Northeastern University
Khoury College of
Computer Sciences

Classification

DS 4400 | Machine Learning and Data Mining I

Zohair Shafi

Spring 2026

Wednesday | January 28, 2026

Recap Continuation

Gradient Descent

Batch vs Mini-Batch vs Stochastic Gradient Descent

Gradient Descent

Batch vs Mini-Batch vs Stochastic Gradient Descent

- Batch Gradient Descent
  - Use **entire training set per epoch**
  - The whole training dataset is used to compute a single parameter update

$$\theta_t = \theta_{t-1} - \alpha \frac{1}{m} \sum_{i=1}^{m} \nabla \ell_{\theta_{t-1}}(x_i, y_i)$$

Gradient Descent

Batch vs Mini-Batch vs Stochastic Gradient Descent

- Batch Gradient Descent
  - Use **entire training set per epoch**
  - The whole training dataset is used to compute a single parameter update
  - One epoch leads to **one** parameter update

$$\theta_t = \theta_{t-1} - \alpha \frac{1}{m} \underbrace{\sum_{i=1}^{m}}_{\text{Sum over the whole training dataset}} \nabla \ell_{\theta_{t-1}}(x_i, y_i)$$

Gradient Descent

Batch vs Mini-Batch vs Stochastic Gradient Descent

- Stochastic Gradient Descent
  - Use **one** randomly selected training data point at each step
  - Parameters are updated after looking at each data point
  - One epoch leads to **m** parameter updates

$$\theta_t = \theta_{t-1} - \alpha \nabla \ell_{\theta_{t-1}}(x_i, y_i)$$

Train / Test Splits

- Generally data is split into a training dataset and a testing data
- Rough rule of thumb is that this is an 80-20 split

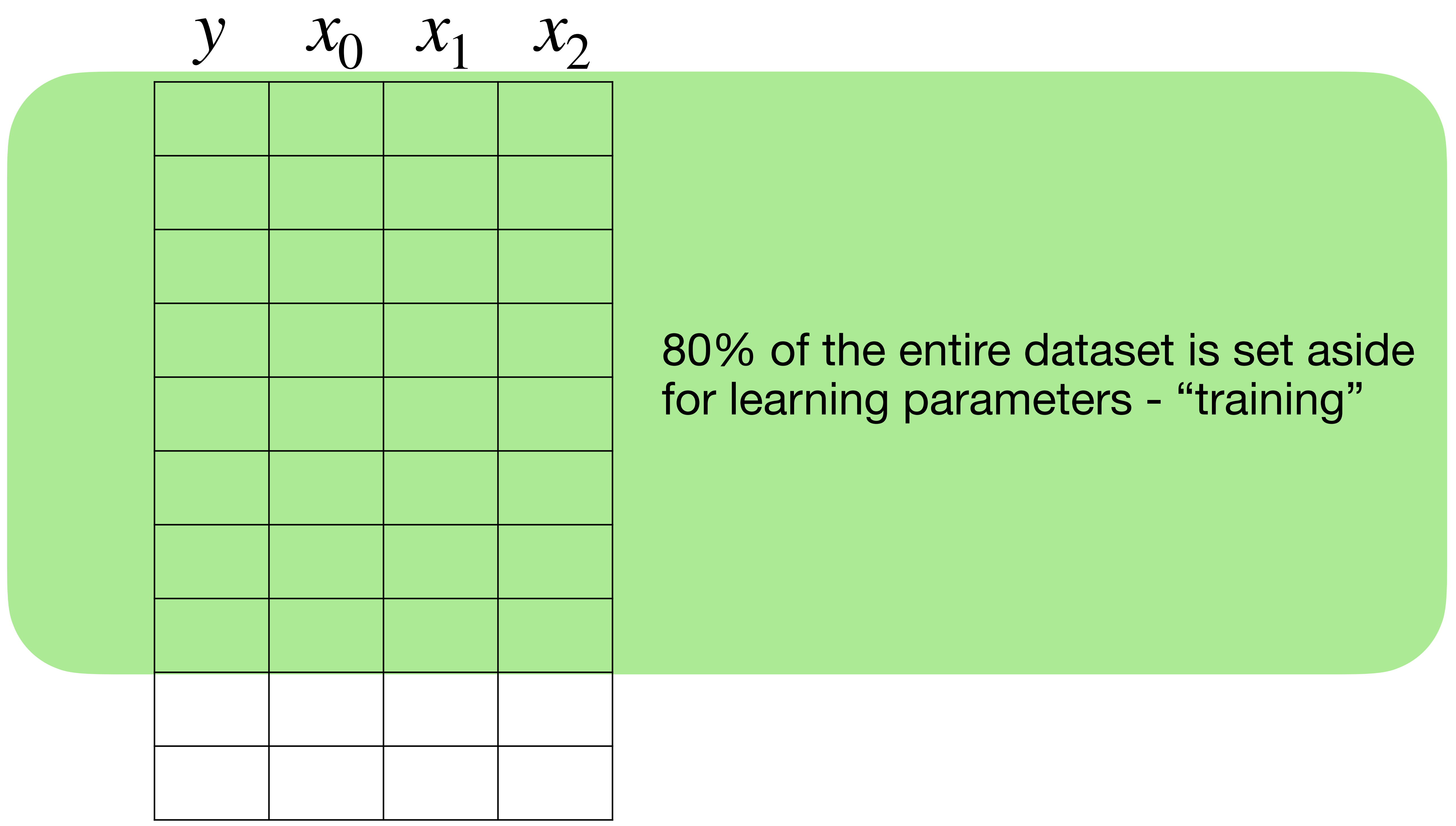
Train / Test Splits

- Generally data is split into a training dataset and a testing data
- Rough rule of thumb is that this is an 80-20 split

y	x_0	x_1	x_2

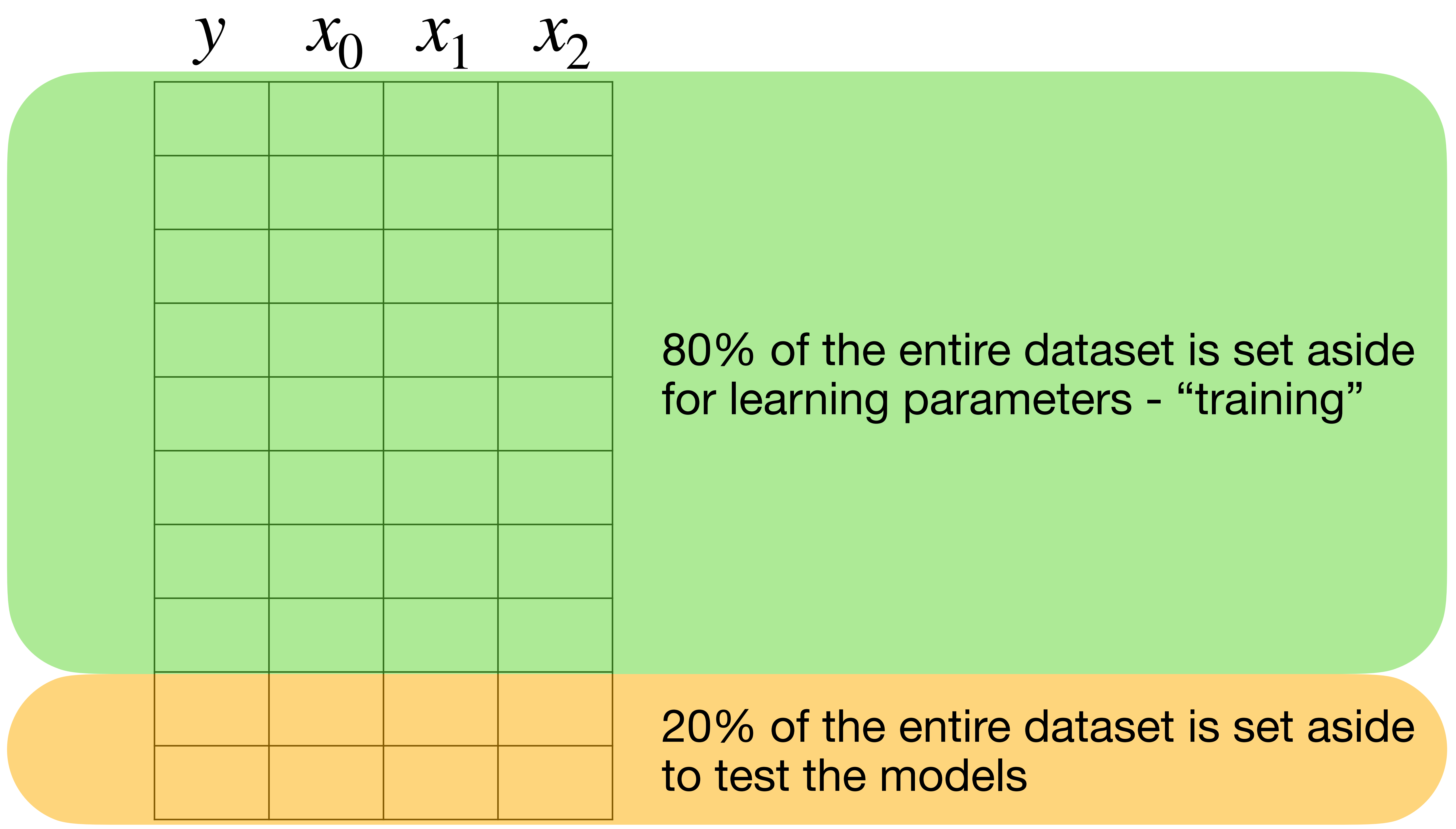
Train / Test Splits

- Generally data is split into a training dataset and a testing data
- Rough rule of thumb is that this is an 80-20 split


y
x_0
x_1
x_2
80% of the entire dataset is set aside for learning parameters - “training”

Train / Test Splits

- Generally data is split into a training dataset and a testing data
- Rough rule of thumb is that this is an 80-20 split



80% of the entire dataset is set aside for learning parameters - “training”

20% of the entire dataset is set aside to test the models

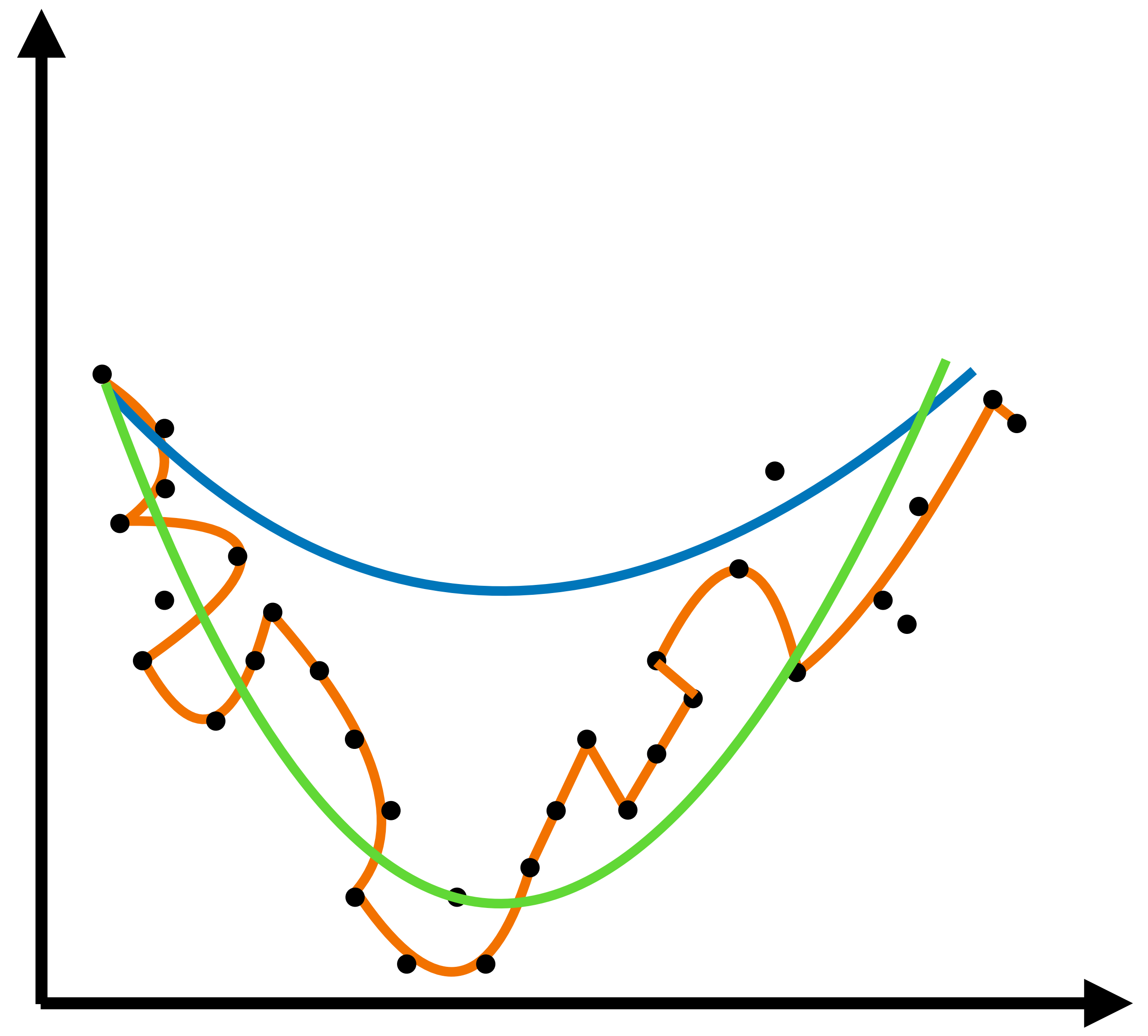
This is **unseen** data and tells you if the model can generalize well

Train / Test Splits

- However, in practice, if you are given only one train and test set, its easy to accidentally pick model architectures that work well on the test set, even though test set data is unseen
- To counter this, we use two unseen datasets - “validation” set and “test” set
- The split is generally of the form 80-10-10 where 80% is training data, 10% is validation data and 10% is test data

Practical Issues in Linear Regression

Overfitting vs Underfitting



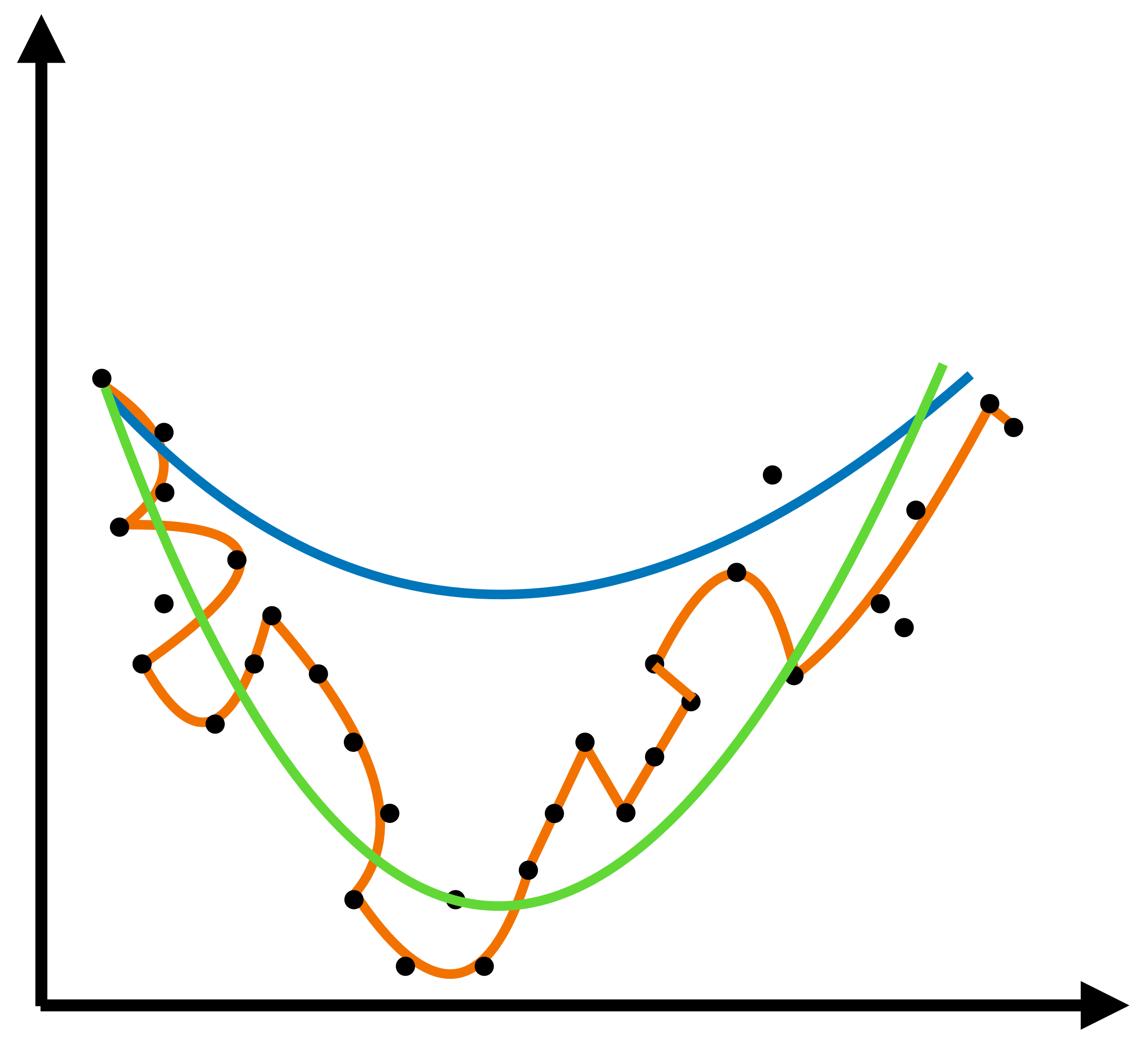
Practical Issues in Linear Regression

Overfitting vs Underfitting

The blue model is **underfitting** the data

The orange model is **overfitting** the data

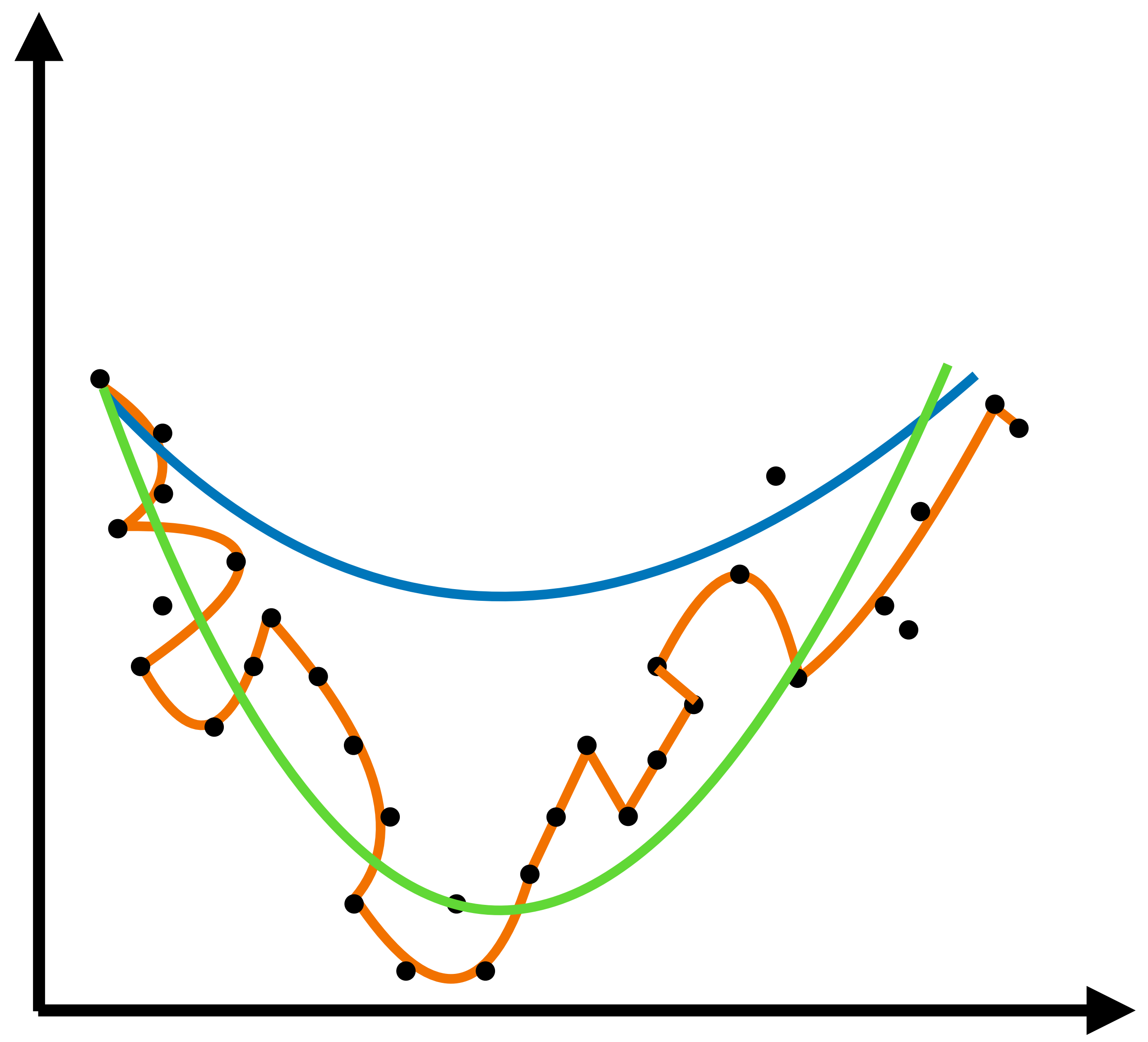
The green model is a good fit of the data



Practical Issues in Linear Regression

Underfitting

- What is happening?
  - The model is too simple to be able to capture the data
- How do you identify it?
  - Training loss is **high**
  - Test loss is **high**
- Solutions
  - Add more features
  - Add polynomial features $(x_1^2, x_2^2, x_1x_2, \cdots)$
  - Use a more complex model

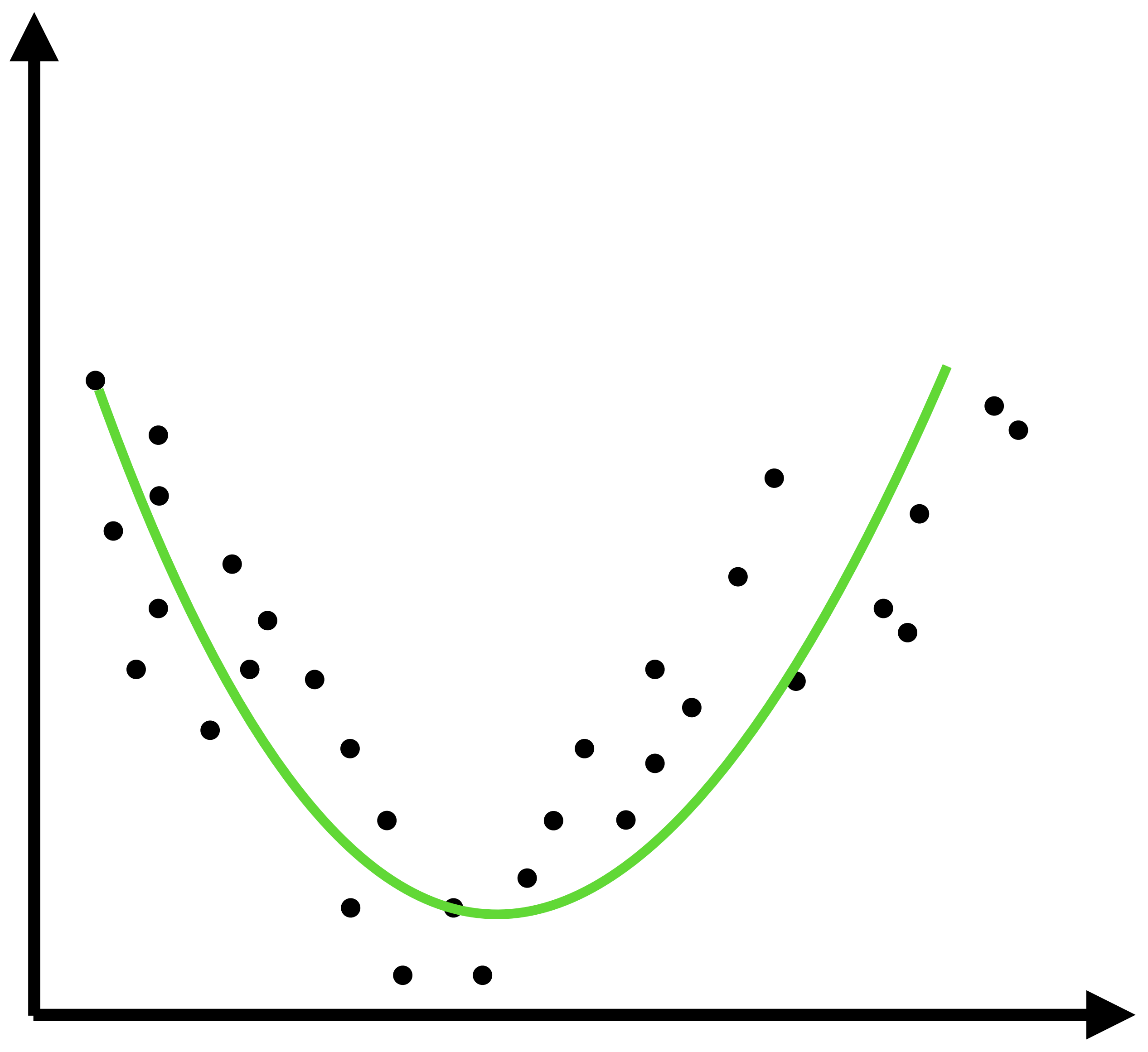


Practical Issues in Linear Regression

Quick Aside

- Add polynomial features $(x_1^2, x_2^2, x_1x_2, \cdots)$

$$f_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2$$

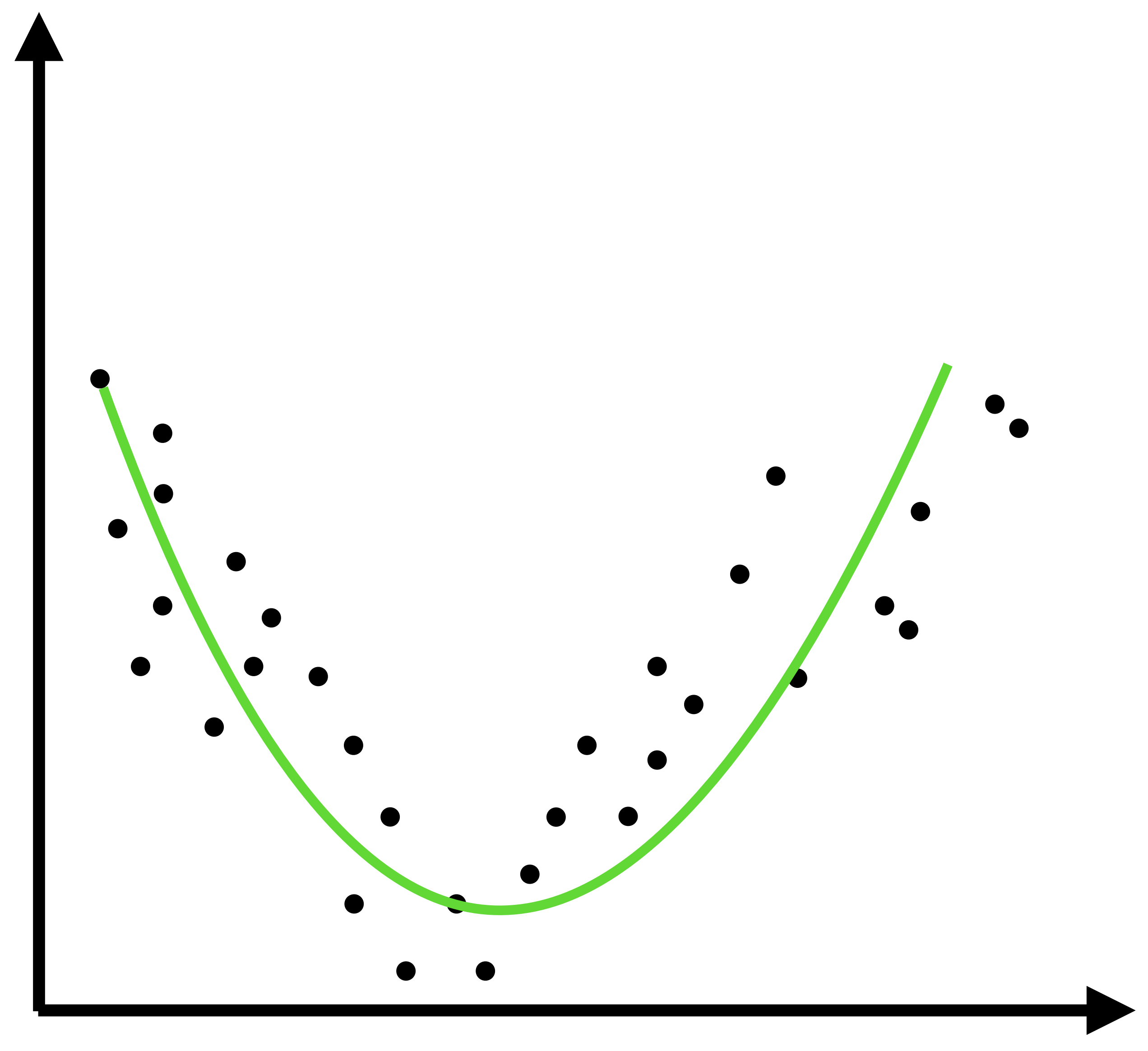


Practical Issues in Linear Regression

Quick Aside

- Add polynomial features $(x_1^2, x_2^2, x_1 x_2, \cdots)$

$$f_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2$$



What about these models?

$$f_\theta(x) = \theta_0^{x_0} + \theta_1^{x_1}$$

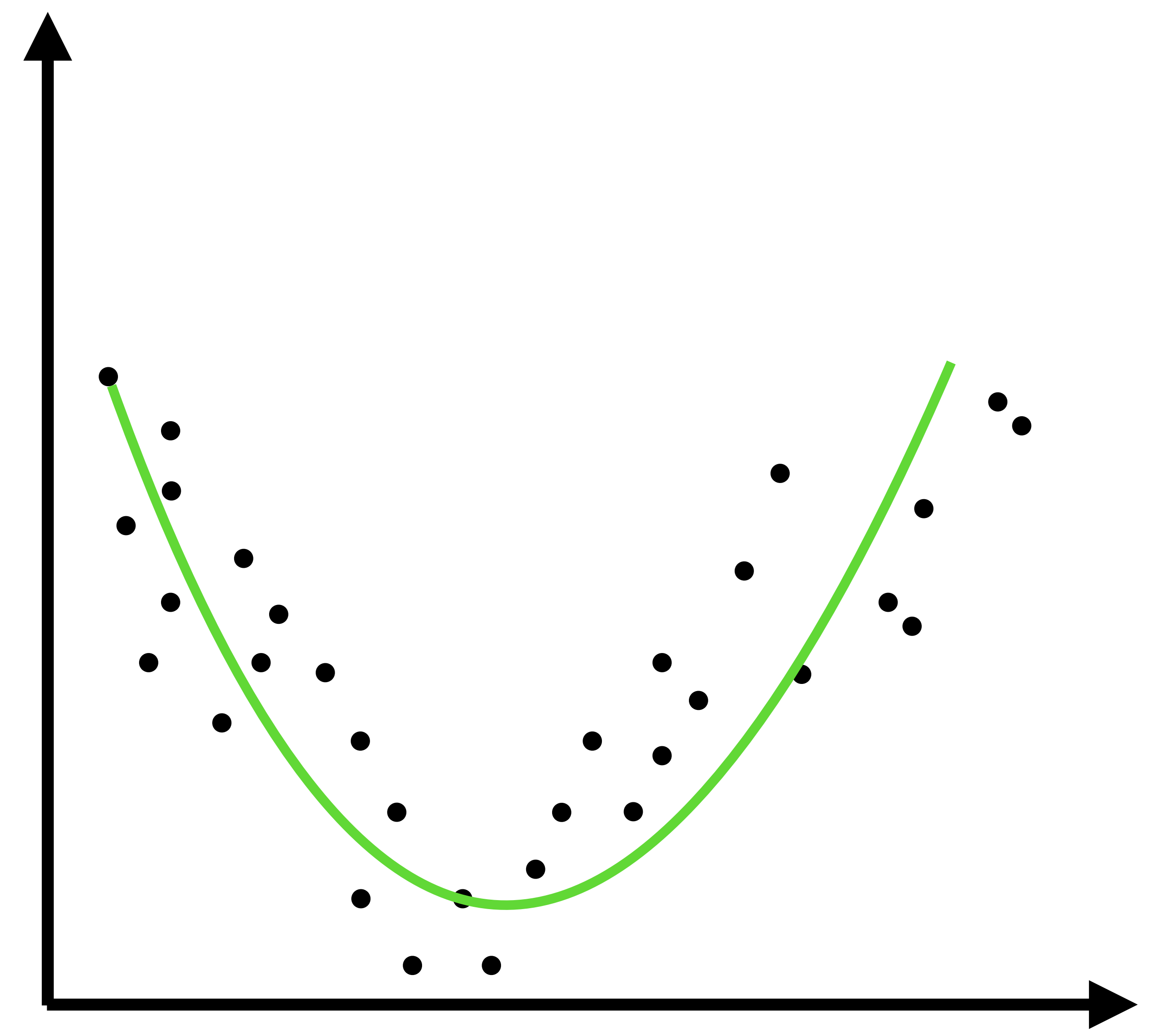
$$f_\theta(x) = x_0^{\theta_0} + x_1^{\theta_1}$$

Practical Issues in Linear Regression

Quick Aside

- Add polynomial features $(x_1^2, x_2^2, x_1 x_2, \cdots)$

$$f_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2$$



What about these models?

$$f_\theta(x) = \theta_0^{x_0} + \theta_1^{x_1}$$

$$f_\theta(x) = x_0^{\theta_0} + x_1^{\theta_1}$$

Linear regression must be of the form

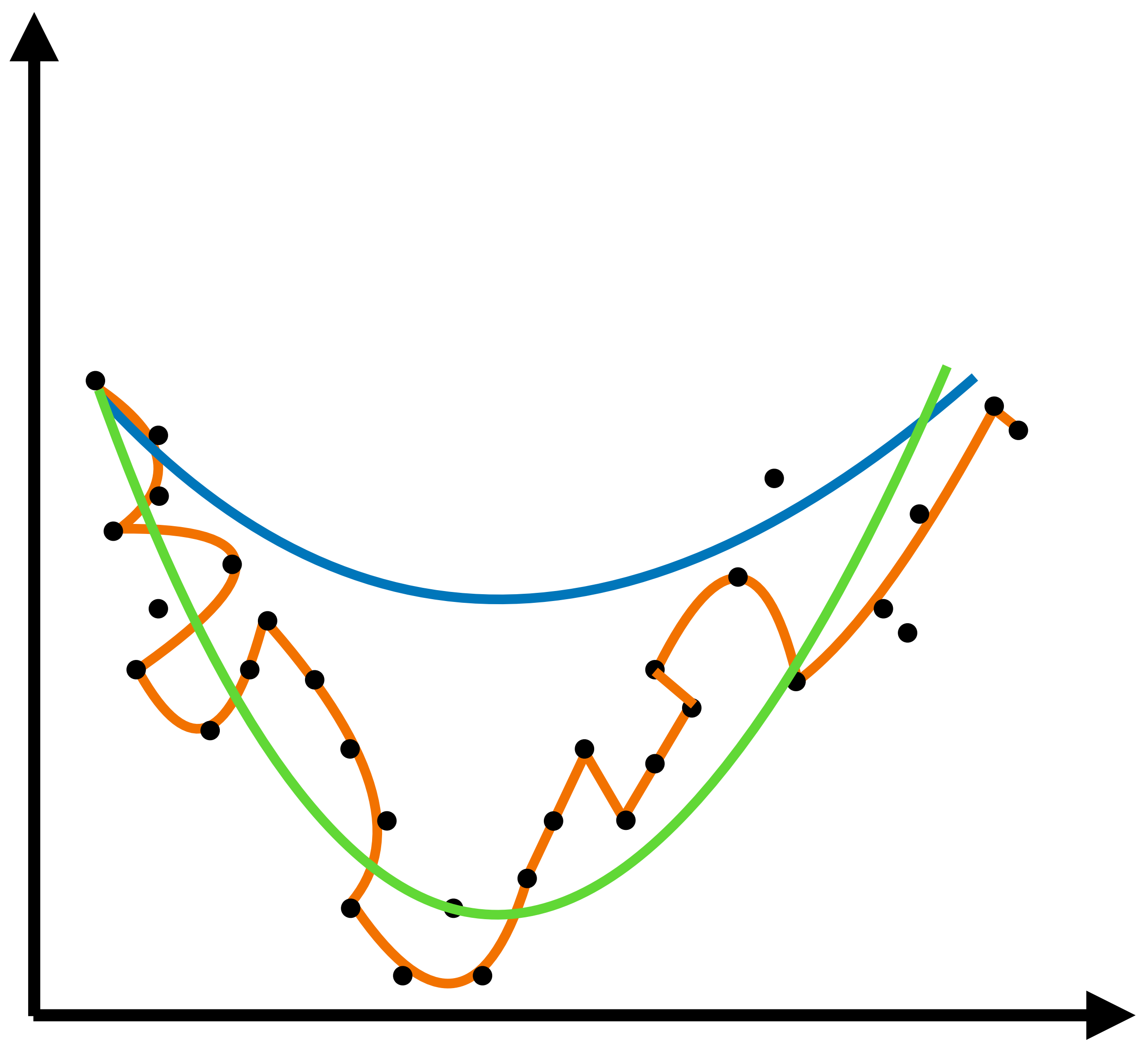
$$f_\theta(x) = \theta_0 \cdot \phi(x_1) + \theta_2 \cdot \phi(x_2)$$

where $\phi(x)$ can be any function

Practical Issues in Linear Regression

Overfitting

- What is happening?
  - The model is too complex, so it learns the noise distribution and outliers and hence does not generalize well to new data points
- How do you identify it?
  - Training loss is **low**
  - Test loss is **high**
  - Coefficients have **large** magnitudes
- Solutions
  - Regularization (L_1, L_2)
  - Cross-validation for model selection
  - Reduce number of features
  - Get more training data



Practical Issues in Linear Regression

A more mathematical look - Bias / Variance Tradeoff

Every model's prediction error/loss can be decomposed into three parts:

$$\text{Expected Loss} = \text{Bias}^2 + \text{Variance} + \text{Irreducible Noise}$$

Practical Issues in Linear Regression

A more mathematical look - Bias / Variance Tradeoff

Every model's prediction error/loss can be decomposed into three parts:

$$\text{Expected Loss} = \text{Bias}^2 + \text{Variance} + \text{Irreducible Noise}$$

Error from wrong assumptions due to the model being too simple

Practical Issues in Linear Regression

A more mathematical look - Bias / Variance Tradeoff

Every model's prediction error/loss can be decomposed into three parts:

$$\text{Expected Loss} = \text{Bias}^2 + \text{Variance} + \text{Irreducible Noise}$$

Error from high sensitivity to each data point and noise due to the model being too complex

Practical Issues in Linear Regression

A more mathematical look - Bias / Variance Tradeoff

Every model's prediction error/loss can be decomposed into three parts:

$$\text{Expected Loss} = \text{Bias}^2 + \text{Variance} + \text{Irreducible Noise}$$

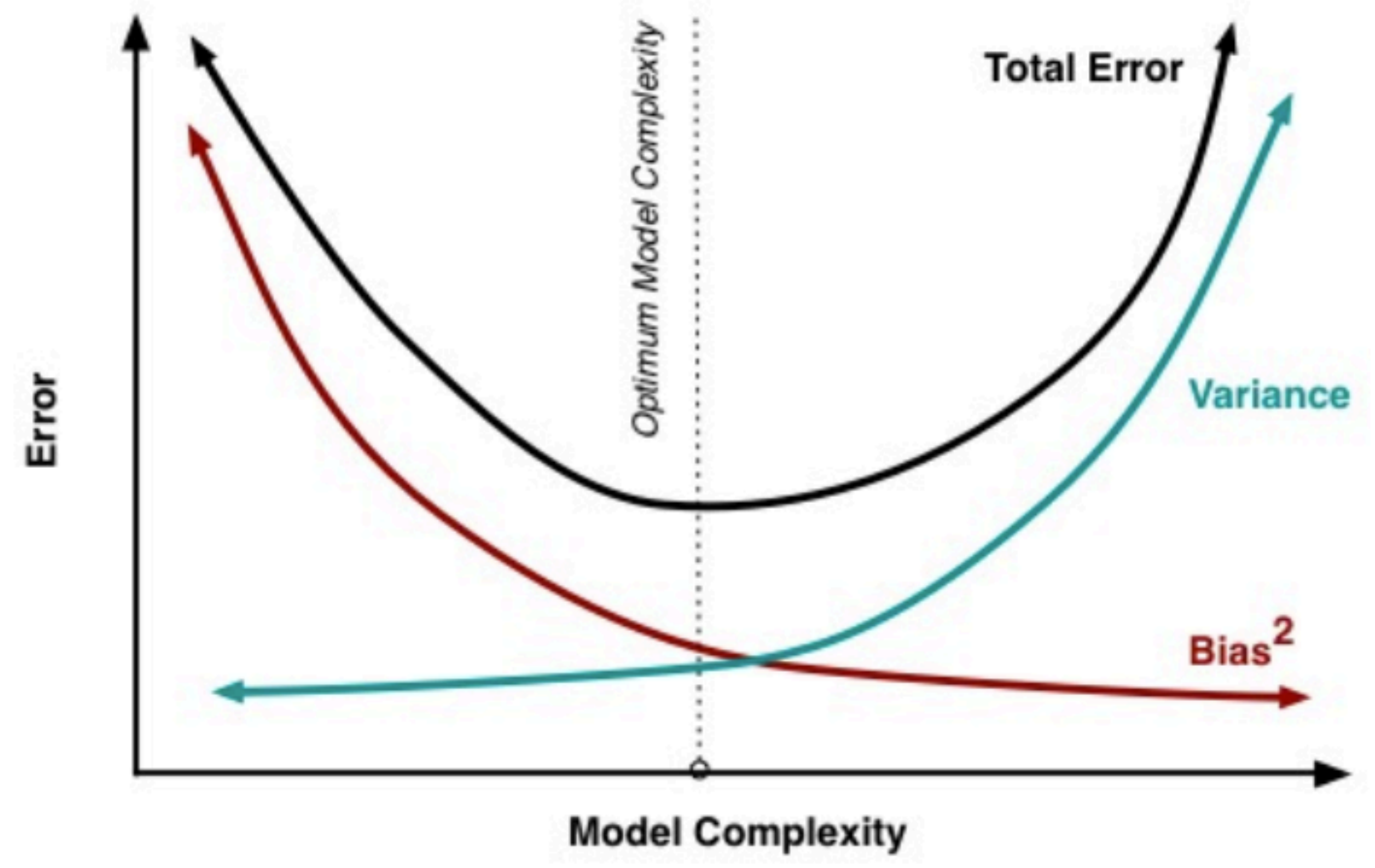
Inherent randomness in data. Cannot be removed.

Practical Issues in Linear Regression

Bias / Variance Tradeoff

Why is it called a **tradeoff**?

Model Complexity	Bias	Variance	Train Error	Test Error
Too Simple	High	Low	High	High
Sweet Spot	Medium	Medium	Medium	Medium
Too Complex	Low	High	Low	High

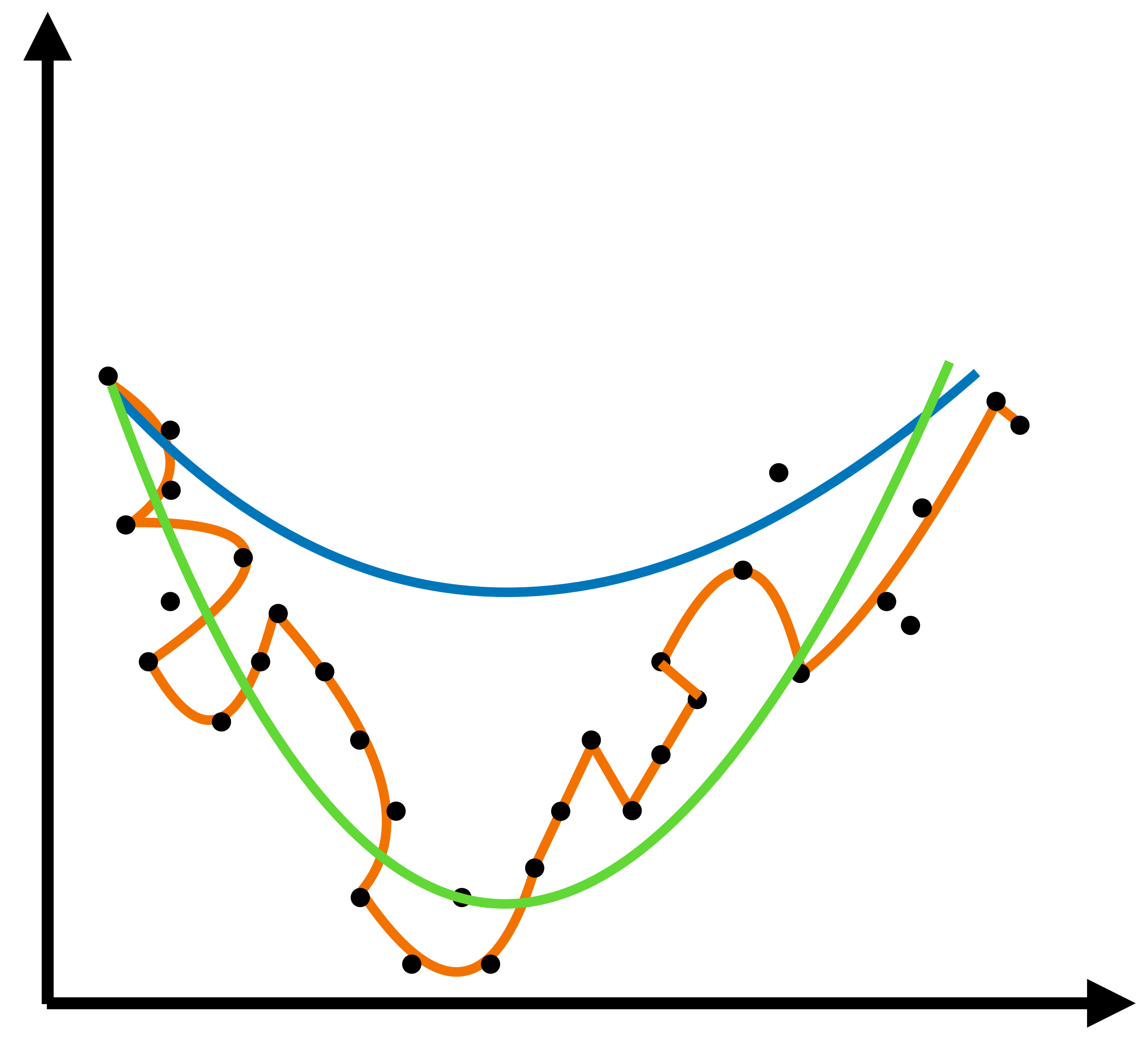


Practical Issues in Linear Regression

Regularization

- Regularization explicitly trades bias for variance.

$$L(\theta) = \frac{1}{m} \sum (Y - X\theta)^2$$



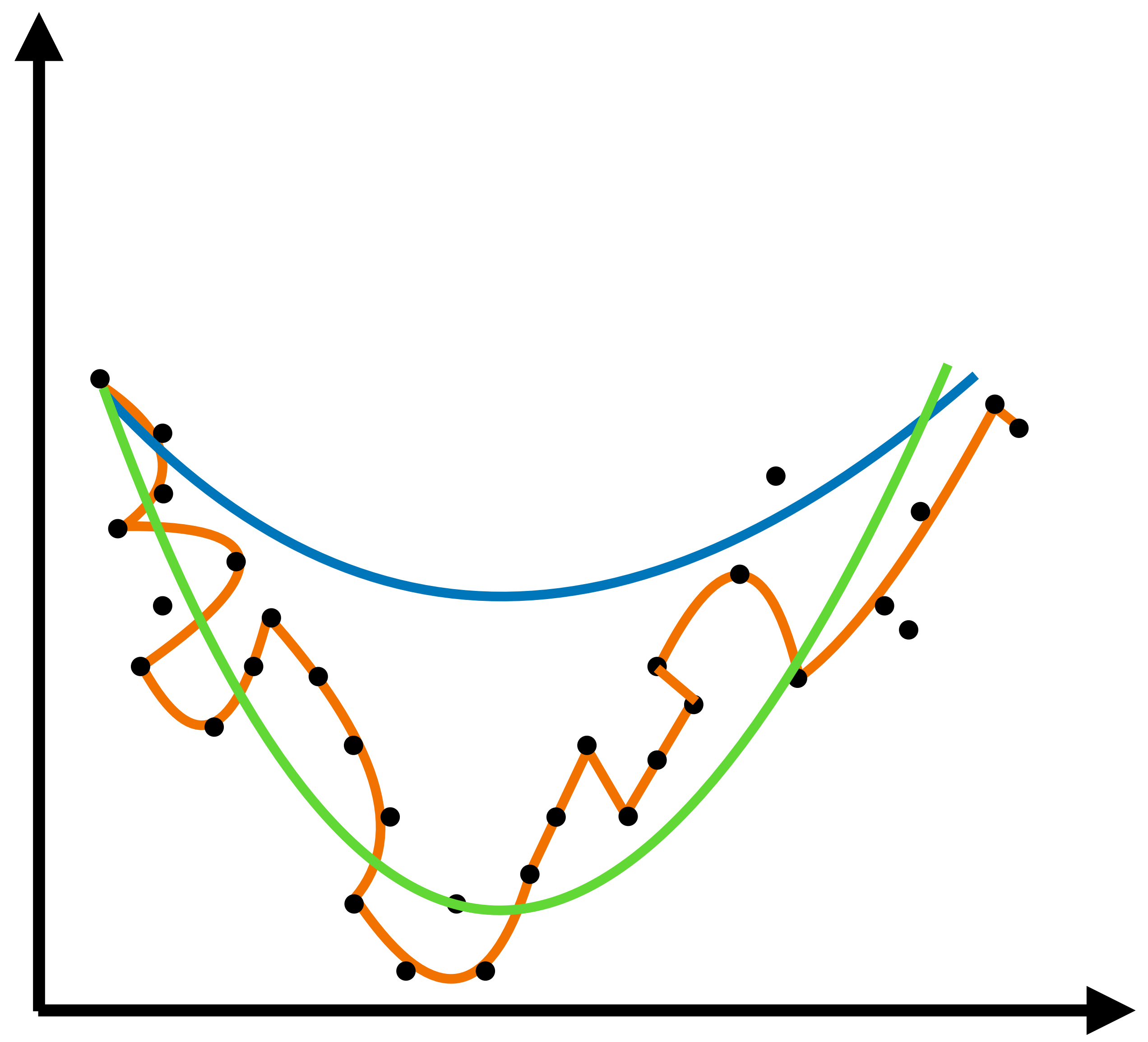
Practical Issues in Linear Regression

Regularization

- Regularization explicitly trades bias for variance.

$$L(\theta) = \frac{1}{m}\sum (Y - X\theta)^2$$

$$L(\theta) = \frac{1}{m}\sum (Y - X\theta)^2 + \lambda ||\theta||^2$$



Practical Issues in Linear Regression

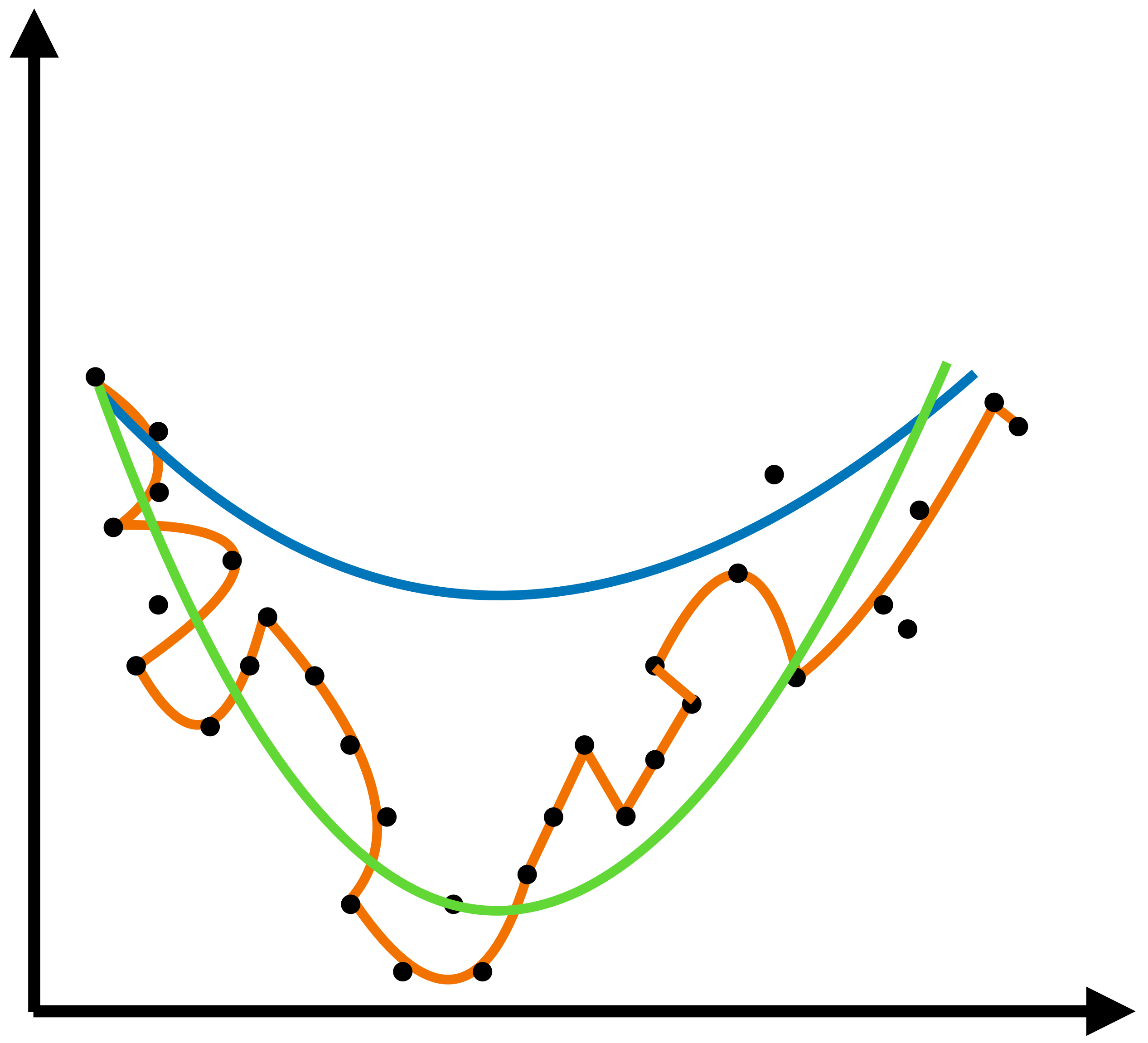
Regularization

- Regularization explicitly trades bias for variance.

$$L(\theta) = \frac{1}{m} \sum (Y - X\theta)^2$$

$$L(\theta) = \frac{1}{m} \sum (Y - X\theta)^2 + \lambda ||\theta||^2$$

$$\theta = (X^T X + \lambda I)^{-1} X^T Y$$



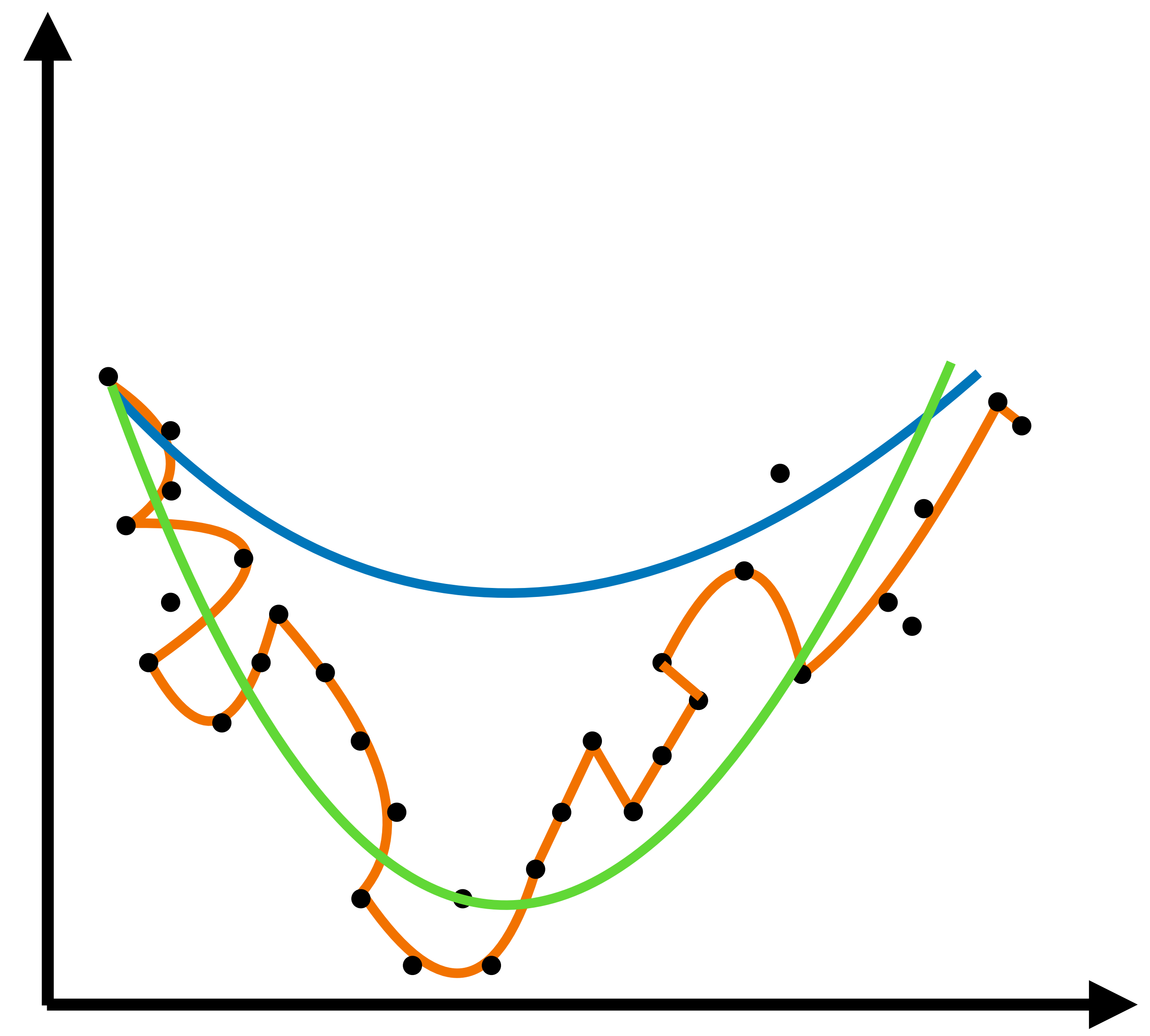
Practical Issues in Linear Regression

Regularization

- Regularization explicitly trades bias for variance.

$$L(\theta) = \frac{1}{m}\sum (Y - X\theta)^2 + \lambda\|\theta\|^2$$

$$\theta = (X^T X + \lambda I)^{-1} X^T Y$$



- As λ increases:
  - Coefficients shrink toward zero
  - Bias increases (we're constraining the model)
  - Variance decreases (less sensitive to data)
  - At some $\lambda*$, test error is minimized

Practical Issues in Linear Regression

Regularization

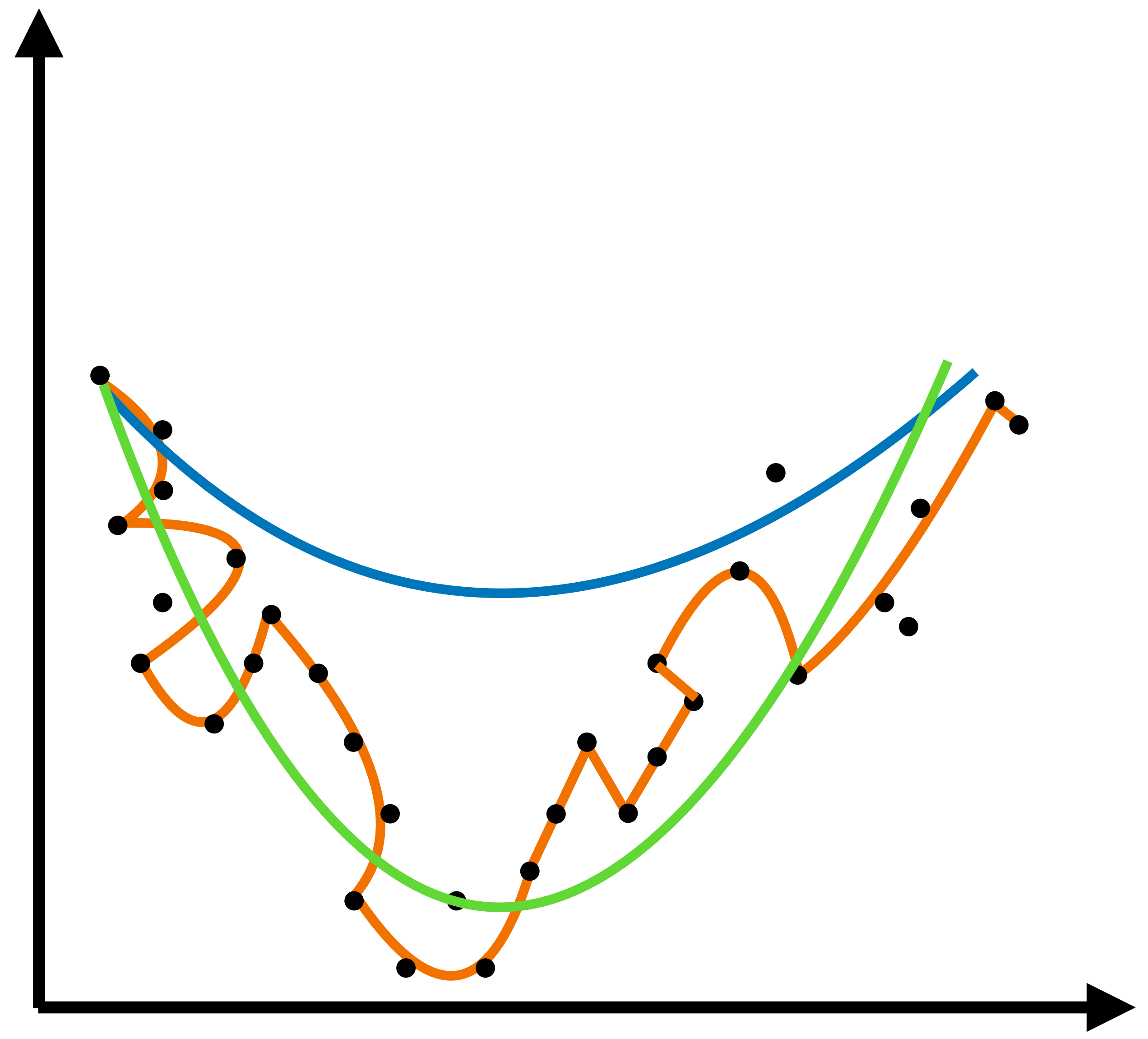
- Regularization explicitly trades bias for variance.

$$L(\theta) = \frac{1}{m} \sum (Y - X\theta)^2 + \lambda ||\theta||^2$$

$$\theta = (X^T X + \lambda I)^{-1} X^T Y$$

These sort of parameters are usually called **hyper-parameters**

They are **not learnable** but are human defined



- As λ increases:
  - Coefficients shrink toward zero
  - Bias increases (we're constraining the model)
  - Variance decreases (less sensitive to data)
  - At some $\lambda*$, test error is minimized

Feature Normalization

Why Normalize?

- If feature x_1 ranges from 0 to 1 and feature x_2 ranges from 0 to 1,000,000, this could lead to numerical instability in the solving process
  - This is particular relevant to gradient descent
- Regularization unfairness
  - If x_2 is much larger, θ_2 must be much smaller to produce similar predictions.
  - The regularization penalty then affects features unequally based on arbitrary scale choices.
- Distance-based algorithms

Feature Normalization

Normalization Methods

1. Min-Max Normalization
2. Mean-Variance Normalization
3. Max-Absolute Normalization
4. Robust Normalization

Feature Normalization

Min-Max Normalization

- For every column in the input data, i.e., for each x_0, x_1, x_2, x_4 etc., this normalization method will scale each column to 0 and 1

$$x' = \frac{x - min(x)}{max(x) - min(x)}$$

- This method preserves zero entries in sparse data
- But is very sensitive to **outliers**

Feature Normalization

Mean-Variance Normalization

- For every column in the input data, i.e., for each x_0, x_1, x_2, x_4 etc., this normalization method will scale to have mean 0 and standard deviation 1

$$x' = \frac{x - \mu(x)}{\sigma(x)}$$

- Most common in practice
- Less sensitive to outliers than min-max
- Does not bound the range to 0 and 1

Feature Normalization

Max-Absolute Normalization

- For every column in the input data, i.e., for each x_0, x_1, x_2, x_4 etc., this normalization method will scale each column to -1 and 1

$$x' = \frac{x}{|max(x)|}$$

- Good for sparse data since it preserves sparsity (zeros stay zero)

Feature Normalization

Robust Normalization

- For every column in the input data, i.e., for each x_0, x_1, x_2, x_4 etc., this normalization method will scale each column as

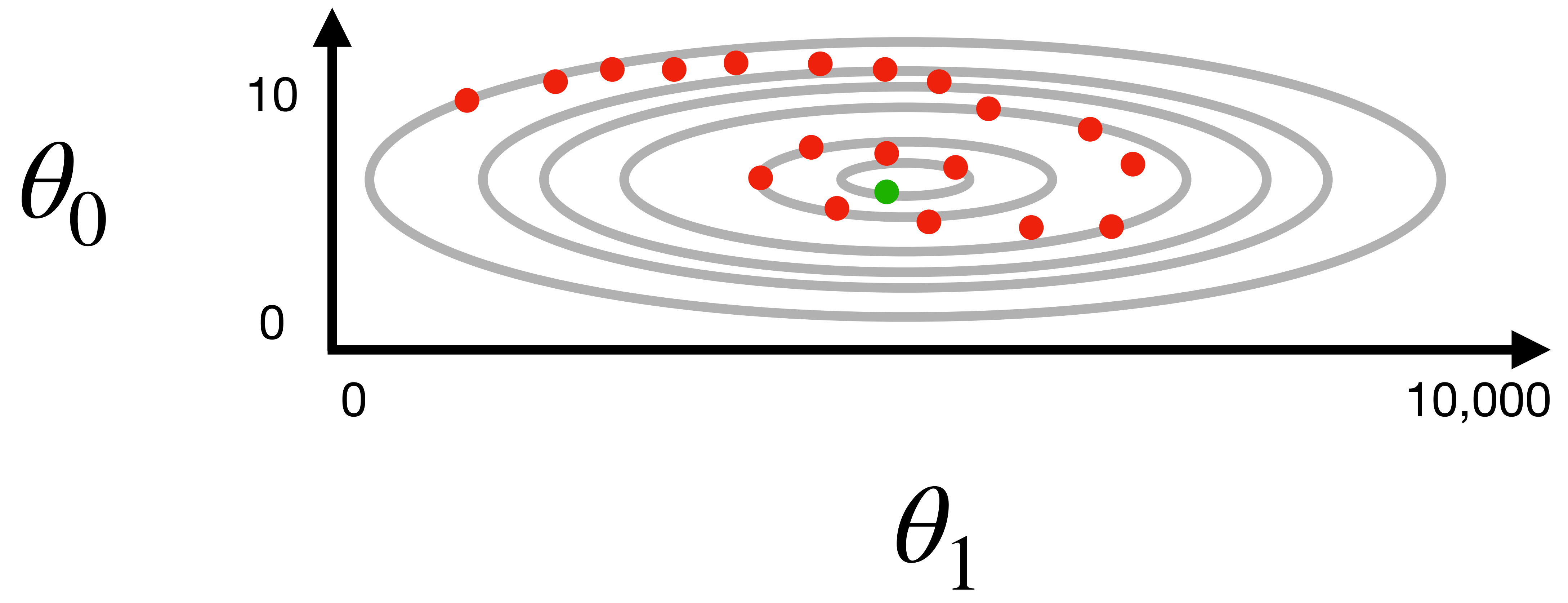
$$x' = \frac{x - median(x)}{IQR(x)}$$

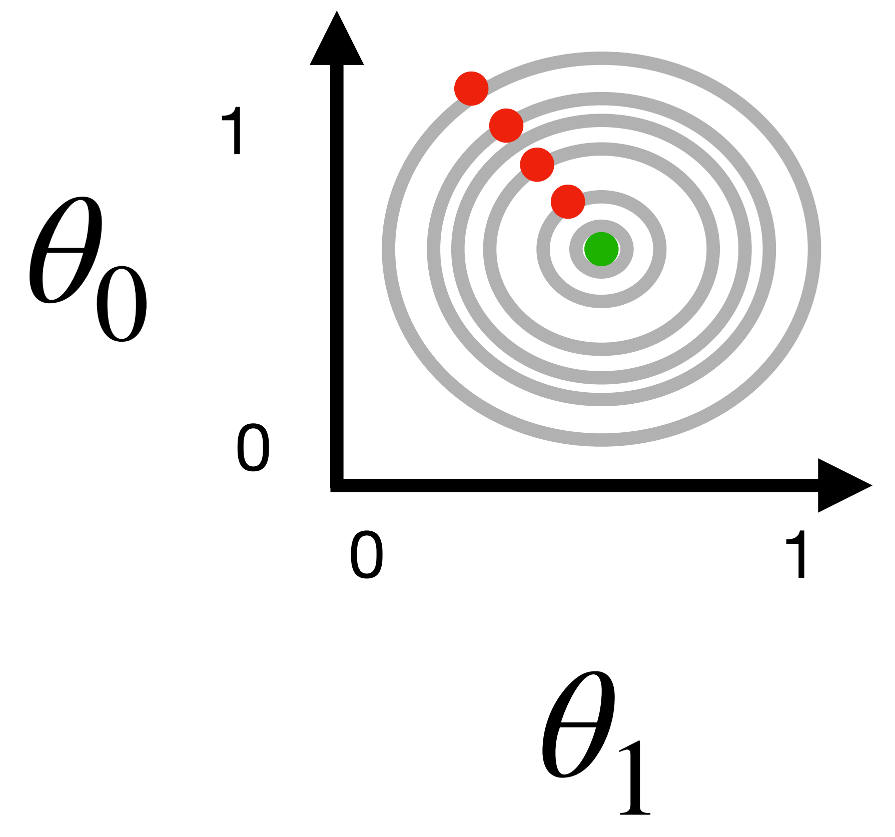
- Robust to outliers
- Use when data has many outliers

Optimizing Loss Functions

Gradient Descent - Practical Fixes

- Feature Scaling
  - Remember we want all input features $x_1, x_2 \cdots x_n$ to be in similar ranges
  - When features have different scales, the loss surface becomes elongated (ill-conditioned).





This dramatically accelerates the optimization process

This also allows having one single learning rate for all parameters

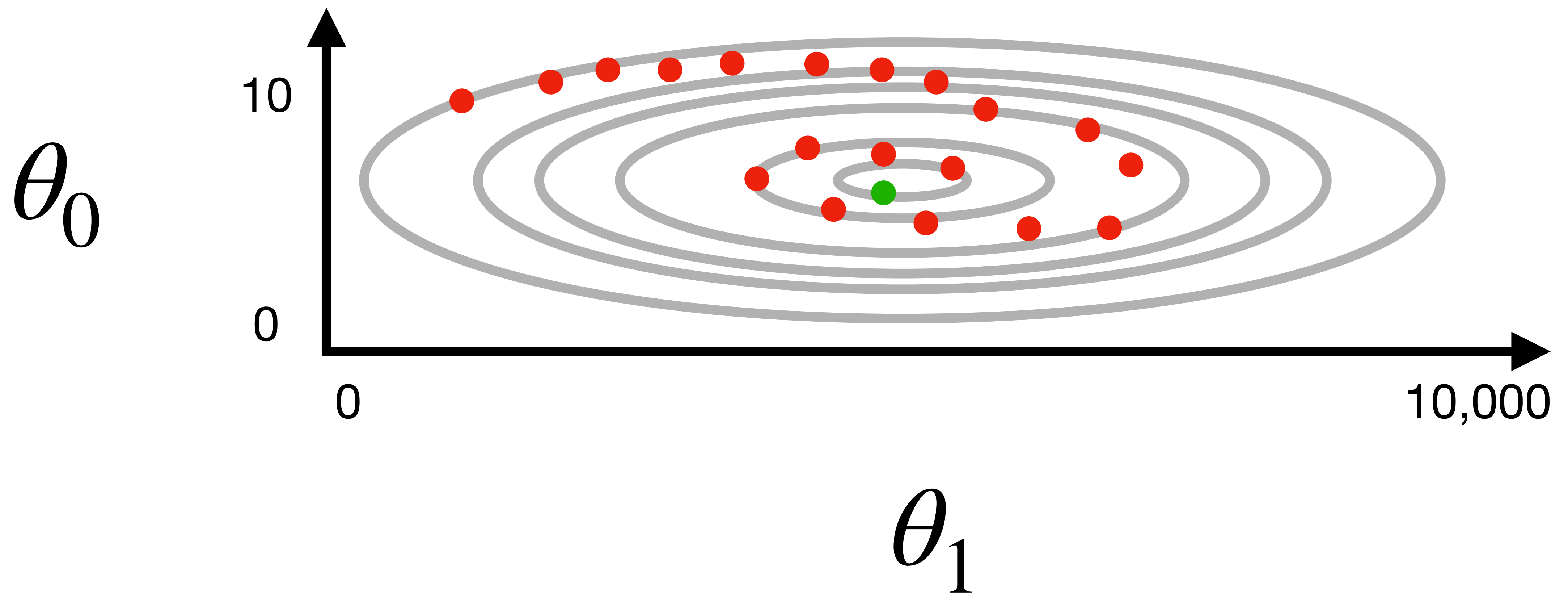
Optimizing Loss Functions

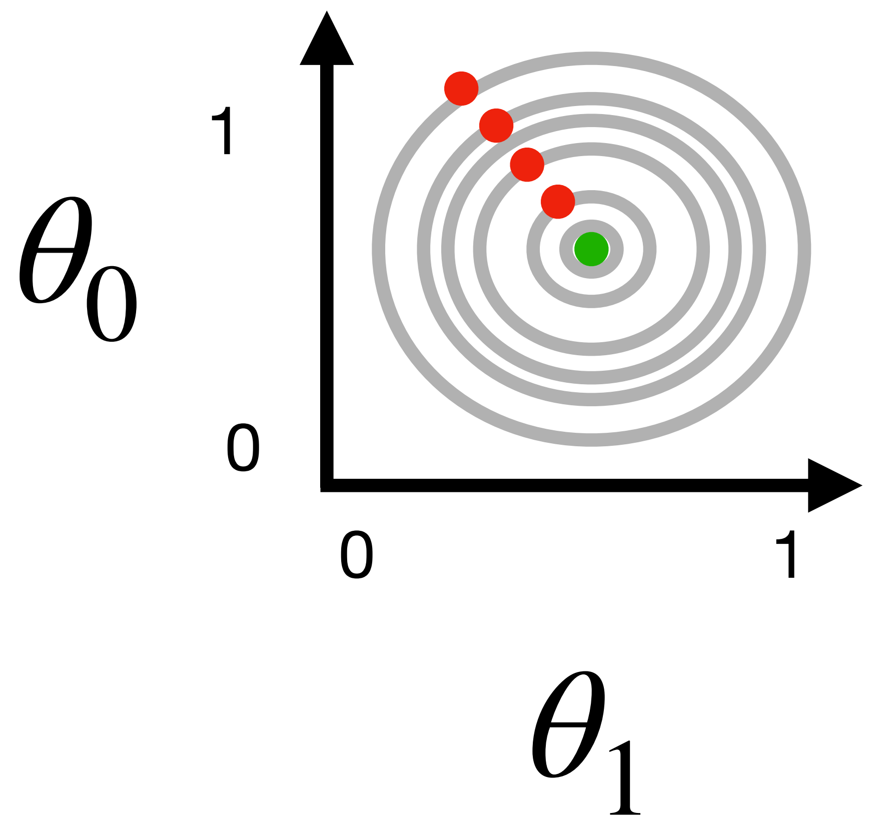
Gradient Descent - Practical Fixes

- Feature Scaling

NOTE: Scaling parameters (mean, standard deviation, min, max) must be computed **<u>only on training data</u>** and then applied to validation and test data to prevent data leakage.

- Remember we want all input features $x_1, x_2 \cdots x_n$ to be in similar ranges
- When features have different scales, the loss surface becomes elongated (ill-conditioned).





This dramatically accelerates the optimization process

This also allows having one single learning rate for all parameters

Today’s Outline

- Classification
- Metrics
- k-Nearest Neighbors

Today’s Outline

- **Classification**
- Metrics
- k-Nearest Neighbors

Classification

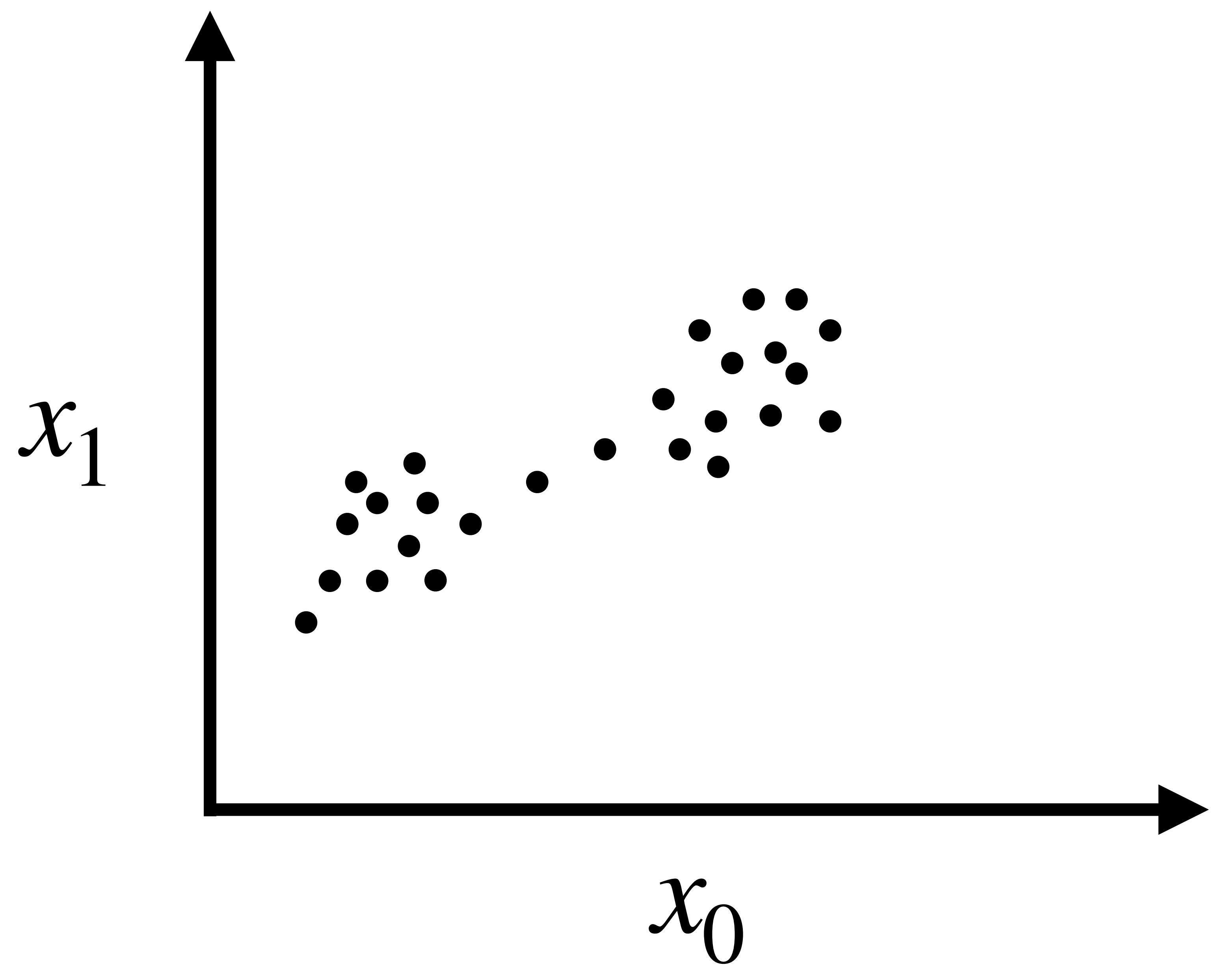
Introduction

- Classification is a supervised learning task where the goal is to predict a discrete class label y for a given input x
- For binary classification, $y \in \{0, 1\}$
- For multi-class classification, $y \in \{1, 2, \ldots, k\}$ where $k > 2$

Classification

Decision Boundary

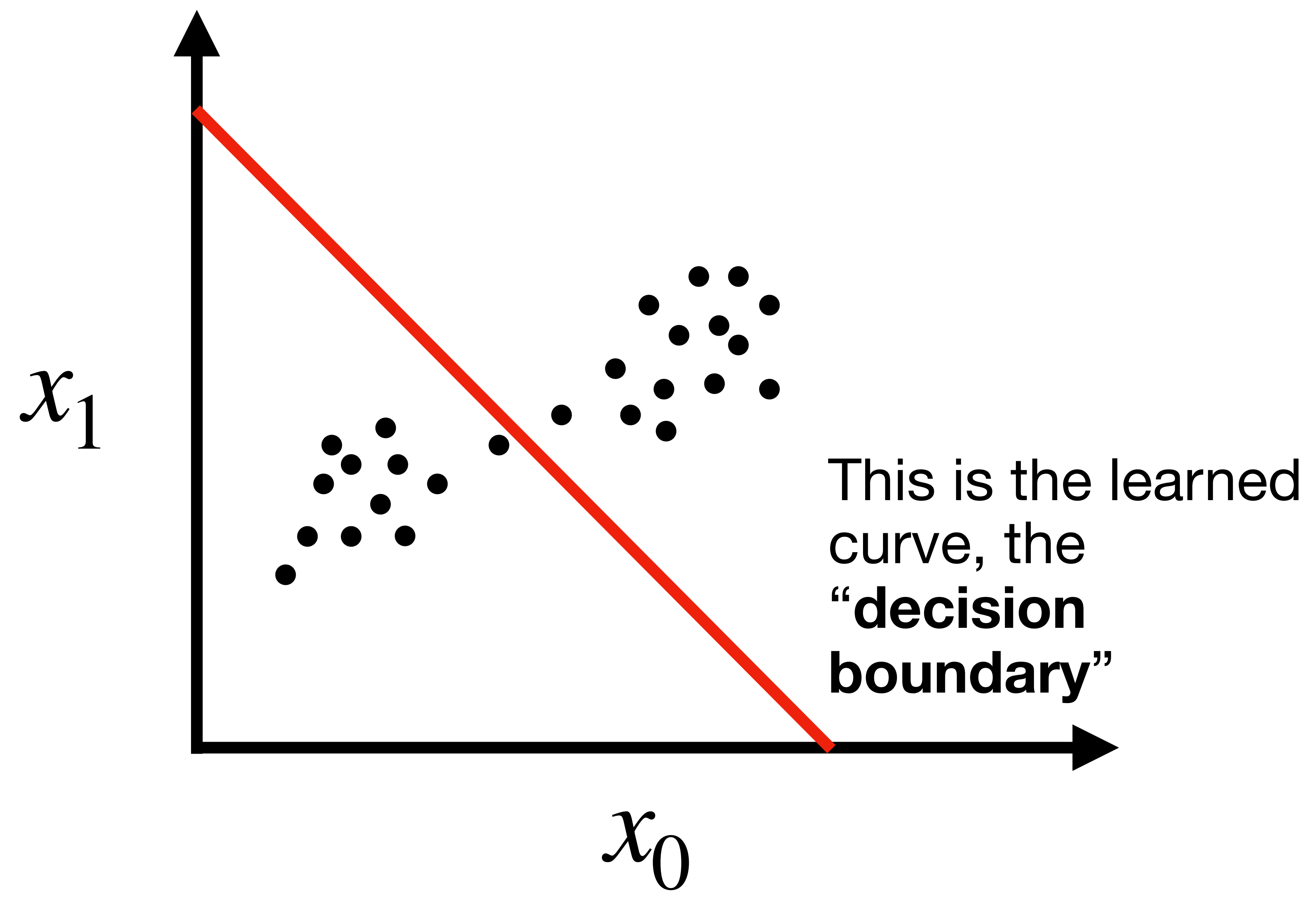
- A classifier partitions space into multiple sections, each section corresponding to a class.



Classification

Decision Boundary

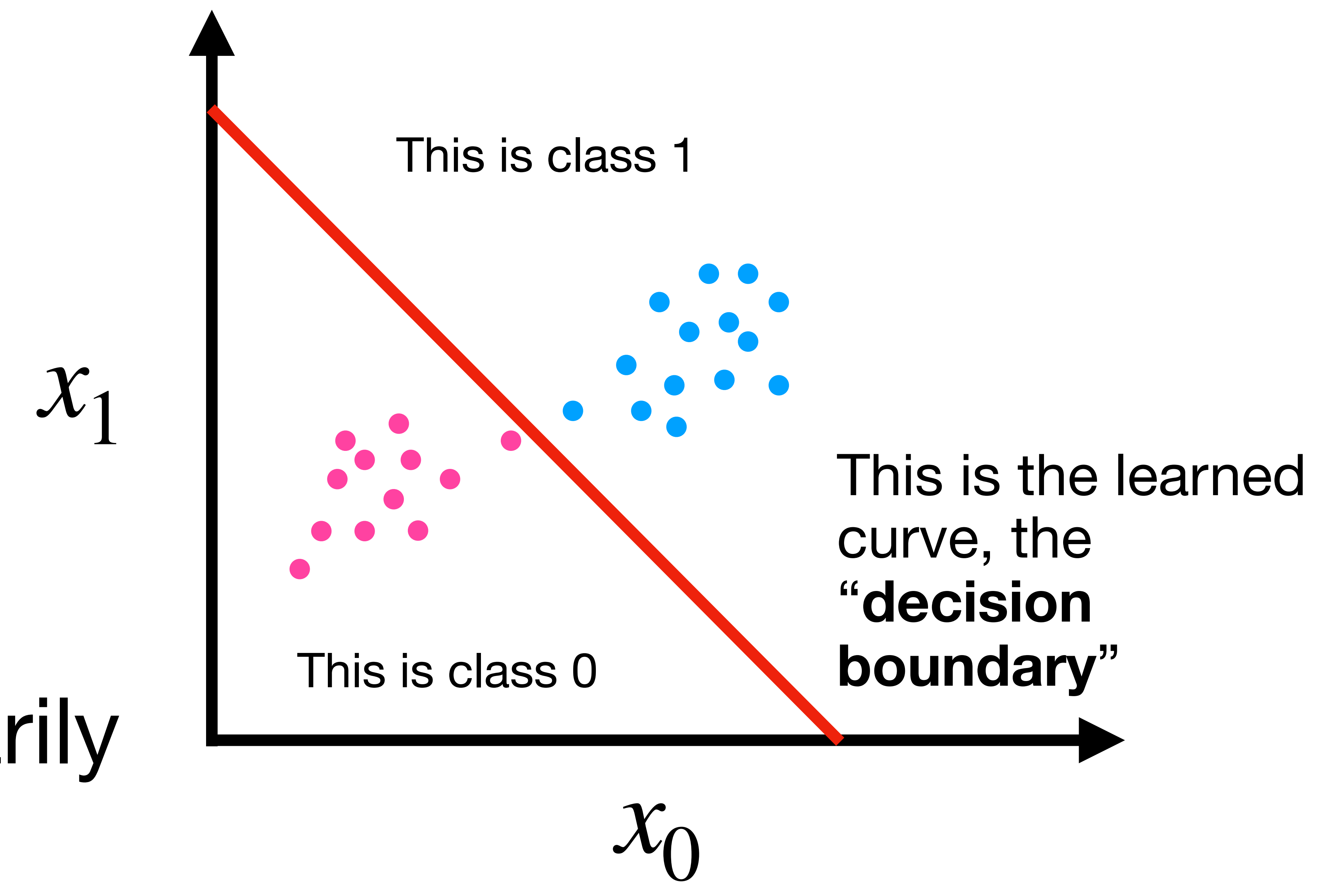
- A classifier partitions space into multiple sections, each section corresponding to a class.



Classification

Decision Boundary

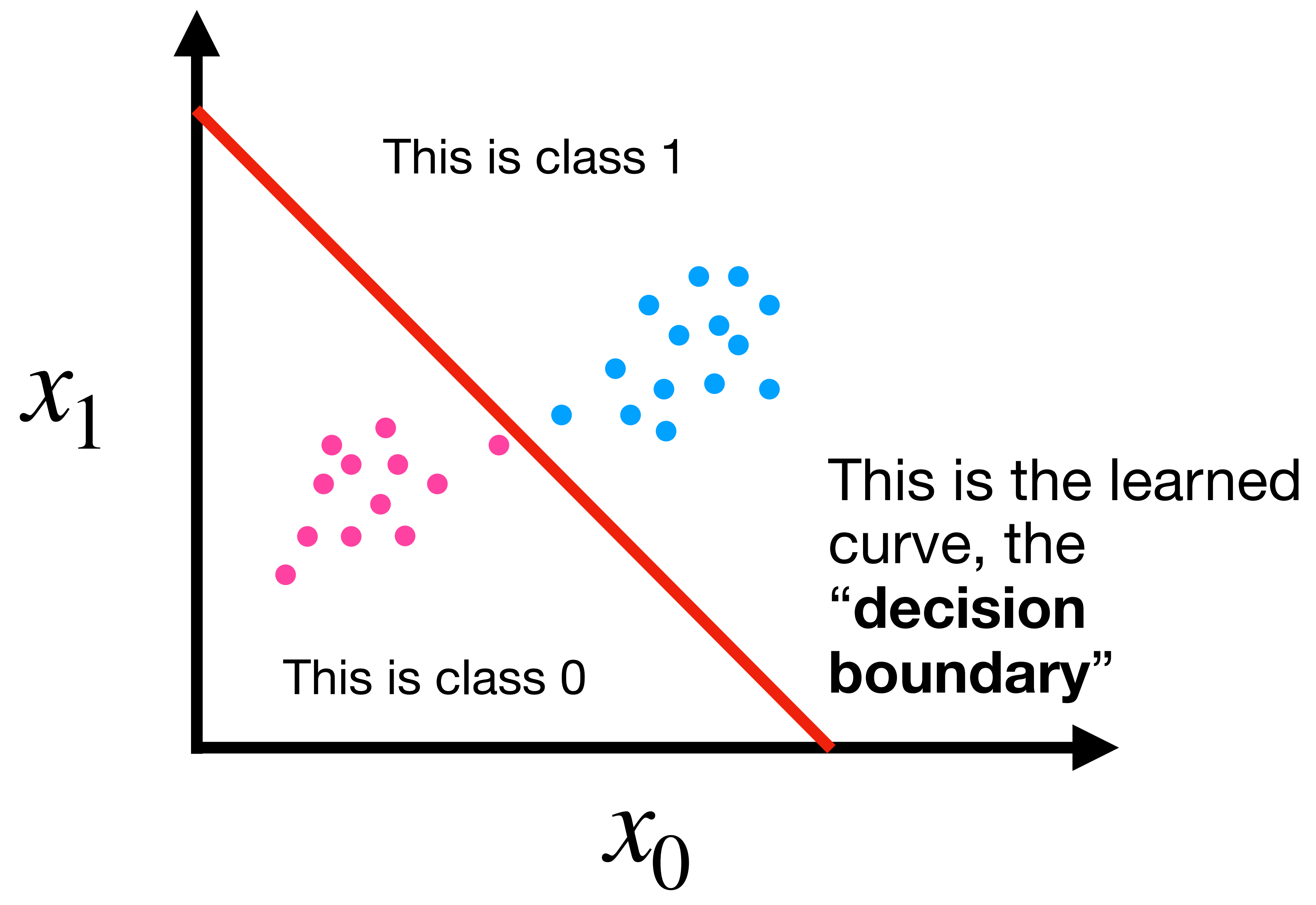
- A classifier partitions space into multiple sections, each section corresponding to a class.
- Different algorithms produce different boundary shapes
  - Linear classifiers produce hyperplanes
  - Non-linear classifiers can produce arbitrarily complex boundaries.



Classification

Decision Boundary

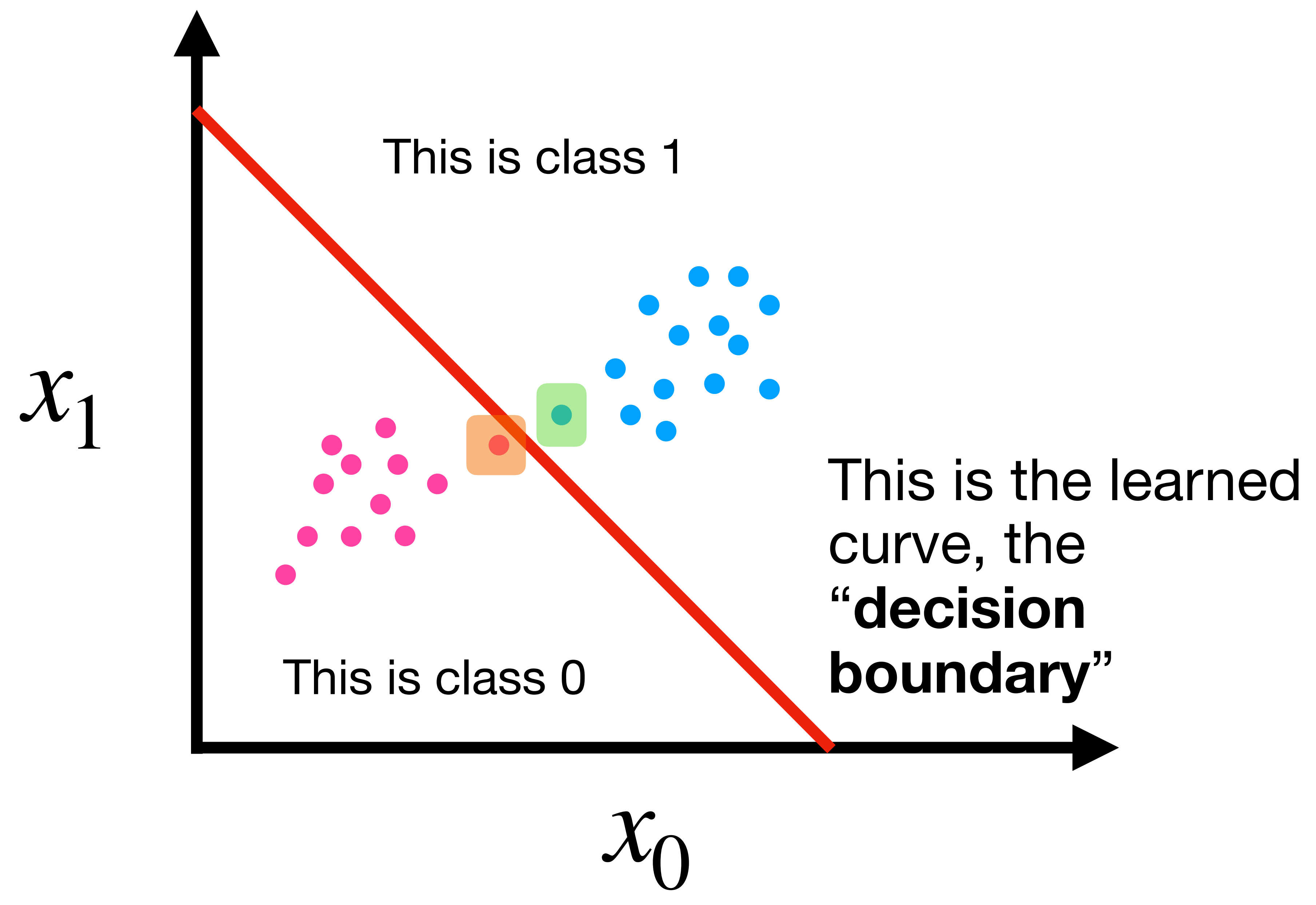
- But practically speaking, your classifier will output a probability value between 0 and 1
- Example:
  - $\mathbb{P}(cat \mid image_1) = 0.61$
  - $\mathbb{P}(cat \mid image_2) = 0.52$



Classification

Decision Boundary

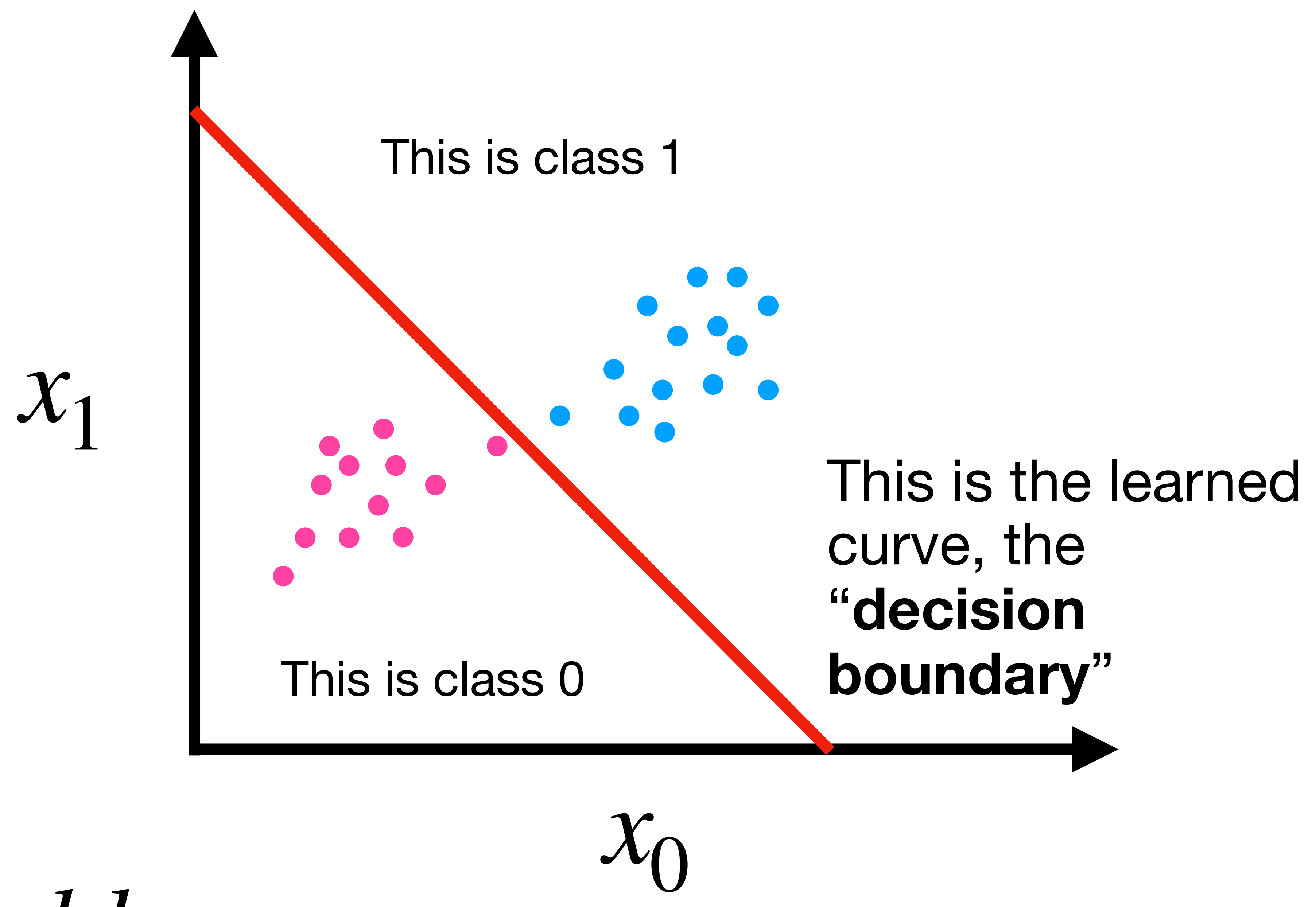
- But practically speaking, your classifier will output a probability value between 0 and 1
- Example:
  - $\mathbb{P}(cat \,|\, image_1) = 0.61$
  - $\mathbb{P}(cat \,|\, image_2) = 0.52$



Classification

Decision Boundary

- But practically speaking, your classifier will output a probability value between 0 and 1
- Example:
  - $\mathbb{P}(cat \mid image_1) = 0.61$
  - $\mathbb{P}(cat \mid image_2) = 0.52$
- Practitioner needs to also set a **threshold**
  - $image_i$ is a cat if $\mathbb{P}(cat \mid image_i) \geq Threshold$



Today’s Outline

- Classification
- **Metrics**
- k-Nearest Neighbors

Metrics

- An obvious metric is **accuracy**

$$Accuracy = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Data Points}}$$

- Say you have a cat classifier with 1000 images. Your classifier gets 797 out of 1000 images correct

$$Accuracy = \frac{797}{1000} = 79\,\%$$

Metrics

- But, accuracy does not tell the whole picture
- Especially when data is skewed
  - For example, if your training data is of size 1000 images
    - 900 of them are of dogs
    - 100 of them are cats
  - **Question:** Is accuracy a good metric in this case?

Metrics

Confusion Matrix

	Predicted Positive	Predicted Negative
Actual Positive		
Actual Negative		

Metrics

Confusion Matrix

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Metrics

Confusion Matrix

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Metrics

Confusion Matrix

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Metrics

Accuracy

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

The proportion of correct predictions.

Simple and intuitive, but misleading for imbalanced data.

A classifier that always predicts the majority class achieves high accuracy on imbalanced datasets while being useless.

Metrics

Precision and Recall

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Metrics

Precision and Recall

$$\text{Precision} = \frac{TP}{TP + FP}$$

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Metrics

Precision and Recall

$$\text{Precision} = \frac{TP}{\boxed{TP + FP}}$$

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Metrics

Precision and Recall

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Metrics

Precision and Recall

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{\boxed{TP + FN}}$$

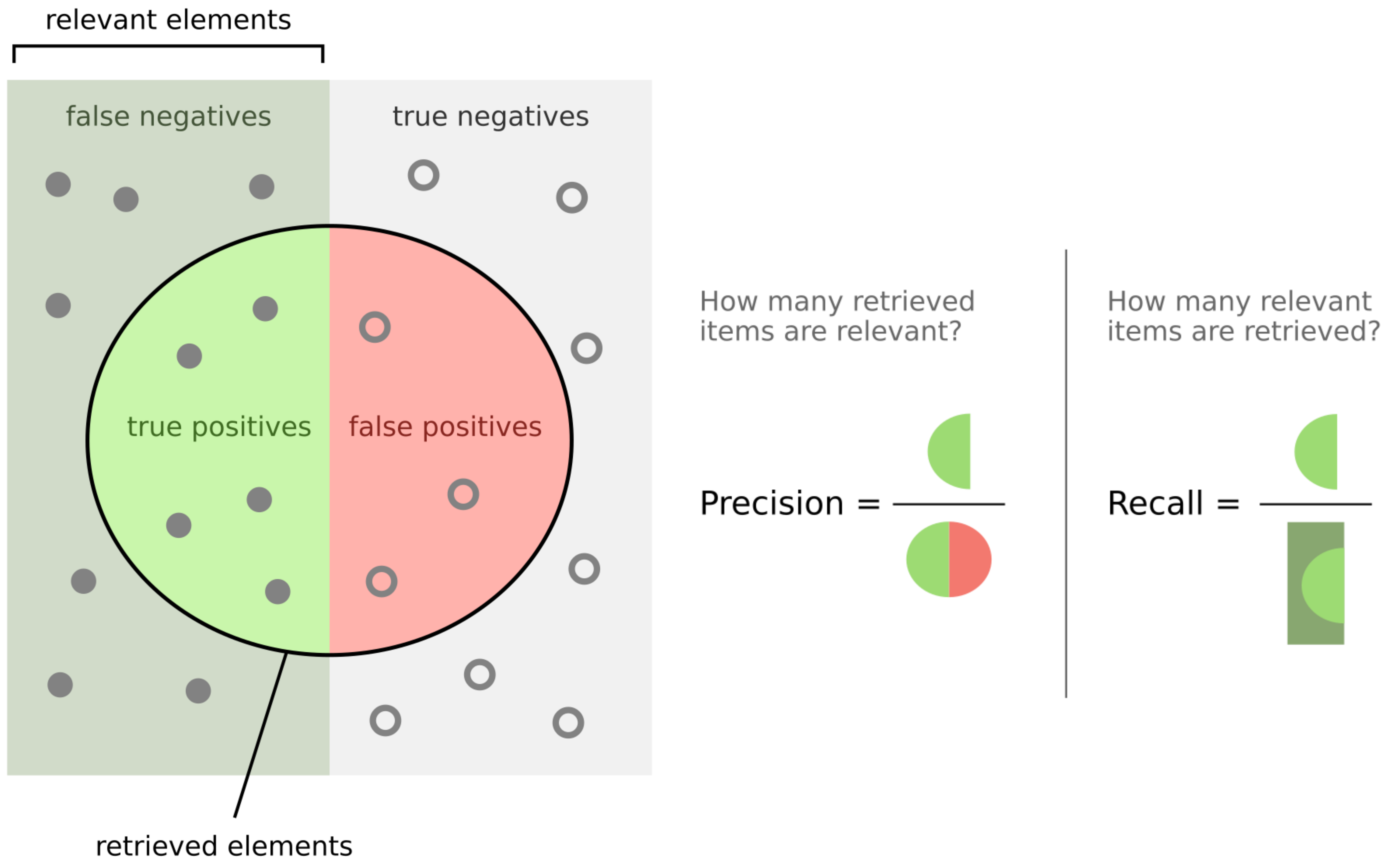
	Predicted Positive	**Predicted Negative**
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Metrics

Precision and Recall

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$



	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Metrics

Precision and Recall

$$\textbf{Precision} = \frac{TP}{TP + FP}$$

Of all instances predicted as positive, what fraction actually are positive? Precision measures the **reliability of positive predictions**. High precision means **few false alarms.**

When to care about precision?
When false positives are costly.

Examples include spam filtering (users hate losing important emails), recommendation systems (irrelevant recommendations erode trust), and legal contexts (wrongful accusations).

$$\text{Recall} = \frac{TP}{TP + FN}$$

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Metrics

Precision and Recall

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\textbf{Recall} = \frac{TP}{TP + FN}$$

Of all actual positive instances, **what fraction did we correctly identify**? Recall measures coverage of positive instances. High recall means **few missed positives**.

When to care about recall?
When false negatives are costly.

Examples include disease screening (missing a diagnosis can be fatal), security threats (missing an attack is catastrophic), and search engines (users want all relevant results).

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Metrics

Precision vs Recall Tradeoff - F1 Score

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

Precision and recall are **inherently in tension**.

Increasing the threshold for positive classification typically **increases precision but decreases recall**.

Decreasing the threshold has the opposite effect.

The optimal balance depends on the application's cost structure.

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)