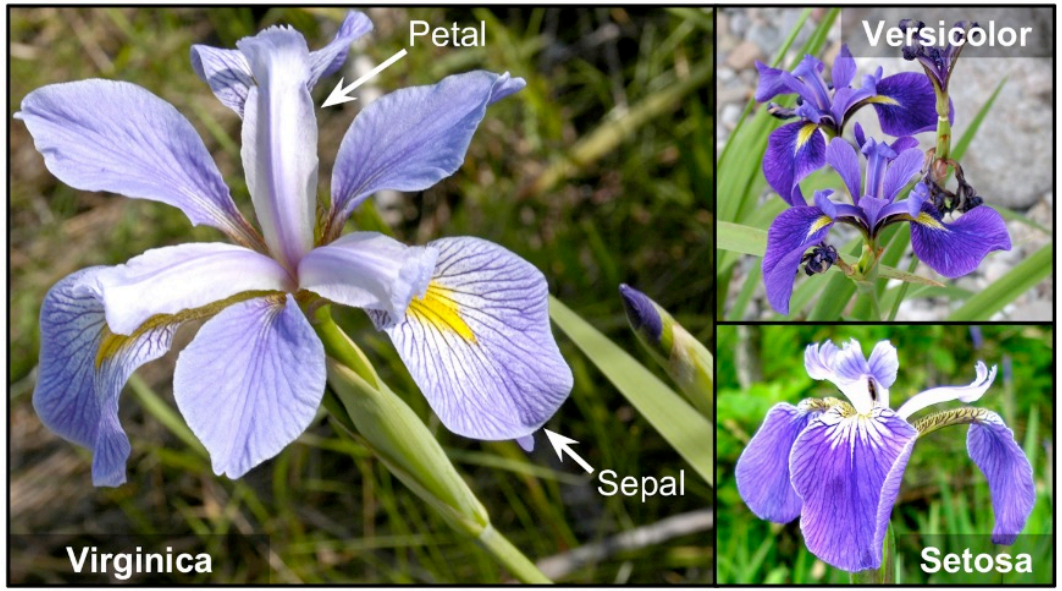


Figure 4-22. Flowers of three iris plant species[14]

- Logistic Regression (for binary classification – providing the probability that a record belongs to one of the two classes).
- This is a famous dataset (Iris dataset) that contains measurements of 150 iris flowers of three different species: Iris setosa, Iris versicolor, and Iris virginica

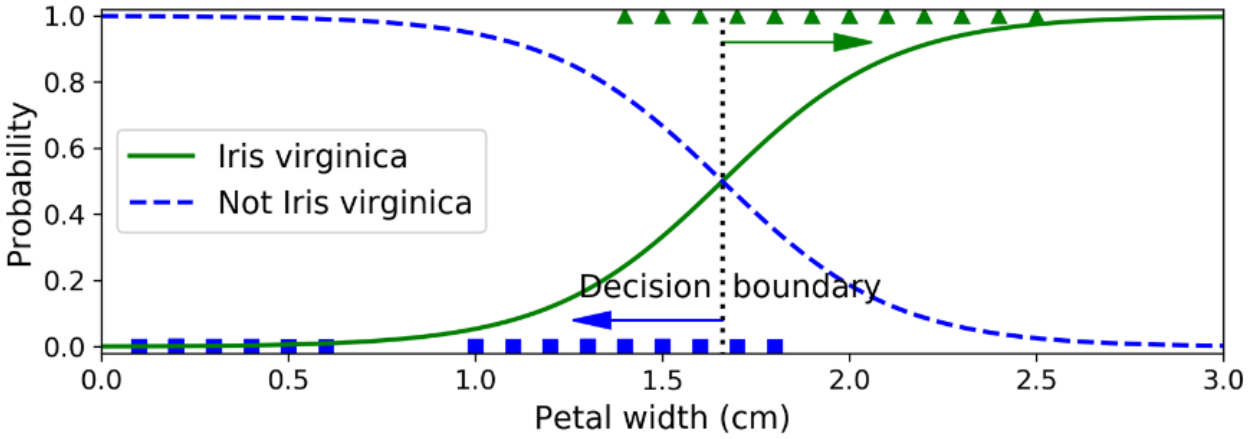


Figure 4-23. Estimated probabilities and decision boundary

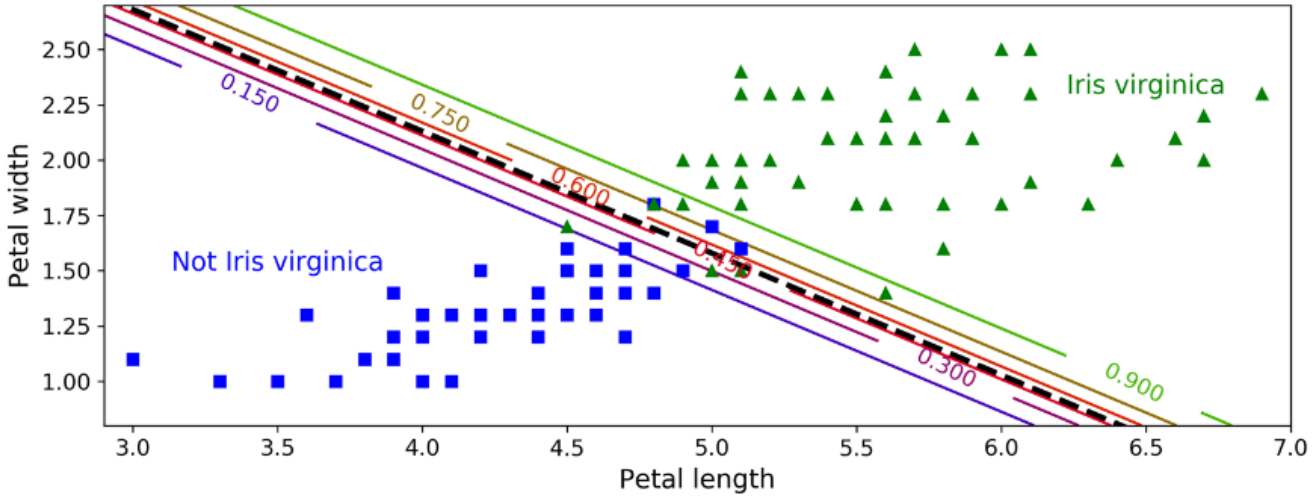


Figure 4-24. Linear decision boundary

Softmax Regression

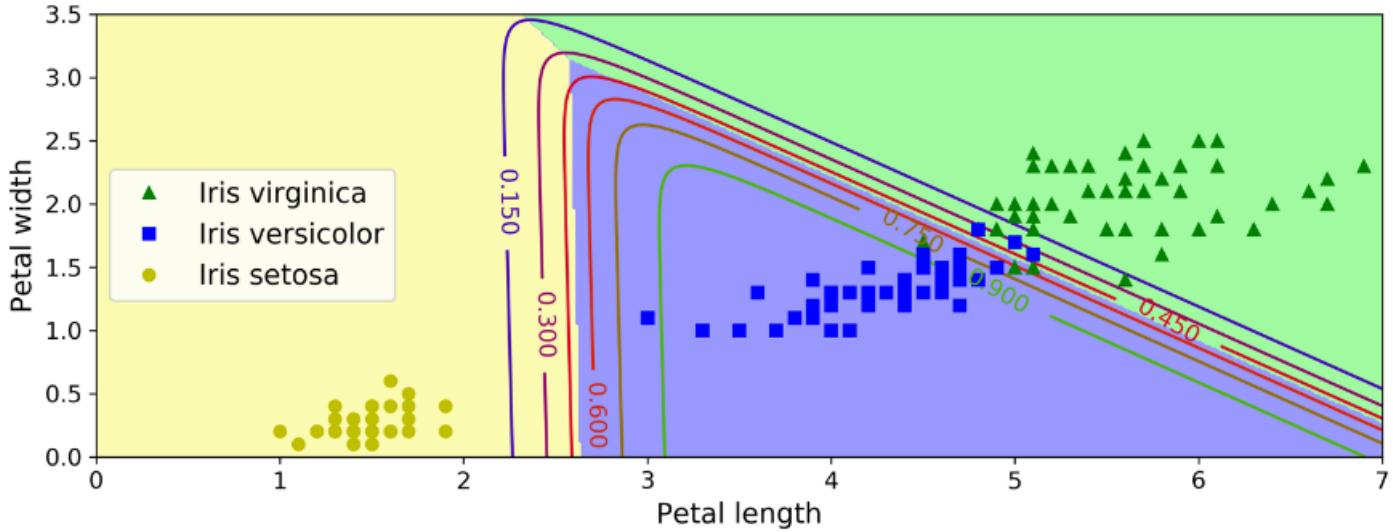


Figure 4-25. Softmax Regression decision boundaries

- The Softmax Regression classifier predicts the class with the highest estimated probability (which is simply the class with the highest score),
- The Softmax Regression classifier predicts only one class at a time (i.e., it is multiclass, not multioutput), so it should be used only with mutually exclusive classes, such as different types of plants. You cannot use it to recognize multiple people in one picture.

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

ML Engineering
Data Collection and Preparation
(MLE, Chapter 3)

Professor Hamza F. Alsarhan
SEAS 8505

Data Collection & Preparation (Chpt 3, MLE)

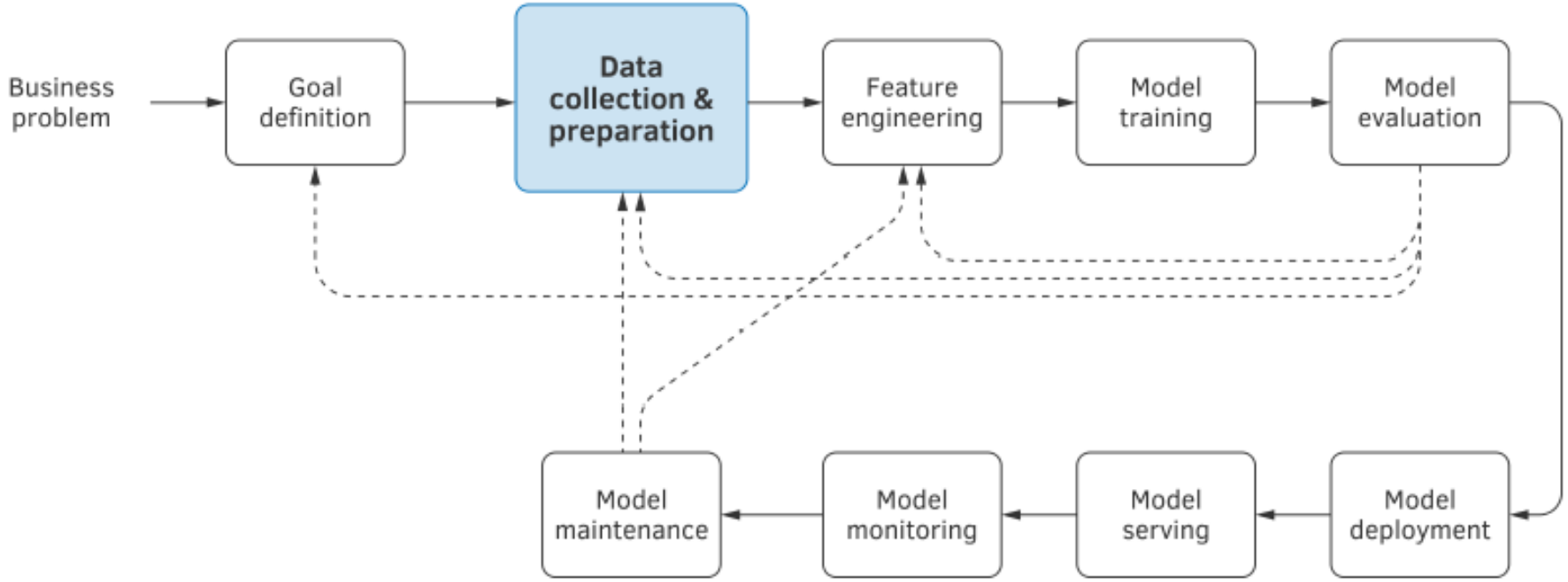
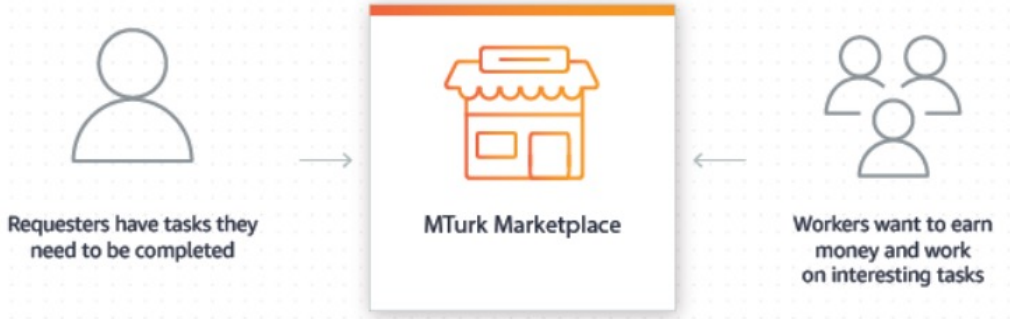


Figure 1: Machine learning project life cycle.

Questions about the Data

- Is the data:
  1. Accessible
  2. Sizeable
  3. Usable
  4. Understandable
  5. Reliable

Common Problems with Data



Amazon Mechanical Turk
https://www.mturk.com/

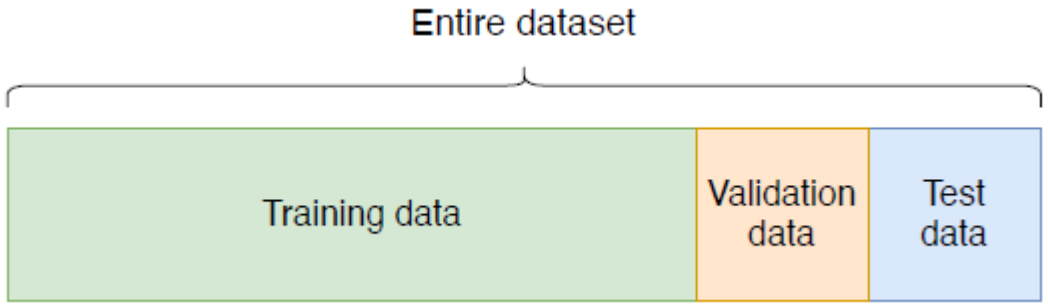
- High cost
- Bad quality
  - Noise
  - Bias
    - Many types: Selection bias, Self-selection bias, Sampling bias, …
    - Causes data to not be representative and will impact generalization, business goals, …
      - Distribution shift – tie back to IID assumption (random variables are independent and identically distributed, and the data used for the ML model is independently and randomly sampled).
    - Ways to Avoid Bias – start with asking questions about how the data was created.
  - Low predictive power
  - Outdated examples
  - Outliers

What is Good Data

- Data has sufficient information to train the model for its purpose
- Data is representative of post deployment expectations:
  - Examples cover the range of scenarios that the model will experience after it is deployed
  - Input data matches what will be seen in production
  - Unbiased (data input during production follows same distribution)
  - Enough variation to allow generalization
- Consistent labeling
- Avoid feedback loops that generate data

Data Partitioning (MLE, 3.6)

- Randomize data
- Split raw data
- Validation and Test sets follow the same distribution (same as production)
- Validation and Test sets need to be large enough to provide dependable statistics; however, these holdout sets are not seen by the learning algorithm.

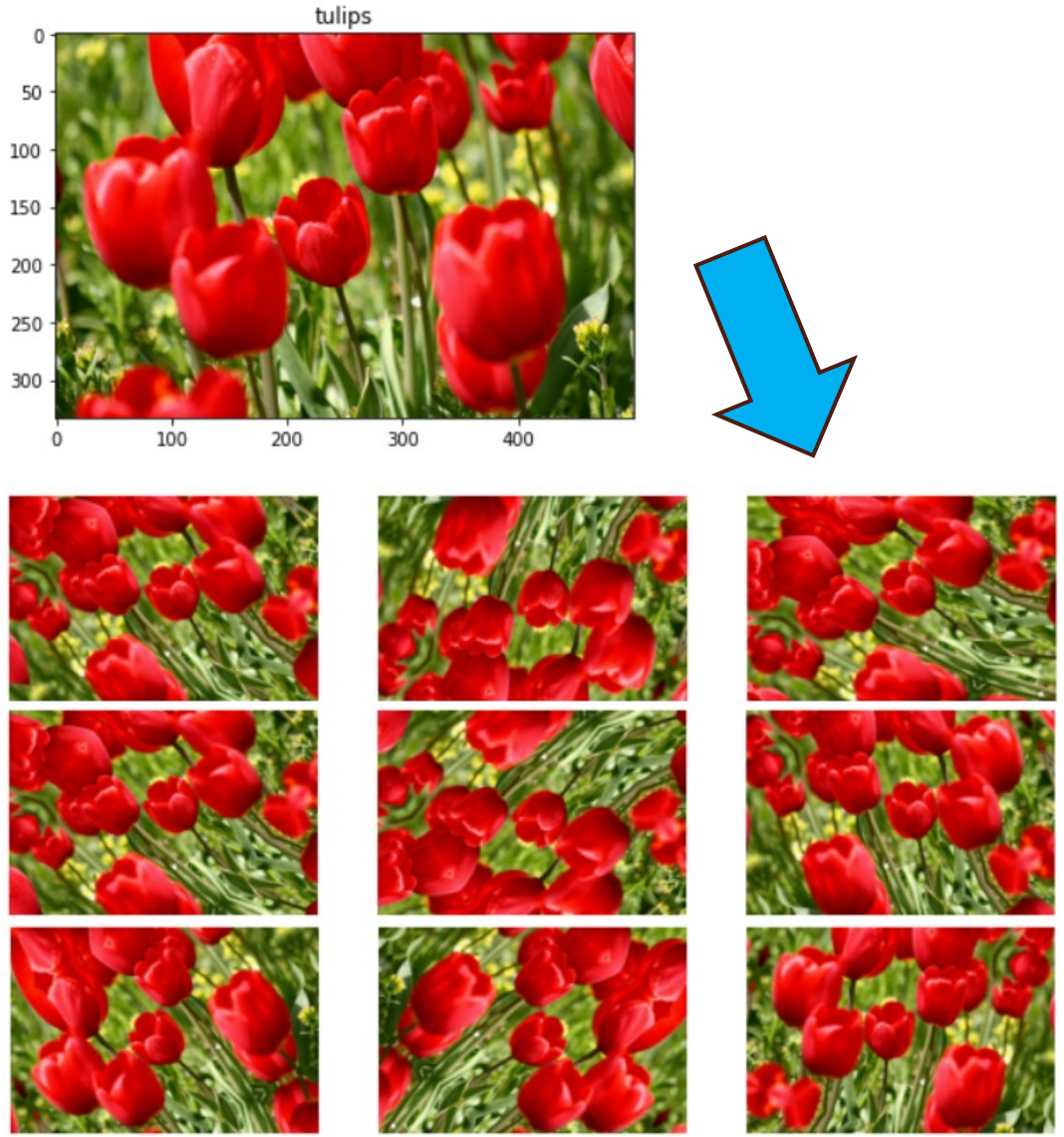


Dealing with Missing Attributes

- Datasets may contain observations with missing attributes
- We may choose to throw these observations away; however, this may not be feasible due to limited dataset size.
- Filling in the missing values is called data imputation
  - Mean imputation
    - When the attribute is numeric, use average value for that attribute within the dataset
    - When the attribute is discrete, use most common value in dataset (most likely)
  - Imputation by regression
    - Build regression model from training dataset to predict missing value

Data Augmentation

- Used to artificially create new data samples from existing samples and increase the diversity and number of records of the training dataset. This enhances the model robustness and ability to generalize on new data.
- Great for images
- Can apply to existing dataset to create more images
  - Flip
    - tf.keras.layers.RandomFlip
  - Crop
    - tf.keras.layers.RandomCrop,
  - Contrast
    - tf.keras.layers.RandomContrast,
  - Zoom
    - tf.keras.layers.RandomZoom
  - Many additional variations



https://www.tensorflow.org/tutorials/images/data_augmentation

Imbalanced Data

- Goal: Balanced number of labels in training data
- If possible, try to have number of instances close to equally split across classes
  - Example: For a 2-class scenario, half and half.
- When dealing with rare events such as disease, the learning algorithm could achieve high accuracy by defaulting to “no disease” output for every input.
  - If disease instances were a small percentage of training data, the reported accuracy would be very high using this simple approach.
  - We want the model to learn to classify “no disease” and “disease” with high accuracy and can force this by having a balanced dataset.
- Can reduce class imbalance by oversampling or undersampling (or a hybrid approach)

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

Model Selection

“"It is better to be vaguely right than exactly wrong"”
– Carveth Read

Professor Hamza F. Alsarhan
SEAS 8505

Generalization

How well does a learned model generalize from the data it was trained on, to a new test set?

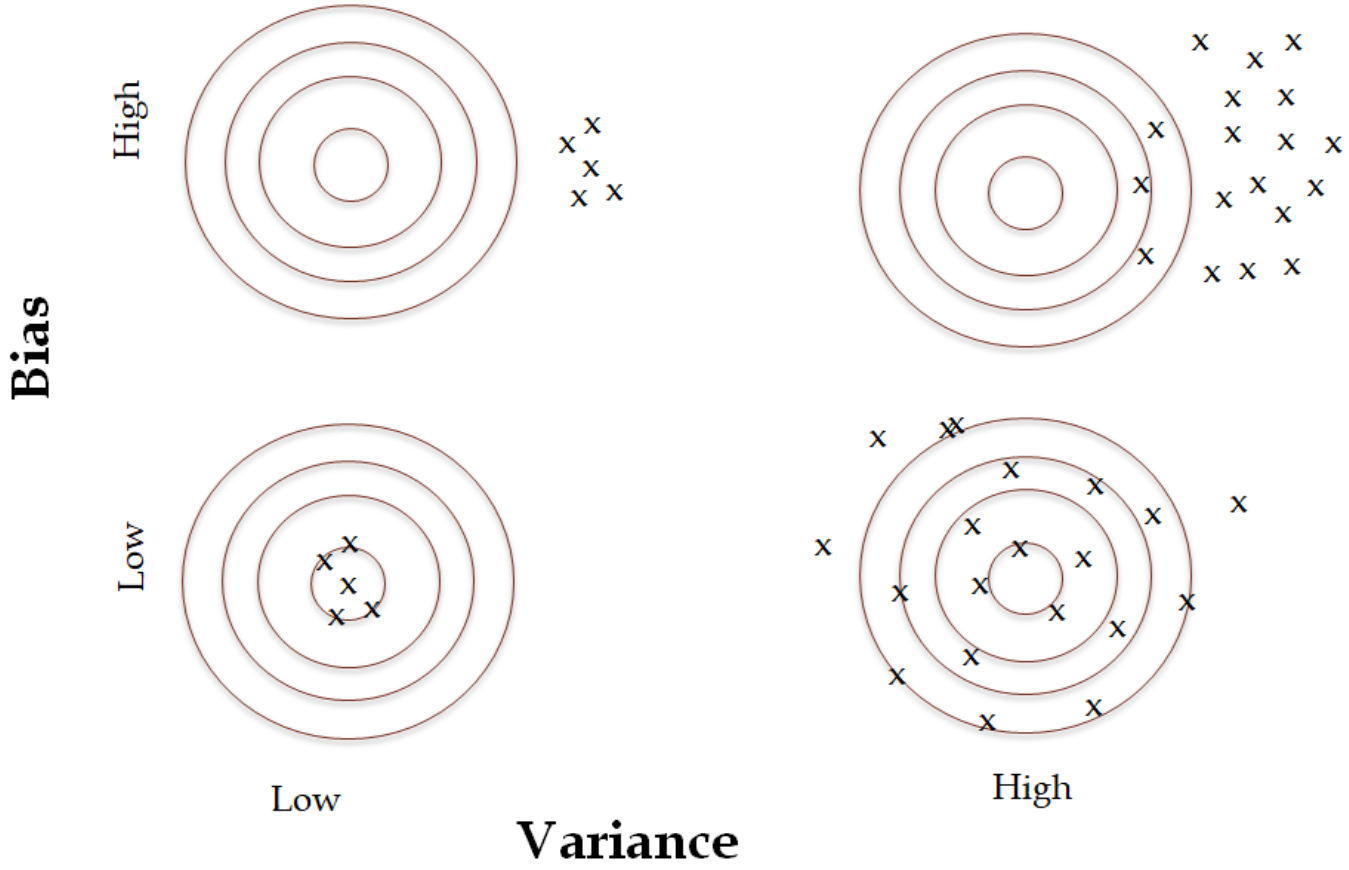
Training set
(labels known)

Test set
(labels unknown)

Slide credit: L. Lazebnik

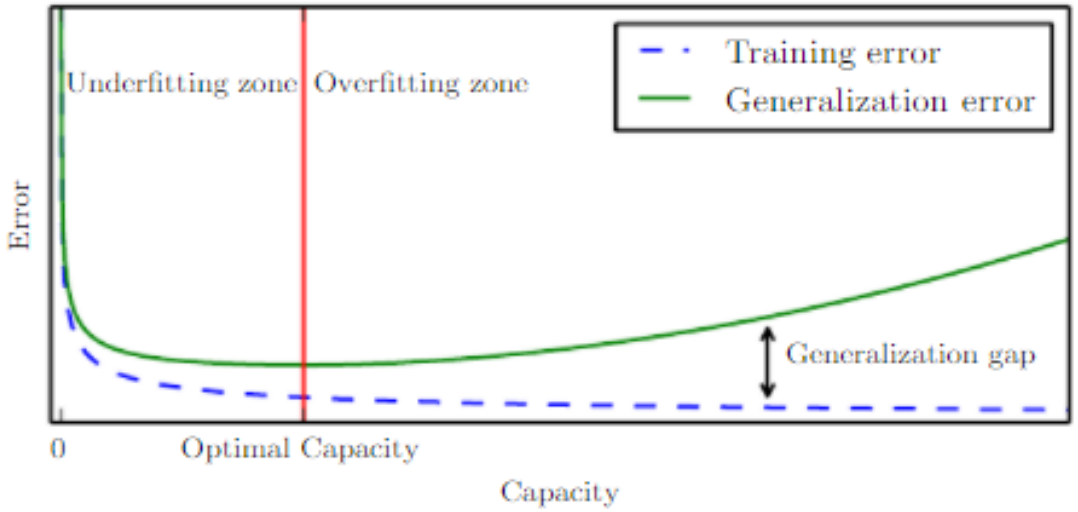
Visual Depiction of Bias and Variance

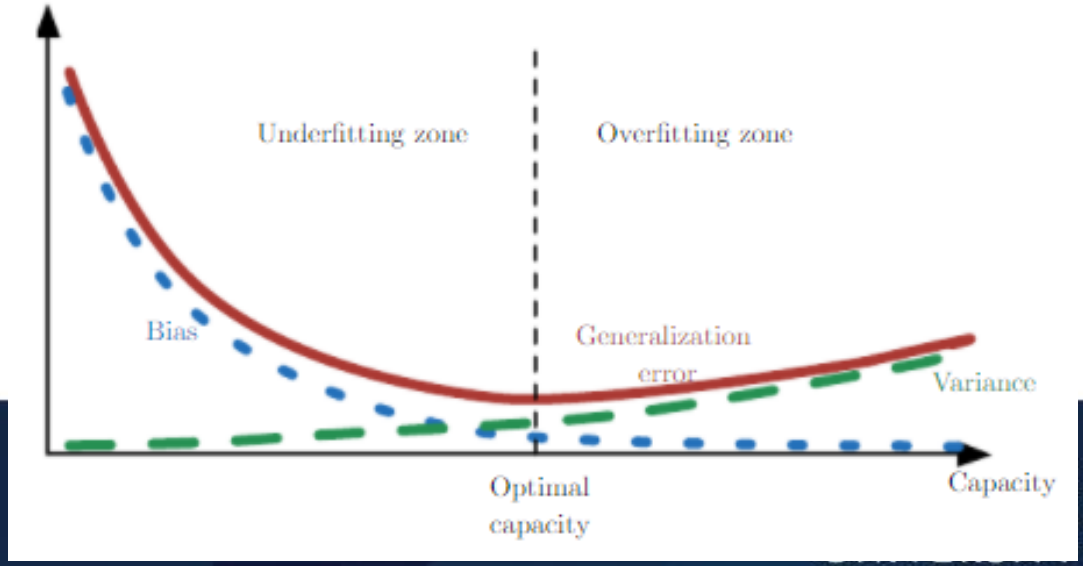
- Bias – how much the model fits the underlying function
  - High bias indicates the model class does not contain the solution
- Variance – how general is the model class
  - Small changes in data result in large changes of model
  - High variance indicates model class is too general



Capacity, Underfitting and Overfitting

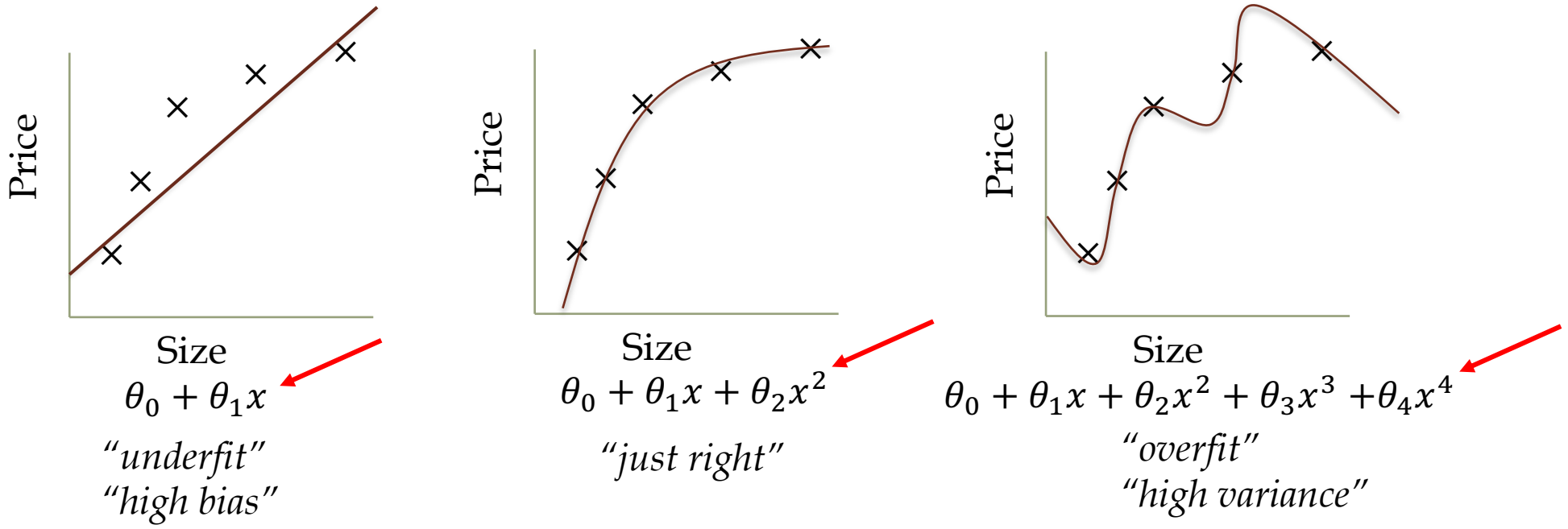
- **Capacity** – how broad is the variety of possible functions the model can assume
- **Underfitting** – when the model performs poorly on training data
  - Example: Your ML project goal is 90% accuracy, however after model training the accuracy only reaches 40%
  - If using linear regression, can include polynomials to increase capacity
- **Overfitting** – when the model performs well on training data but performs poorly on test data it hasn't seen before
  - Example: training error is 0.0001% (training accuracy of 99.9999%) however error on the test data subset is 40% (test accuracy of 60%)
  - Memorized the training data but cannot generalize to new data. Learned the noise in the training data.





DLB, p109

Finding the Optimal Model



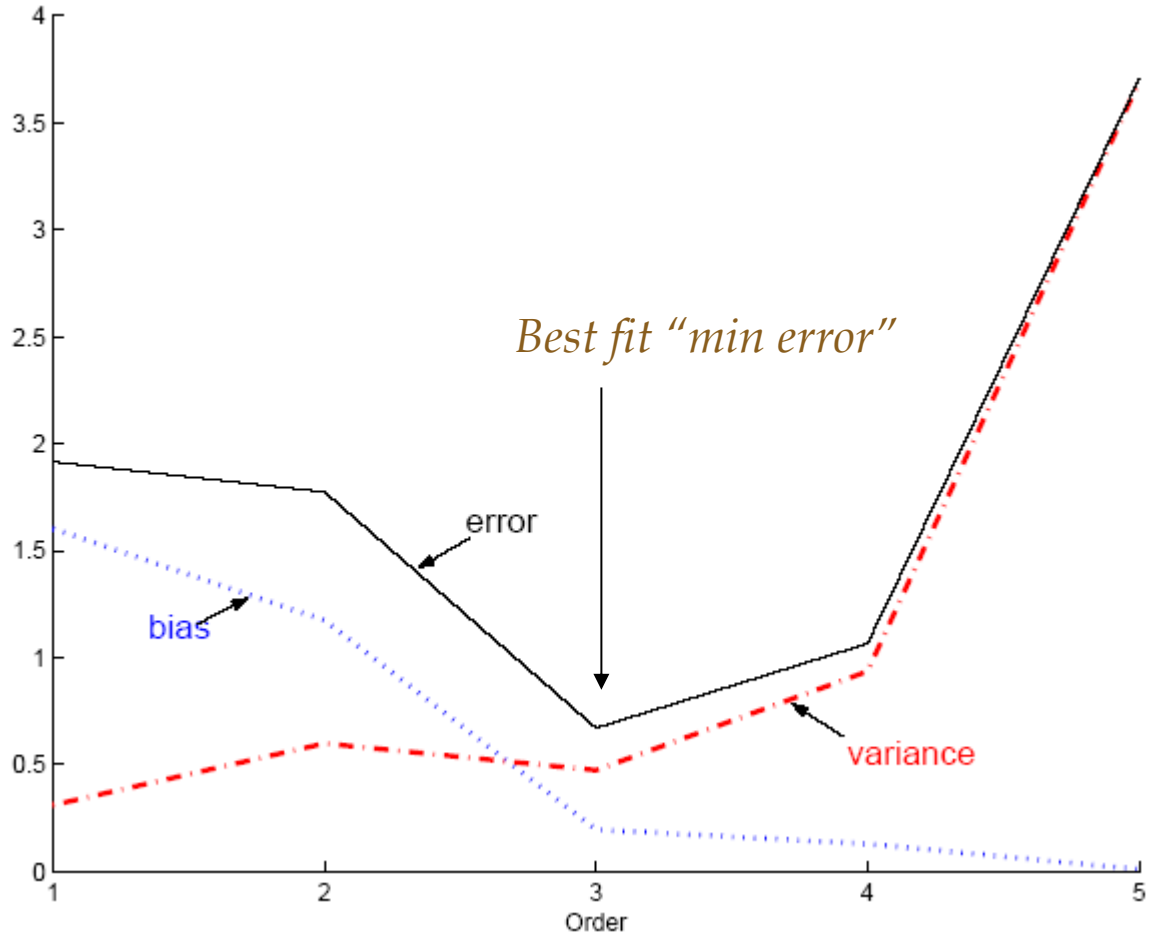
As we increase model complexity, bias decreases (better fit to data) and variance increases (fit varies more with data)

Overfitting: If the hypothesis space is too large, then you can choose a high order polynomial to fit almost any function. The learned hypothesis may fit the training dataset very well, but fail to generalize to new examples i.e. accurately predict prices on new house examples

Based on Regularization | The Problem of Overfitting – Andrew Ng
https://www.youtube.com/watch?v=u73PU6Qwl1I

Polynomial Regression

- As the order of the polynomial increases (i.e. complexity)
  - Bias decreases – making the error on the training data small
  - Variance increases – model becomes too sensitive to variation in the inputs and loses ability to generalize
  - Error initially decreases, then increases



Adapted from Alpaydin

Triple Trade-Off

- There is a trade-off between three factors (Dietterich, 2003):

  1. Complexity of Hypothesis class $\mathcal{H}$, c ($\mathcal{H}$),
  2. Training set size, N,
  3. Generalization error, E, on new data

- As N↑, E↓
- As c (H)↑, first E ↓ and then E↑

- In practice, increasing dataset size has an outsized impact when compared to model improvements

Training, Validation, and Test Sets

- Cannot find the bias and variance directly. To estimate generalization error, we need data unseen during training. We split the data as:
  - Training set
    - Used for learning to set the parameters of the model
  - Validation set
    - Choose the learning algorithm
    - Used for tuning the hyperparameters (i.e. configuration of the learning algorithm)
  - Test set
    - Used for publication of the results
- No absolute rules about how to split the data between training, validation and test sets. Some options: 50%/25%/25%; 70%/15%/15%; or 100 examples for each class within validation and test sets
- Cross-validation when you do not have enough data for a dedicated validation set

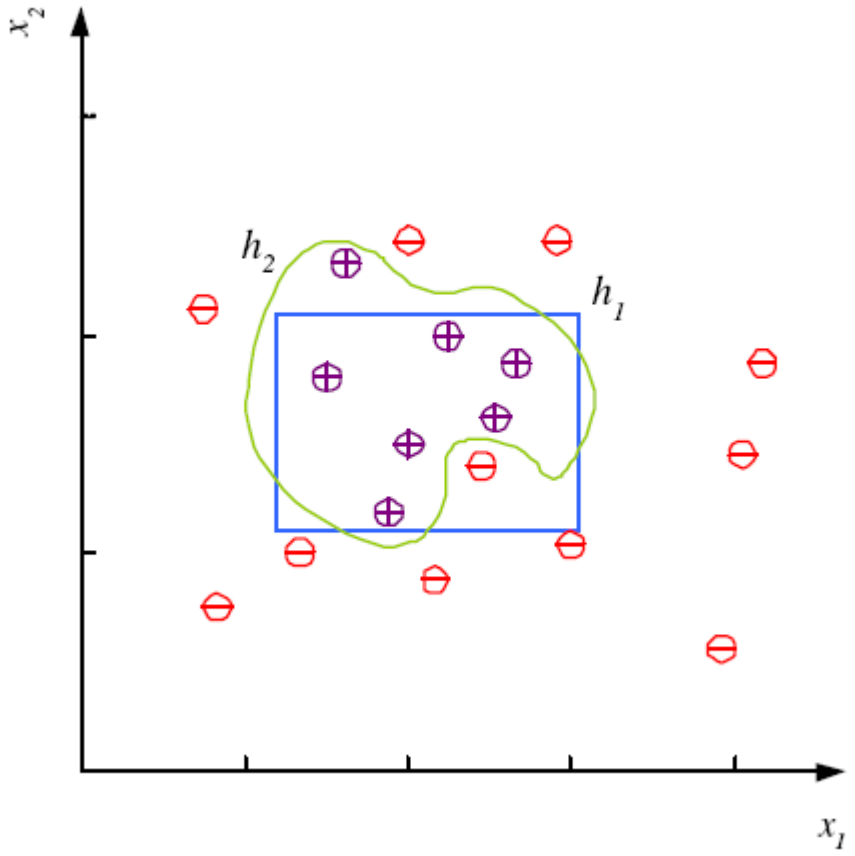
Noise and Model Complexity

Noise is any unwanted anomaly in the data

- Example: corrupted labels

Benefits of Simple Models:

- Simpler to use
  - Lower computational complexity
- Easier to train
  - Lower space complexity
- Easier to explain
  - More interpretable
- Generalize better
  - Lower variance - Occam's razor



Generalization

- There is no best algorithm for **all** possible tasks
  - No Free Lunch Theorem (Wolpert, 1996)
- We need to understand our application (task) and then design our algorithm accordingly (specificity)
- ML finds rules that are probably correct about most members of the set
- ML algorithms must perform well on new, previously unseen inputs
  - This ability is called generalization
- ML vs Optimization
  - Both apply to training a ML model – want low training error
  - ML also wants generalization error on the test set (i.e. unseen data or input) to be low

Model Selection Criteria

- Model selection criteria attempt to find a good compromise between:
  - The complexity of a model
  - Its prediction accuracy on the data
- Reasoning: a good model is a simple model that achieves high accuracy on the given data
- Also known as **Occam's Razor:** the best theory is the smallest one that describes all the facts
  - William of Ockham, born in the village of Ockham in England about 1285, was one of the most influential philosophers of the 14th century.

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

Setup – Before you Start

Before You Start (Preconditions)

- Validate Schema Conformity
  - Ensure original data and current data are the same
- Define an Achievable Performance Level
  - When further model improvement is not needed
- Choose a Performance Metric
  - Best metric depends upon data and problem
- Choose the Right Baseline
  - Model or algorithm to provide a reference point
- Split Data Into Three Sets (training; validation; test)

MLE, Chapter 5; MLB, Chapter 5

Feature Engineering

- One-Hot Encoding
  - Numerical encoding (binary; zero or one)
  - Length of vector is number of possible values – if we had a column called color, and it has three values (red, yellow, and green), each of those colors would have a new column created for it including a binary value based on the color of that data point
  - Used to encode categorial data points that have no ordinal relationship
- Binning
  - Converting continuous features into bins of various ranges
- Normalization
  - Convert range of values into a standard range (usually between 0 and 1)
  - Brings all features to a common scale to ensure they each have equal representation in the model
- Standardization
  - Squeezes normal values into a small range similar to Normal distribution (0 mean and 1 standard deviation and variance) – centers the distribution around 0

Selecting the Learning Algorithm

Main Properties of a Learning Algorithm:

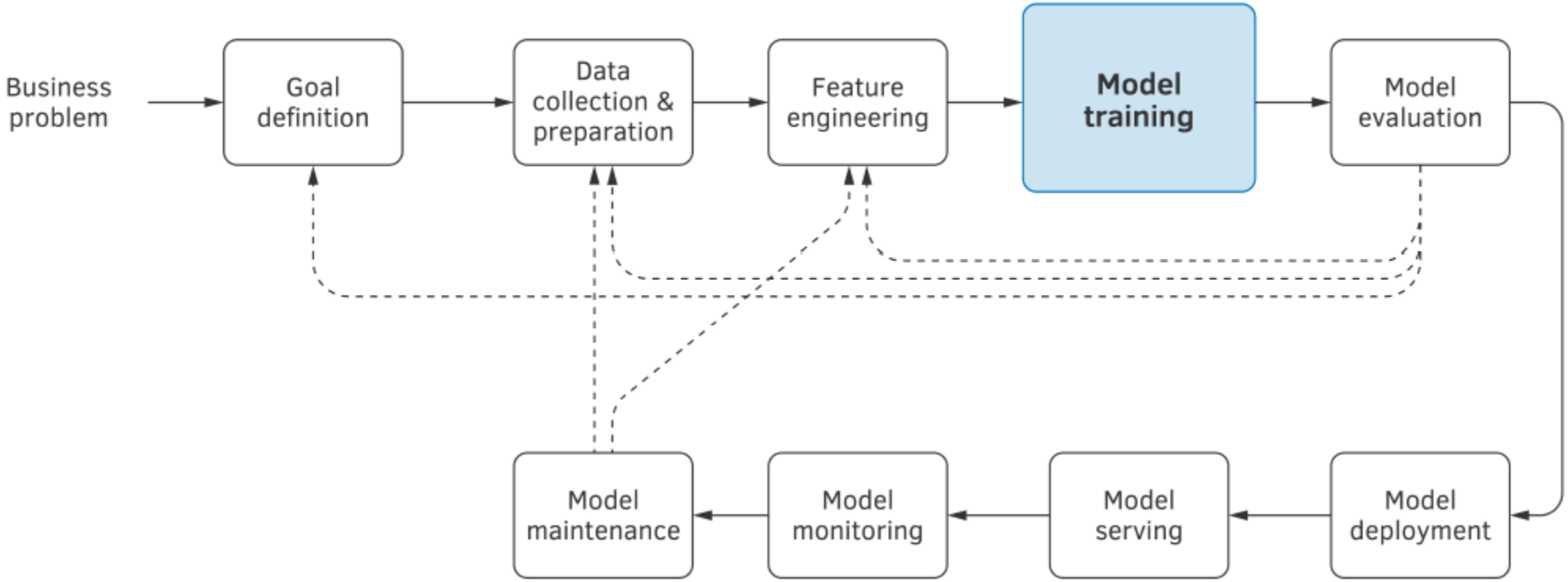
- Explainability
  - Is an explanation needed for how the model produced its output?
- In-memory vs. out-of-memory
  - Does the dataset fit within computer memory
- Number of features and examples within dataset
- Nonlinearity of the data (i.e. is data linearly separable)
  - Essential to determine whether a linear model could work
- Training speed
  - Ensure that model refresh needs can be met with training times
- Prediction speed
  - How much data will need to be processed and how quickly is a response needed
  - Essential for real time applications such as intrusion detection

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

Hyperparameters and Cross-Validation

Professor Hamza F. Alsarhan
SEAS 8505

Model Training



MLE, Chapter 5

Hyperparameter Tuning

- Hyperparameters effect:
  - Speed of training
  - Bias-variance
  - Precision-recall

- Tuned by the analyst looking for the best combination (not by learning algorithm)
- Tuning techniques
  - Grid Search – simple approach when range of hyperparameters is not too large
  - Random Search – random sampling
  - Coarse-to-Fine Search – grid search and random search
  - Bayesian – uses prior knowledge

Hyperparameter Selection

- *Hyperparameter:* external parameter that can be tuned to optimize the performance of a machine learning algorithm
  - Different from basic parameter that is part of a model, such as a coefficient in a linear regression model
  - Example hyperparameter: *k* in the *k*-nearest neighbor classifier
- We are not allowed to peek at the final test data to choose the value of this parameter
  - Adjusting hyperparameters to the test data will lead to optimistic performance estimates on test data
  - Parameter tuning needs to be viewed as part of the learning algorithm and must be done using the training data only
- But how to get a useful estimate of performance for different parameter values so that we can choose a value?
  - Answer: split the data into a smaller “training” set and a validation set” (normally, the data is shuffled first)
  - Build models using different values of *k* on the new, smaller training set and evaluate them on the validation set
  - Pick the best value of k and rebuild the model on the full original training set

Cross-Validation Method

- Get a number of training and validation set pairs from a dataset X
- Leave out part as the test set
- Randomly divide data into K parts, then randomly divide each part into two and use one half for training and the other half for validation.
- K is typically 5 to 30
- Problem: Datasets are rarely large enough for this
- Solution: Cross-validation

Cross-Validation

- K-fold cross-validation avoids overlapping validation sets
  - First step: split data into k subsets of equal size
  - Second step: use each subset in turn for validation, the remainder for training
  - This means the learning algorithm is applied to k different training sets
- Often the subsets are stratified before the cross-validation – breaking the dataset based on a specific criteria or class label
- Error estimates are averaged to yield an overall error estimate; also, standard deviation is often computed
- Alternatively, predictions and actual target values from the k folds are pooled to compute one estimate
  - Does not yield an estimate of standard deviation – as that describes the variability within each fold
- Assess final model according to test set

Making the Most of the Data

- Once evaluation is complete, *all the training+validation data* can be used to build the final classifier – **never** include the test data in the training process even after evaluation
- Generally, the larger the training data, the better the classifier (but returns diminish)
- The larger the test data the more accurate the error estimate
- *Holdout* procedure: method of splitting original data into training and test set
- Dilemma: ideally both training set *and* test set should be large!

Comparing Algorithms: Going Beyond Error Rate

- Criteria (Application-dependent):
  - Misclassification error, or risk (loss functions)
  - Training time/space complexity
  - Testing time/space complexity
  - Interpretability
  - Easy programmability

- Cost-sensitive learning
  - Class imbalance – assigning different costs to different classes, encouraging the model to pay attention to minority classes

Many Datasets Publicly Available

- UC Irvine Machine Learning Repository
- Kaggle datasets
- Amazon's AWS datasets
- OpenDataMonitor
- Quandl
- Wikipedia's list of Machine Learning datasets
- Quora.com
- The datasets subreddit

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

Decision Trees

Professor Hamza F. Alsarhan
SEAS 8505

Overview – Industrial-Strength Algorithms

- Very simple to understand and implement
- Output is easy to understand
- Very scalable
- For an algorithm to be useful in a wide range of real- world applications it must:
  - Permit numeric attributes
  - Allow missing values
  - Be robust in the presence of noise
- Basic scheme needs to be extended to fulfill these requirements

Overview

- Decision Trees are versatile Machine Learning algorithms that can perform both classification and regression tasks, and even multioutput tasks.

- Powerful algorithms capable of fitting complex datasets
  - One of the many qualities of Decision Trees is that they require very little data preparation
  - Don’t require feature scaling or centering at all for various reasons, one of which is because they do not measure the distance between data points

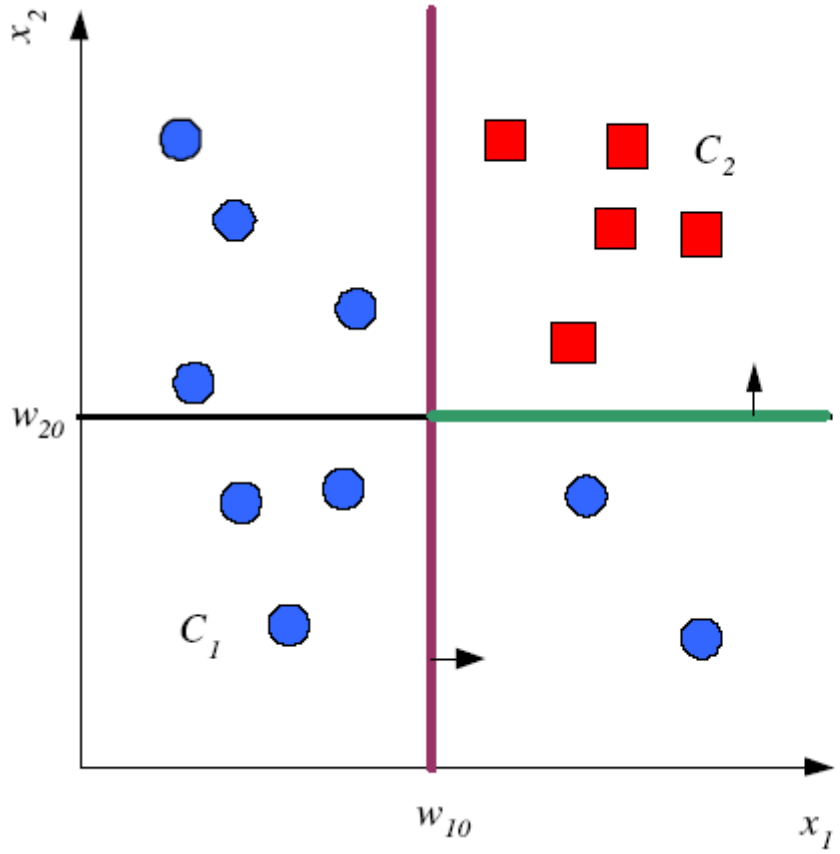
- Decision Trees are also the fundamental components of Random Forests, which are among the most powerful Machine Learning algorithms available today

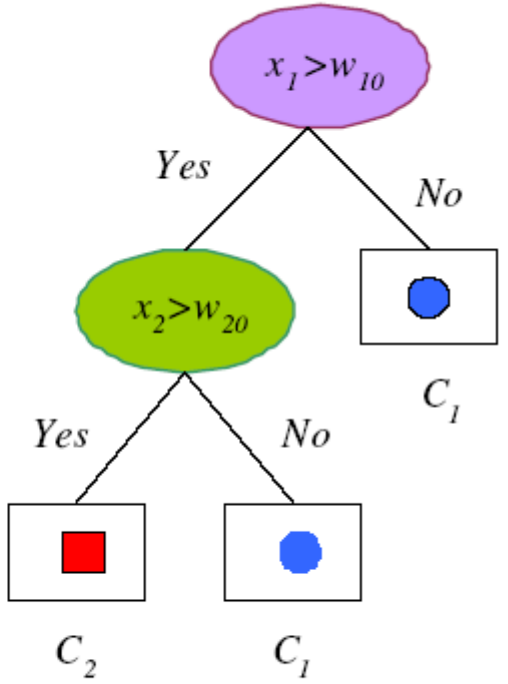
- **More about DTs:**
  - Hierarchical Model for Supervised Learning
  - Can represent any Boolean function
  - Variable size
  - Deterministic
  - Can have discrete or continuous parameters
  - Eager Learner
  - Batch Processing (mainly)

Eager vs. Lazy Learning

- Tree is built by adding nodes (considered to be a “constructive search”)
- Eager Learning (builds a model from the training data)
- Given a set of training data, it constructs a classification model before receiving new (e.g. test sample) data to classify
- Main advantage: target function is approximated globally during training (requires less memory than using a lazy learning system)
- Eager: Generally Batch Processing (some online forms)
- Some examples are :
  - Lazy (delays the generalization process until a query is made): K - Nearest Neighbor
  - Eager (generalizes the training data into a model before receiving any queries): Decision Tree, Naive Bayes, Artificial Neural Networks

DTs Use Nodes, Branches, and Leaves



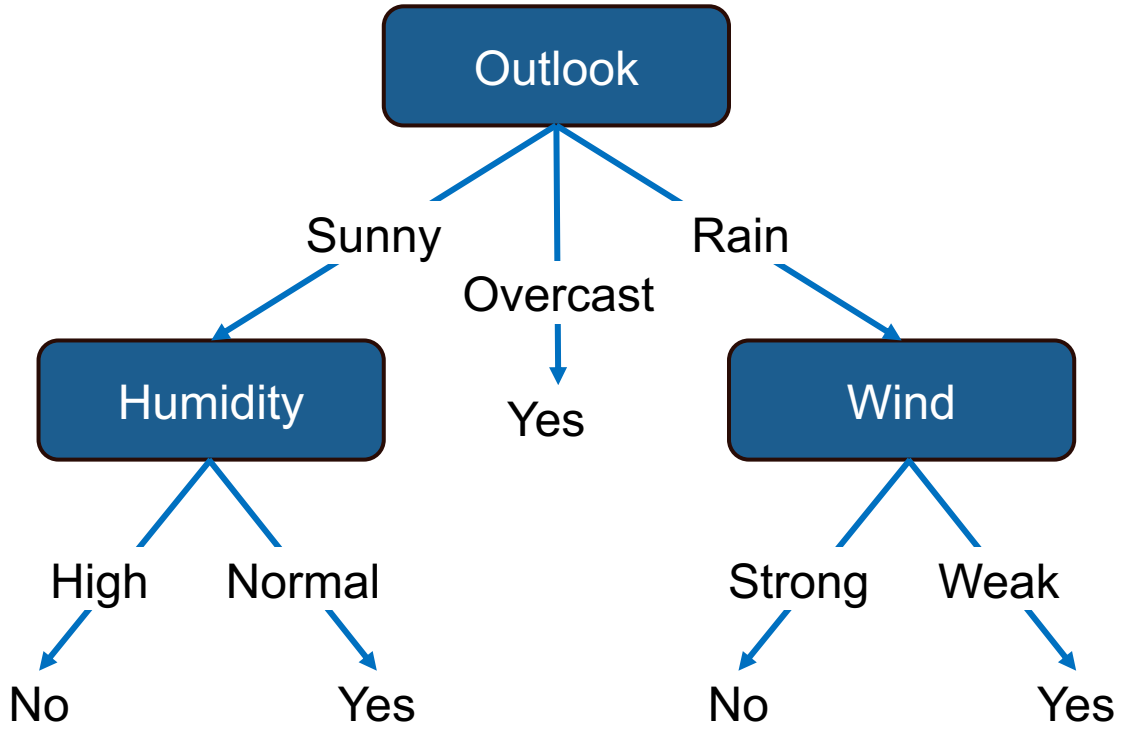


Playing Tennis Example: Training Dataset

Day	Outlook	Temperature	Humidity	Wind	PlayTennis?
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

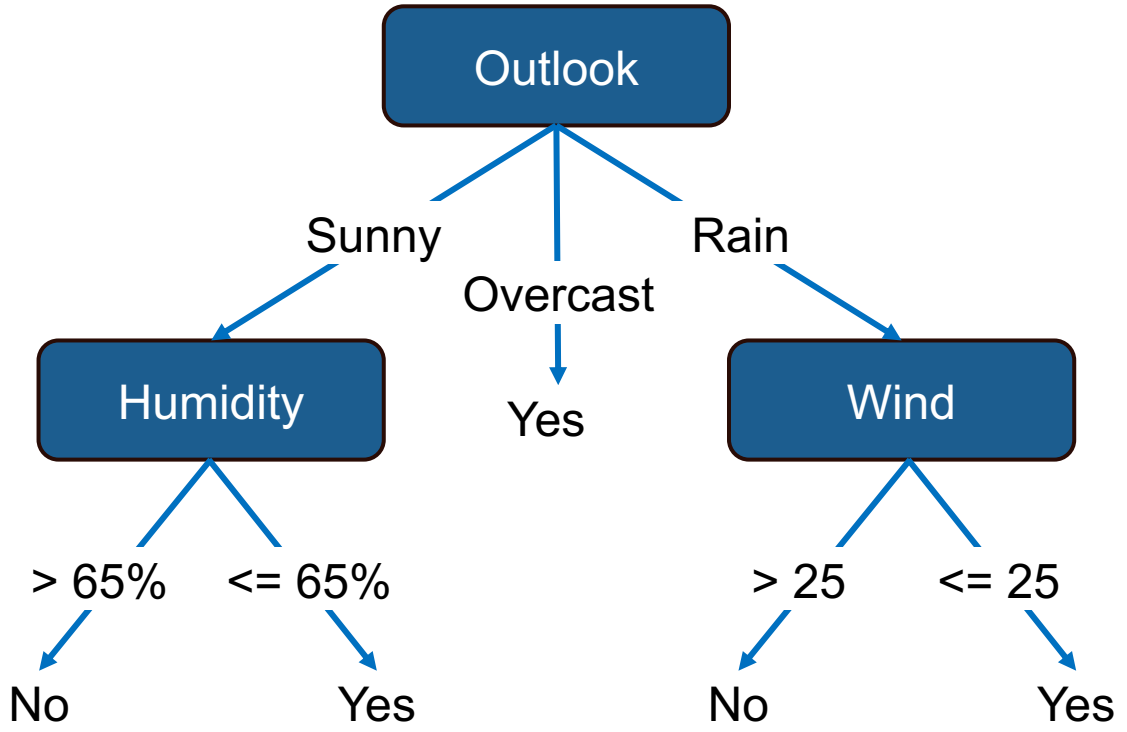
Playing Tennis Example: Features and the DT

- Features:
  - Outlook
  - Temperature
  - Humidity
  - Wind

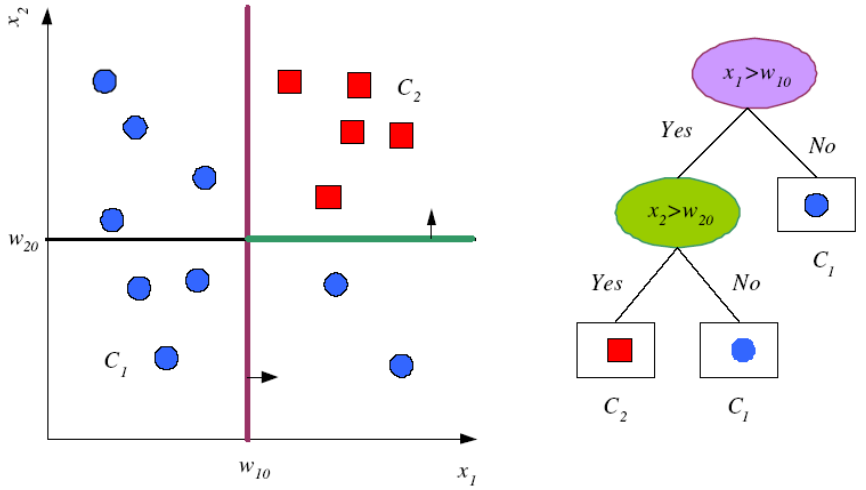


Playing Tennis Example: Features and the DT

- Features:
  - Outlook
  - Temperature
  - Humidity
  - Wind

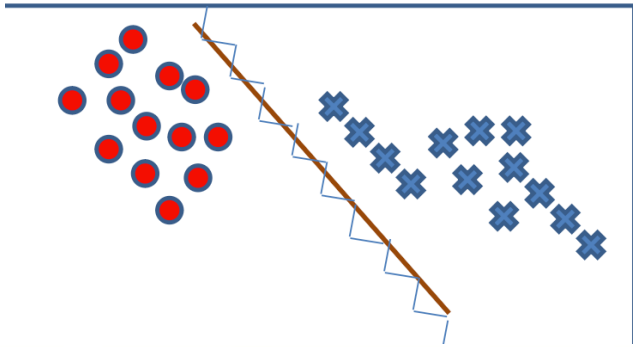


What Can Decision Trees Learn?



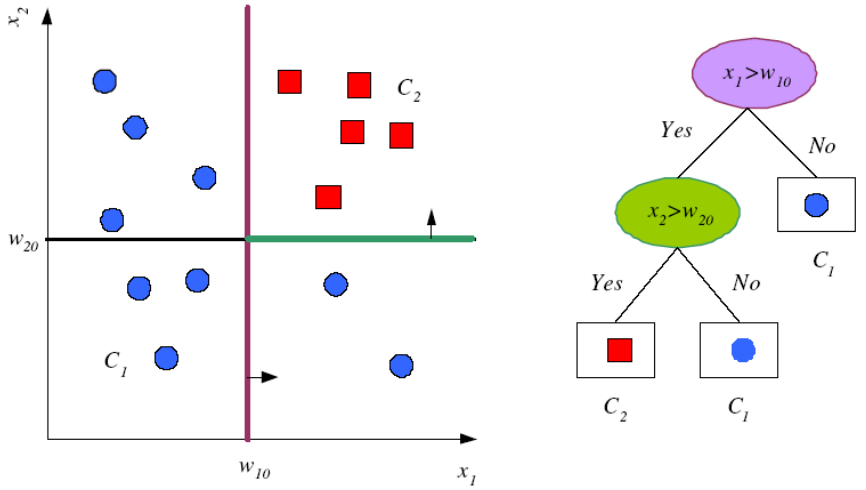
- DTs make axis-aligned splits (perpendicular to the feature axes

How about this?



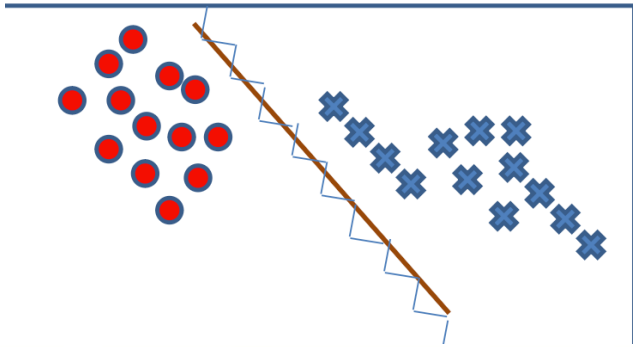
- How do we think they will handle diagonal splits?
- What would that lead to?

What Can Decision Trees Learn?



- DTs make axis-aligned splits (perpendicular to the feature axes

How about this?



- They will represent diagonal boundaries with many splits, leading to deeper and more complex trees
- This would lead to overfitting

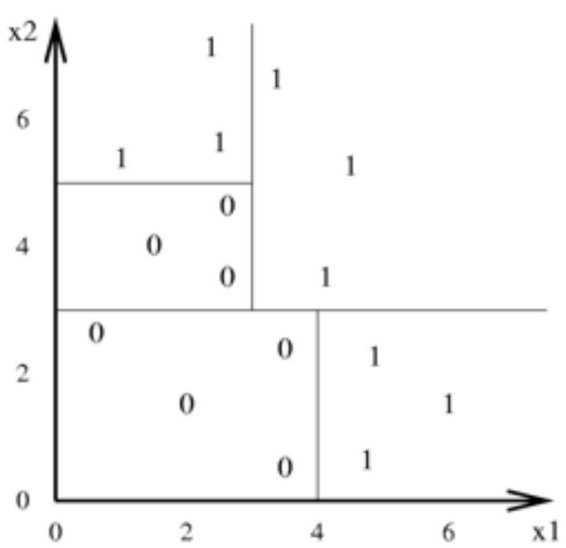
Divide and Conquer

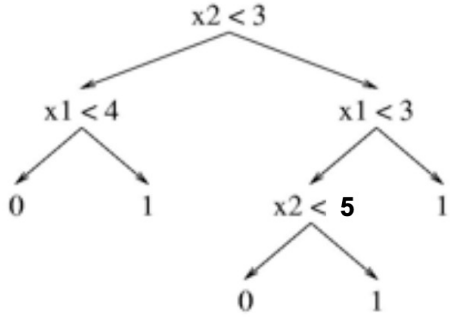
- Internal decision nodes
  - Univariate: Uses a single attribute, x_i
    - Numeric xi: Binary split: $x_i > w_m$
    - Discrete xi: n-way split for n possible values
  - Multivariate (rarely used): Creates a linear combination of multiple features, uses attributes, $\boldsymbol{x}$
- Leaves
  - Classification: Class labels, or proportions
  - Regression: Numeric; *r* average, or local fit
- Learning is greedy; find the best split recursively
  - Local versus Global Optimality
  - Always makes the choice that seems to be the best at that moment
  - Has only one shot to compute the optimal solution

Eager vs. Greedy in Machine Learning

- Eager Learning Algorithm:
  - Processes and analyzes entire training dataset upfront.
  - Invests significant computational effort to build a generalized model.
  - Ready for immediate deployment; quickly generates predictions.
  - Examples: Decision Trees, Naive Bayes, Artificial Neural Networks.
- Greedy Algorithm:
  - Makes locally optimal choices at each step of optimization.
  - Prioritizes immediate gains without considering long-term consequences.
  - Provides quick solutions but may not always yield the most optimal outcome.
  - Commonly used in feature selection and model optimization tasks.
  - Examples: Forward Selection, Backward Elimination.
- Summary:
  - Eager learning processes entire dataset upfront, while greedy algorithms make locally optimal choices
  - iteratively.
  - Eager algorithms build a generalized model, while greedy algorithms prioritize immediate gains.

Illustrative Example

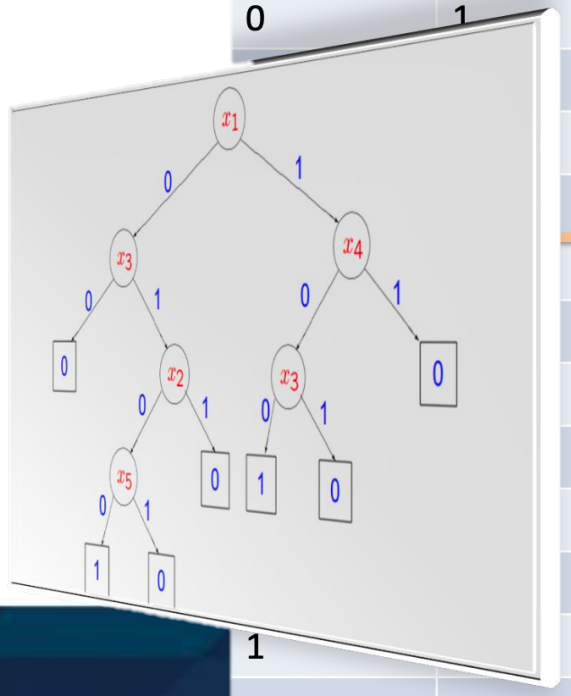




- While Decision Trees are primarily eager learners, they also exhibit some characteristics of greedy algorithms during the construction phase.
- Specifically, at each node of the tree, a decision is made based on a greedy criterion, such as maximizing information gain or minimizing impurity.
- This means that the algorithm selects the feature that provides the most immediate improvement in splitting the data at each step, without considering the potential future consequences of that decision.
- Therefore, while tree learning algorithms are primarily eager learners, they incorporate greedy decision-making during the construction of the tree structure.

Can we Represent any Boolean Function with a DT?

0	0	0	0	x_n
0	0	0	1	…
0	0	1	0	…
0	0	1	1	…
0	1	0	0	…
		0	1	…
		1	0	…
		1	1	…
		0	0	…
		0	1	…
		1	0	…
		1	1	…
		0	0	…
		0	1	…
1		1	0	…
1	1	1	1	…

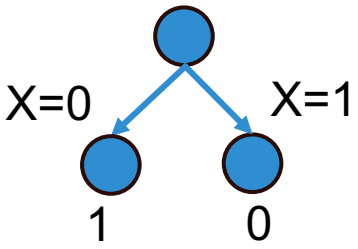


Properties

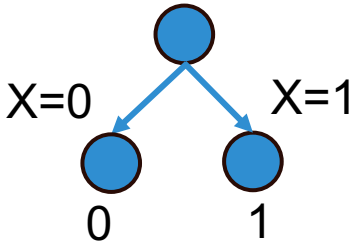
- As the number of nodes increases the hypothesis space grows
- We call this the “depth” of the tree
- Depth 1: Decision Stump (e.g. Boolean function of one feature)
- Depth 2: Can represent a Boolean function of two features and some other functions
- So how do we learn these decision trees?

Illustrative Examples

x	y
0	1
1	0

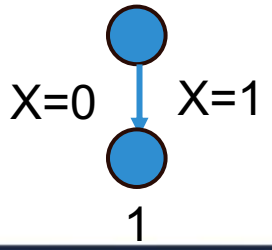


x	y
0	0
1	1

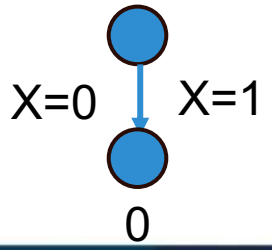


Nodes: 3
Depth: 1

x	y
0	1
1	1



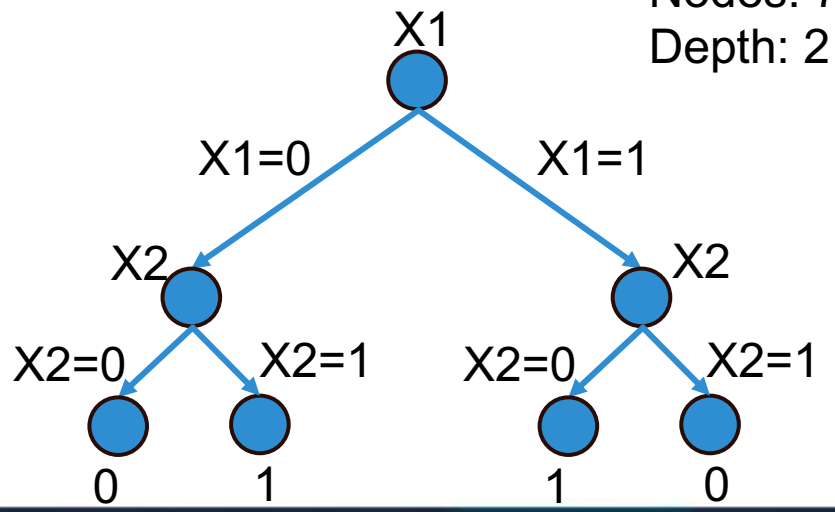
x	y
0	0
1	0



Nodes: 2
Depth: 1

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0

Nodes: 7
Depth: 2

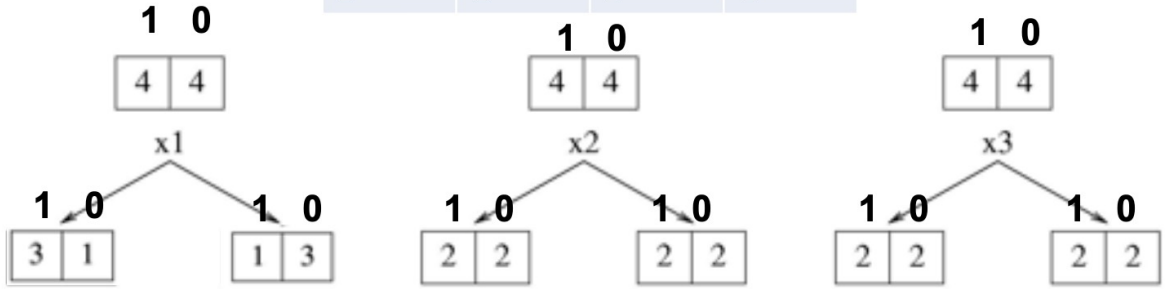


Decision Tree Learning

- **Problem Setting:**
- Set of possible instances *X*:
  - Each instance *x* in *X* is a feature vector
  - $x = \langle x_1, x_2, \ldots x_n \rangle$
- Unknown Target Function f:X→Y (Y is discrete valued)
- Set of function hypotheses H={h | h: X→ Y}
- Input:
  - Training Examples of unknown target function f
- Output:
  - Hypothesis h ∈ H that best approximates target function f

How to Choose the Best Attribute for Splitting

x1	x2	x3	y
0	0	0	**0**
0	0	1	**1**
0	1	0	**0**
0	1	1	**0**
1	0	0	**1**
1	0	1	**0**
1	1	0	**1**
1	1	1	**1**

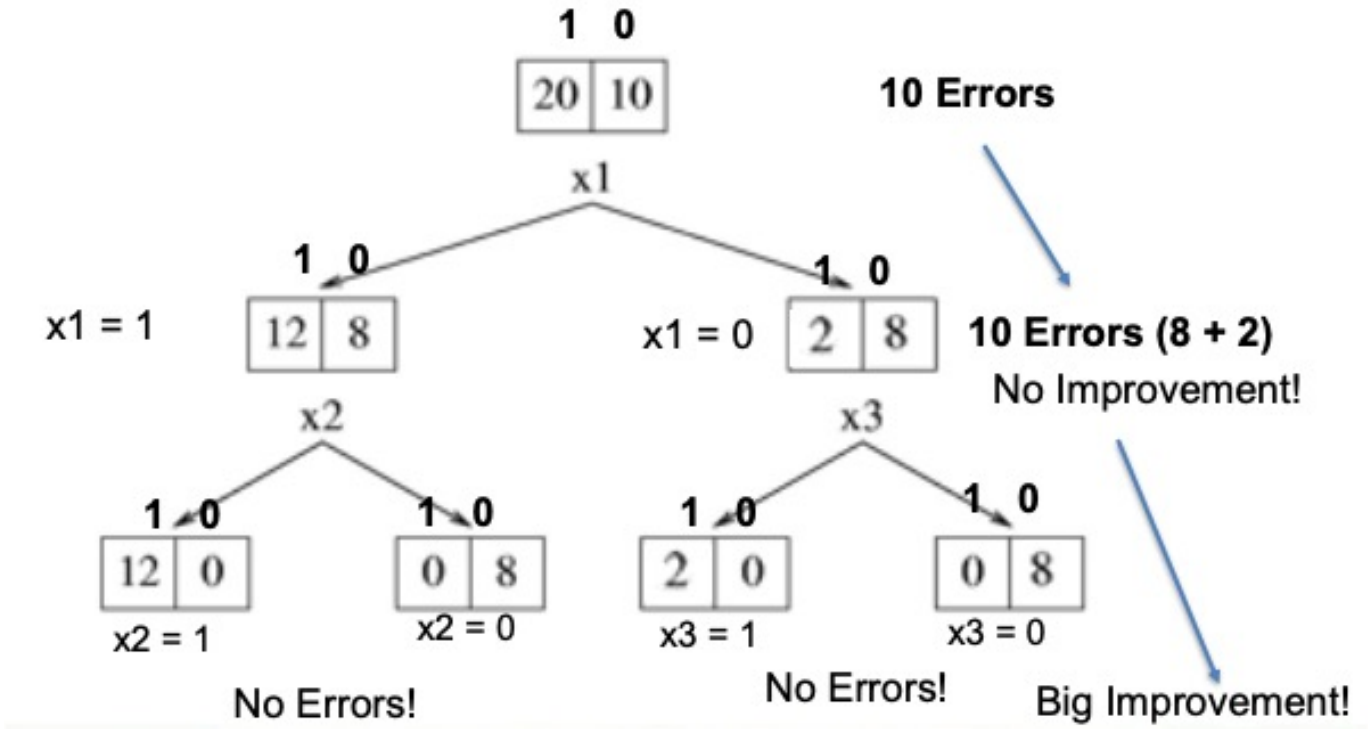


2 Errors
Pick this one!

4 Errors

4 Errors

Choosing the Best Attribute



x1	x2	x3	x4	x5	y
0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	1	0	0
0	0	0	1	1	0
0	0	1	0	0	1
0	0	1	0	1	1
0	1	0	1	0	0
0	1	0	1	1	0
0	1	0	0	0	0
0	1	0	0	1	0
1	0	0	1	0	0
1	0	0	1	1	0
1	0	1	0	0	0
1	0	1	0	1	0
1	0	1	1	0	0
1	0	1	1	1	0
1	0	0	0	0	0
1	0	0	0	1	0
1	1	0	1	0	1
1	1	0	1	1	1
1	1	1	0	0	1
1	1	1	0	1	1
1	1	1	1	0	1
1	1	1	1	1	1
1	1	0	0	0	1
1	1	0	0	1	1
1	1	0	1	0	1
1	1	0	1	1	1
1	1	1	0	0	1
1	1	1	0	1	1

- So even though it looked like choosing attribute x1 was making no progress, it actually was! **Is there a better way to do this?**

How do We Choose the Best Attribute?

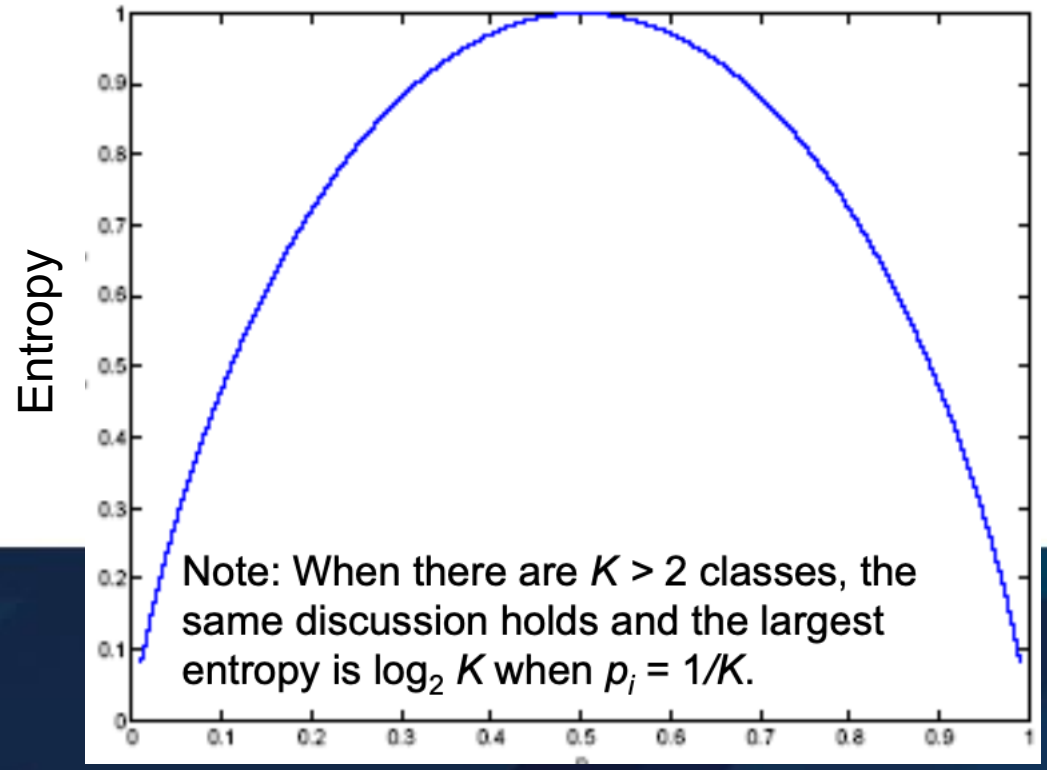
- Let V be a random variable with the following probability distribution:

P(X = 0)	P(X = 1)
0.2	0.8

- The information content or entropy, H(X=x) of each value of X is defined as:
  - H(X=x) = – P(X=x)log P(X=x)
- Ex. Suppose you have a system that follows the probabilities defined above, which of these will have the most “surprise” X = 0 or X = 1
- Ex. Suppose you have a system where P(X=1) =1 and you find X = 1, how surprised are you?
- Ex. Suppose you have a system where P(X=1) =1, and you find X = 0, how surprised are you?
- **We want to maximize the information we are going to gain**

Entropy (K=2)

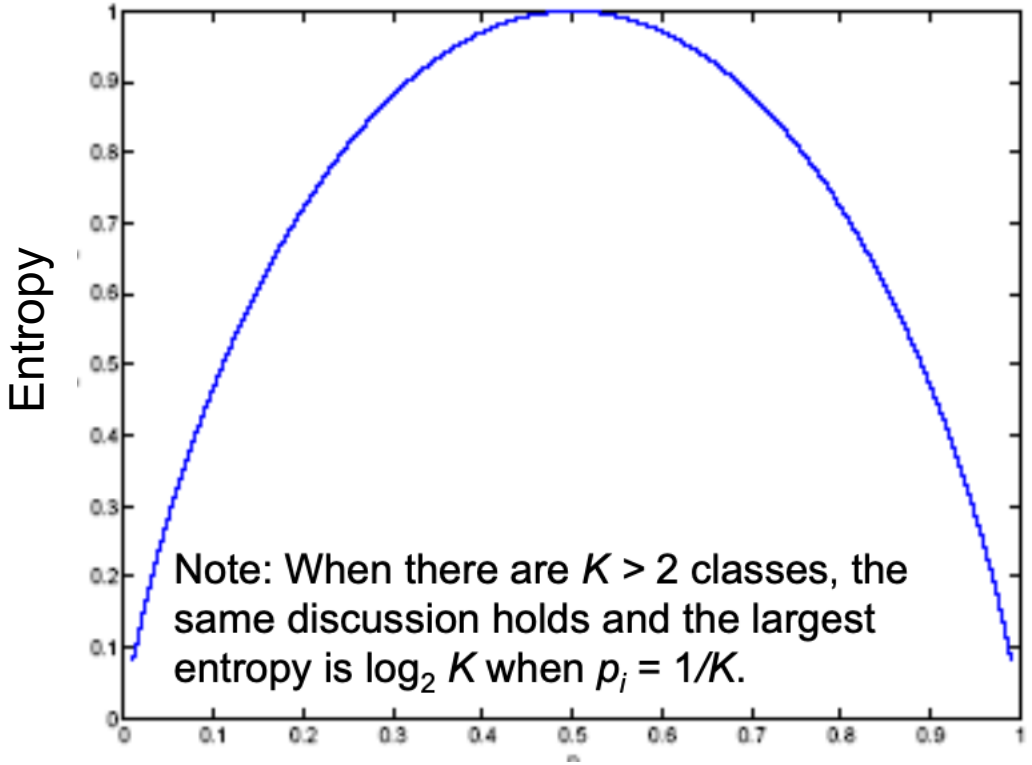
- Entropy related measures are sometimes thought of as the average of “surprise” or a measure of “impurity”
- $I_m = -P(X= x) \log P(X=x) - (1 - p)\log(1 - p)$
  - Note: P(X = x) = p



Classification Trees

- For node m, N_m instances reach m, N^i_m belong to C_i
- Node m is pure if p^i_m is 0 or 1
  Measure of impurity is based on entropy
- Figure to the right is for K=2 classes.
  - Note: When there are K > 2 classes, the same discussion holds and the largest entropy is log2K when pi = 1/K.

$$I_m = -\sum_{i=1}^{K} p^i_m \log_2 p^i_m$$



Best Split

- If node m is pure, generate a leaf and stop, otherwise split and continue recursively.

- Find the variable and split that min impurity (among all variables and split positions for numeric variables).

Other Measures

1. Entropy

$\varphi(p, 1-p) = -p \log_2 p - (1-p) \log_2(1-p)$

Generalization to $K > 2$ classes:

→ $$\mathcal{I}_m = -\sum_{i=1}^{K} p_m^i \log_2 p_m^i$$

2. *Gini index* (Breiman et al. 1984)

$\varphi(p, 1-p) = 2p(1-p)$

3. Misclassification error

$\varphi(p, 1-p) = 1 - \max(p, 1-p)$

Gini Impurity

- A node's gini attribute measures its impurity
- A node is "pure" (gini=0) if all training instances it applies to belong to the same class
- For example, since the depth-1 left node applies only to Iris setosa training instances, it is pure and its gini score is 0

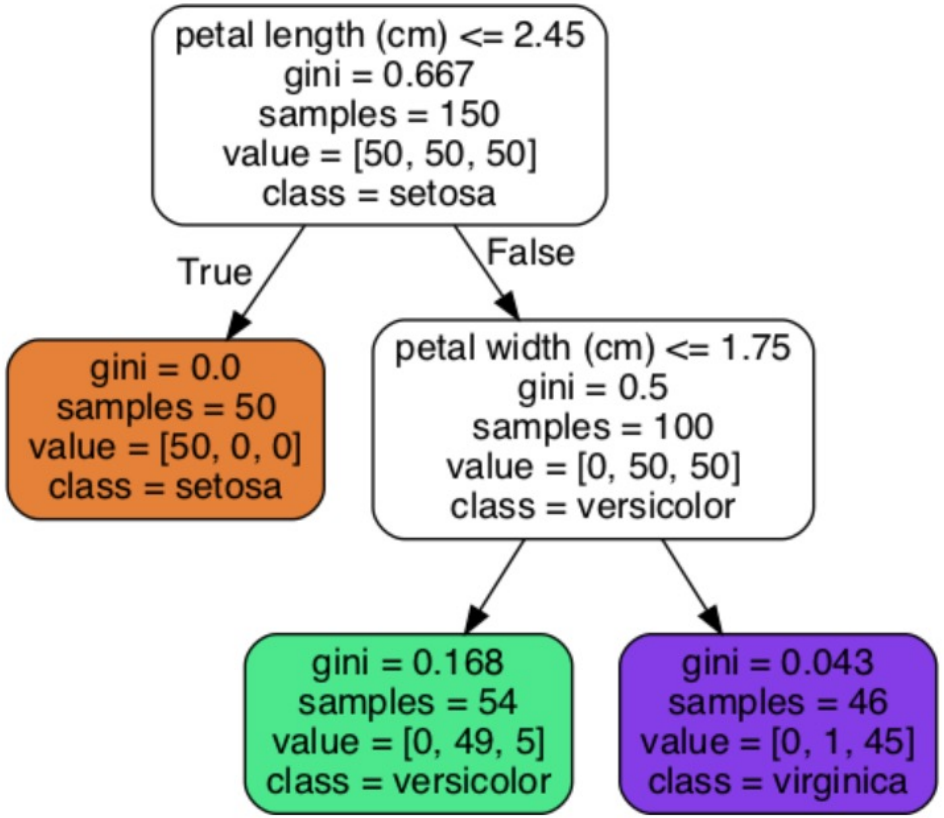
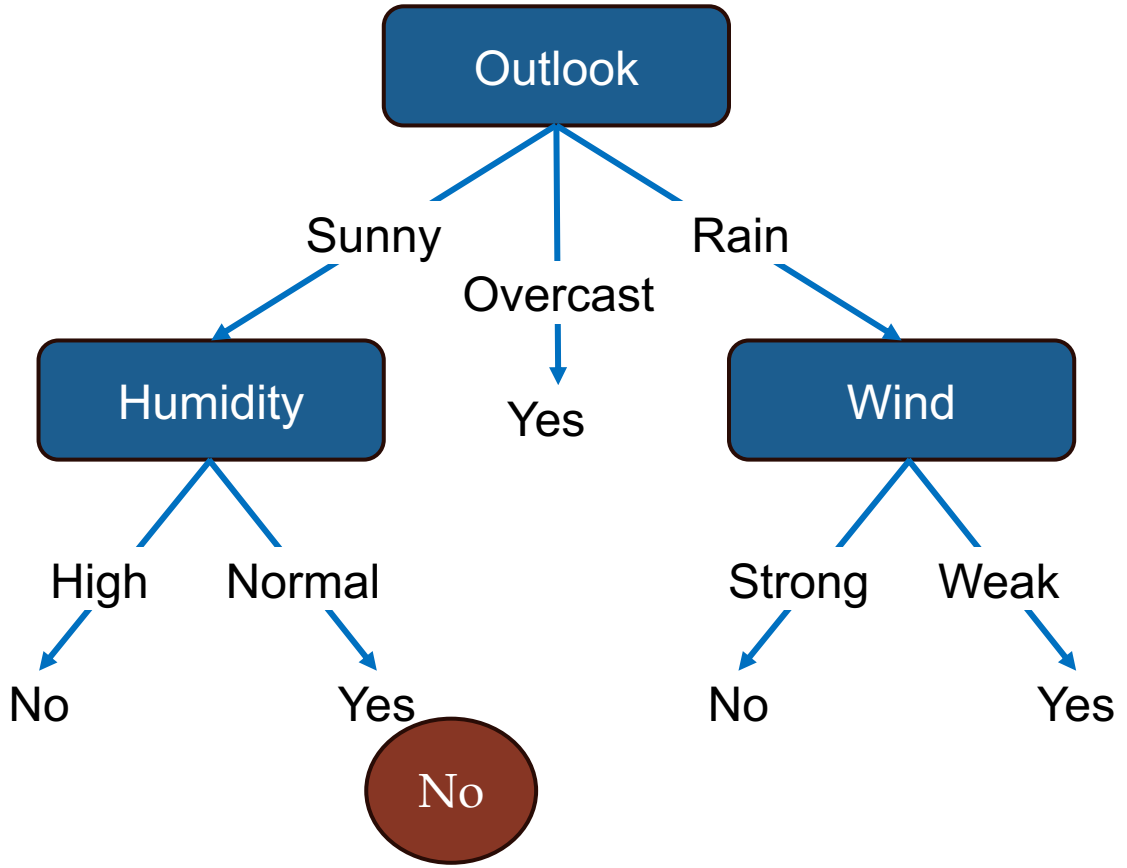


Figure 6-1. Iris Decision Tree

Playing Tennis Example: Overfitting

- **Features:**
  - Outlook
  - Temperature
  - Humidity
  - Wind
- **Noisy example:**
  - Sunny, Hot, and Normal Humidity
  - → Play Tennis = No
    - What is the impact of such data points on the tree?



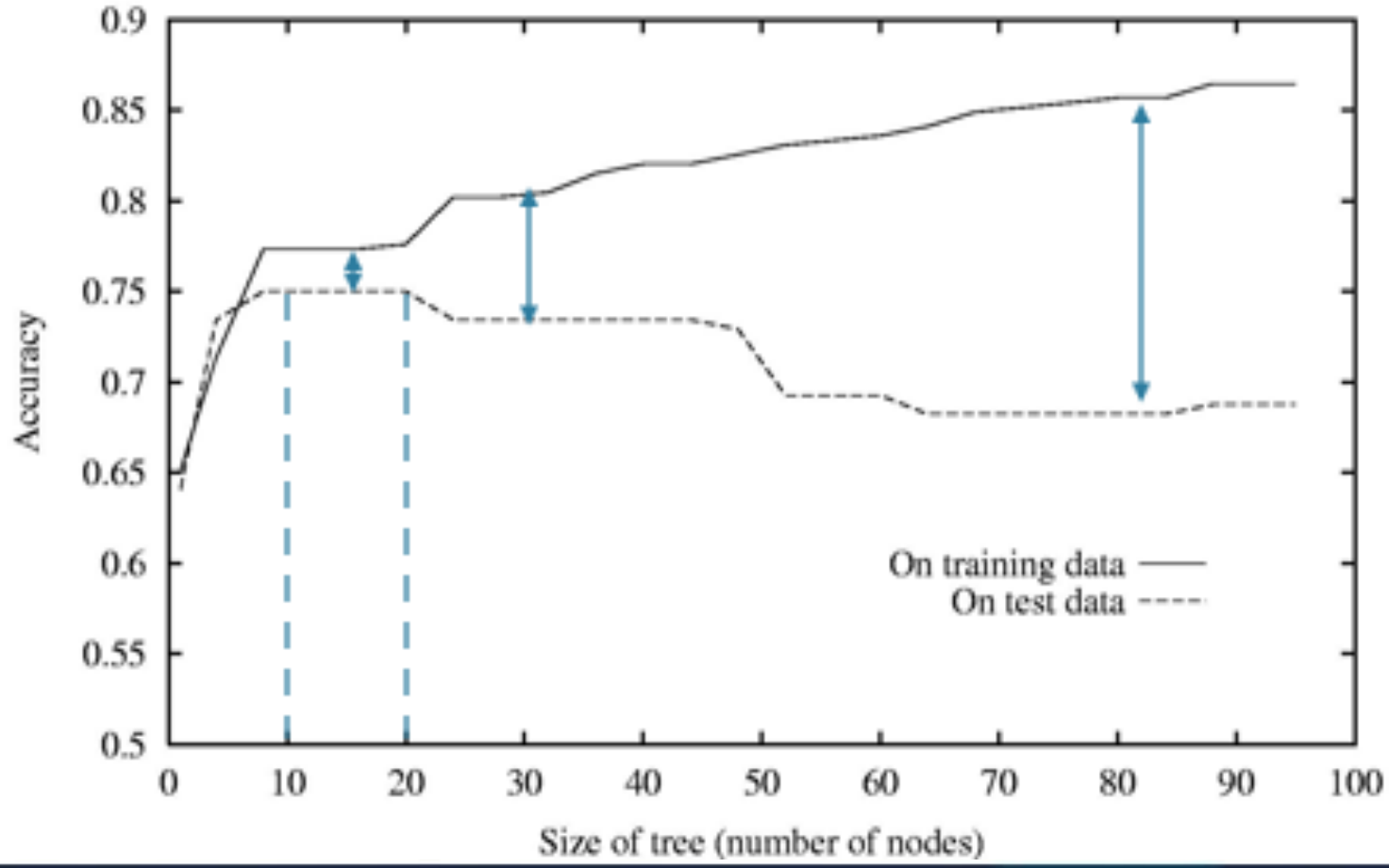
More Formal Definition of Overfitting

- Comparing the error of two hypotheses:
  - A. Error over training data: $error_{train}(h)$
  - B. Error over distribution $\mathcal{D}$ of all data: $error_{\mathcal{D}}(h)$
- **Big challenge in Machine Learning:**

  $\rightarrow$B is what we want, A is what we have!
- Hypothesis h $\in$ H overfits the training data if there is an alternative hypothesis h' $\in$ H such that:
  - $error_{train}(h) < error_{train}(h')$
  - $error_{\mathcal{D}}(h) > error_{\mathcal{D}}(h')$

Overfitting in DT Learning

- Decision Trees are prone to overfitting since they make split decisions that are focused on local optima, rather than global optima.



Mitchell, 1997

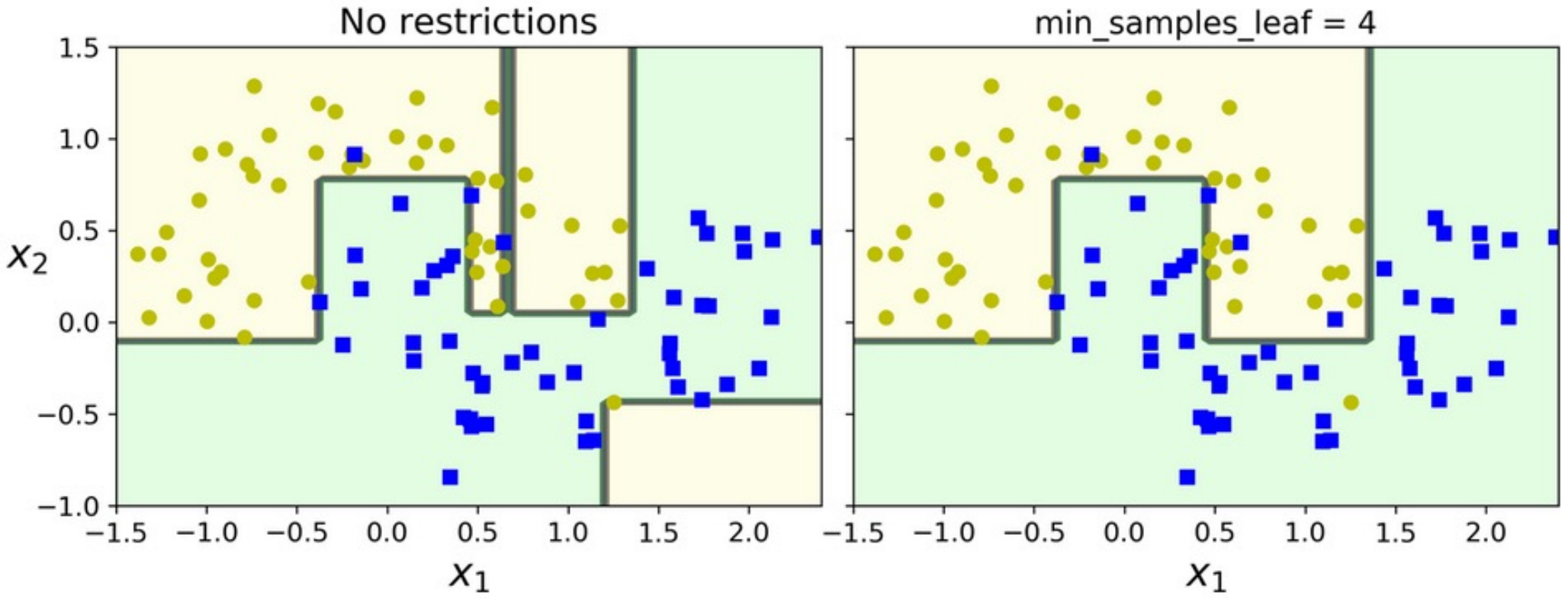


Figure 6-3. Regularization using `min_samples_leaf`

- The model on the left is overfitting, and the model on the right will probably generalize better

Overfitting For Decision Tree Regression

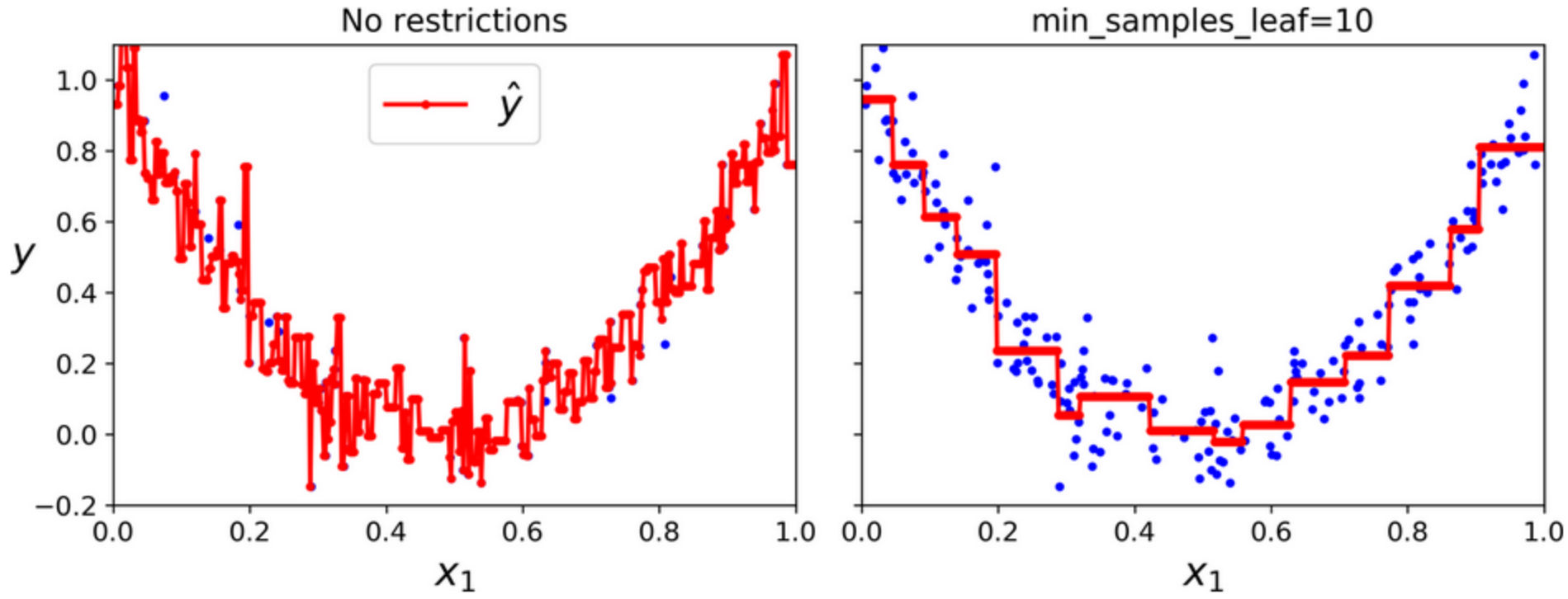


Figure 6-6. Regularizing a Decision Tree regressor

Avoiding Overfitting

- How can we avoid overfitting?
  - Stop growing when data split is not statistically significant (pre-pruning)
  - Grow full tree and then post-prune
- How to select the “best” tree:
  - Measure performance over training data
  - Measure performance over separate validation data set
  - Add a complexity penalty to performance measure

Pruning

- First, training the Decision Tree without restrictions, then pruning (deleting) unnecessary nodes.
- A node whose children are all leaf nodes is considered unnecessary if the purity improvement it provides is not statistically significant.
- Standard statistical tests, such as the $\chi 2$ test (chi-squared test), are used to estimate the probability that the improvement is purely the result of chance (which is called the null hypothesis).
- If this probability, called the p-value, is higher than a given threshold (typically 5%, controlled by a hyperparameter), then the node is considered unnecessary, and its children nodes are deleted. The pruning continues until all unnecessary nodes have been pruned.

Pre-Pruning

- Based on statistical significance test
- Stop growing the tree when there is no *statistically significant* association between any attribute and the class at a particular node
- Most popular test: *chi-squared test*
- Only statistically significant attributes are allowed to be selected by the information gain procedure

Early Stopping

- Pre-pruning may stop the growth process prematurely: *early stopping*
- Classic example: XOR/Parity-problem
  - No *individual* attribute exhibits any significant association with the class
  - Structure is only visible in fully expanded tree
- But: XOR-type problems rare in practice
- And: pre-pruning is faster than post-pruning

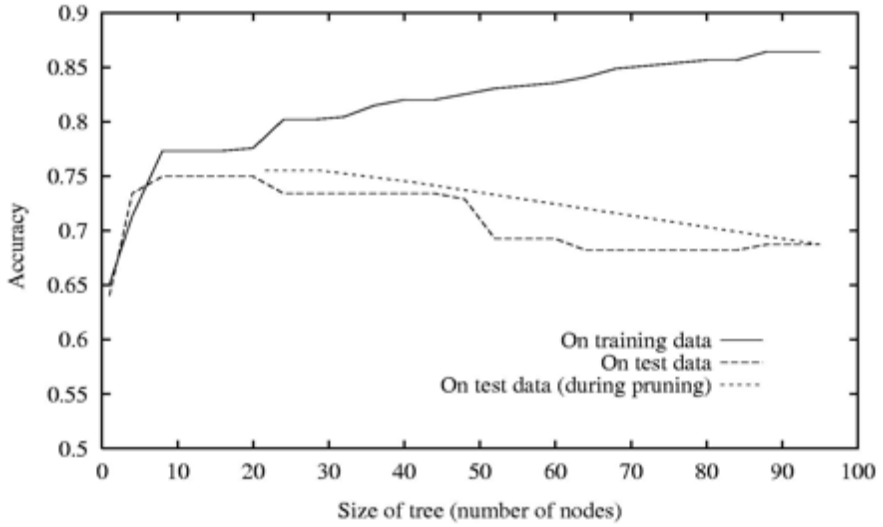
	a	b	class
1	0	0	0
2	0	1	1
3	1	0	1
4	1	1	0

Post-Pruning

- First, training the Decision Tree without restrictions, then pruning (deleting) unnecessary nodes.
- First, build full tree
- Then, prune it
  - Fully-grown tree shows all attribute interactions
- Problem: some subtrees might be due to chance effects
- Two pruning operations:
  - Subtree replacement (replacing a subtree with a leaf node)
  - Subtree raising (moving a subtree up the tree – child node will replace parent node)
- Possible strategies:
  - Error estimation (compare error on validation set before and after pruning)
  - Significance testing (check whether a split genuinely contributes to model accuracy)

Reduced Error Pruning

- Split data into training and validation set
- Do this until further pruning is harmful
- Evaluate the impact on validation set of pruning each possible node (plus those below it)
- Greedily remove the one that most improves validation accuracy



Using Decision Trees to Make Predictions

- Start at the root node (depth 0, at the top)
- Is the flower's petal length smaller than 2.45 cm?
  - If it is, then you move down to the root's left child node (depth 1, left).
- In this case, it is a **leaf node** simply look at the predicted class for that node, and the Decision Tree predicts that your flower is an Iris setosa

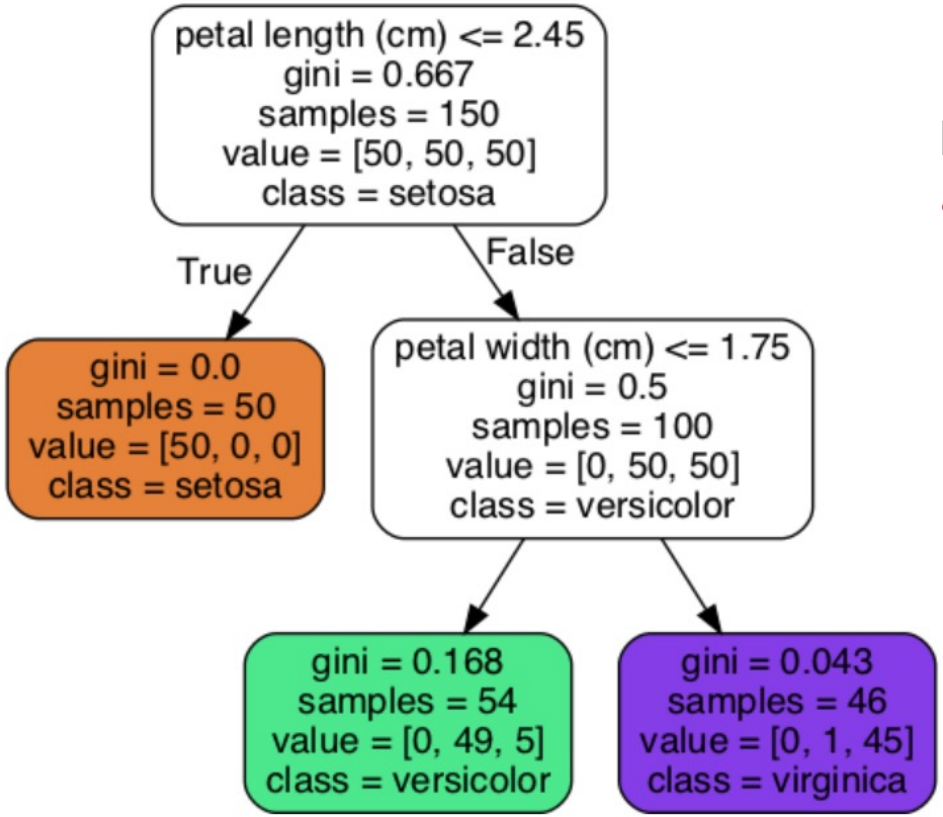


Figure 6-1. Iris Decision Tree

For a different flower with a petal length greater than 2.45 cm:

- Move down to the root's right child node (depth 1, right), which is not a leaf node, so the node asks another question: is the petal width smaller than 1.75 cm?
  - If it is, then your flower is most likely an Iris versicolor (depth 2, left).
  - If not, it is likely an Iris virginica (depth 2, right)

Decision Tree Decision Boundaries

- Thick vertical line represents the decision boundary of the root node (depth 0): petal length = 2.45 cm
- The righthand area is impure
  - The depth-1 right node splits it at petal width = 1.75 cm (represented by the dashed line)
- Since max_depth was set to 2, the Decision Tree stops right there

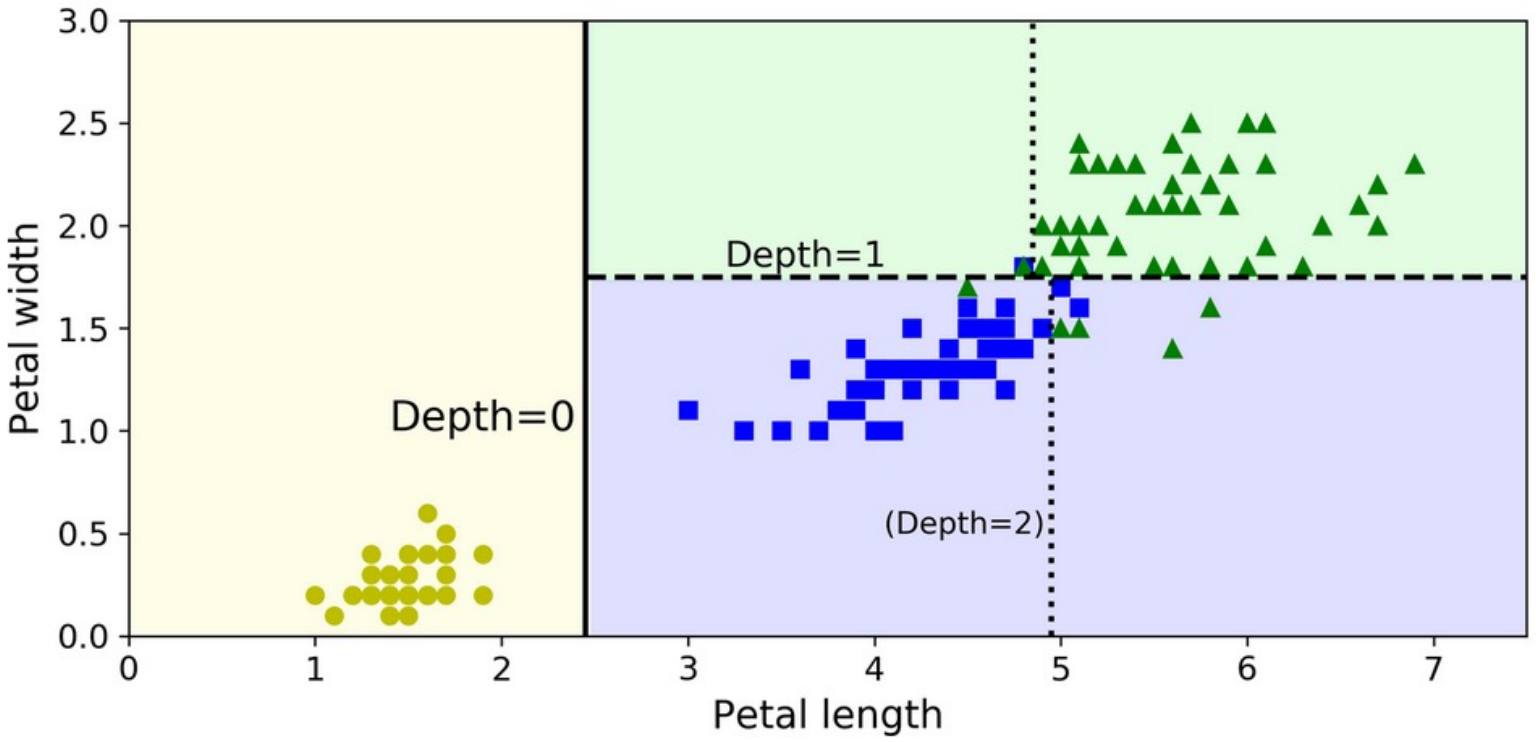


Figure 6-2. Decision Tree decision boundaries

Estimating Class Probabilities

- A Decision Tree can also estimate the probability that an instance belongs to a particular class k.
- First, it traverses the tree to find the leaf node for this instance, and then it returns the ratio of training instances of class k in this node.
- And if you ask it to predict the class, it should output the class with the highest probability.

Regression

- Very similar to a classification
- Instead of predicting a class in each node, it predicts a value.

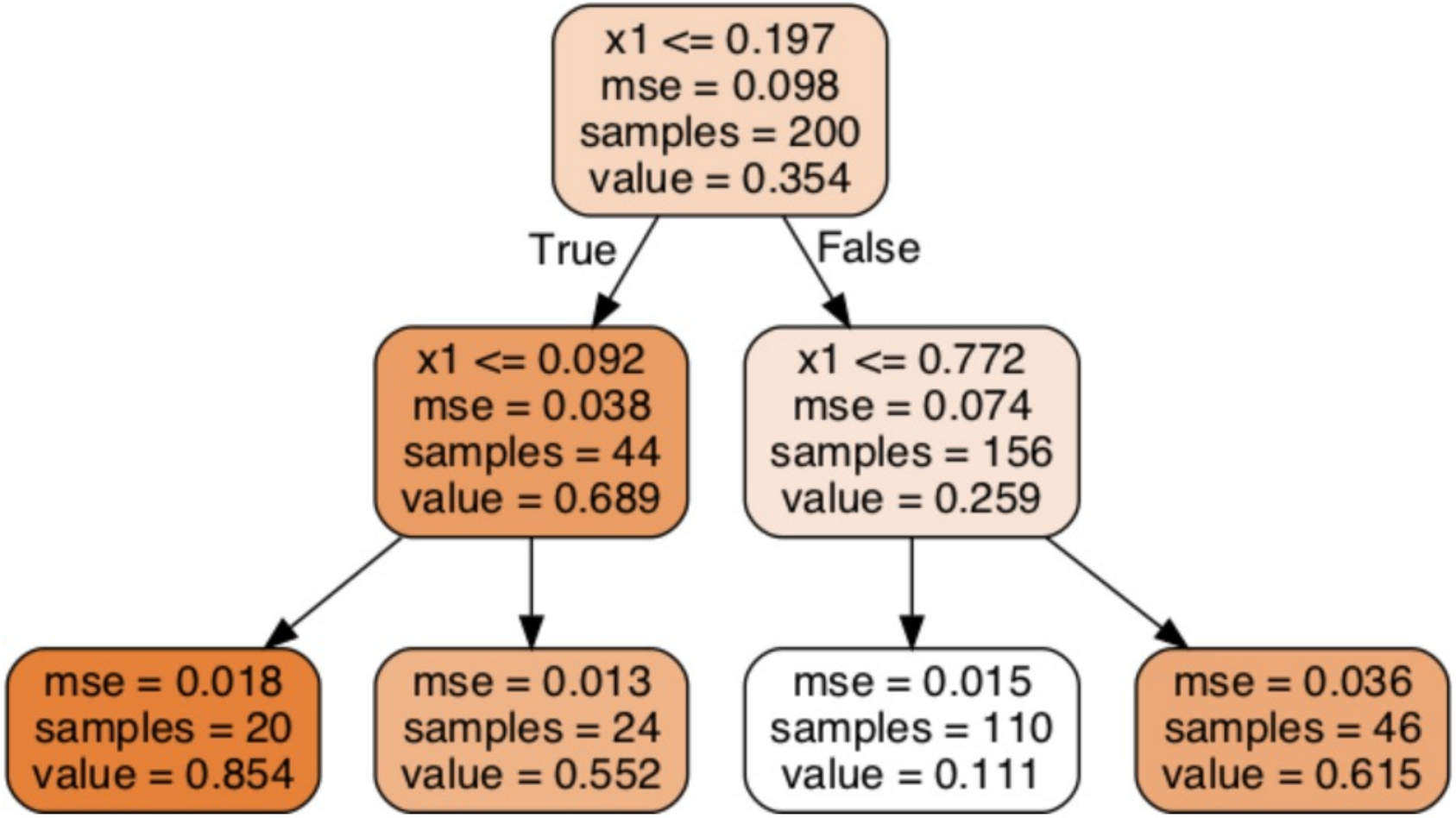


Figure 6-4. A Decision Tree for regression

Using Decision Trees for Regression

- The predicted value for each region is the average target value of the instances in that region.
- The algorithm splits each region in a way that makes most training instances as close as possible to that predicted value.

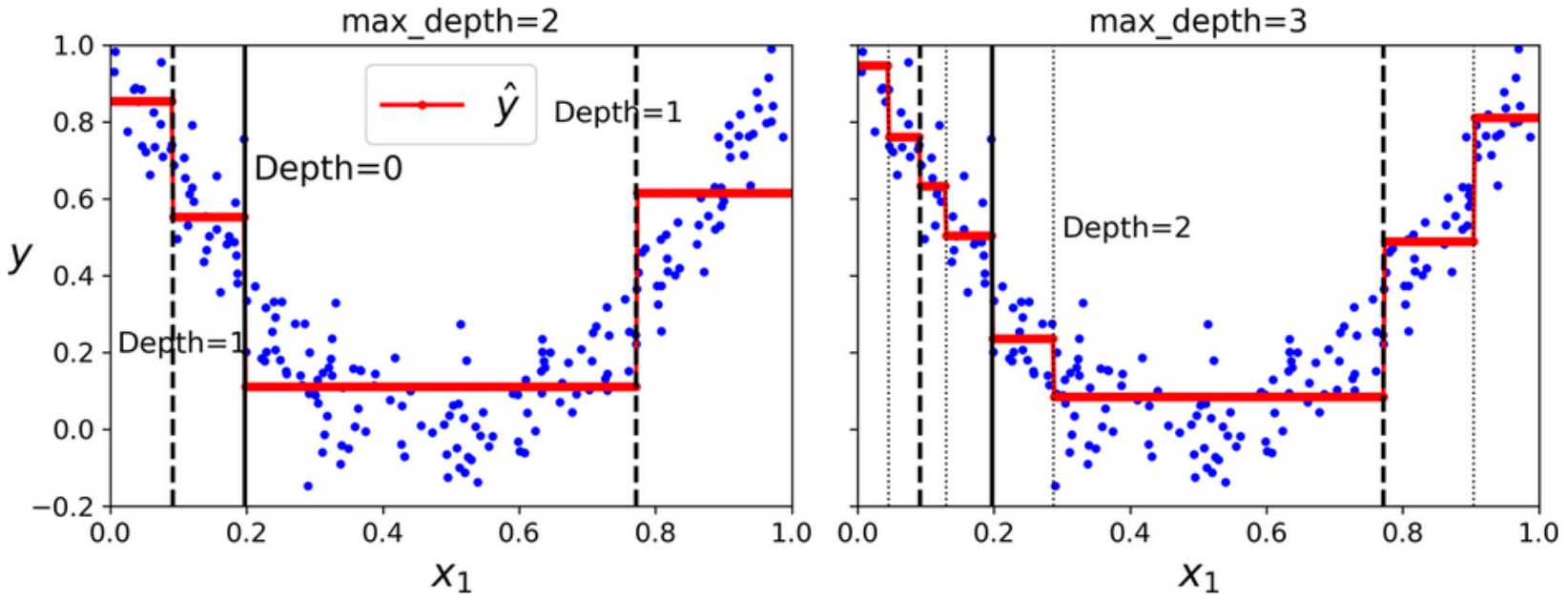


Figure 6-5. Predictions of two Decision Tree regression models

Regularization Hyperparameters

- Decision Trees make very few assumptions about the training data
- Decision trees tend to overfit if they are not constrained
  - The tree structure will adapt itself to the training data, fitting it very closely
- To avoid overfitting to the training data, you need to restrict the Decision Tree's freedom during training (i.e. regularization)
- The regularization hyperparameters depend on the algorithm used, but generally you can at least restrict the maximum depth of the Decision Tree

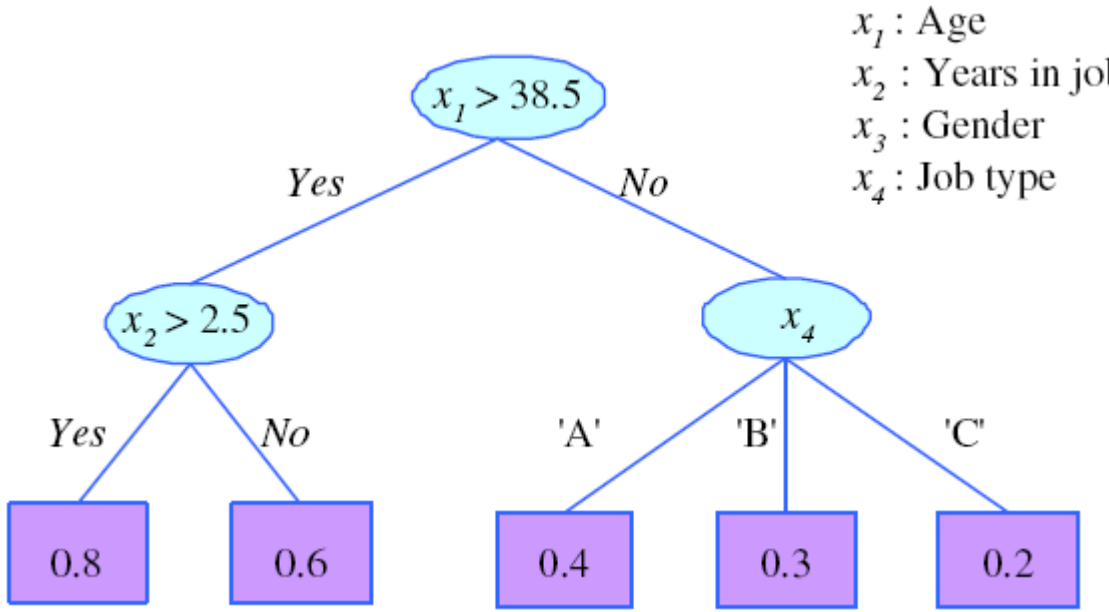
Instability

- Decision trees are simple to understand and interpret, easy to use, versatile, and powerful; however, they do have a few limitations:
  - Decision Trees love orthogonal decision boundaries (all splits are perpendicular to an axis), which makes them sensitive to training set rotation
  - Decision Trees are very sensitive to small variations in the training data, and those changes may generate very different models on the same training data (i.e. decision trees that look different)
- This instability is a motivation for random forests (covered later in the course)

Model Interpretation: White Box Versus Black Box

- Decision Trees are intuitive, and their decisions are easy to interpret
- Intuitive and easy to interpret models are often called white box models
- Black box models are usually hard to explain in simple terms why they performed the way they did (e.g. Random Forests and neural networks)
  - Random Forests and neural networks make great predictions, and you can easily check the calculations that they performed to make these predictions; nevertheless, it is usually hard to explain in simple terms why the predictions were made.
  - For example, if a neural network says that a particular person appears on a picture, it is hard to know what contributed to this prediction: did the model recognize that person's eyes? Their mouth? Their nose? Their shoes? Or even the couch that they were sitting on? Conversely, Decision Trees provide nice, simple classification rules that can even be applied manually if need be (e.g., for flower classification).

Rule Extraction from Trees



R1: IF (age>38.5) AND (years-in-job>2.5) THEN y =0.8
R2: IF (age>38.5) AND (years-in-job≤2.5) THEN y =0.6
R3: IF (age≤38.5) AND (job-type='A') THEN y =0.4
R4: IF (age≤38.5) AND (job-type='B') THEN y =0.3
R5: IF (age≤38.5) AND (job-type='C') THEN y =0.2

Quinlan, 1993

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

Homework Overview

Professor Hamza F. Alsarhan
SEAS 8505

This Week

- Reading:
  - Chapters 4 and 6 in the textbook
- HW #2 (Run Python Script in Google Colab first, and then answer the homework questions)
- Discussion #2
- Start getting familiar with TensorFlow and Google Colab

- Reminder: No extensions provided. Start assignments early!

Next Steps

- Come to office hours with any questions you may have.
- Work on your HW and Discussion and submit them by 9:00 am ET on Saturday.
- See you next class!

Thank you!