

# Table of Contents

Lecture 01 - knn

Lecture 02 - linear - models

Lecture 03 - linear - models - II

Lecture 04 - gradient - descent

Lecture 05 - convexity

Lecture 06 - svm

Lecture 07 - regularization

Lecture 08 - kernels

Lecture 09 - neural - networks

Lecture 10 - bayes - decision - theory

Lecture 11 - density - estimation

Lecture 12 - mle

Lecture 13 - mv - gaussians

Lecture 14 - naive - bayes

Lecture 15 - regression - probabilistic

Lecture 16 - decision - trees

# DSC 140A

*Probabilistic Modeling & Machine Learning*

Lecture 01 | Part 1

**What is Machine Learning??**

# **What is Machine Learning?**

- ▶ Computers can do things very quickly.

# **What is Machine Learning?**

- ▶ Computers can do things very quickly.
- ▶ But must be given really specific instructions.

# What is Machine Learning?

- ▶ Computers can do things very quickly.
- ▶ But must be given really specific instructions.
- ▶ **Problem:** Not all tasks are easy to dictate.

# Example



How old is this person?

# The Trick: Use Data



age = 28



age = 42



age = 63



age = 24



age = 37



age = 39



age = ?



age = 35

# What is Machine Learning?

- ▶ Before: Computer is **told** how to do a task.
- ▶ Instead: **learn** how to do a task using data.

# What is Machine Learning?

- ▶ Before: Computer is **told** how to do a task.
- ▶ Instead: **learn** how to do a task using data.
- ▶ We still have to **tell** the computer how to learn.

A **machine learning algorithm** is a set of precise instructions telling the computer how to learn from data.

A **machine learning algorithm** is a set of precise instructions telling the computer how to learn from data.

Spoiler: the algorithms are usually pretty simple. It's the **data** that does the real work.

# Machine Learning Tasks

- ▶ Prediction
- ▶ Clustering
- ▶ Representation Learning
- ▶ Reinforcement Learning
- ▶ Anomaly Detection
- ▶ ...

# Machine Learning Tasks

- ▶ Prediction
- ▶ Clustering
- ▶ Representation Learning
- ▶ Reinforcement Learning
- ▶ Anomaly Detection
- ▶ ...

**But first...**

Syllabus: [dsc140a.com](http://dsc140a.com)

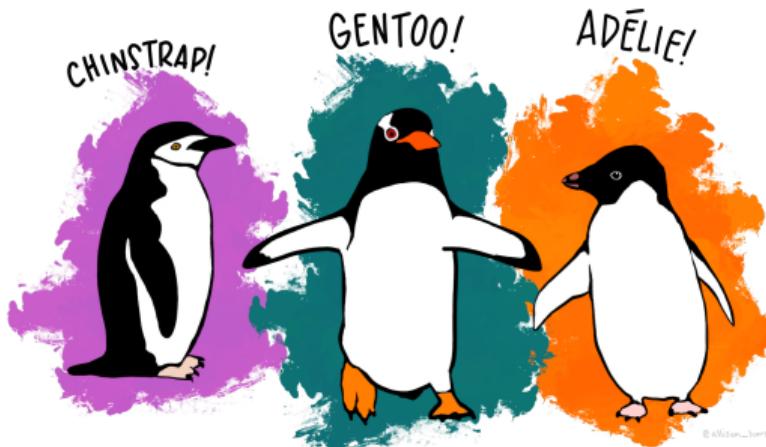
# DSC 140A

*Probabilistic Modeling & Machine Learning*

Lecture 01 | Part 2

## A Simple Prediction Algorithm

# Penguin Prediction



1

- ▶ **Task:** given a new penguin, predict its species.

---

<sup>1</sup>Artwork by @allison\_horst

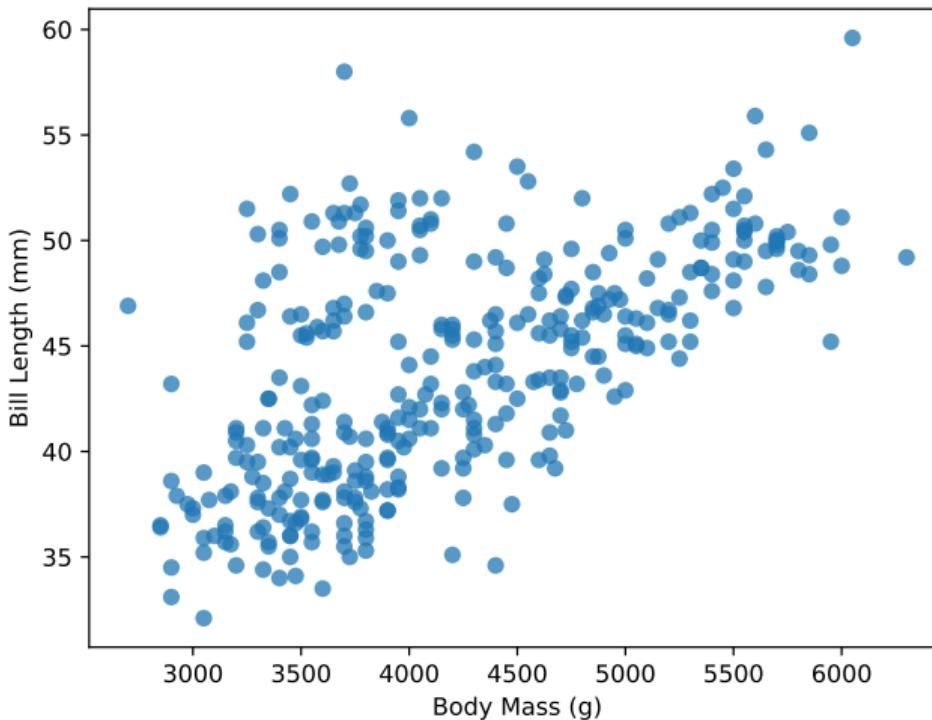
# First Step: Featurization

- ▶ We represent each penguin as a collection of measurements (a **feature vector**).
- ▶ Most often, features are numerical.
- ▶ Why?
  1. Computers process numbers (not penguins).
  2. Allows us to use mathematical machinery.

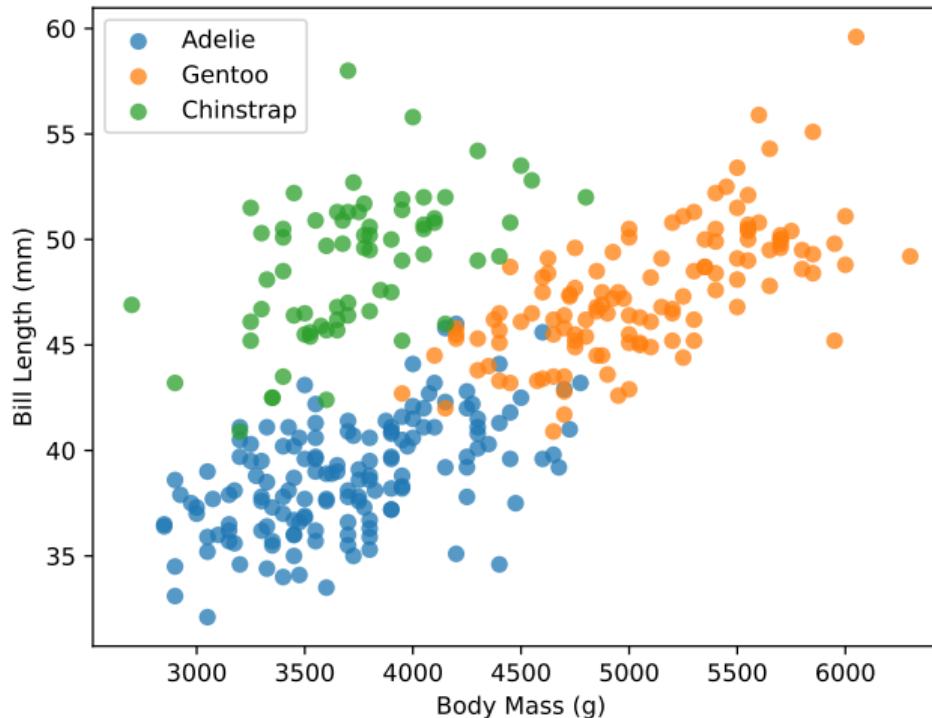
# First Step: Featurization

- ▶ Chosen features should contain enough info to distinguish between species.
- ▶ We will represent each penguin by two numbers:
  1. Body Mass (in grams)
  2. Bill Length (in millimeters)
- ▶ Allows us to **embed** penguins as **point cloud** in  $\mathbb{R}^2$ .

# Penguin Embedding

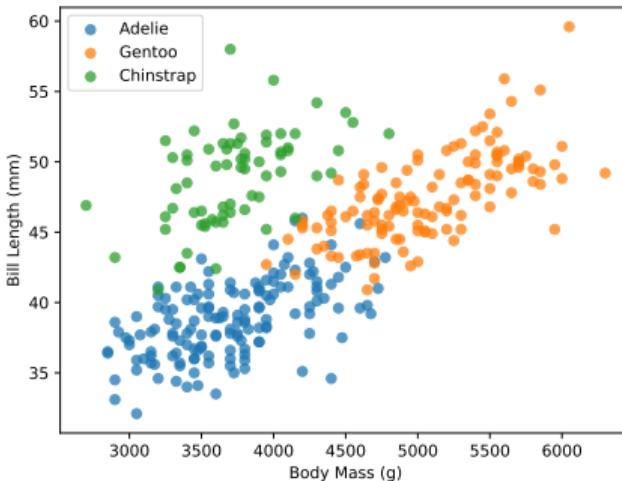


# Penguin Embedding



## Exercise

We see a new penguin with body mass of 5300 g and bill length of 46 mm. What is its species, most likely?



# A Simple Intuition

- ▶ New penguin's embedding is close to Gentoo penguins  $\implies$  it is mostly likely also Gentoo.
- ▶ **Our Assumption:** *locality*. Similar inputs have similar outputs.

# A Simple Prediction Algorithm

- ▶ **Data:** a set of penguins (as feature vectors) and their species.
- ▶ **Given:** a new penguin whose species is unknown.
- ▶ **Predict:**
  1. Find the *nearest* penguin whose species is known.
  2. Use that penguin's species as our prediction.

# Nearest Neighbor Classification

- ▶ **Data:** a set  $\mathcal{D}$  of  $n$  feature vectors with labels:  
 $\{(\vec{x}^{(i)}, y_i)\} = \{(\vec{x}^{(1)}, y_1), \dots, (\vec{x}^{(n)}, y_n)\}$
- ▶ **Given:** a new point,  $\vec{z}$  with unknown label.
- ▶ **Predict:**
  1. Find the closest point to  $\vec{z}$  in  $\mathcal{D}$ :

$$i^* = \arg \min_{i \in \{1, \dots, n\}} \|\vec{x}^{(i)} - \vec{z}\|$$

2. Use  $y_{i^*}$  as the predicted label.

# A Note About Distances

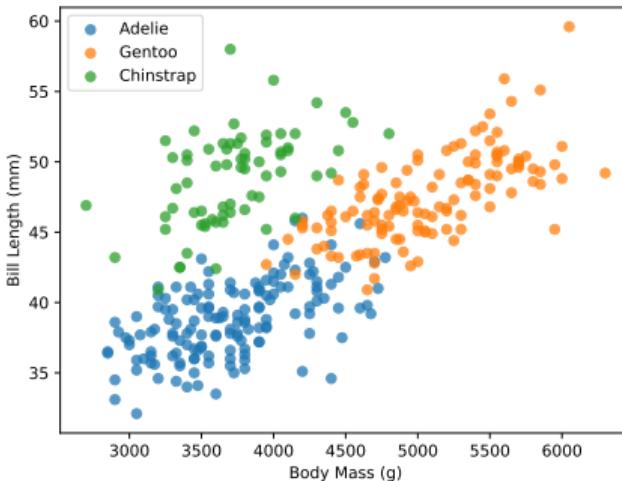
- We found the nearest neighbor with **Euclidean distance**:

$$\begin{aligned}\|\vec{p} - \vec{q}\| &= \sqrt{(p_1 - q_1)^2 + \dots + (p_d - q_d)^2} \\ &= \sqrt{\sum_{k=1}^d (p_k - q_k)^2} \\ &= \sqrt{(\vec{p} - \vec{q}) \cdot (\vec{p} - \vec{q})}\end{aligned}$$

- Note that this is just one choice – there are other valid distances. E.g., cosine distance.

## Exercise

We see a new penguin with body mass of 4800 g and bill length of 53 mm. What is its species, most likely?



# Scale Matters!

- ▶ Not just a visual trick – Euclidean distance treats all directions the same.
- ▶ **Example.** Suppose  $P = (5000g, 45mm)$ . Both of these penguins are the same distance away:

$$Q_1 = (5050g, 45mm) \quad Q_2 = (5000g, 95mm)$$

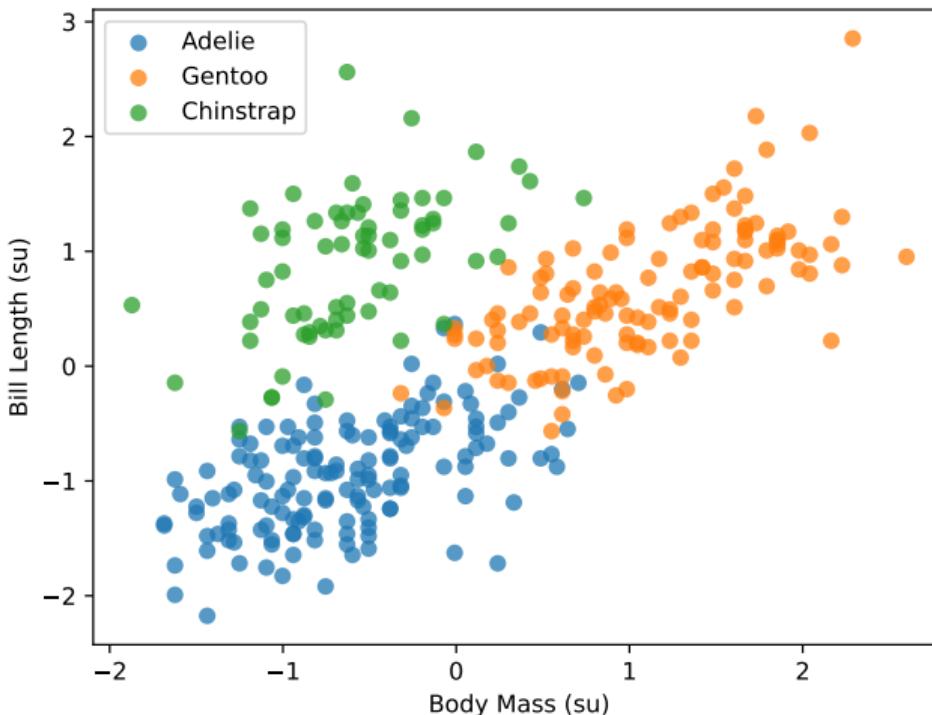
# A Common Fix

- ▶ It is sometimes useful to **scale** each feature independently.
- ▶ E.g., through **standardization**:

$$(\text{new body mass}) = \frac{(\text{old body mass}) - (\text{mean body mass})}{(\text{std. body mass})}$$

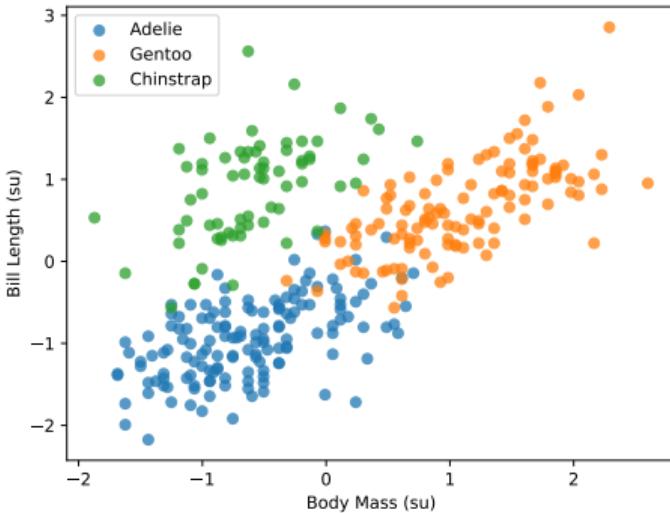
- ▶ Not always the right approach, though!

# Standardized Penguins



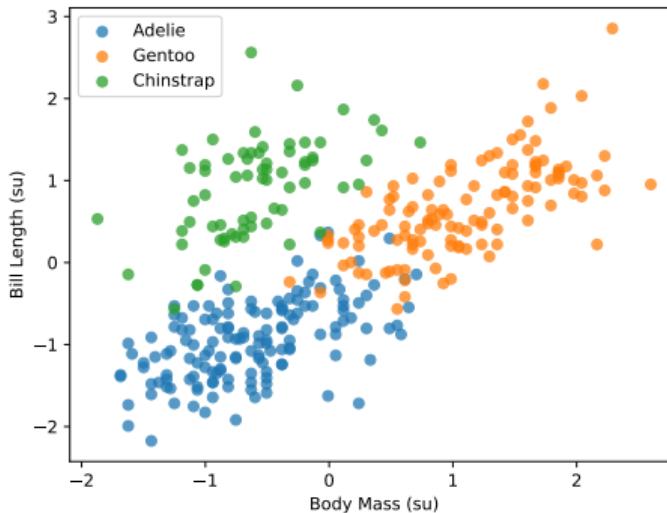
# The Decision Boundary

- ▶ We can visualize the prediction for every possible input.
- ▶ **Decision boundary:** where the prediction changes.

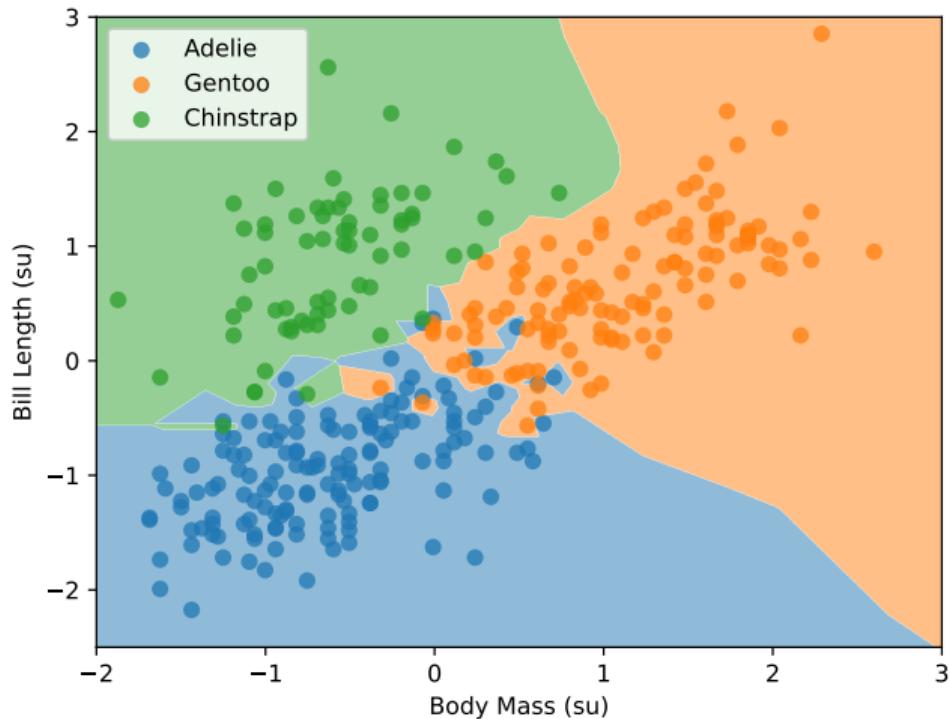


## Exercise

What will the decision boundary look like for our NN penguin classifier?



# The Decision Boundary

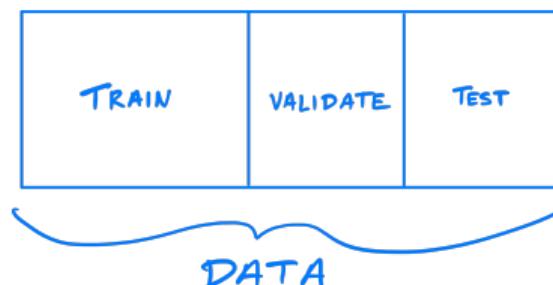


# ***k*-Nearest Neighbors**

- ▶ Before: *single* closest neighbor determined prediction.
- ▶ Idea: have  $k$  closest neighbors “vote”.
- ▶ Can be useful to reduce noise.

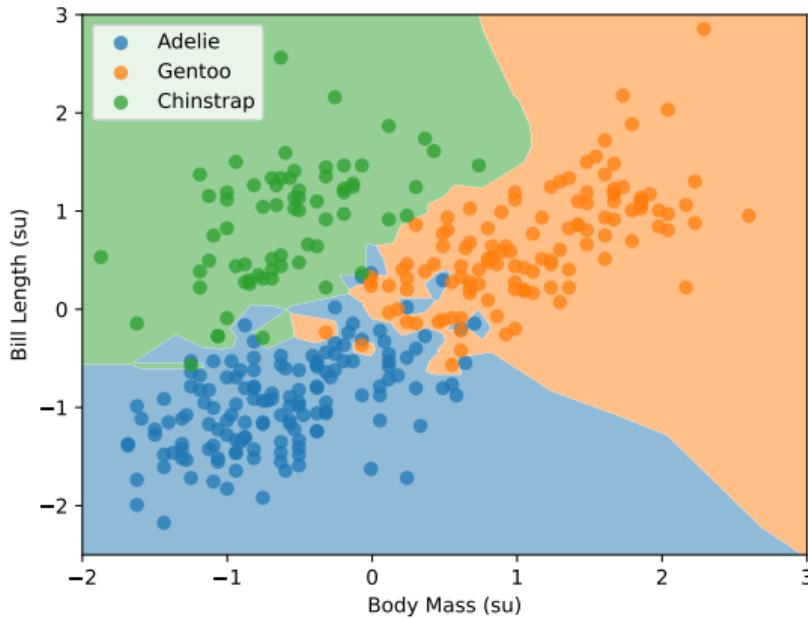
# Choosing $k$

- ▶ The number of neighbors,  $k$ , is a **hyperparameter** of the algorithm.
- ▶ We typically choose  $k$  to be odd to break ties.
- ▶ One approach: choose  $k$  with a **validation set**.



# $k$ and the Decision Boundary

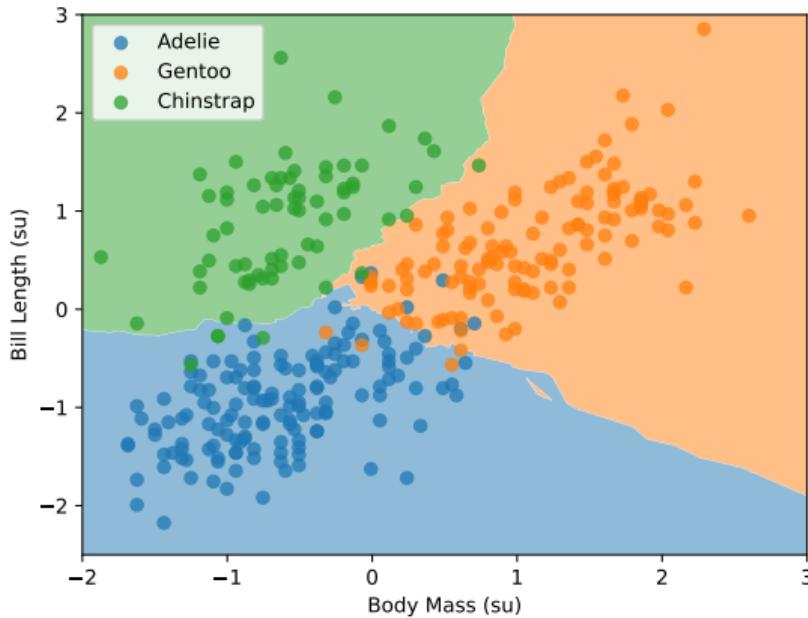
- ▶ How might the decision boundary change as we increase  $k$ ?



$$k = 1$$

# $k$ and the Decision Boundary

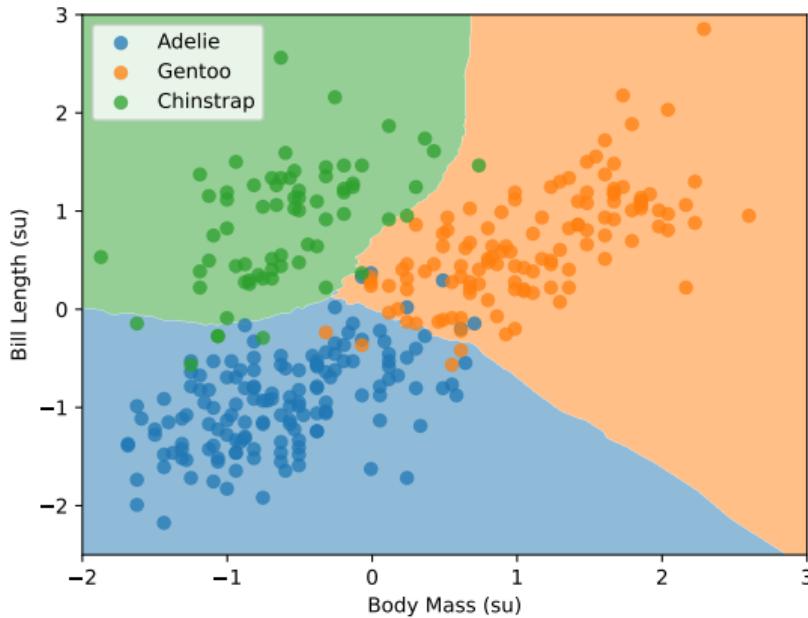
- ▶ How might the decision boundary change as we increase  $k$ ?



$$k = 10$$

# $k$ and the Decision Boundary

- ▶ How might the decision boundary change as we increase  $k$ ?



$$k = 20$$

# $k$ and “Complexity”

- ▶  $k$  controls the “complexity” of the decision boundary.
- ▶ Recall **overfitting**: when predictor learns patterns that do not appear outside of the training data.
- ▶  $k$  can be used to control overfitting.

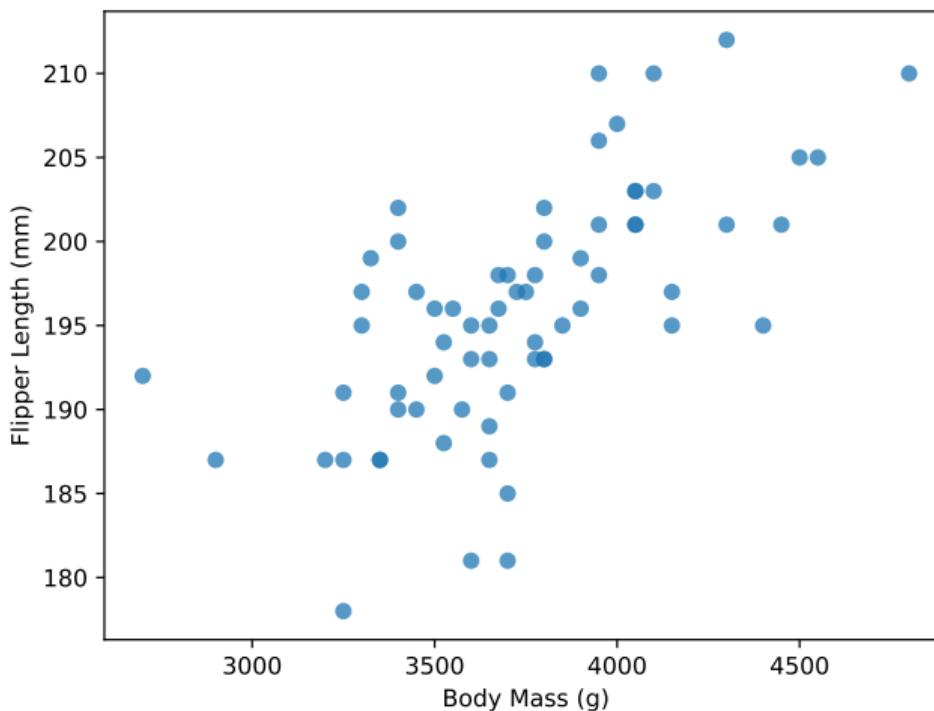
## Exercise

What the prediction be if we set  $k = n$ ?

# Nearest Neighbor Regression

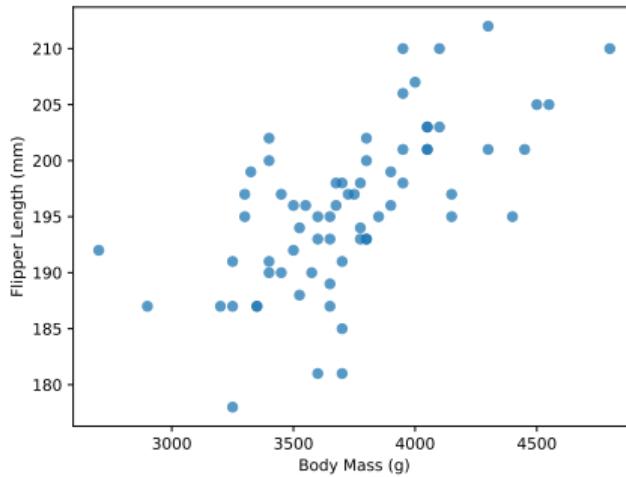
- ▶ The nearest neighbor rule can be used for **regression**, too.

# A Simple Prediction Algorithm



## Exercise

We see a new penguin with body mass of 4000 g.  
What is a likely flipper length for this penguin?



# A Simple Prediction Algorithm

- ▶ **Data:** a set of penguins (as feature vectors) and their flipper lengths.
- ▶ **Given:** a new penguin whose flipper length is unknown.
- ▶ **Predict:**
  1. Find the *nearest* penguin whose species is known.
  2. Use that penguin's flipper length as our prediction.

# Nearest Neighbor Regression

- ▶ **Data:** a set  $\mathcal{D}$  of  $n$  feature vectors with targets:  
 $\{(\vec{x}^{(i)}, y_i)\} = \{(\vec{x}^{(1)}, y_1), \dots, (\vec{x}^{(n)}, y_n)\}$
- ▶ **Given:** a new point,  $\vec{z}$  with unknown target.
- ▶ **Predict:**
  1. Find the closest point to  $\vec{z}$  in  $\mathcal{D}$ :

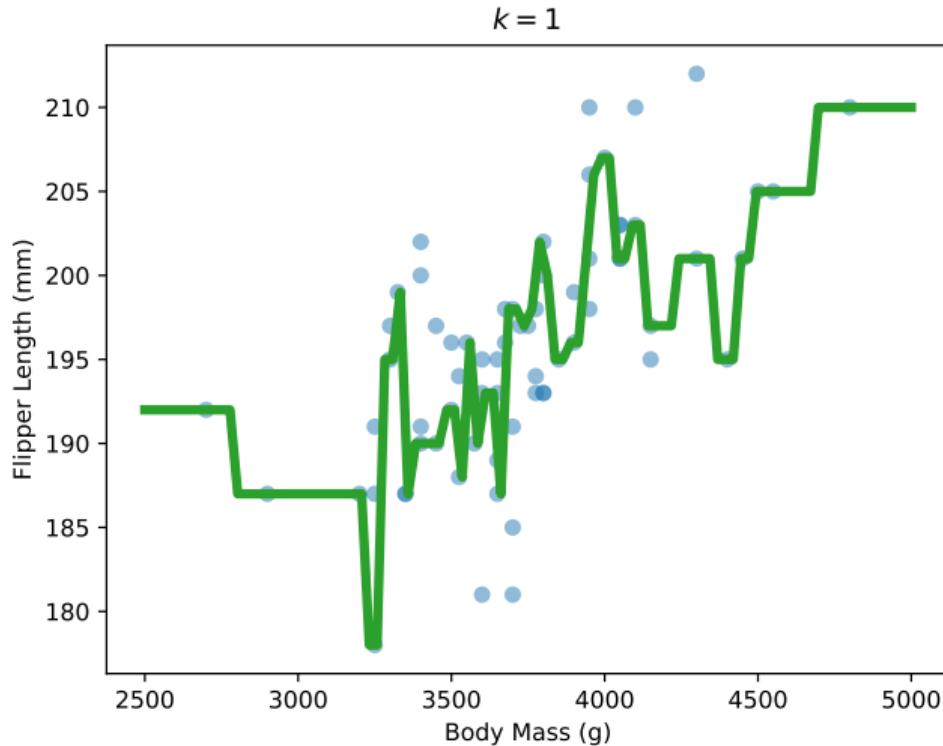
$$i^* = \arg \min_{i \in \{1, \dots, n\}} \|\vec{x}^{(i)} - \vec{z}\|$$

2. Use  $y_{i^*}$  as the predicted target.

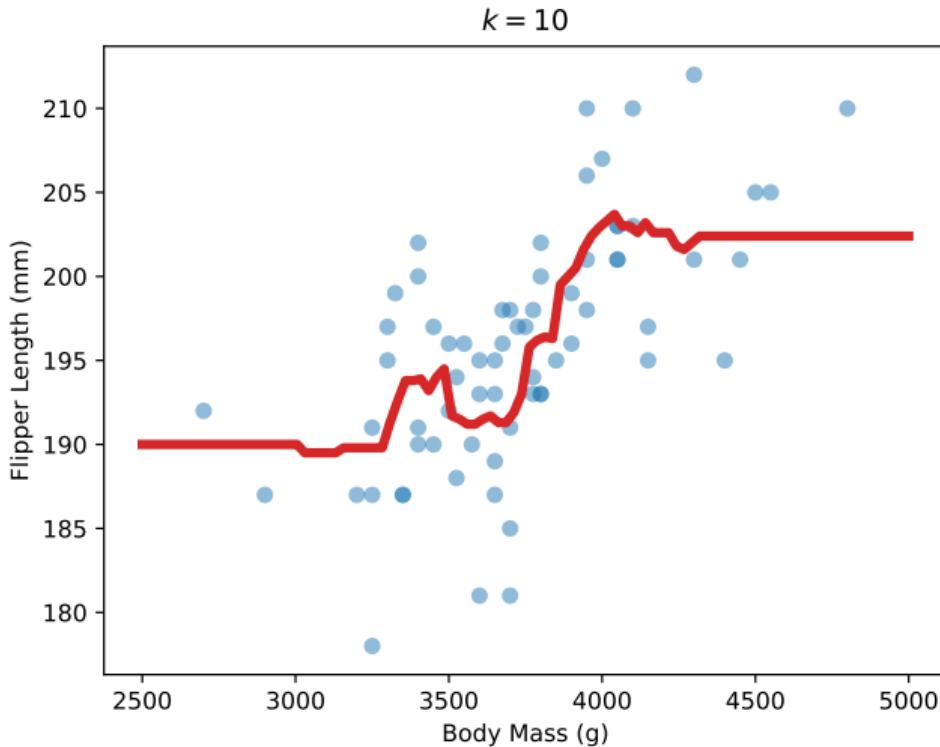
# $k$ NN Regression

- ▶ As with classification, can generalize to  $k$  nearest neighbors.
- ▶ Natural prediction: the **mean** of the targets of the  $k$  closest neighbors.

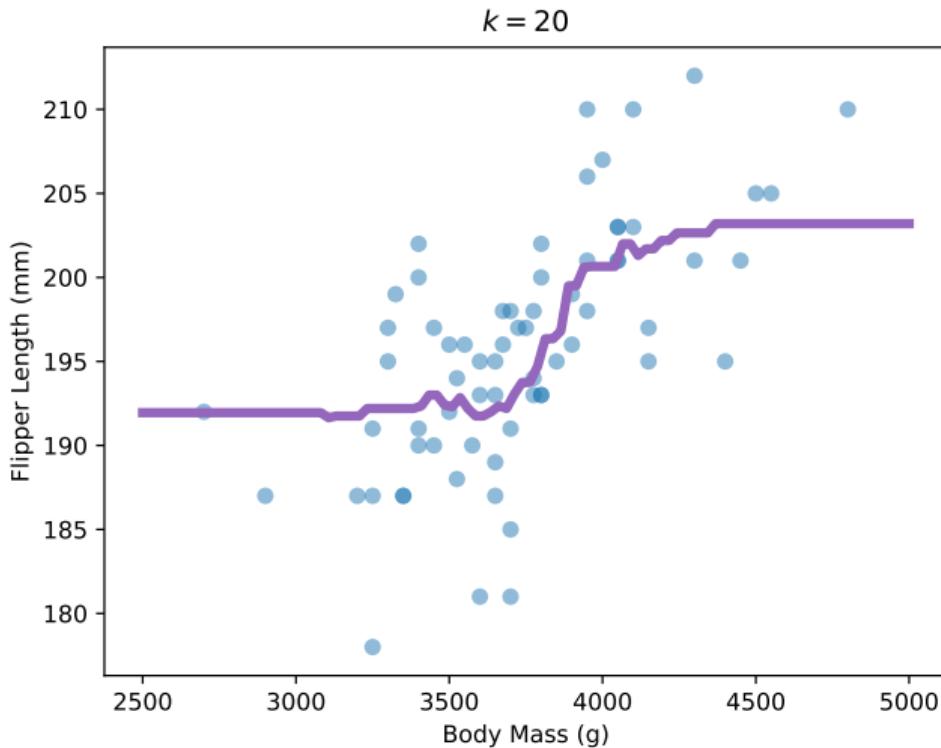
# $kNN$ Penguin Regression



# $kNN$ Penguin Regression



# $kNN$ Penguin Regression



# DSC 140A

*Probabilistic Modeling & Machine Learning*

Lecture 01 | Part 3

**Demo: Classifying Handwritten Digits**

# The Problem

- ▶ **Given** an image of a handwritten digit.
- ▶ **Classify** the image as a one, two, three, etc.



# The Machine Learning Approach

- ▶ Gather a **training set** of images with **labels**.
- ▶ Let the computer **learn** the underlying patterns.
- ▶ We'll use a freely available data set, **MNIST**:



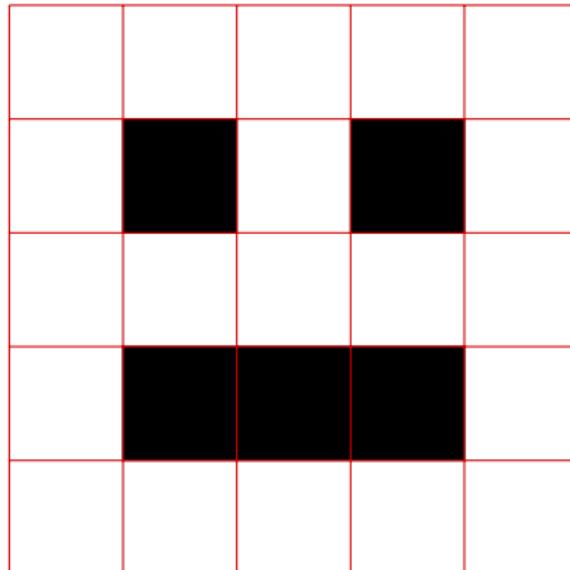
# MNIST

- ▶ 60,000 training images and associated labels.
- ▶ Each image is  $28 \times 28$  pixels.
- ▶ Each pixel is 8 bit grayscale (0 - 255)

# kNN on MNIST

- ▶ We'll use kNN to do **character recognition**.
  - ▶  $\mathcal{X}$  = set of  $28 \times 28$ , 8-bit grayscale images
  - ▶  $\mathcal{Y} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ .
- ▶ How do we represent an image as a feature vector?

# Images as Feature Vectors

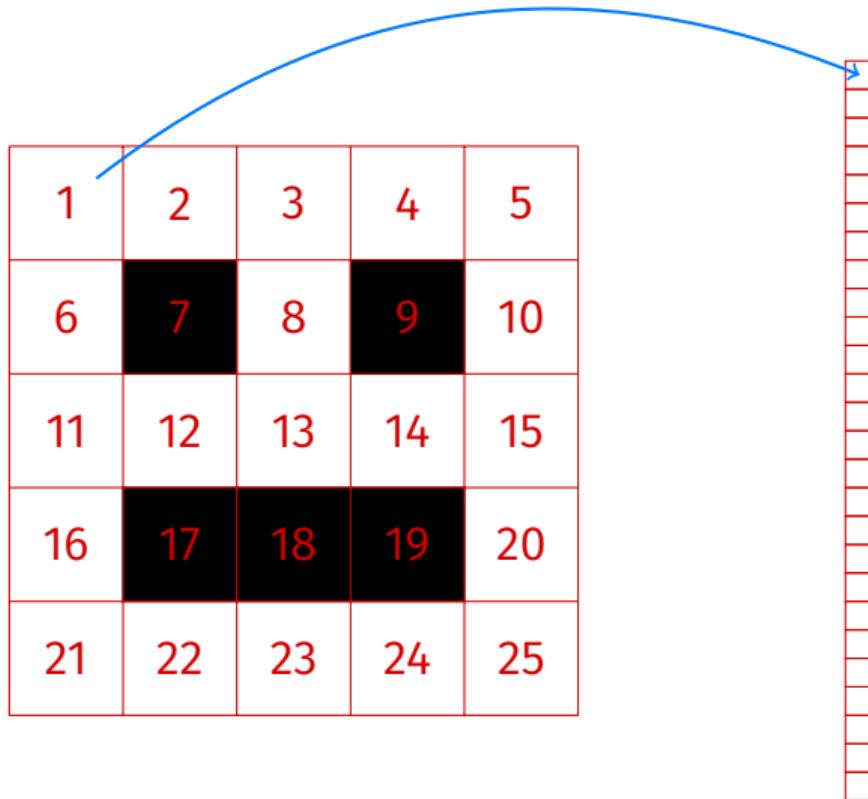


# Images as Feature Vectors

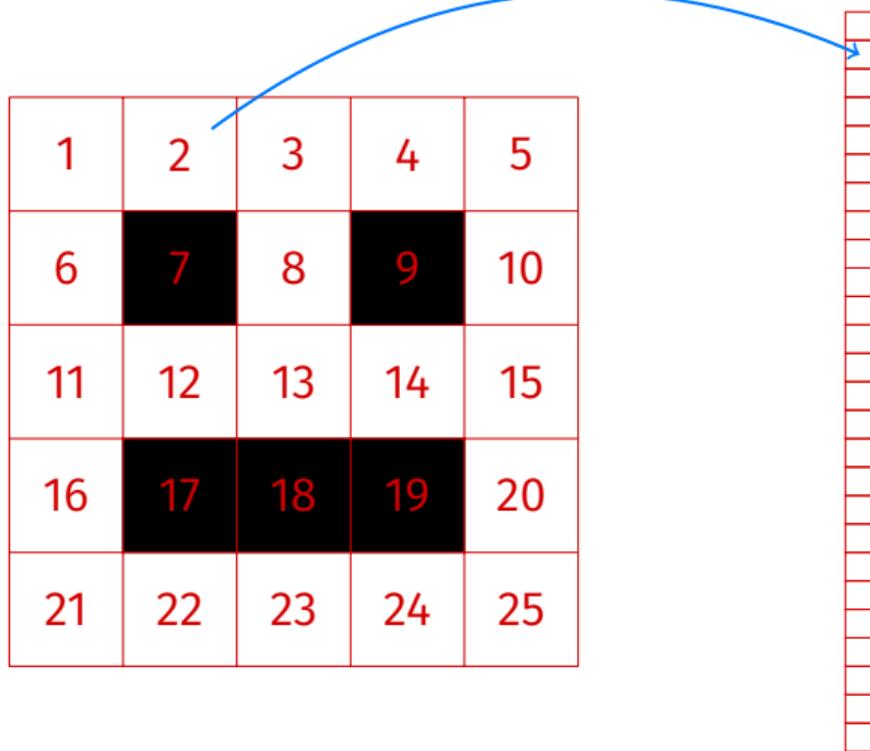
1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25



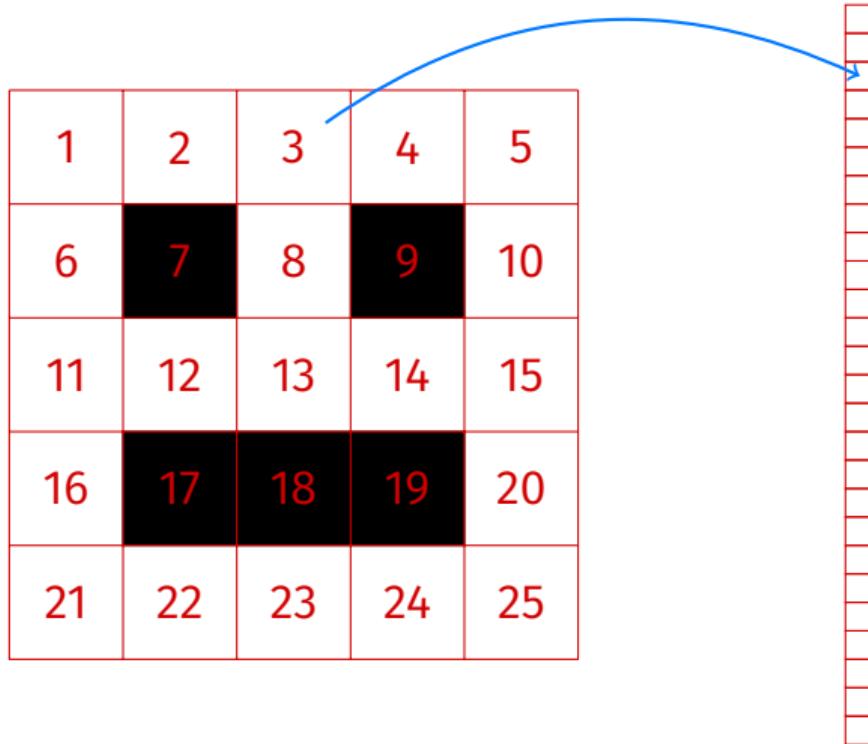
# Images as Feature Vectors



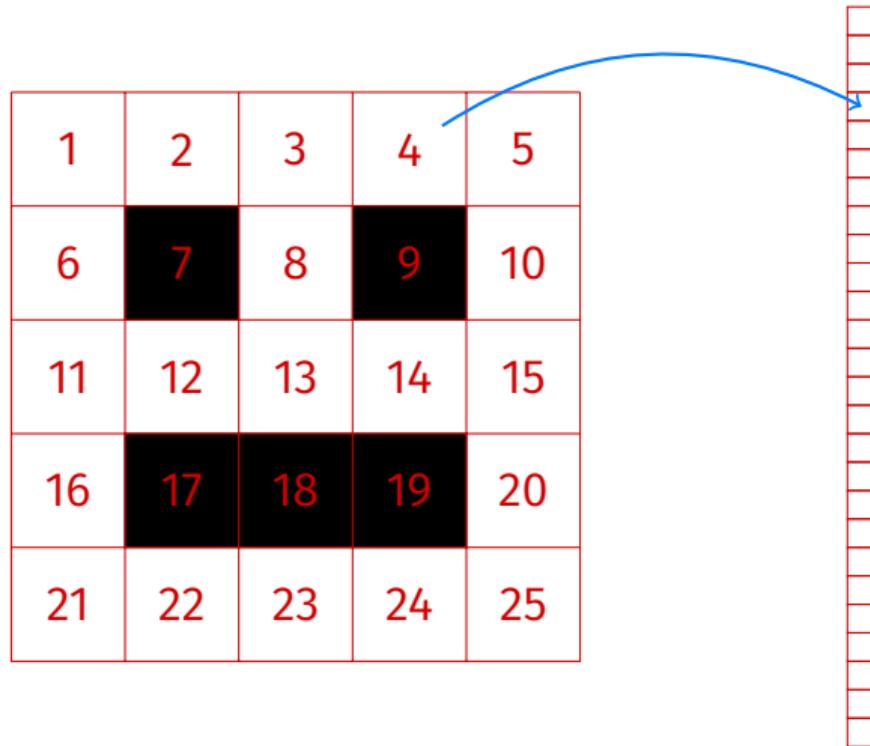
# Images as Feature Vectors



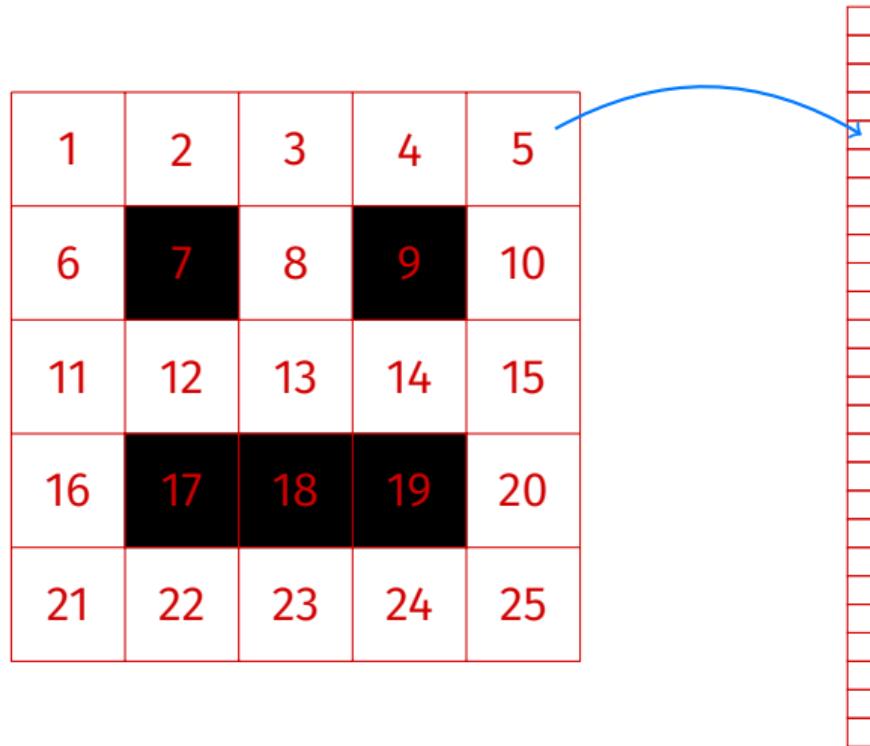
# Images as Feature Vectors



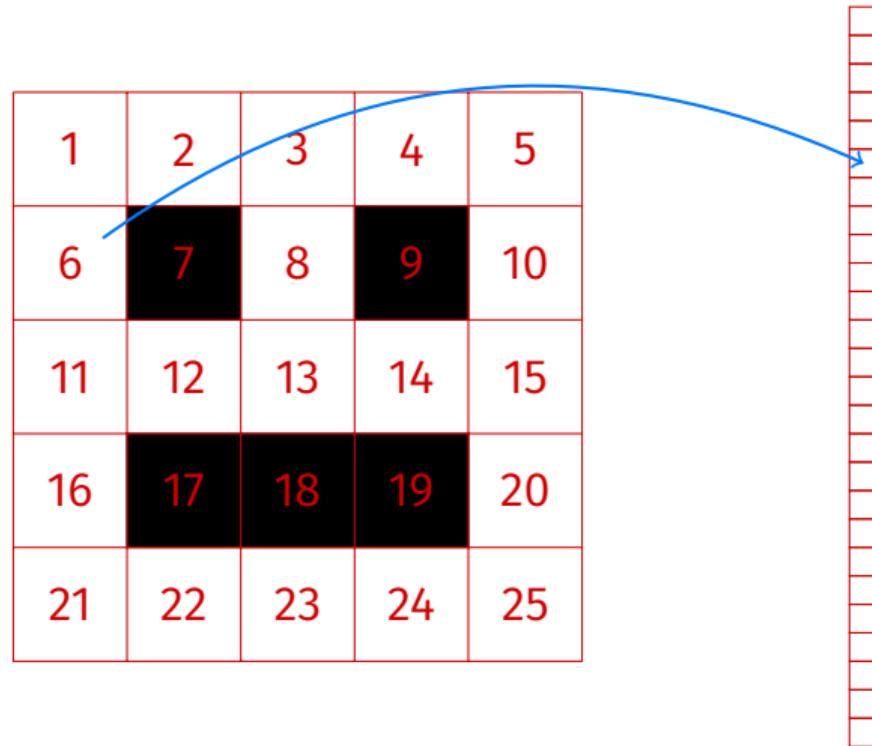
# Images as Feature Vectors



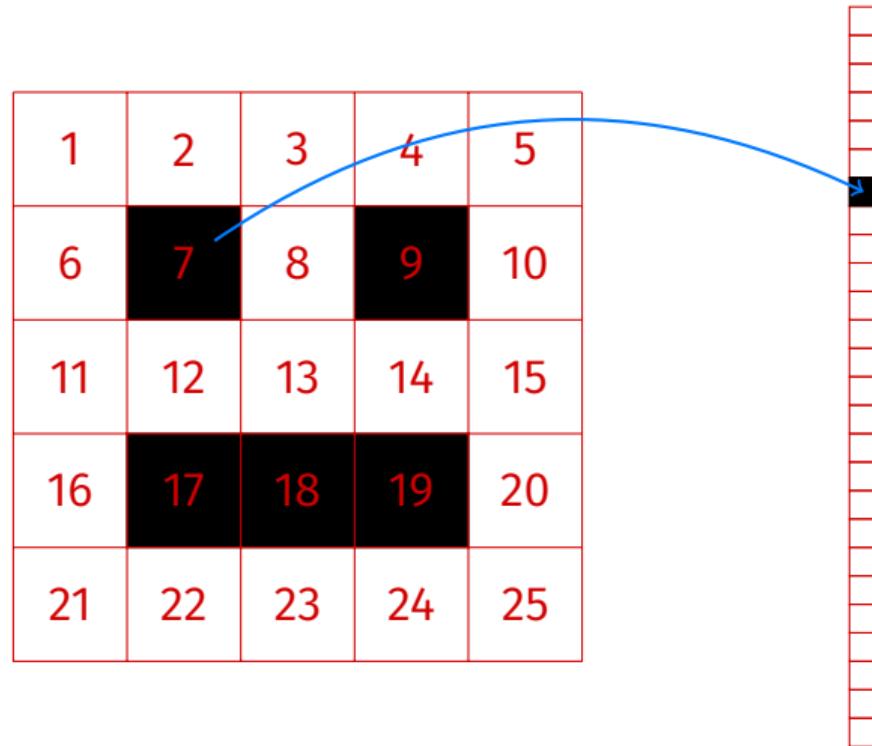
# Images as Feature Vectors



# Images as Feature Vectors



# Images as Feature Vectors



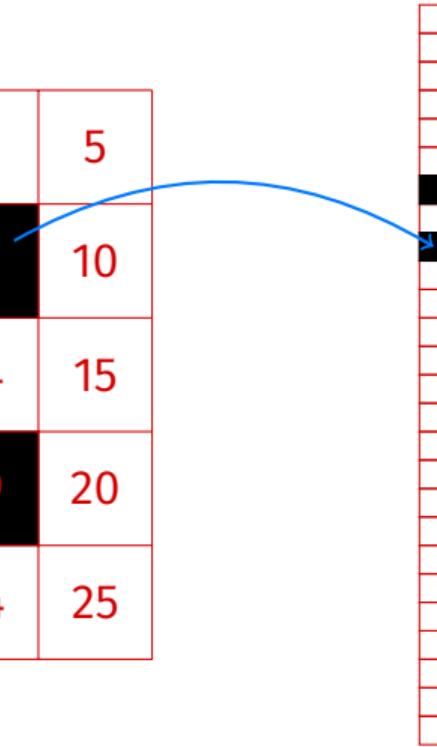
# Images as Feature Vectors

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25



# Images as Feature Vectors

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25



# Images as Feature Vectors

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25



# Euclidean Distance Between Images

- ▶ We “unroll” a  $28 \times 28$  image into a vector in  $\mathbb{R}^{784}$ .
- ▶ Euclidean distance between two images  $\vec{p}^{(1)}, \vec{p}^{(2)}$  in  $\mathbb{R}^{784}$ :

$$\begin{aligned}\|\vec{p}^{(1)} - \vec{p}^{(2)}\| &= \sqrt{\left(p_1^{(1)} - p_1^{(2)}\right)^2 + \left(p_2^{(1)} - p_2^{(2)}\right)^2 + \dots + \left(p_{784}^{(1)} - p_{784}^{(2)}\right)^2} \\ &= \sqrt{\sum_{i=1}^{784} \left(p_i^{(1)} - p_i^{(2)}\right)^2}\end{aligned}$$

# How well does it work?

- ▶ Does this make accurate predictions?
- ▶ Use 1-NN to classify each image in **training set**.
- ▶ **Question:** What will be the accuracy?

$$\text{error} = \frac{\# \text{ incorrect} \text{ predictions}}{60,000}$$

# How well does it work?

- ▶ The error will be 0!
- ▶ Accuracy on training set is **misleading**, overly-optimistic.
- ▶ MNIST also includes a **test set** of 10,000 images and labels. Let's test on this set.

# The Test Error

- ▶ Test set contains 1,000 images from each digit.
- ▶ Suppose our classifier always guesses 7.  
**Question:** what will be the test error?

# The Test Error

- ▶ Test set contains 1,000 images from each digit.
- ▶ Suppose our classifier always guesses 7.  
**Question:** what will be the test error?
- ▶ **Answer:** 90%.

# The Test Error

- ▶ Test set contains 1,000 images from each digit.
- ▶ Suppose our classifier guesses at random.  
**Question:** what is the expected test error?

# The Test Error

- ▶ Test set contains 1,000 images from each digit.
- ▶ Suppose our classifier guesses at random.  
**Question:** what is the expected test error?
- ▶ **Answer:** 90%.

# The Test Error

- ▶ The test error of nearest neighbor is only 3.09%.
- ▶ Examples of errors:

Input:



NN:



# Does k-NN do better?

$k$	1	3	5	7	9	11
Test Error (%):	3.09	2.94	3.13	3.10	3.43	3.34

# DSC 1410A

*Probabilistic Modeling & Machine Learning*

Lecture 01 | Part 4

The End?

# The End?

- ▶ We have developed a simple prediction algorithm:  $k$ -nearest neighbors.
- ▶ Often works well!
- ▶ Have we “solved” machine learning?

# No

- ▶ Nearest neighbor predictors have significant limitations in two areas:
  1. Computational efficiency
  2. Predictive performance

# Computational Efficiency

- ▶ Three main areas to consider when evaluating efficiency:
  1. Time taken to **train** the model.
  2. Memory taken to **store** the trained model.
  3. Time taken to **predict**.

# Time Taken in Training

- ▶ “Training” a NN predictor is trivial.
  - ▶ The training data is simply stored.
- ▶ **Takes very little time.**

# Memory Taken by Trained Model

- ▶ In NN predictors, the model *is* the training data.
- ▶ We store the **entire training data**.
- ▶ Potentially **very costly**.

# Time Taken in Prediction

- ▶ To predict, we search the **entire training set** for the nearest neighbor(s):  $\Theta(nd)$  time.
- ▶ Also potentially **very costly**.

# Analogy: Studying

- ▶ Approach #1: do many practice problem to learn core principles, recognize patterns.
- ▶ I.e., learn “compressed” knowledge.
- ▶ During the exam, answer unseen questions by reasoning.

# **Analogy: Studying**

- ▶ Approach #2: memorize answers to practice problems.
- ▶ During the exam, answer each question by finding most similar practice question.

## **Analogy: Studying**

- ▶ Approach #2 is most similar to nearest neighbor.
- ▶ Can we find methods that work like approach #1?

# Note

- ▶ There are methods for improving the efficiency of NN predictors.
  - ▶ Approximate NN search,  $k - d$  trees, thinning the data, etc.
- ▶ However, they break down in high dimensions (many features).

# Predictive Performance

- ▶ NN predictors can work quite well.
- ▶ Still, often outperformed by other methods, especially when many features are used.

# The Main Problem

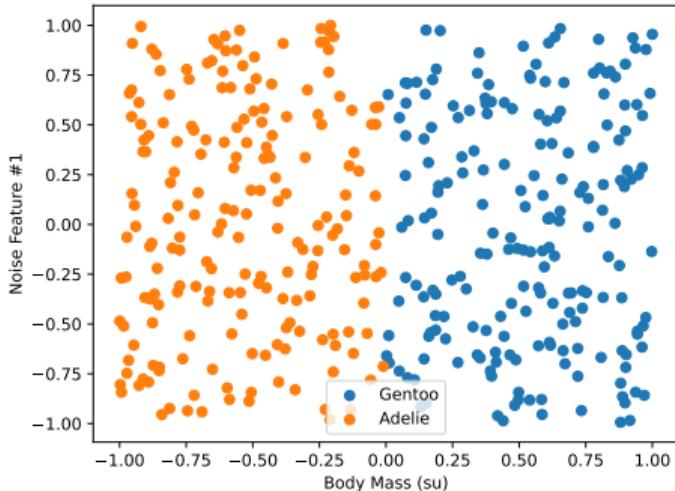
- ▶ Nearest neighbor approaches **do not learn** which features are **useful** and which **are not**.

# Example

- ▶ Suppose all Adelie penguins weigh less than all Gentoo penguins.
- ▶ I.e., we can **predict perfectly** based on body mass alone.

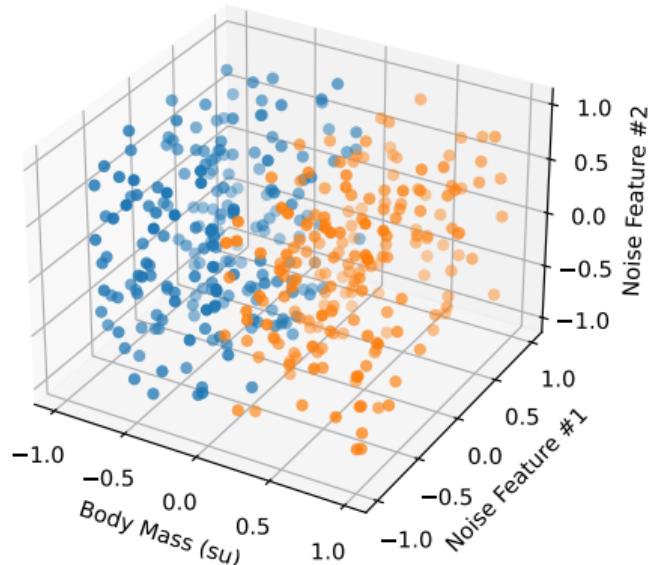
# Example: One Noisy Feature

- ▶ Suppose we add a feature that is total noise.
- ▶ Still enough information to perfectly classify.
- ▶ 1-NN: 98% test accuracy.



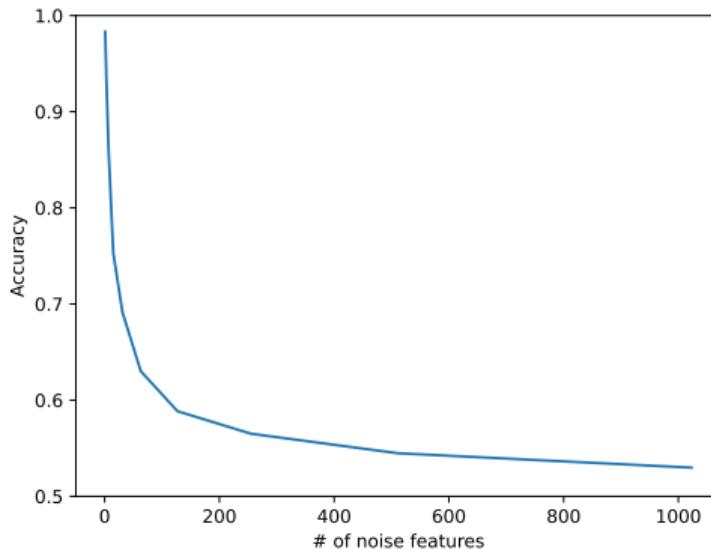
# Example: Two Noisy Features

- ▶ Suppose we add a second feature that is total noise.
- ▶ Still enough information to perfectly classify.
- ▶ 1-NN: 95% test accuracy (**-3%**).



# Example: Noisy Features

- ▶ No matter how many noisy features we add, there is enough information to classify perfectly.
- ▶ But 1-NN performance **degrades** with # of (noisy) features:



# Explanation

- ▶ Euclidean distance treats all features the same.
  - ▶ Even those that are pure noise.
- ▶ NN does not **learn** which features are useful.<sup>2</sup>
- ▶ Distance becomes less meaningful as *noisy* features are added.

---

<sup>2</sup>For extensions of kNN which learn a distance metric from data, see:  
(Weinberger and Saul, 2009; Goldberger et al., 2005; Shalev-Shwartz et al., 2004)

# Summary

- ▶  $k$ NN prediction is simple and can work well.
- ▶ It may be computationally intensive.
- ▶ It does not:
  - ▶ “learn” in the sense of “compressing knowledge”.
  - ▶ learn which features are useful.

## **Next time...**

- ▶ A different approach that attempts to learn a “weight” for each feature.

# DSC 1410A

*Probabilistic Modeling & Machine Learning*

Lecture 02 | Part 1

[News](#)

# News

- ▶ Lab 01 released. Due Sunday @ 11:59 pm.
- ▶ HW 01 released. Due Wednesday @ 11:59 pm.
  - ▶  $\text{\LaTeX}$  template available (optional).

# DSC 140A

Probabilistic Modeling & Machine Learning

Lecture 02 | Part 2

Linear Models

# Last Time: Nearest Neighbors

- ▶ Nearest neighbor methods are simple; can work well.
- ▶ However, they:
  1. “memorize” the training data (**inefficient**);
  2. do not learn relative important of features.

# Example: Predicting Salary

- ▶ **Goal:** predict a data scientist's salary from three features:
  - ▶  $x_1$ : years of experience
  - ▶  $x_2$ : # of interview questions missed
  - ▶  $x_3$ : favorite number
- ▶ **Observations:**
  - ▶  $x_1$  is **positively** associated with salary
  - ▶  $x_2$  is **negatively** associated with salary
  - ▶  $x_3$  is **not** associated with salary

# Prediction Functions

- ▶ **Informally:** we think years of experience, etc., are predictive of salary.
- ▶ **Formally:** we think there is a function  $H$  that takes  $\vec{x} = (x_1, x_2, x_3)$  and outputs a good prediction of salary.

$$H(\vec{x}) \rightarrow \text{prediction}$$

- ▶  $H$  is called a **prediction function**.<sup>1</sup>

---

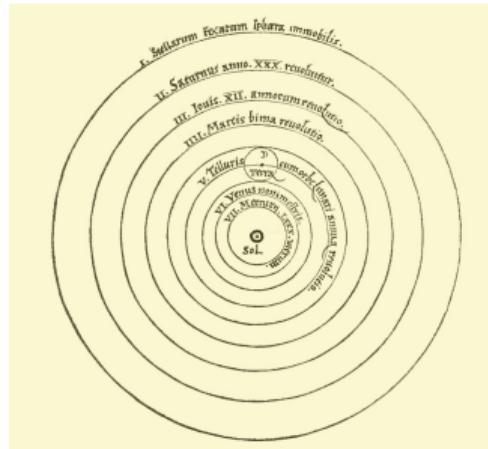
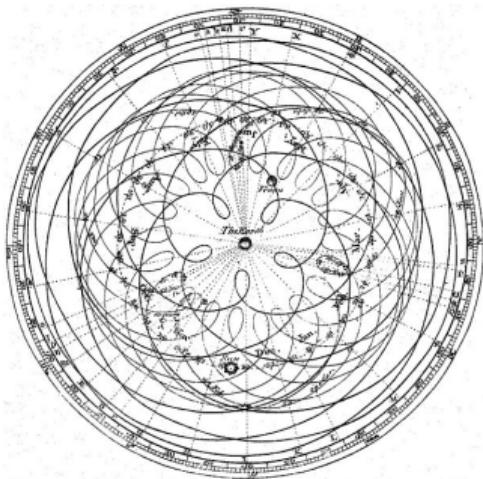
<sup>1</sup>Or, sometimes, a **hypothesis function**

# Prediction Functions

- ▶ **Goal:** find an accurate prediction function.
- ▶ What should our prediction function *look like*?
- ▶ That is, we must choose a **model**.
  - ▶ In context of prediction functions: a **hypothesis class**.

# Occam's Razor

- ▶ **Occam's Razor:** when faced with two competing explanations (models), favor the simpler one.<sup>2</sup>



---

<sup>2</sup>As long as it works, of course.

# Linear Functions

- ▶ **Idea:** model salary as a **weighted sum** of factors.
- ▶ That is, as a **linear function**:

$$H(\vec{x}) = w_0 + w_1x_1 + w_2x_2 + w_3x_3$$

- ▶  $w_0, w_1, \dots, w_3$  are the **parameters** or **weights**.
- ▶ **TODO:** how do we choose the weights?

## Exercise

Recall:

- ▶  $x_1$ : years of experience
- ▶  $x_2$ : # of interview questions missed
- ▶  $x_3$ : favorite number

What are reasonable values of the weights in the linear prediction function  $H(\vec{x}) = w_0 + w_1x_1 + w_2x_2 + w_3x_3$  if it is to be a good predictor of salary?

# Parameter Vectors

- ▶ The parameters of a linear function can be packaged into a **parameter vector**,  $\vec{w}$ .
- ▶ **Example:** if  $H(\vec{x}) = w_0 + w_1x_1 + w_2x_2 + w_3x_3$  then  $\vec{w} = (w_0, \dots, w_3)^T$ .

# Parameterization

- ▶ A linear function  $H(\vec{x})$  is **completely determined** by its parameter vector.
  - ▶ Can work either with the function,  $H$ , or vector,  $\vec{w}$ .
- ▶ Sometimes write  $H(\vec{x}; \vec{w})$ .
- ▶ Example:  $\vec{w} = (8, 3, 1, 5, -2, -7)^T$  specifies

$$H(\vec{x}; \vec{w}) = 8 + 3x_1 + 1x_2 + 5x_3 - 2x_4 - 7x_5$$

# Number of Parameters

- ▶ If a linear predictor  $H(\vec{x}; \vec{w})$  takes in  $d$ -dimensional feature vectors, it has  $d + 1$  parameters.

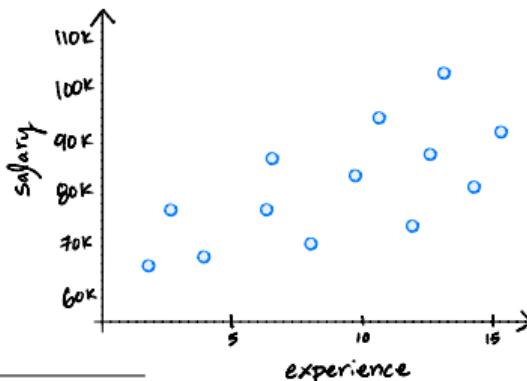
$$\begin{aligned} H(\vec{x}; \vec{w}) &= w_0 + w_1 x_1 + w_2 x_2 + \dots + w_d x_d \\ &= w_0 + \sum_{i=1}^d w_i x_i \end{aligned}$$

- ▶ That is, if  $\vec{x} \in \mathbb{R}^d$ , then  $\vec{w} \in \mathbb{R}^{d+1}$ .

# Visualization

- ▶ Linear prediction rules have linear graphs.<sup>3</sup>
- ▶ **Example:** A linear prediction function for salary.

$$H_1(\vec{x}) = \$50,000 + (\text{experience}) \times \$8,000$$



<sup>3</sup>When visualized in feature space.

# Visualization ( $d > 1$ )

- ▶ The **surface** of a prediction function  $H$  is made by plotting  $H(\vec{x})$  for all  $\vec{x}$ .
- ▶ If  $H$  is a linear prediction function, and
  - ▶  $\vec{x} \in \mathbb{R}^1$ , then  $H(x)$  is a straight line.
  - ▶  $\vec{x} \in \mathbb{R}^2$ , then  $H(\vec{x})$  is a plane.
  - ▶  $\vec{x} \in \mathbb{R}^d$ , then  $H(\vec{x})$  is a  $d$ -dimensional **hyperplane**.

# Note: Compact Form

- ▶ Recall the **dot product** of vectors  $\vec{a}$  and  $\vec{b}$ :

$$\vec{a} = (a_1, a_2, \dots, a_d)^T \quad \vec{b} = (b_1, b_2, \dots, b_d)^T$$

$$\vec{a} \cdot \vec{b} = a_1 b_1 + a_2 b_2 + \dots + a_d b_d$$

- ▶ Observe:

$$\begin{aligned} H(\vec{x}; \vec{w}) &= w_0 + w_1 x_1 + \dots + w_d x_d \\ &= \underbrace{(w_0, w_1, \dots, w_d)^T}_{\vec{w}} \cdot \underbrace{(1, x_1, \dots, x_d)^T}_{?} \end{aligned}$$

# Note: Compact Form

- ▶ The **augmented feature vector**  $\text{Aug}(\vec{x})$  is the vector obtained by adding a 1 to the front of  $\vec{x}$ :

$$\vec{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix} \quad \text{Aug}(\vec{x}) = \begin{pmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix}$$

- ▶ With augmentation, we can write:

$$\begin{aligned} H(\vec{x}) &= w_0 + w_1 x_1 + w_2 x_2 + \dots + w_d x_d \\ &= \vec{w} \cdot \text{Aug}(\vec{x}) \end{aligned}$$

# Classification?

- ▶ We have been focusing on **regression**.
- ▶ Linear prediction functions can be used for **classification**, too.
- ▶ We will come back to this.

# DSC 140A

*Probabilistic Modeling & Machine Learning*

Lecture 02 | Part 3

**Empirical Risk Minimization**

# Picking a Prediction Function

- ▶ Suppose we model salary as a linear function:

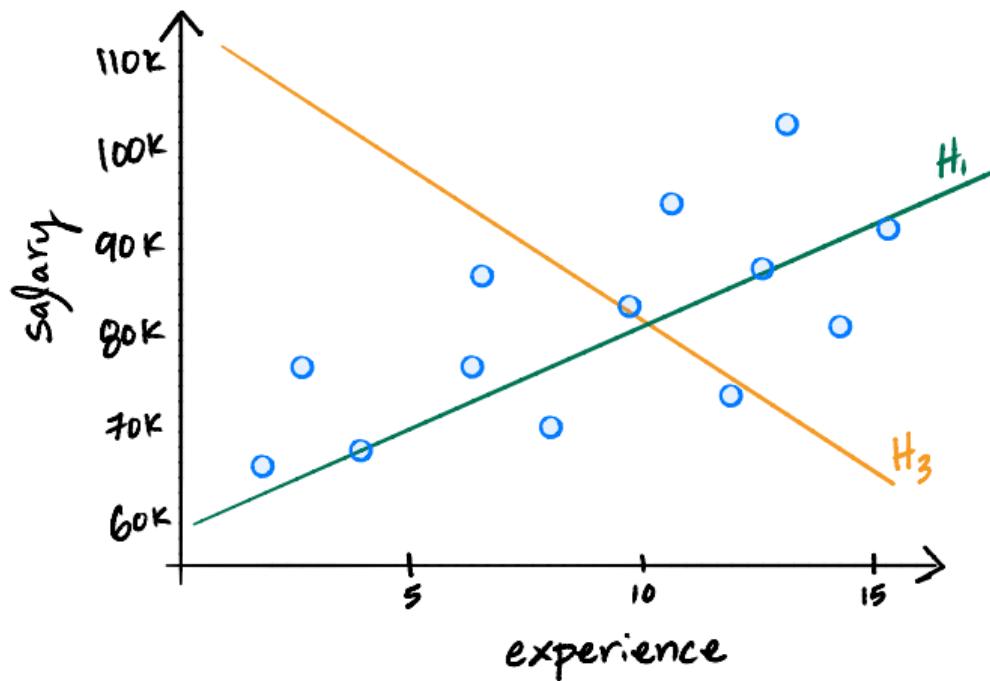
$$H(\vec{x}; \vec{w}) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3$$

- ▶ **Question:** how do we choose weights  $w_0, \dots, w_3$  so that  $H$  makes good predictions?

# Learning

- ▶ **Assumption:** the future will look like the past.
- ▶ *If so*, we should pick a prediction function that worked well on past data.
- ▶ That is, we should **learn** a function from data.

# Example

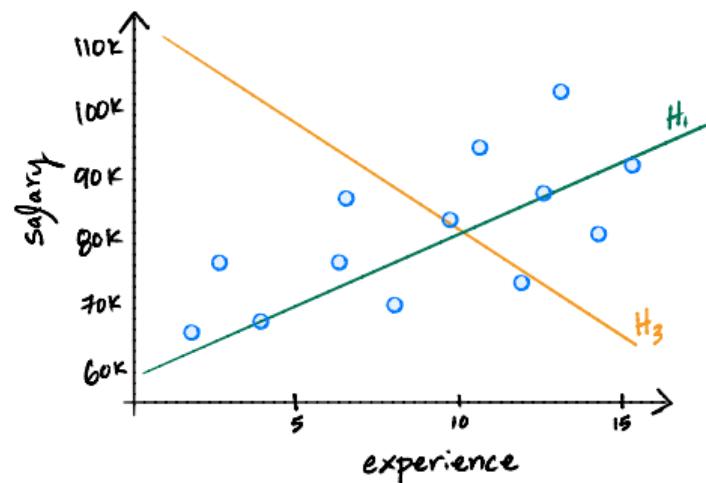


# Training Data

- ▶ To learn, we gather **training data**.
- ▶ A set  $\mathcal{D}$  of  $n$  pairs:  $(\vec{x}^{(i)}, y_i)$ 
  - ▶  $\vec{x}^{(i)}$  is the  $i$ th **feature vector**
  - ▶  $y_i$  is its **label** (the correct answer)
- ▶ In regression,  $y_i$  is a continuous number; in classification, it is discrete.
- ▶ This regime is called **supervised learning**.

# An Optimization Problem

- ▶ Some prediction functions “fit” the data better than others.
- ▶ **Idea:** find the function that “fits best”



# Quantifying Fit

- ▶ How do we measure “fit”?
- ▶ Formally: measure difference between our prediction  $H(\vec{x}^{(i)})$  and the “right answer”,  $y_i$ .
- ▶ A **loss function** quantifies how wrong a single prediction is.
- ▶ **Example:** the **absolute loss**  
 $\ell_{\text{abs}}(H(\vec{x}^{(i)}), y_i) = |H(\vec{x}^{(i)}) - y_i|$

# Quantifying Overall Fit

- ▶ **Idea:** a good  $H$  makes good predictions *on average* over entire data set.
- ▶ Find  $H$  minimizing the **expected loss**, also called the **empirical risk**:

$$R(H) = \sum_{i=1}^n \ell(H(\vec{x}^{(i)}), y_i)$$

- ▶ Note:  $R$  depends on both  $H$  and the data!

# Empirical Risk Minimization

- ▶ This strategy is called **empirical risk minimization (ERM)**.
- ▶ Step 1: choose a **hypothesis class**
  - ▶ Let's assume we've chosen linear predictors
- ▶ Step 2: choose a **loss function**
- ▶ Step 3: minimize **expected loss (empirical risk)**

# ERM for Regression

- ▶ We have chosen as our hypothesis class the set of **linear functions**  $\mathbb{R}^d \rightarrow \mathbb{R}$ .
- ▶ Suppose we choose **absolute loss**:

$$\ell_{\text{abs}}(H(\vec{x}^{(i)}), y_i) = |H(\vec{x}^{(i)}) - y_i|$$

- ▶ **Goal:** find  $H$  minimizing **mean absolute error**:

$$R_{\text{abs}}(H) = \sum_{i=1}^n |H(\vec{x}^{(i)}) - y_i|$$

# Minimizing Mean *Absolute* Error

- ▶ **Goal:** out of all **linear** functions  $\mathbb{R}^d \rightarrow \mathbb{R}$ , find the function  $H^*$  with the smallest mean absolute error on the training set.
- ▶ That is, find:

$$H^* = \arg \min_{\text{linear } H} \frac{1}{n} \sum_{i=1}^n |H(x_i) - y_i|$$

# Minimizing Mean Absolute Error

- ▶ Assume for now that  $d = 1$  (one feature). Then  $w \in \mathbb{R}^2$  and:

$$H(x; \vec{w}) = w_0 + w_1 x$$

- ▶ Recall that  $H$  is completely determined by  $w_0, w_1$ .
- ▶ Equivalent goal: find  $w_0$  and  $w_1$  minimizing

$$\frac{1}{n} \sum_{i=1}^n |H(x; w_0, w_1) - y_i|$$

# Minimizing Mean Absolute Error

- ▶ To find optimal  $w_0$  and  $w_1$ , might use calculus.
  - ▶ Set  $\partial R / \partial w_0 = 0$  and  $\partial R / \partial w_1 = 0$  and solve.
- ▶ Problem: absolute value is **not differentiable!**
- ▶ It is hard to minimize the mean absolute error.<sup>4</sup>
- ▶ What can we do?

---

<sup>4</sup>Though it can be done with linear programming.

# Minimizing Mean *Squared* Error

- The **square loss** is differentiable:

$$\ell_{\text{sq}}(H(\vec{x}), y) = (H(\vec{x}) - y)^2$$

- Let's try minimizing the mean squared error instead.

## Main Idea

We often choose a loss function out of practical considerations.

# DSC 140A

*Probabilistic Modeling & Machine Learning*

Lecture 02 | Part 4

**Minimizing the MSE**

# Our Goal

- ▶ Out of all **linear** functions  $\mathbb{R} \rightarrow \mathbb{R}$ , find the function  $H^*$  with the smallest **mean squared error**.

- ▶ That is, find:

$$H^* = \arg \min_{\text{linear } H} \frac{1}{n} \sum_{i=1}^n (H(x_i) - y_i)^2$$

- ▶ This problem is called **least squares regression**.

## For now...

- ▶ For simplicity, assume that there is only one feature (predictor variable).
  - ▶  $H(x; \vec{w}) = w_0 + w_1 x$
  - ▶ I.e., one-dimensional linear regression.
- ▶ We will come back to multi-dimensional case in the next lecture.

# Minimizing the MSE

- ▶ The MSE is a function of a function:

$$R_{\text{sq}}(H) = \frac{1}{n} \sum_{i=1}^n (H(x_i) - y_i)^2$$

- ▶ But since  $H$  is linear,  $H(x) = w_1 x + w_0$ .

$$R_{\text{sq}}(w_1, w_0) = \frac{1}{n} \sum_{i=1}^n ((w_1 x_i + w_0) - y_i)^2$$

- ▶ Now it's a function of  $w_1, w_0$ .

# Updated Goal

- ▶ Find slope  $w_1$  and intercept  $w_0$  which minimize the MSE,  $R_{\text{sq}}(w_1, w_0)$ :

$$R_{\text{sq}}(w_1, w_0) = \frac{1}{n} \sum_{i=1}^n ((w_1 x_i + w_0) - y_i)^2$$

- ▶ Strategy: multivariate calculus.

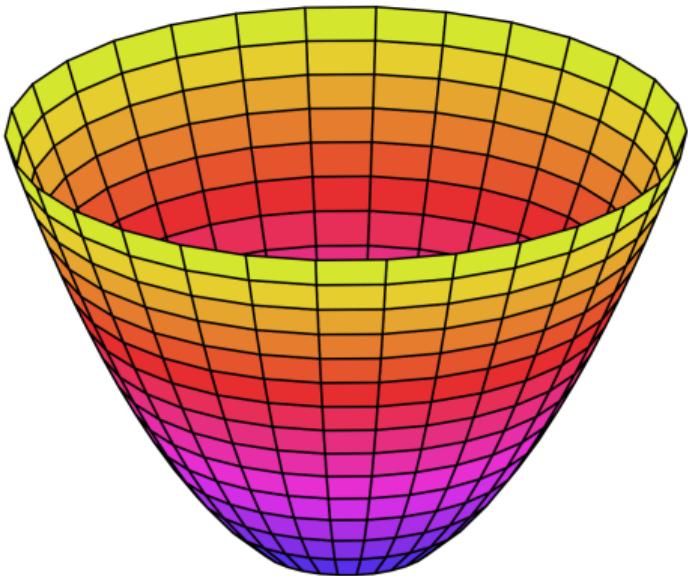
## Exercise

Suppose we plotted  $R_{sq}(w_1, w_0)$ . What would it look like?

$$R_{sq}(w_1, w_0) = \frac{1}{n} \sum_{i=1}^n ((w_1 x_i + w_0) - y_i)^2$$

- ▶ Can  $R_{sq}$  be negative?
- ▶ Can it be zero?
- ▶ How many minima / maxima?

# Answer



# Recall: the gradient

- ▶ If  $f(x, y)$  is a function of two variables, the **gradient** of  $f$  at the point  $(x_0, y_0)$  is a **vector** of **partial derivatives**:

$$\nabla f(x_0, y_0) = \begin{pmatrix} \frac{\partial f}{\partial x}(x_0) \\ \frac{\partial f}{\partial y}(y_0) \end{pmatrix}$$

- ▶ **Key Fact:** gradient is zero at critical points.

# Strategy

To minimize  $R(w_1, w_0)$ : compute the gradient, set equal to zero, solve.

$$R_{\text{sq}}(w_1, w_0) = \frac{1}{n} \sum_{i=1}^n ((w_1 x_i + w_0) - y_i)^2$$

$$\frac{\partial R_{\text{sq}}}{\partial w_1} =$$

$$R_{\text{sq}}(w_1, w_0) = \frac{1}{n} \sum_{i=1}^n ((w_1 x_i + w_0) - y_i)^2$$

$$\frac{\partial R_{\text{sq}}}{\partial w_0} =$$

# Strategy

$$0 = \frac{2}{n} \sum_{i=1}^n ((w_1 x_i + w_0) - y_i) x_i \quad 0 = \frac{2}{n} \sum_{i=1}^n ((w_1 x_i + w_0) - y_i)$$

1. Solve for  $w_0$  in second equation.
2. Plug solution for  $w_0$  into first equation, solve for  $w_1$ .

# Solve for $w_0$

$$0 = \frac{2}{n} \sum_{i=1}^n ((w_1 x_i + w_0) - y_i)$$

# Solve for $w_0$

$$0 = \frac{2}{n} \sum_{i=1}^n ((w_1 x_i + w_0) - y_i)$$

# Key Fact

- ▶ Define

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i \quad \bar{y} = \frac{1}{n} \sum_{i=1}^n y_i$$

- ▶ Then

$$\sum_{i=1} (x_i - \bar{x}) = 0 \quad \sum_{i=1} (y_i - \bar{y}) = 0$$

# Solve for $w_1$

$$0 = \frac{2}{n} \sum_{i=1}^n ((w_1 x_i + w_0) - y_i) x_i \quad w_0 = \bar{y} - w_1 \bar{x}$$

# Solve for $w_1$

$$0 = \frac{2}{n} \sum_{i=1}^n ((w_1 x_i + w_0) - y_i) x_i \quad w_0 = \bar{y} - w_1 \bar{x}$$

# Least Squares Solutions

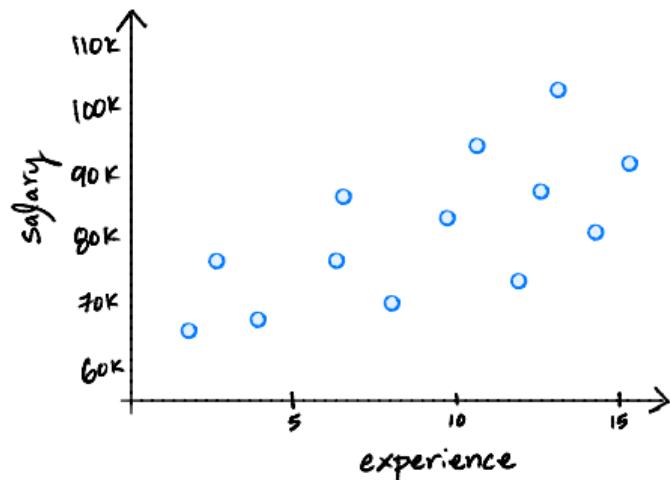
- ▶ The **least squares solutions** for the slope  $w_1$  and intercept  $w_0$  are:

$$w_1 = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2} \quad w_0 = \bar{y} - w_1 \bar{x}$$

$$\text{where } \bar{x} = \frac{1}{n} \sum_{i=1}^n x_i \quad \bar{y} = \frac{1}{n} \sum_{i=1}^n y_i$$

# Interpretation of Slope

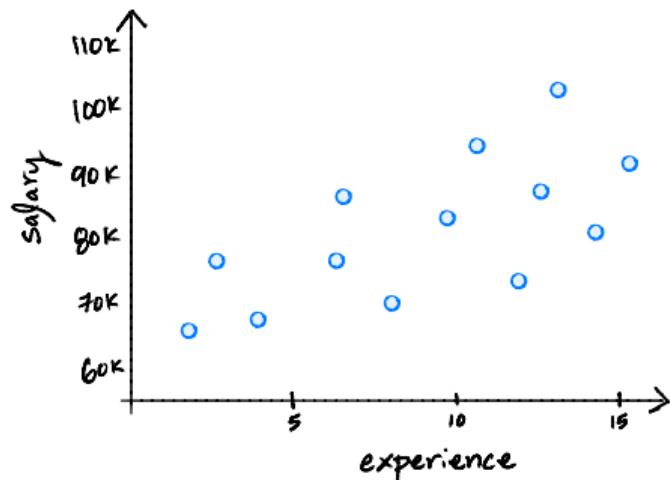
$$w_1 = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2}$$



- ▶ What is the sign of  $(x_i - \bar{x})(y_i - \bar{y})$  when:
  - ▶  $x_i > \bar{x}$  and  $y_i > \bar{y}$ ?

# Interpretation of Slope

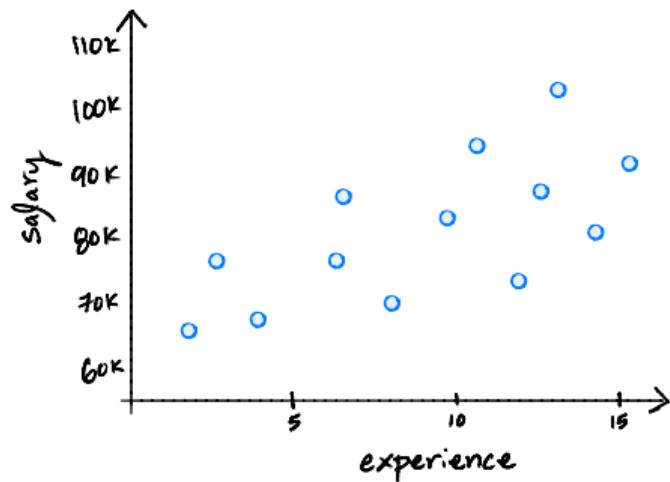
$$w_1 = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2}$$



- ▶ What is the sign of  $(x_i - \bar{x})(y_i - \bar{y})$  when:
  - ▶  $x_i < \bar{x}$  and  $y_i < \bar{y}$ ?

# Interpretation of Slope

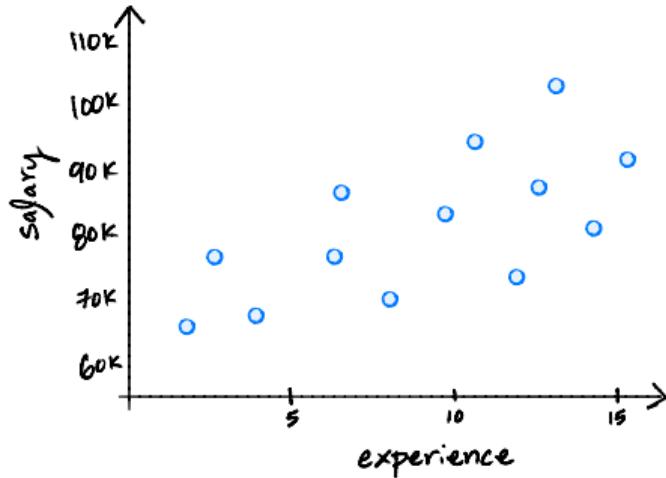
$$w_1 = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2}$$



- ▶ What is the sign of  $(x_i - \bar{x})(y_i - \bar{y})$  when:
  - ▶  $x_i > \bar{x}$  and  $y_i < \bar{y}$ ?

# Interpretation of Slope

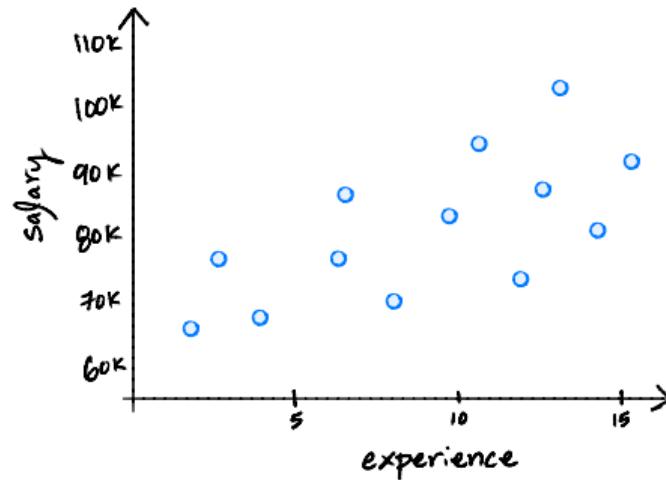
$$w_1 = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2}$$



- ▶ What is the sign of  $(x_i - \bar{x})(y_i - \bar{y})$  when:
  - ▶  $x_i < \bar{x}$  and  $y_i > \bar{y}$ ?

# Interpretation of Intercept

$$w_0 = \bar{y} - w_1 \bar{x}$$

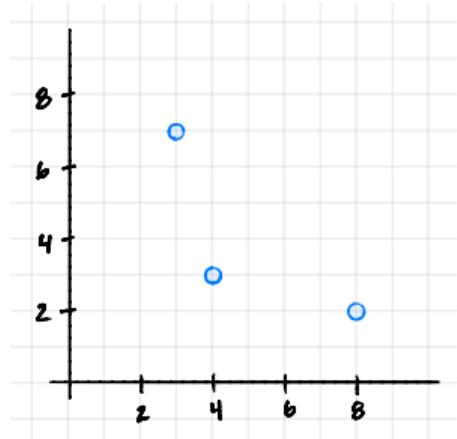


- ▶ What is  $H(\bar{x})$ ?

# Question

We fit a linear prediction rule for salary given years of experience. Then everyone gets a \$5,000 raise. What happens to slope/intercept?

# Example



$$\bar{x} =$$

$$\bar{y} =$$

$$w_1 = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2} =$$

$$w_0 = \bar{y} - w_1 \bar{x}$$

$x_i$	$y_i$	$(x_i - \bar{x})$	$(y_i - \bar{y})$	$(x_i - \bar{x})(y_i - \bar{y})$	$(x_i - \bar{x})^2$
3	7	-1	4	-4	1
4	3	0	-4	0	0
8	2	5	-6	-30	25

# DSC 140A

*Probabilistic Modeling & Machine Learning*

Lecture 02 | Part 5

**Fitting Non-Linear Trends**

# Non-Linear Trends

- ▶ We have fit a straight line of the form:

$$H(x) = w_0 + w_1 x$$

- ▶ What if we believe, e.g., salary grows with the **square** of experience?
- ▶ I.e., how do we fit a function of the form:

$$H(x) = w_0 + w_1 x^2?$$

# “Linear” Models

- ▶ The **linear** in **linear prediction function** refers to the weights, not the features.
- ▶ These are all **linear** prediction functions:
  - ▶  $H(x) = w_0 + w_1x + w_2x^2$
  - ▶  $H(x) = w_0 + w_1e^x$
  - ▶  $H(x) = w_0 + w_1\sqrt{x} + w_2 \sin x$
- ▶ These are **not**:
  - ▶  $H(x) = w_0 + w_1e^{w_2x}$
  - ▶  $H(x) = w_0 + w_1 \sin(w_2x)$

# In General

- ▶  $H(x) = w_0 + w_1\phi(x)$  is a linear model, no matter what  $\phi$  is.<sup>5</sup>
- ▶  $\phi$  is called a **basis function** (or **feature map**).
- ▶ Example:  $\phi(x) = x^2$

---

<sup>5</sup>Provided  $\phi$  does not involve  $w_0$  and  $w_1$

# Minimizing Mean Squared Error

- ▶ Fix a basis function  $\phi(x)$ .
- ▶ **Goal:** pick  $w_0$  and  $w_1$  so as to minimize the mean squared error of  $H$ :

$$R_{\text{sq}}(w_0, w_1) = \frac{1}{n} \sum_{i=1}^n [(w_0 + w_1 \phi(x_i)) - y_i]^2$$

# Minimizing Mean Squared Error

- ▶ Notation: define  $z_i = \phi(x_i)$ .
- ▶ Strategy: compute  $\partial R_{\text{sq}} / \partial w_0$  and  $\partial R_{\text{sq}} / \partial w_1$ , set to zero, solve.

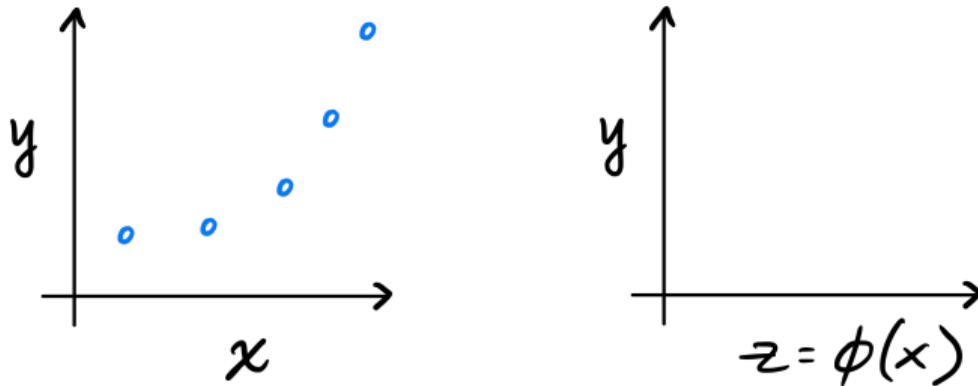
# Solution

- ▶ **Observation:** This is the **exact same** calculation we've done, but with  $x_i$  replaced by  $z_i$ .
- ▶ The **least squares solutions:**

$$w_1 = \frac{\sum_{i=1}^n (z_i - \bar{z})(y_i - \bar{y})}{\sum_{i=1}^n (z_i - \bar{z})^2} \quad w_0 = \bar{y} - w_1 \bar{z}$$

- ▶ where  $\bar{z} \equiv \frac{1}{n} \sum_{i=1}^n \phi(x_i)$

# Intuition



$x$	1	2	3	4
<hr/>				
$y$	2	8	18	32
<hr/>				
$z = x^2$	1	4	9	16

# Interpretation

- ▶ To fit a function  $H(x) = w_0 + w_1\phi(x)$ :
  1. Create new data set  $\{(z_i, y_i)\}$ , where  $z_i = \phi(x_i)$ .
  2. Fit a straight line  $H(z) = w_0 + w_1 z$  on this new data.
  3. Use  $w_0$  and  $w_1$  in  $H(x) = w_0 + w_1\phi(x)$

# Summary

- ▶ We have seen how to fit linear prediction functions of the form:

$$H(x) = w_0 + w_1 \phi(x)$$

- ▶ **Next time:** how do we fit functions of the form:

$$H(x_1, x_2, \dots) = w_0 + w_1 \phi(x_1) + w_2 \phi(x_2) + \dots$$

- ▶ How does this compare to nearest neighbor methods?

# DSC 140A

*Probabilistic Modeling & Machine Learning*

Lecture 3 | Part 1

**Least Squares Regression ( $d > 1$ )**

# Last Time

- ▶ Introduced **linear models**:

$$\begin{aligned}H(\vec{x}) &= w_0 + w_1x_1 + w_2x_2 + \dots + w_dx_d \\&= \text{Aug}(\vec{x}) \cdot \vec{w}\end{aligned}$$

# Last Time

- ▶ Introduced **empirical risk minimization (ERM)**:
- ▶ Step 1: choose a **hypothesis class**
  - ▶ Let's assume we've chosen linear predictors
- ▶ Step 2: choose a **loss function**
- ▶ Step 3: minimize **expected loss (empirical risk)**

# Last Time

- ▶ Chose the **square loss**.
- ▶ Minimized the expected square loss for models with *one* feature (predictor variable):

$$H(x) = w_0 + w_1 x$$

- ▶ The **least squares solutions**:

$$w_1 = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2}$$
$$w_0 = \bar{y} - w_1 \bar{x}$$

# **Today**

- ▶ Least squares solutions for > 1 feature.
- ▶ What about classification?

# Setting

- ▶ A training set consisting of  $n$  pairs,  $(\vec{x}^{(i)}, y_i)$ :
  - ▶  $\vec{x}^{(i)} \in \mathbb{R}^d$  is the  $i$ th **feature vector**;
  - ▶  $y_i \in \mathbb{R}$  is the  $i$ th **target**

# Our Goal

- ▶ Out of all **linear** functions  $\mathbb{R}^d \rightarrow \mathbb{R}$ , find the function  $H^*$  with the smallest **mean squared error** w.r.t. the training data.
- ▶ That is, find linear  $H$  minimizing:

$$R_{\text{sq}}(H) = \frac{1}{n} \sum_{i=1}^n (H(\vec{x}^{(i)}) - y_i)^2$$

- ▶ This problem is called **least squares regression**.

# Recall

- ▶ A linear function  $H(\vec{x}; \vec{w}) = w_0 + w_1x_1 + \dots + w_dx_d$  is **completely determined** by its parameter vector,  $\vec{w}$ :

$$H(\vec{x}; \vec{w}) = \text{Aug}(\vec{x}) \cdot \vec{w}$$

- ▶ **Updated goal:** find  $\vec{w} \in \mathbb{R}^{d+1}$  minimizing:

$$R_{\text{sq}}(\vec{w}) = \frac{1}{n} \sum_{i=1}^n (\text{Aug}(\vec{x}^{(i)}) \cdot \vec{w} - y_i)^2$$

# **Two Approaches**

- ▶ **Approach #1:** Vector Calculus
- ▶ **Approach #2:** Orthogonal Projection (Geometry)
- ▶ Different ways of arriving at same solutions.

# Approach #1: Vector Calculus

- ▶ Find gradient of  $R_{\text{sq}}(\vec{w})$ ; set to zero; solve for  $\vec{w}$ .
- ▶ Essentially what was done last time.
- ▶ Now: there are many more than 2 parameters.
- ▶ Rely on results from vector calculus.

# First Step: Rewrite Risk

- ▶ Step one: rewrite  $R_{\text{sq}}$  in vector form.
- ▶ We'll find:

$$\begin{aligned} R_{\text{sq}}(\vec{w}) &= \frac{1}{n} \sum_{i=1}^n (\text{Aug}(\vec{x}^{(i)}) \cdot \vec{w} - y_i)^2 \\ &= \frac{1}{n} \|X\vec{w} - \vec{y}\|^2 \end{aligned}$$

# Recall

- ▶ If  $\vec{u} = (u_1, u_2, \dots, u_k)^T$ , then:

$$\|\vec{u}\|^2 = \vec{u} \cdot \vec{u} = \sum_{i=1}^k u_i^2$$

- ▶ So, if  $\vec{a} = (a_1, \dots, a_k)^T$  and  $\vec{b} = (b_1, \dots, b_k)^T$ :

$$\begin{aligned}\|\vec{a} - \vec{b}\|^2 &= (\vec{a} - \vec{b}) \cdot (\vec{a} - \vec{b}) \\ &= \sum_{i=1}^k (a_i - b_i)^2\end{aligned}$$

# First Step: Rewrite Risk

- ▶ Define  $p_i = \text{Aug}(\vec{x}^{(i)}) \cdot \vec{w}$ , and let  $\vec{p} = (p_1, \dots, p_n)^T$ .
  - ▶  $\vec{p}$  is a vector of the predictions on training set.
  - ▶ Note:  $\vec{p} \in \mathbb{R}^n$ , not  $\mathbb{R}^d$ !
- ▶ Then:

$$\begin{aligned} R_{\text{sq}}(\vec{w}) &= \frac{1}{n} \sum_{i=1}^n (\text{Aug}(\vec{x}^{(i)}) \cdot \vec{w} - y_i)^2 \\ &= \frac{1}{n} \sum_{i=1}^n (p_i - y_i)^2 \\ &= \frac{1}{n} \|\vec{p} - \vec{y}\|^2 \end{aligned}$$

# First Step: Rewrite Risk

- ▶ Define the (augmented) **design matrix**,  $X$ :

$$X = \begin{pmatrix} \text{Aug}(\vec{x}^{(1)}) & \longrightarrow \\ \text{Aug}(\vec{x}^{(2)}) & \longrightarrow \\ \vdots & \vdots \\ \text{Aug}(\vec{x}^{(n)}) & \longrightarrow \end{pmatrix} = \begin{pmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \dots & x_d^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \dots & x_d^{(2)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_1^{(n)} & x_2^{(n)} & \dots & x_d^{(n)} \end{pmatrix}$$

# First Step: Rewrite Risk

► Observe:  $\vec{p} = X\vec{w}$ .

$$\underbrace{\begin{pmatrix} \text{Aug}(\vec{x}^{(1)}) \\ \text{Aug}(\vec{x}^{(2)}) \\ \vdots \\ \text{Aug}(\vec{x}^{(n)}) \end{pmatrix}}_X \underbrace{\begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{pmatrix}}_{\vec{w}} = \underbrace{\begin{pmatrix} \text{Aug}(\vec{x}^{(1)}) \cdot \vec{w} \\ \text{Aug}(\vec{x}^{(2)}) \cdot \vec{w} \\ \vdots \\ \text{Aug}(\vec{x}^{(n)}) \cdot \vec{w} \end{pmatrix}}_{\vec{p}}$$

# First Step: Rewrite Risk

- ▶ Therefore, the MSE can be written:

$$\begin{aligned} R_{\text{sq}}(\vec{w}) &= \frac{1}{n} \sum_{i=1}^n (\text{Aug}(\vec{x}^{(i)}) \cdot \vec{w} - y_i)^2 \\ &= \frac{1}{n} \sum_{i=1}^n (p_i - y_i)^2 \\ &= \frac{1}{n} \|\vec{p} - \vec{y}\|^2 \\ &= \frac{1}{n} \|X\vec{w} - \vec{y}\|^2 \end{aligned}$$

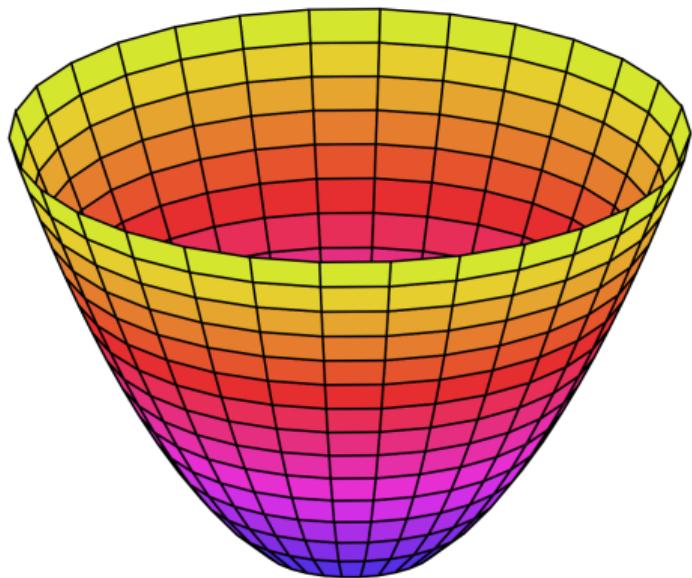
# Goal

- ▶ Find  $\vec{w} \in \mathbb{R}^{d+1}$  minimizing:

$$R_{\text{sq}}(\vec{w}) = \frac{1}{n} \|X\vec{w} - \vec{y}\|^2$$

- ▶ **Step Two:** find gradient, set to zero, solve.

# Recall: the Risk Surface



# Step Two: Find Gradient

- We want to compute:

$$\frac{d}{d\vec{w}} [R_{\text{sq}}(\vec{w})] = \frac{d}{d\vec{w}} \left[ \frac{1}{n} \|X\vec{w} - \vec{y}\|^2 \right]$$

- $\frac{dR_{\text{sq}}}{d\vec{w}}$  is the **gradient** of  $R_{\text{sq}}$ .
- It is the vector of partial derivatives:

$$\frac{dR_{\text{sq}}}{d\vec{w}} = \left( \frac{\partial R_{\text{sq}}}{\partial w_0}, \frac{\partial R_{\text{sq}}}{\partial w_1}, \dots, \frac{\partial R_{\text{sq}}}{\partial w_d} \right)^T$$

# Idea

- ▶ While we could compute each of:  $\frac{\partial R_{\text{sq}}}{\partial w_0}, \frac{\partial R_{\text{sq}}}{\partial w_1}, \dots$
- ▶ Instead use rules from vector calculus.
- ▶ Example:  $\frac{d}{d\vec{w}} [\vec{y}^T X \vec{w}] = X^T \vec{y}$

# Good to know...

$$(A + B)^T = A^T + B^T$$

$$(AB)^T = B^T A^T$$

$$\vec{u} \cdot \vec{v} = \vec{v} \cdot \vec{u} = \vec{u}^T \vec{v} = \vec{v}^T \vec{u}$$

$$(\vec{u} + \vec{v}) \cdot (\vec{w} + \vec{z}) = \vec{u} \cdot \vec{w} + \vec{u} \cdot \vec{z} + \vec{v} \cdot \vec{w} + \vec{v} \cdot \vec{z}$$

$$\|\vec{u}\|^2 = \vec{u} \cdot \vec{u}$$

# Step Two: Find Gradient

- ▶ Expand norm to make gradient easier.

$$\|\vec{Xw} - \vec{y}\|^2 =$$

=

=

=

## Step Two: Find Gradient

$$\frac{d}{d\vec{w}} [R_{\text{sq}}(\vec{w})] = \frac{1}{n} \frac{d}{d\vec{w}} [\vec{w}^T X^T X \vec{w} - 2\vec{y}^T X \vec{w} + \vec{y}^T \vec{y}]$$

= ?

# Claims

- ▶  $\frac{d}{d\vec{w}} [\vec{w}^T X^T X \vec{w}] = 2X^T X \vec{w}$
- ▶  $\frac{d}{d\vec{w}} [\vec{y}^T X \vec{w}] = X^T \vec{y}$
- ▶  $\frac{d}{d\vec{w}} [\vec{y}^T \vec{y}] = 0$

# Example

- ▶ Show  $\frac{d}{d\vec{u}} [\vec{v}^T \vec{u}] = \vec{v}$ .
- ▶ General procedure:
  1. Expand  $\vec{v}^T \vec{u}$  until coordinates  $u_1, \dots, u_k$  are visible.
  2. Compute  $\partial d / \partial u_1, \partial d / \partial u_2, \dots, \partial d / \partial u_k$ .
  3. Gather result in vector form.

## Step Two: Find Gradient

$$\frac{d}{d\vec{w}} [R_{\text{sq}}(\vec{w})] = \frac{1}{n} \frac{d}{d\vec{w}} [\vec{w}^T X^T X \vec{w} - 2\vec{y}^T X \vec{w} + \vec{y}^T \vec{y}]$$

=

# Solution

- We have found:

$$\frac{d}{d\vec{w}} [R_{\text{sq}}(\vec{w})] = \frac{1}{n} (2X^T X \vec{w} - 2X^T \vec{y})$$

- To minimize  $R_{\text{sq}}(\vec{w})$ , set gradient to zero, solve:

$$2X^T X \vec{w} - 2X^T \vec{y} = 0 \implies X^T X \vec{w} = X^T \vec{y}$$

- This is a system of equations in matrix form, called the **normal equations**.

# The Normal Equations

- ▶ The least squares solutions for  $\vec{w}$  are found by solving the **normal equations**:

$$X^T X \vec{w} = X^T \vec{y}$$

- ▶ Mathematically, solved by:

$$\vec{w}^* = (X^T X)^{-1} X^T \vec{y}$$

- ▶ In practice: **don't invert!** Use the **Singular Value Decomposition** (SVD).

# Aside

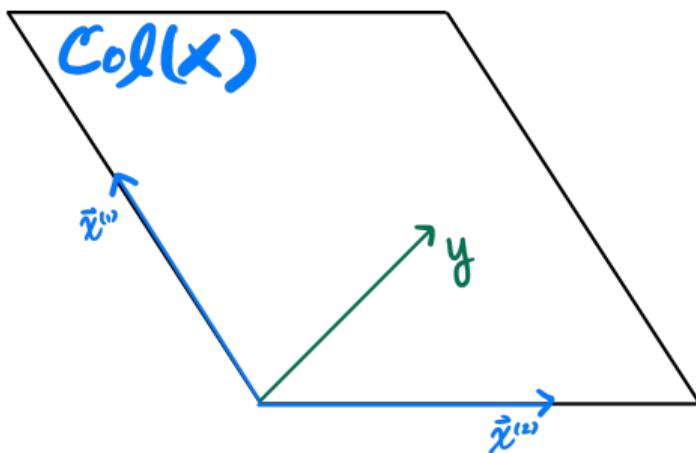
- ▶  $X^+ = (X^T X)^{-1} X^T$  is known as the **Moore-Penrose pseudoinverse** of  $X$ .
- ▶ A generalization of the matrix inverse for non-square matrices.
- ▶ With this, the least squares solution is:  $\vec{w}^* = X^+ \vec{y}$

# **Two Approaches**

- ▶ **Approach #1:** Vector Calculus
- ▶ **Approach #2:** Orthogonal Projection (Geometry)
- ▶ Different ways of arriving at same solutions.

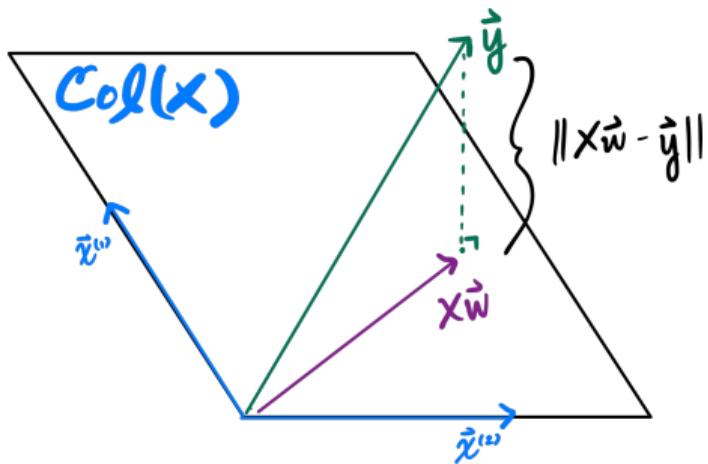
## Approach #2: Geometry

- ▶ In the collinear case,  $\vec{y}$  can be predicted exactly.
- ▶ That is, there is a  $\vec{w}$  such that  $X\vec{w} = \vec{y}$ .



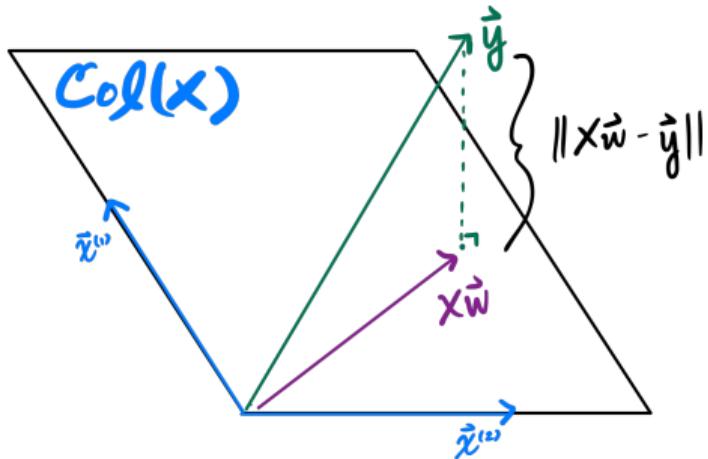
# Approach #2: Geometry

- ▶ In practice,  $\vec{y}$  is **not** in the **column space** of  $X$ .
- ▶ **Goal:** choose  $\vec{w}$  so that  $X\vec{w} \approx \vec{y}$ .



## Approach #2: Geometry

- ▶ Idea:  $\|X\vec{w} - \vec{y}\|$  is smallest when  $X\vec{w}$  is the **orthogonal projection** of  $\vec{y}$  onto the column space of  $X$ .



# Recall: Projections

- ▶ Let  $A$  be a matrix and  $\vec{u}$  be a vector.
- ▶ The **orthogonal projection** of  $\vec{u}$  onto the column space of  $A$  is given by:

$$A(A^T A)^{-1} A^T \vec{u}$$

## Approach #2: Geometry

- ▶ So, the orthogonal projection of  $\vec{y}$  onto the column space of  $X$  is given by:

$$\underbrace{X(X^T X)^{-1} X^T \vec{y}}_{\vec{w}^*}$$

# Least Squares Regression

- We can now fit functions of the form:

$$H(\vec{x}) = w_0 + w_1x_1 + w_2x_2 + \dots + w_dx_d$$

by minimizing expected square loss.

- Example:
  - $x_1$ : years of experience
  - $x_2$ : # of interview questions missed
  - $x_3$ : favorite number

# Procedure

1. Form (augmented)  $n \times (d + 1)$  **design matrix**:

$$X = \begin{pmatrix} \text{Aug}(\vec{x}^{(1)}) \\ \text{Aug}(\vec{x}^{(2)}) \\ \vdots \\ \text{Aug}(\vec{x}^{(n)}) \end{pmatrix} \longrightarrow = \begin{pmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \dots & x_d^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \dots & x_d^{(2)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_1^{(n)} & x_2^{(n)} & \dots & x_d^{(n)} \end{pmatrix}$$

2. Solve the **normal equations**:

$$\vec{w}^* = (X^T X)^{-1} X^T \vec{y}$$

# DSC 140A

*Probabilistic Modeling & Machine Learning*

Lecture 3 | Part 2

**Example: Gaussian Basis Functions**

# Recall: Basis Functions

- ▶ We can fit any function of the form:

$$H(\vec{x}; \vec{w}) = w_0 + w_1\phi_1(\vec{x}) + w_2\phi_2(\vec{x}) + \dots + w_k\phi_k(\vec{x})$$

- ▶  $\phi_i(\vec{x}) : \mathbb{R}^d \rightarrow \mathbb{R}$  is called a **basis function**.

# Procedure

1. Define  $\vec{\phi}^{(i)} = (\phi_1(\vec{x}^{(i)}), \phi_2(\vec{x}^{(i)}), \dots, \phi_k(\vec{x}^{(i)}))^T$

2. Form  $n \times k$  **design matrix**:

$$\Phi = \begin{pmatrix} \text{Aug}(\vec{\phi}^{(1)}) & \longrightarrow \\ \text{Aug}(\vec{\phi}^{(2)}) & \longrightarrow \\ \vdots & \vdots \\ \text{Aug}(\vec{\phi}^{(n)}) & \longrightarrow \end{pmatrix} = \begin{pmatrix} \phi_1(\vec{x}^{(1)}) & \phi_2(\vec{x}^{(1)}) & \dots & \phi_k(\vec{x}^{(1)}) \\ \phi_1(\vec{x}^{(2)}) & \phi_2(\vec{x}^{(2)}) & \dots & \phi_k(\vec{x}^{(2)}) \\ \vdots & \vdots & \vdots & \vdots \\ \phi_1(\vec{x}^{(n)}) & \phi_2(\vec{x}^{(n)}) & \dots & \phi_k(\vec{x}^{(n)}) \end{pmatrix}$$

3. Solve the **normal equations**:

$$\vec{w}^* = (\Phi^T \Phi)^{-1} \Phi^T \vec{y}$$

# Example: Polynomial Curve Fitting

- ▶ Fit a function of the form:

$$H(x; \vec{w}) = w_0 + w_1x + w_2x^2 + w_3x^3$$

- ▶ Use basis functions:

$$\phi_0(x) = 1 \quad \phi_1(x) = x \quad \phi_2(x) = x^2 \quad \phi_3(x) = x^3$$

# Example: Polynomial Curve Fitting

- ▶ Design matrix becomes:

$$\Phi = \begin{pmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ \dots & \dots & \ddots & \vdots \\ 1 & x_n & x_n^2 & x_n^3 \end{pmatrix}$$

# Gaussian Basis Functions

- ▶ **Gaussians** make for useful basis functions.

$$\phi_i(x) = \exp\left(-\frac{(x - \mu_i)^2}{\sigma_i^2}\right)$$

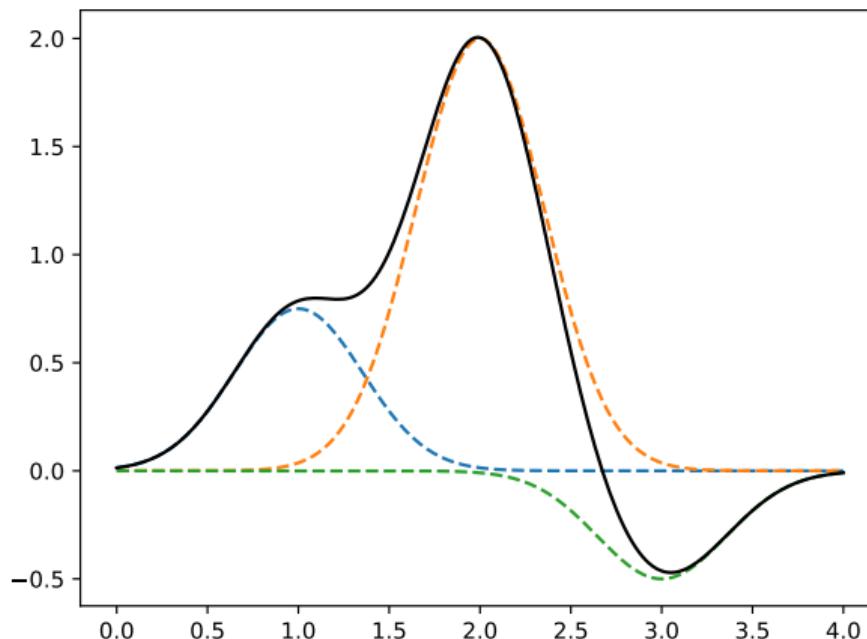
- ▶ Must specify<sup>1</sup> **center**  $\mu_i$  and **width**  $\sigma_i$  for each Gaussian basis function.

---

<sup>1</sup>You pick these; they are not learned!

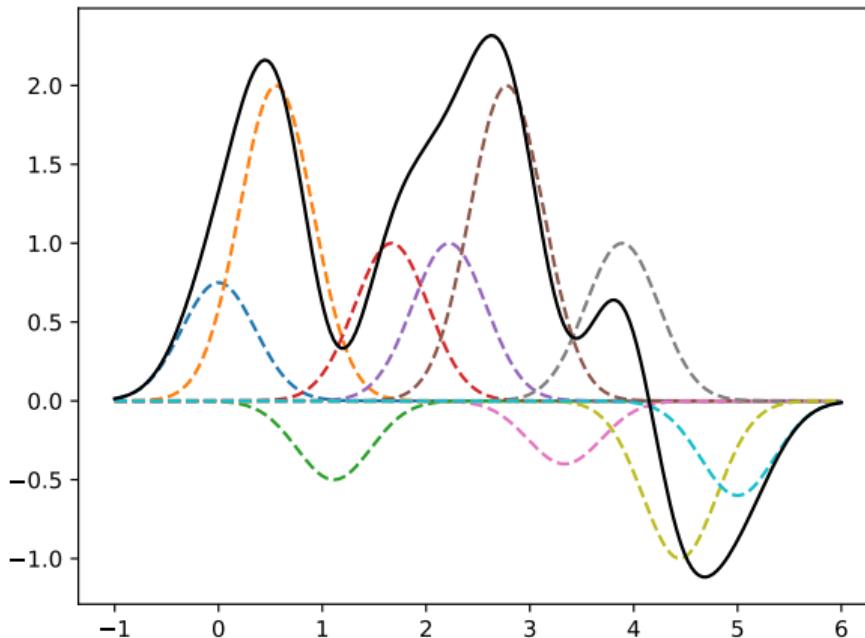
# Example: $k = 3$

- ▶ A function of the form:  $H(x) = w_1\phi_1(x) + w_2\phi_2(x) + w_3\phi_3(x)$ , using 3 Gaussian basis functions.

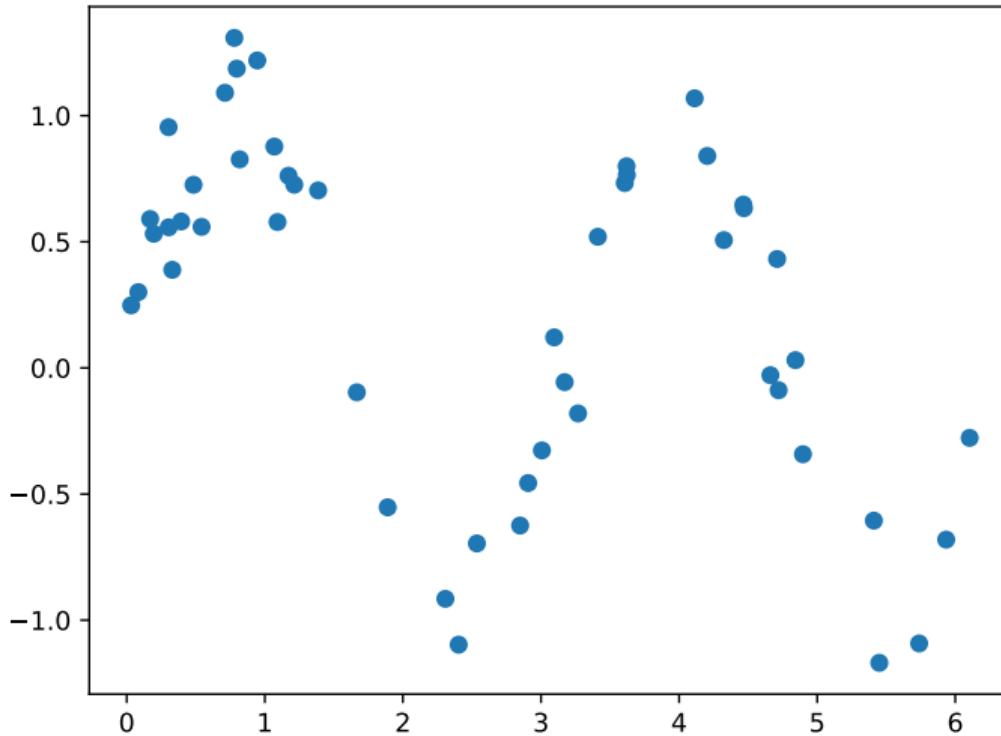


# Example: $k = 10$

- ▶ The more basis functions, the more complex  $H$  can be.



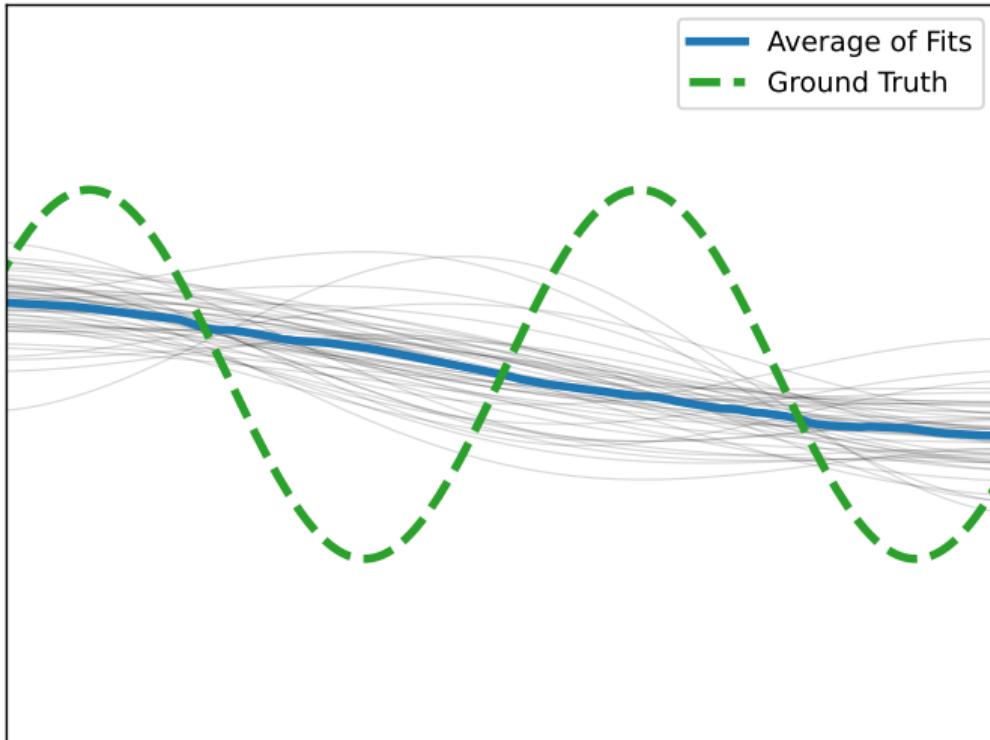
# Example



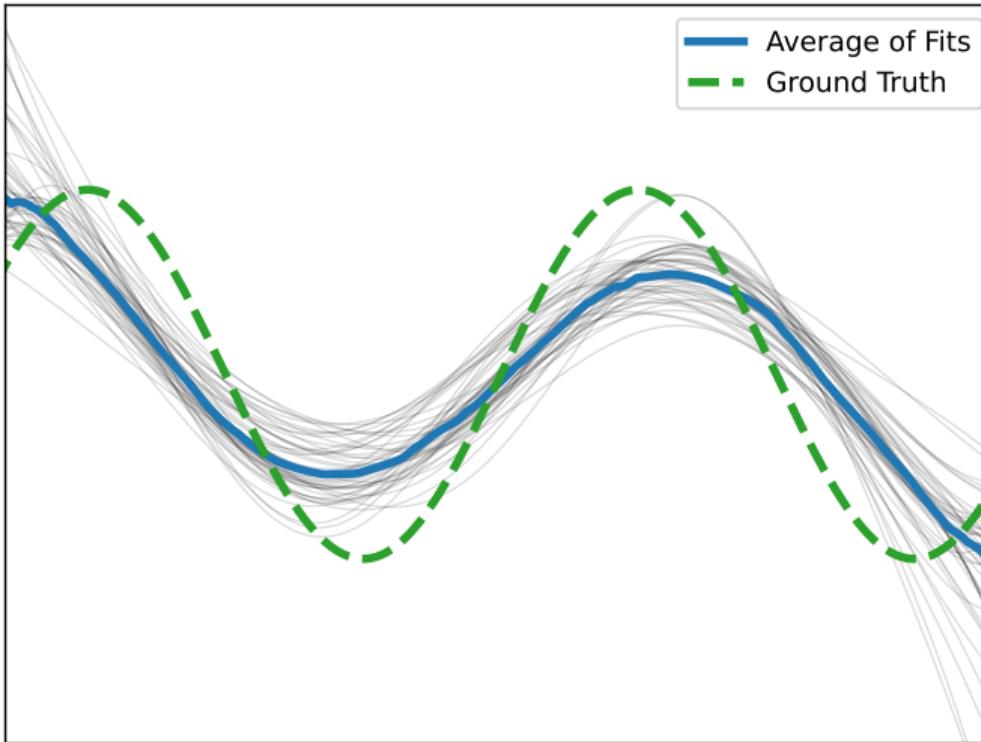
# Example

- ▶ Will use  $k$  Gaussian basis functions, equally spaced in interval  $[0, 2\pi]$ .
- ▶ Same width for each:  $\sigma_i = \sigma = 2\pi/k$ .
- ▶ Will fit on 50 random sinusoidal datasets.

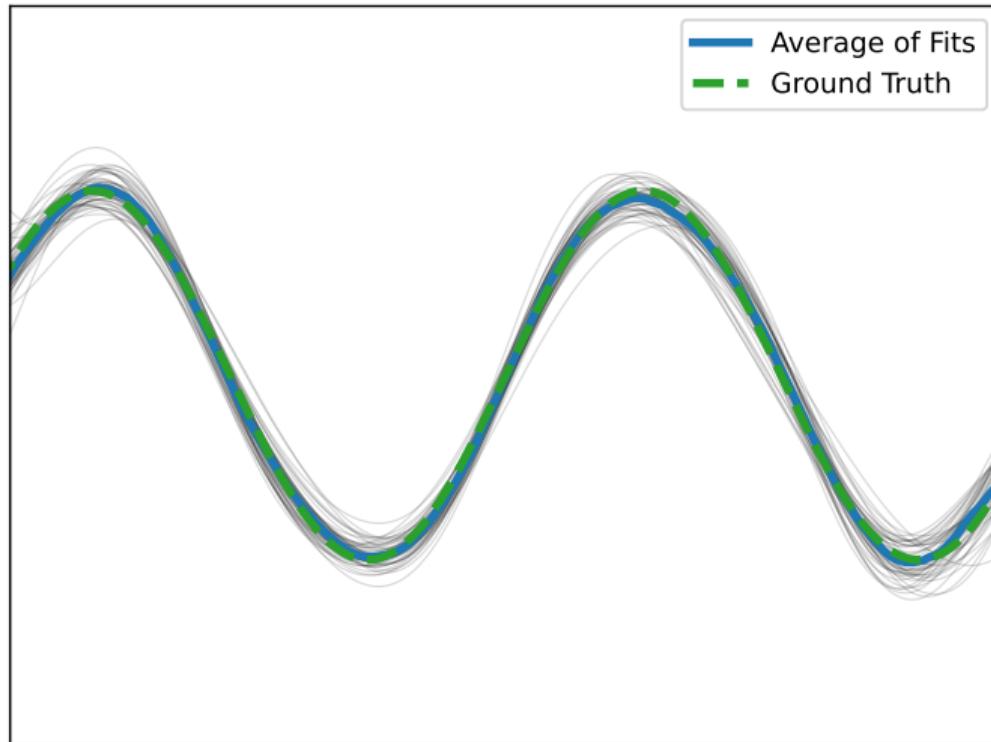
$k = 2$



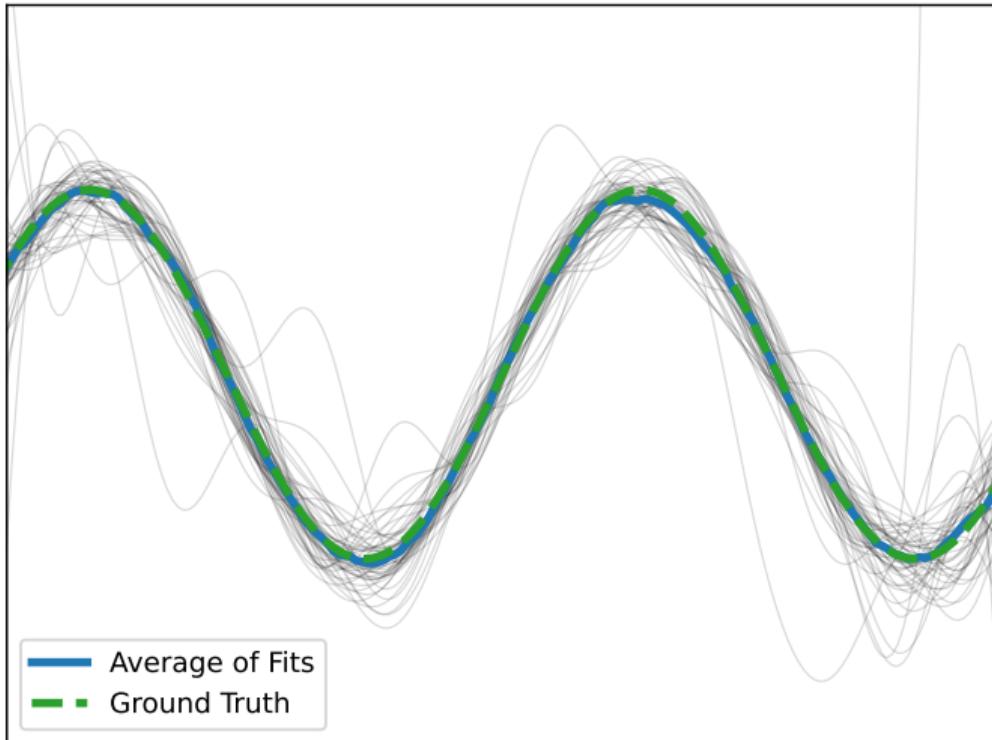
$k = 4$



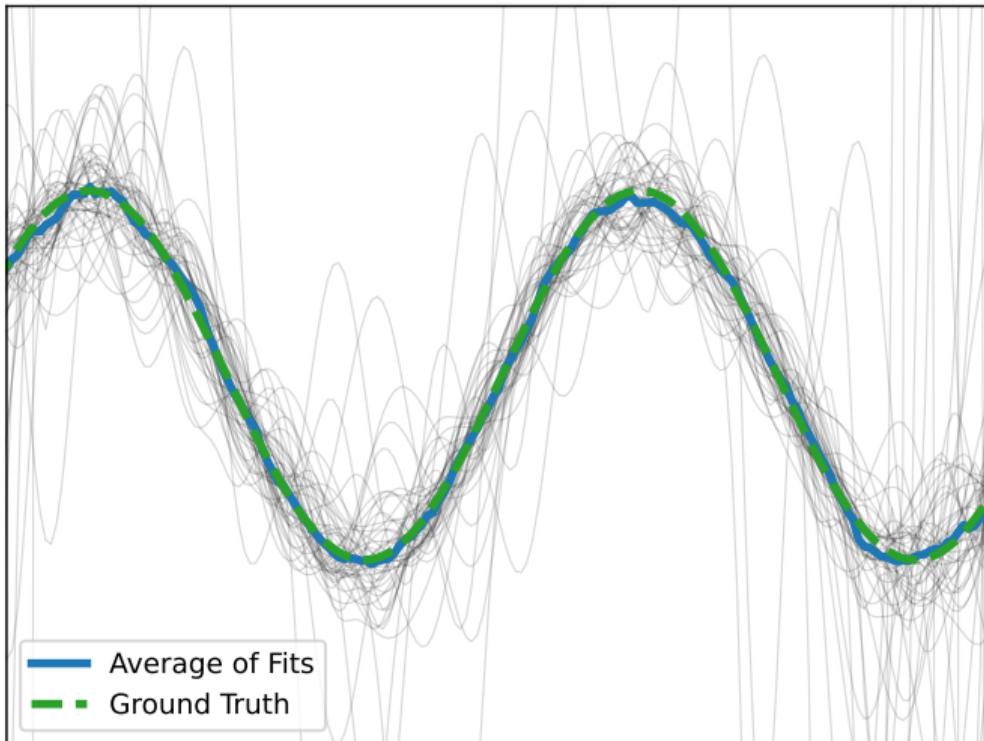
$k = 8$



$k = 16$

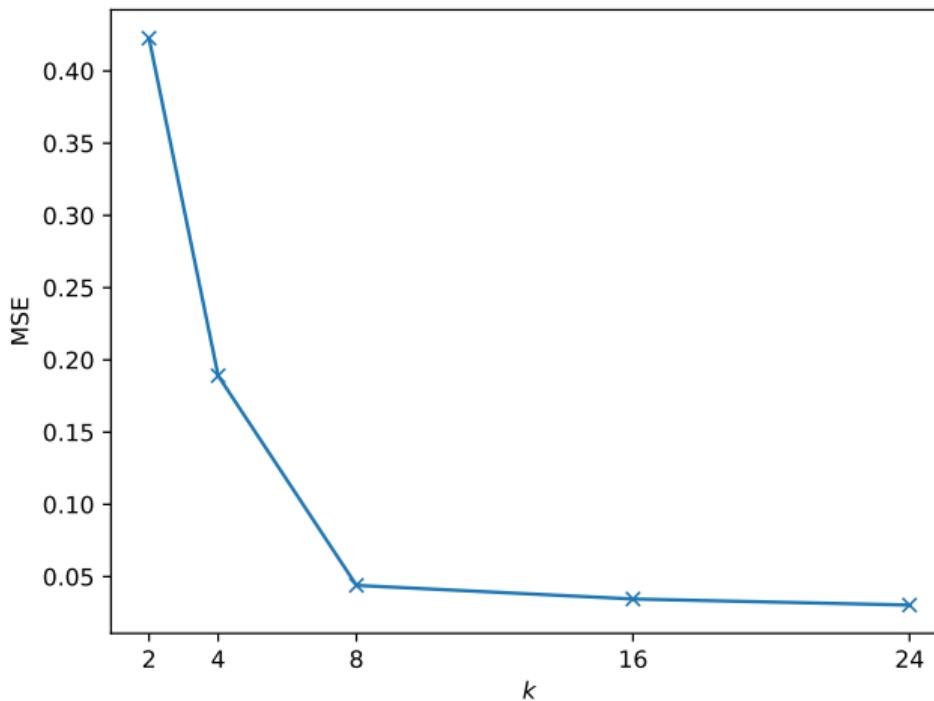


$k = 24$

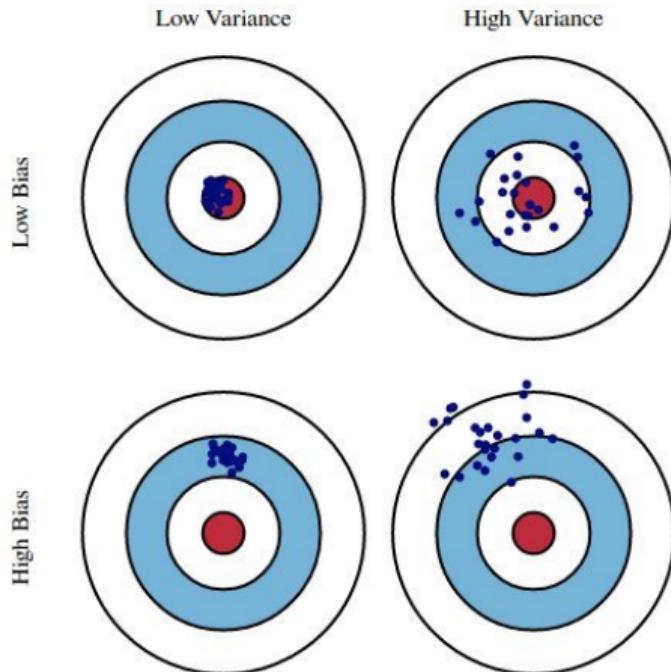


# Observation #1

- **Mean Squared Error** decreases as  $k$  increases.



# Observation #2



## Observation #2

- ▶ When  $k$  is small: **high bias, low variance**
- ▶ When  $k$  is large **low bias, high variance**
- ▶ There appears to be a tradeoff.

# DSC 140A

*Probabilistic Modeling & Machine Learning*

Lecture 3 | Part 3

**Linear Classification**

# Classification

- ▶ We've been considering **regression**. What about **classification**?
- ▶ In classification, output is a discrete **label**.

# Example: Mango Ripeness

- ▶ Predict whether a mango is ripe given greenness and hardness.
- ▶ Convention: encode “yes” as 1 and “no” as -1.

Greenness	Hardness		Ripe
0.7	0.9		1
0.2	0.5		-1
0.3	0.1		-1
:	:		:

# Binary vs. Multiclass Classification

- ▶ **Binary** classification: predict “yes” / “no”.
  - ▶ E.g., is this picture of a dog? (“yes” / “no”)
- ▶ **Multiclass** classification: predict one of  $k$  possible labels.
  - ▶ E.g., what animal is in this picture?
  - ▶ cat, dog, giraffe, mongoose, etc.
- ▶ We'll focus on **binary** for simplicity.

# Linear Classification

- ▶ Linear prediction functions output real numbers:

$$H(\vec{x}) = w_0 + w_1x_1 + \dots + w_dx_d$$

- ▶ Can turn output into a “yes” / “no” answer by thresholding:

$$\text{sign}(z) = \begin{cases} 1 & z > 0 \\ -1 & z < 0 \\ 0 & \text{otherwise} \end{cases}$$

- ▶ Final prediction:  $\text{sign}(H(\vec{x}))$

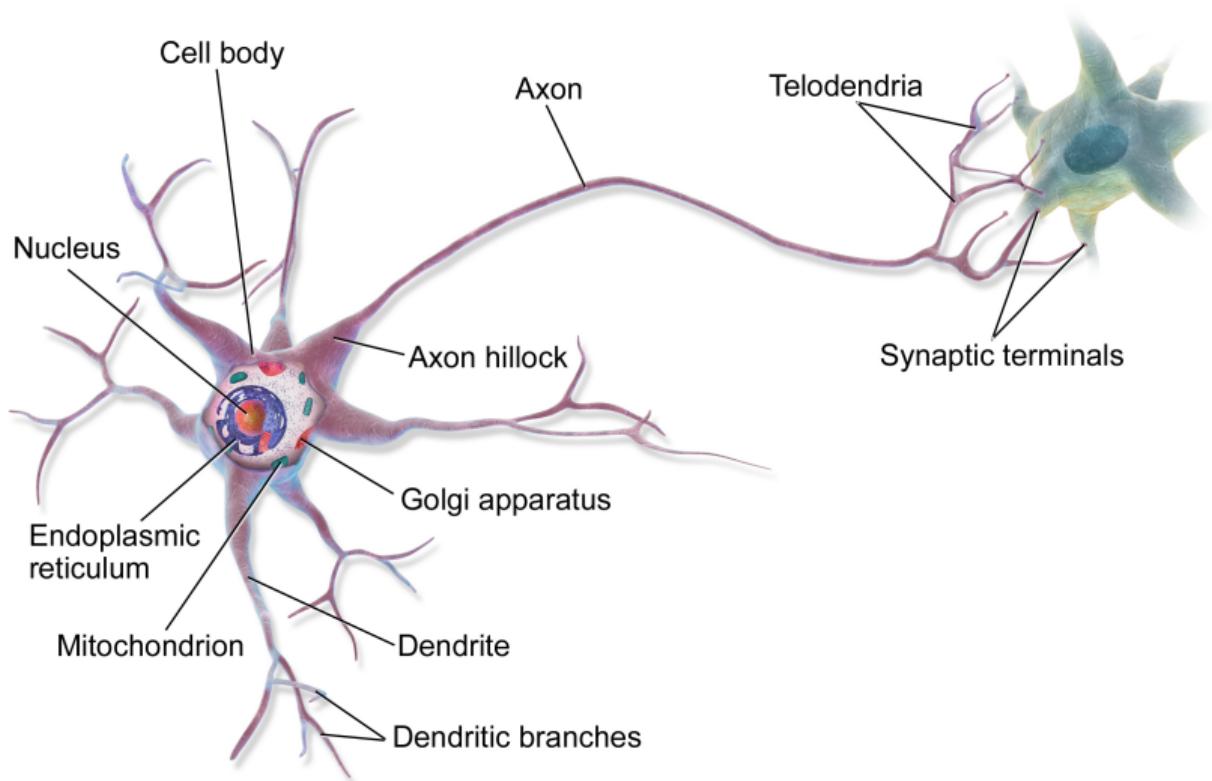
## Interpretation: Weighted Vote

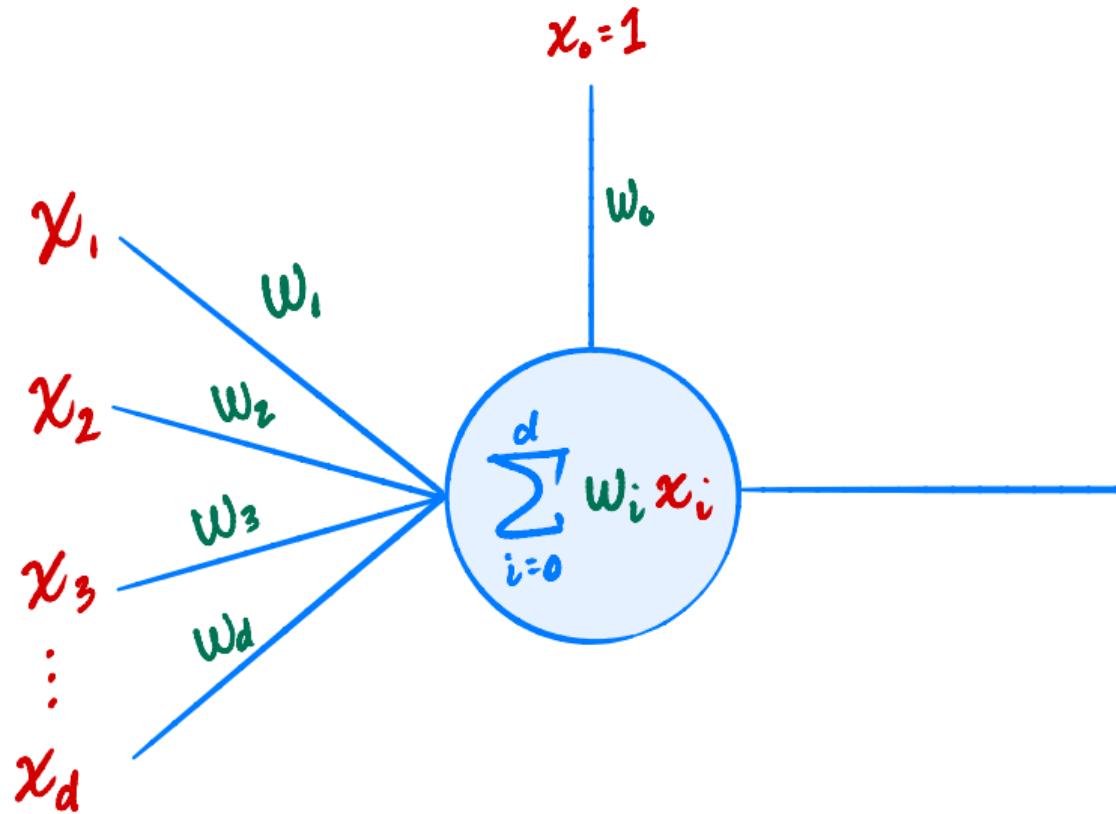
- ▶ A linear decision function can be thought of as a **weighted vote**:

$$H(\vec{x}) = w_0 + w_1x_1 + \dots + w_dx_d$$

# Interpretation: A Simple Neuron

- ▶ A linear decision function is a simple model for a neuron.
- ▶ Are the basis of modern neural networks.





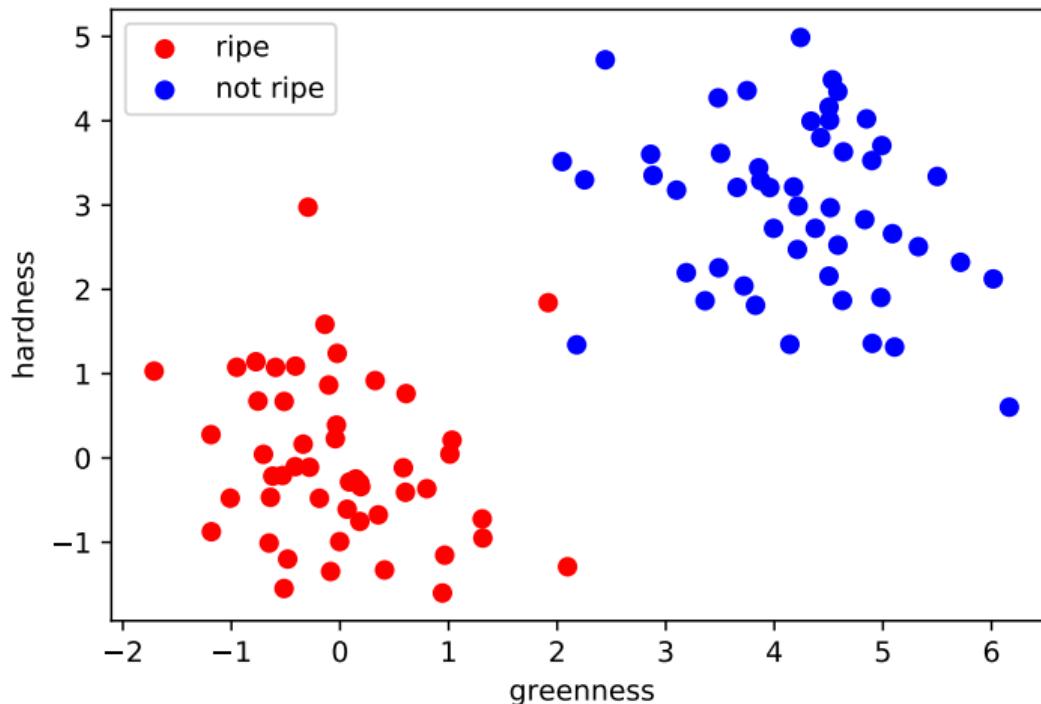
# Decision Boundary

- ▶ The **decision boundary** is the place where the output of  $H(x)$  switches from “yes” to “no”.
  - ▶ If  $H > 0 \mapsto$  “yes” and  $H < 0 \mapsto$  “no”, the decision boundary is where  $H = 0$ .
- ▶ If  $H$  is a linear predictor and<sup>2</sup>
  - ▶  $\vec{x} \in \mathbb{R}^1$ , then the decision boundary is just a number.
  - ▶  $\vec{x} \in \mathbb{R}^2$ , the boundary is a straight line.
  - ▶  $\vec{x} \in \mathbb{R}^d$ , the boundary is a  $d - 1$  dimensional (hyper) plane.

---

<sup>2</sup>when plotted in the original feature coordinate space!

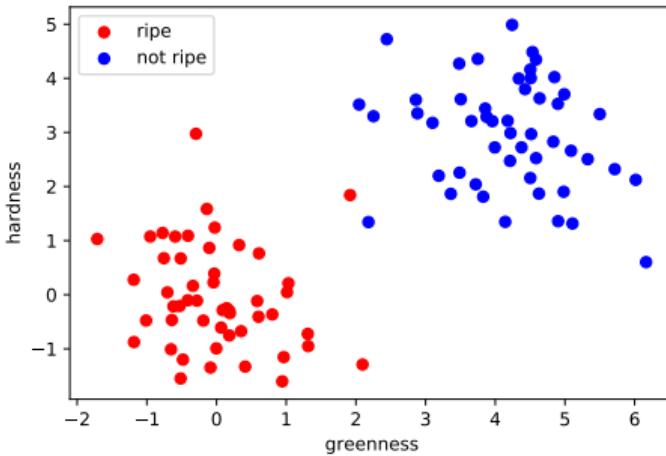
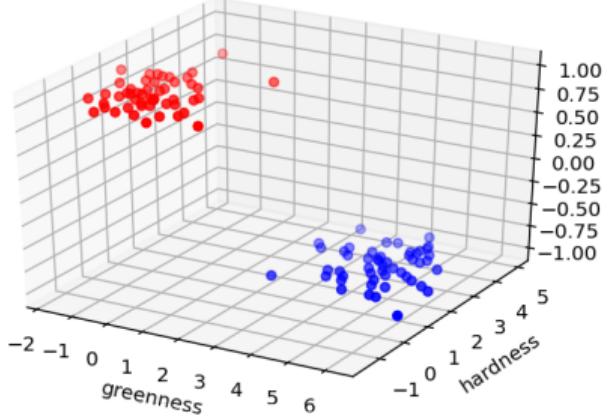
# Example



# Prediction Surface

- ▶ Although our final prediction uses only the sign, the magnitude of  $H$  is useful.
- ▶ Recall: the plot of a linear prediction function  $H$  is a (hyper)plane.
- ▶ The problem of binary classification can be thought of as regression where the targets are 1 and -1.

# Example



# Useful Fact

- ▶  $|H(\vec{x})|$  is proportional to the distance from the decision boundary.
- ▶ If  $H(\vec{x}^{(1)}) = H(\vec{x}^{(2)})$ ,  $\vec{x}^{(1)}$  and  $\vec{x}^{(2)}$  are equally far away from the boundary.
- ▶ If  $H(\vec{x}) = 0$ ,  $\vec{x}$  is on the boundary.
- ▶  $|H(\vec{x})|$  can be used as a measure of “confidence”.

# DSC 140A

*Probabilistic Modeling & Machine Learning*

Lecture 3 | Part 4

**ERM for Linear Classifiers**

# Learning a Classifier

- ▶ We can learn a linear classifier using the same ERM paradigm as before.

# Empirical Risk Minimization

- ▶ Step 1: choose a **hypothesis class**
  - ▶ Let's assume we've chosen linear predictors
- ▶ Step 2: choose a **loss function**
- ▶ Step 3: minimize **expected loss (empirical risk)**

## Exercise

Can we use the square loss for classification?

$$(H(\vec{x}^{(i)}) - y_i)^2$$

# Square Loss for Classification

- ▶ **Yes!** We can use the square loss, but it may not be the best choice.
- ▶ E.g., suppose  $H(\vec{x}^{(i)}) = 11$  and  $y_i = 1$ .
  - ▶ Square loss: 100. **Large!**
- ▶ E.g., suppose  $H(\vec{x}^{(i)}) = -9$  and  $y_i = 1$ .
  - ▶ Square loss:

## Main Idea

While the square loss can be used for classification, it may not be the best choice because it penalizes predictions that are “very correct”.

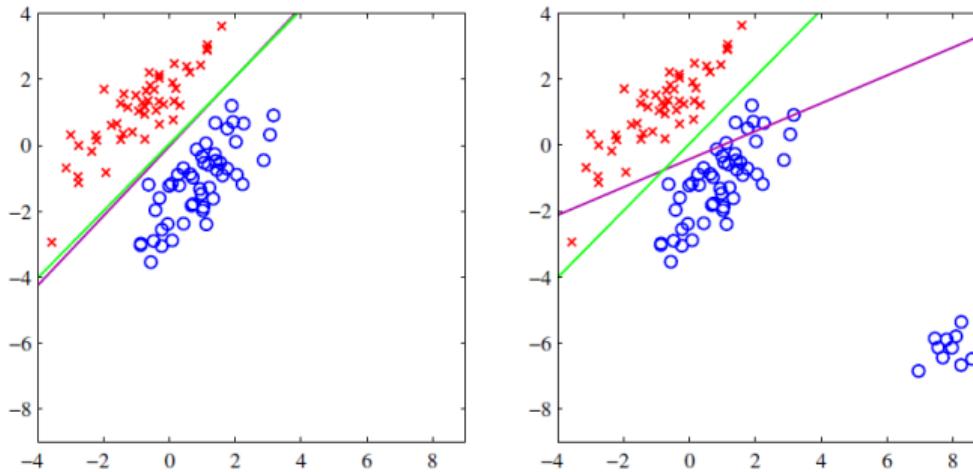
It is designed for regression.

# Least Squares Classifier

- ▶ **Least squares classification** is performed exactly the same as least squares regression.
  - ▶ I.e., solve the normal equations:  $\vec{w}^* = (X^T X)^{-1} X^T \vec{y}$
- ▶ Except the prediction is thresholded:

$$H(\vec{x}) = \text{sign}(\text{Aug}(\vec{x}) \cdot \vec{w}^*)$$

# Least Squares and Outliers



**Figure 4.4** The left plot shows data from two classes, denoted by red crosses and blue circles, together with the decision boundary found by least squares (magenta curve) and also by the logistic regression model (green curve), which is discussed later in Section 4.3.2. The right-hand plot shows the corresponding results obtained when extra data points are added at the bottom left of the diagram, showing that least squares is highly sensitive to outliers, unlike logistic regression.

# Another Loss Function

- ▶ What about the **0-1 loss**?
  - ▶ Loss = 0 if prediction is **correct**.
  - ▶ Loss = 1 if prediction is **incorrect**.
- ▶ More formally:

$$L_{0-1}(H(\vec{x}^{(i)}), y_i) = \begin{cases} 0 & \text{if } \text{sign}(H(\vec{x}^{(i)})) = y_i \\ 1 & \text{if } \text{sign}(H(\vec{x}^{(i)})) \neq y_i \end{cases}$$

# Expected 0-1 Loss

- ▶ The expected 0-1 loss (empirical risk) has a nice interpretation:

$$R_{0-1}(H) = \frac{1}{n} \sum_{i=1}^n \begin{cases} 0 & \text{if } \text{sign}(H(\vec{x}^{(i)})) = y_i \\ 1 & \text{if } \text{sign}(H(\vec{x}^{(i)})) \neq y_i \end{cases}$$

## Exercise

What is it?

# Answer

- ▶ The empirical risk with respect to the 0-1 loss is  $(1 - \text{the accuracy})$  of the classifier.

$$R_{0-1}(H) = \frac{1}{n} \sum_{i=1}^n \begin{cases} 0 & \text{if } \text{sign}(H(\vec{x}^{(i)})) = y_i \\ 1 & \text{if } \text{sign}(H(\vec{x}^{(i)})) \neq y_i \end{cases}$$
$$= \frac{\# \text{ of incorrect predictions}}{n}$$

# ERM for the 0-1 Loss

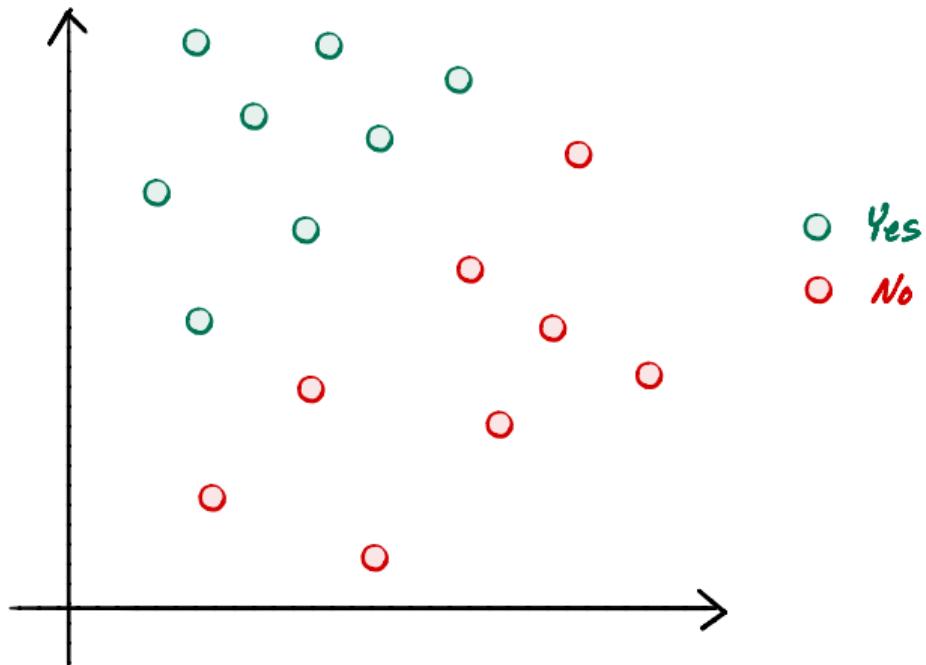
- ▶ Assume the 0-1 loss, linear prediction functions.
- ▶ Next step: find linear  $H$  minimizing the empirical risk.
- ▶ That is: find  $H$  which maximizes the **accuracy** on the training set.

# Problem

- ▶ The 0-1 loss is not differentiable.
- ▶ Can't even use gradient descent...

# Why?

- The 0-1 risk surface is flat almost everywhere.



# Computationally Difficult

- ▶ It is **not feasible** to minimize 0-1 risk in general.
- ▶ More formally: NP-Hard to optimize expected 0-1 loss in general.<sup>4</sup>

---

<sup>4</sup>It is efficiently doable if the classes are linearly separable by finding convex hulls of each class. If non-separable, it is difficult.

## Main Idea

It is computationally difficult (NP-Hard) to maximize the accuracy of a linear classifier.

# Surrogate Loss

- ▶ Instead of the 0-1 loss, we use a **surrogate loss**.
- ▶ That is, a loss that is similar in spirit, but has better mathematical properties.

# The Perceptron Loss

- ▶ The **perceptron loss** is designed for binary classification.
  - ▶ Loss = 0 if prediction is correct.
  - ▶ Loss > 0 if prediction incorrect.
  - ▶ Loss increases with distance from boundary.
- ▶ More formally:

$$L_{\text{tron}}(H(\vec{x}), y) = \begin{cases} 0, & \text{sign}(H(\vec{x})) = \text{sign}(y) \\ |H(\vec{x})|, & \text{sign}(H(\vec{x})) \neq \text{sign}(y) \end{cases}$$

# ERM for the Perceptron

- ▶ **Goal:** minimize the empirical expected perceptron loss (risk):

$$H_{\text{tron}}(H(\vec{x}), y) = \begin{cases} 0, & \text{sign}(H(\vec{x})) = \text{sign}(y) \\ |H(\vec{x})|, & \text{sign}(H(\vec{x})) \neq \text{sign}(y) \end{cases}$$

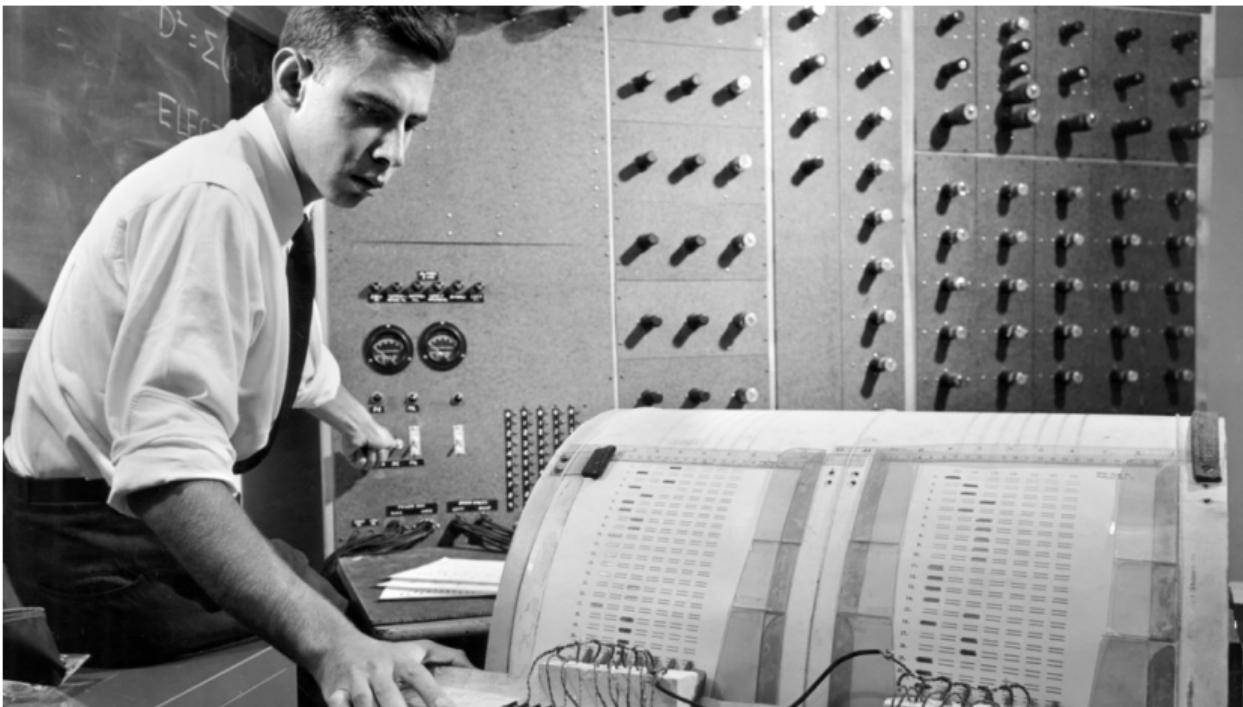
# Problem

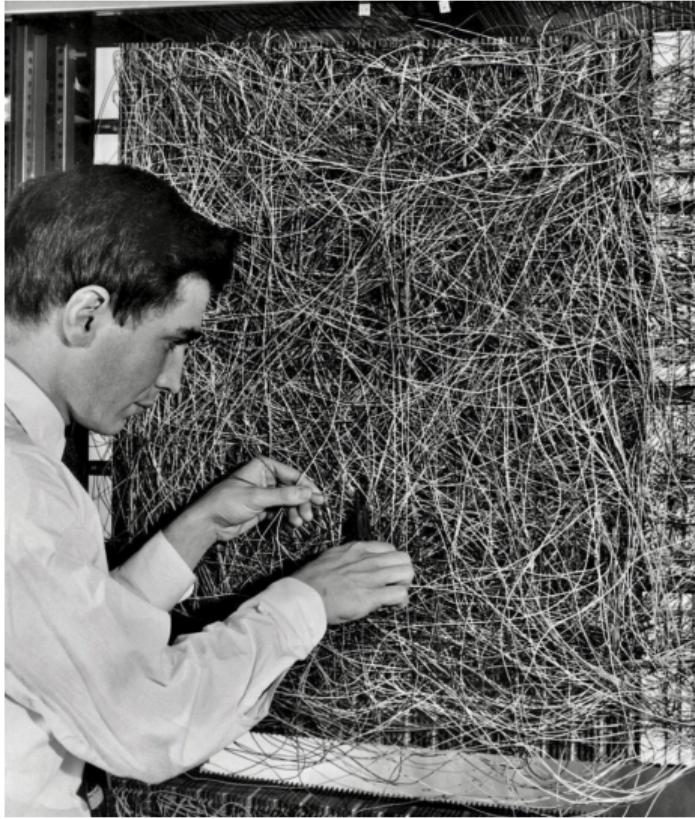
- ▶ The perceptron loss is **not differentiable**.
- ▶ However:
  - ▶ it is **not flat!**
  - ▶ its derivative exists almost everywhere.
- ▶ We can train iteratively using **gradient descent** (next time).

# Some History

- ▶ Perceptrons were one of the first “machine learning” models.
- ▶ The basis of modern neural networks.

# Rosenblatt's Perceptron





# DSC 140A

*Probabilistic Modeling & Machine Learning*

Lecture 4 | Part 1

**Linear Classification**

# Classification

- ▶ We've been considering **regression**. What about **classification**?

# Empirical Risk Minimization

- ▶ Step 1: choose a **hypothesis class**
  - ▶ Let's assume we've chosen linear predictors
- ▶ Step 2: choose a **loss function**
- ▶ Step 3: minimize **expected loss (empirical risk)**

## Exercise

Can we use the square loss for classification?

$$(H(\vec{x}^{(i)}) - y_i)^2$$

# Square Loss for Classification

- ▶ **Yes!** We can use the square loss, but it may not be the best choice.
- ▶ E.g., suppose  $H(\vec{x}^{(i)}) = 11$  and  $y_i = 1$ .
  - ▶ Square loss: 100. **Large!**
- ▶ E.g., suppose  $H(\vec{x}^{(i)}) = -9$  and  $y_i = 1$ .
  - ▶ Square loss:

## Main Idea

While the square loss can be used for classification, it may not be the best choice because it penalizes predictions that are “very correct”.

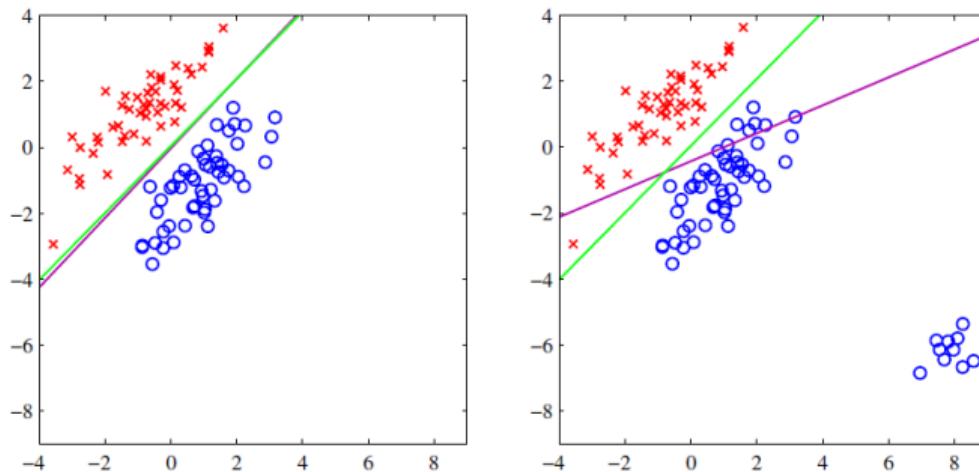
It is designed for regression.

# Least Squares Classifier

- ▶ **Least squares classification** is performed exactly the same as least squares regression.
  - ▶ I.e., solve the normal equations:  $\vec{w}^* = (X^T X)^{-1} X^T \vec{y}$
- ▶ Except the prediction is thresholded:

$$H(\vec{x}) = \text{sign}(\text{Aug}(\vec{x}) \cdot \vec{w}^*)$$

# Least Squares and Outliers



**Figure 4.4** The left plot shows data from two classes, denoted by red crosses and blue circles, together with the decision boundary found by least squares (magenta curve) and also by the logistic regression model (green curve), which is discussed later in Section 4.3.2. The right-hand plot shows the corresponding results obtained when extra data points are added at the bottom left of the diagram, showing that least squares is highly sensitive to outliers, unlike logistic regression.

# Another Loss Function

- ▶ What about the **0-1 loss**?
  - ▶ Loss = 0 if prediction is **correct**.
  - ▶ Loss = 1 if prediction is **incorrect**.
- ▶ More formally:

$$L_{0-1}(H(\vec{x}^{(i)}), y_i) = \begin{cases} 0 & \text{if } \text{sign}(H(\vec{x}^{(i)})) = y_i \\ 1 & \text{if } \text{sign}(H(\vec{x}^{(i)})) \neq y_i \end{cases}$$

# Expected 0-1 Loss

- ▶ The expected 0-1 loss (empirical risk) has a nice interpretation:

$$R_{0-1}(H) = \frac{1}{n} \sum_{i=1}^n \begin{cases} 0 & \text{if } \text{sign}(H(\vec{x}^{(i)})) = y_i \\ 1 & \text{if } \text{sign}(H(\vec{x}^{(i)})) \neq y_i \end{cases}$$

## Exercise

What is it?

# Answer

- ▶ The empirical risk with respect to the 0-1 loss is  $(1 - \text{the accuracy})$  of the classifier.

$$R_{0-1}(H) = \frac{1}{n} \sum_{i=1}^n \begin{cases} 0 & \text{if } \text{sign}(H(\vec{x}^{(i)})) = y_i \\ 1 & \text{if } \text{sign}(H(\vec{x}^{(i)})) \neq y_i \end{cases}$$
$$= \frac{\# \text{ of incorrect predictions}}{n}$$

# ERM for the 0-1 Loss

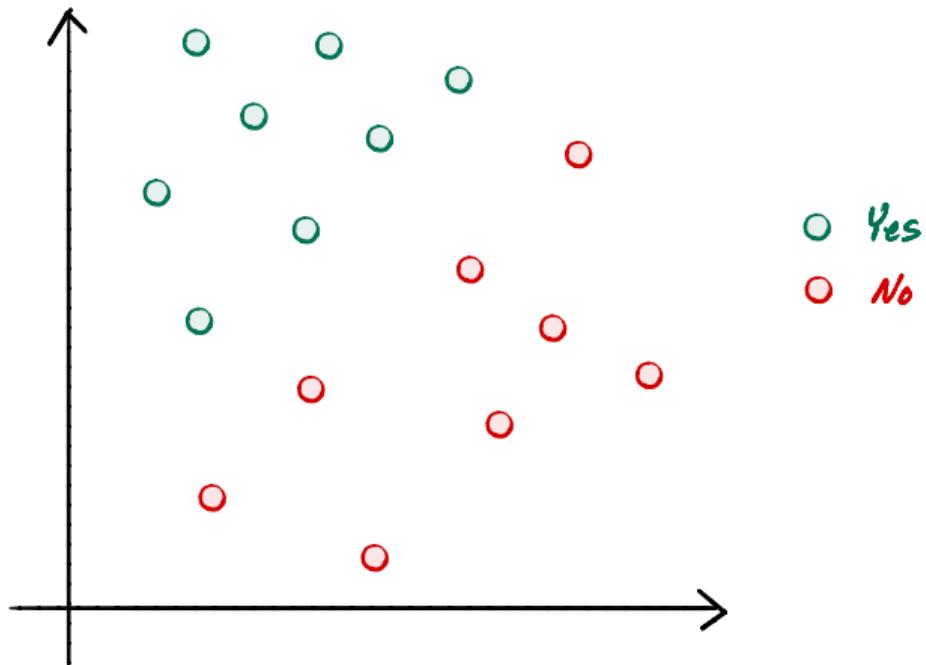
- ▶ Assume the 0-1 loss, linear prediction functions.
- ▶ Next step: find linear  $H$  minimizing the empirical risk.
- ▶ That is: find  $H$  which maximizes the **accuracy** on the training set.

# Problem

- ▶ The 0-1 loss is not differentiable.
- ▶ Can't even use gradient descent...

# Why?

- The 0-1 risk surface is flat almost everywhere.



# Computationally Difficult

- ▶ It is **not feasible** to minimize 0-1 risk in general.
- ▶ More formally: NP-Hard to optimize expected 0-1 loss in general.<sup>2</sup>

---

<sup>2</sup>It is efficiently doable if the classes are linearly separable by finding convex hulls of each class. If non-separable, it is difficult.

## Main Idea

It is computationally difficult (NP-Hard) to maximize the accuracy of a linear classifier.

# Surrogate Loss

- ▶ Instead of the 0-1 loss, we use a **surrogate loss**.
- ▶ That is, a loss that is similar in spirit, but has better mathematical properties.

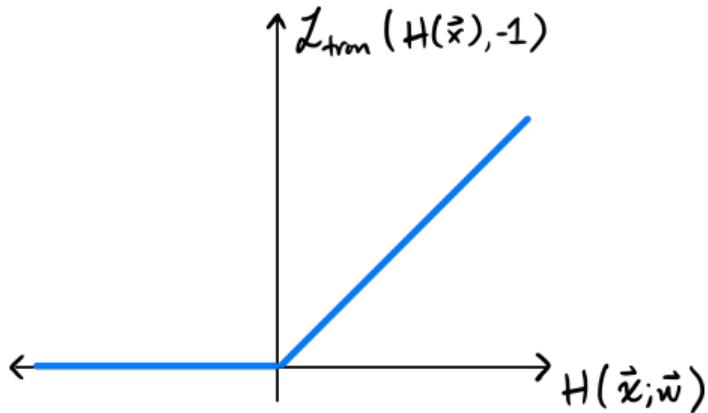
# The Perceptron Loss

- ▶ The **perceptron loss** is designed for binary classification.
  - ▶ Loss = 0 if prediction is correct.
  - ▶ Loss > 0 if prediction incorrect.
  - ▶ Loss increases with distance from boundary.
- ▶ More formally:

$$L_{\text{tron}}(H(\vec{x}), y) = \begin{cases} 0, & \text{sign}(H(\vec{x})) = \text{sign}(y) \\ |H(\vec{x})|, & \text{sign}(H(\vec{x})) \neq \text{sign}(y) \end{cases}$$

# Perceptron Loss

$$L_{\text{tron}}(H(\vec{x}; \vec{w}), y) = \begin{cases} 0, & \text{sign}(H(\vec{x})) = \text{sign}(y) \\ |H(\vec{x})|, & \text{sign}(H(\vec{x})) \neq \text{sign}(y) \end{cases}$$



# ERM for the Perceptron

- ▶ **Goal:** minimize the empirical expected perceptron loss (risk):

$$L_{\text{tron}}(H(\vec{x}), y) = \begin{cases} 0, & \text{sign}(H(\vec{x})) = \text{sign}(y) \\ |H(\vec{x})|, & \text{sign}(H(\vec{x})) \neq \text{sign}(y) \end{cases}$$

# Optimization

- The perceptron risk is:

$$\begin{aligned} R_{\text{tron}}(\vec{w}) &= \frac{1}{n} \sum_{i=1}^n L_{\text{tron}}(H(\vec{x}_i), y_i) \\ &= \frac{1}{n} \sum_{i=1}^n \begin{cases} 0, & \text{sign}(H(\vec{x}_i)) = \text{sign}(y_i) \\ |H(\vec{x}_i)|, & \text{sign}(H(\vec{x}_i)) \neq \text{sign}(y_i) \end{cases} \end{aligned}$$

## Exercise

Suppose the training data are **linearly separable**.  
What is the minimum possible value of  $R_{\text{tron}}$ ?

# Optimization

- ▶ To optimize, solve  $\vec{\nabla}R_{\text{tron}}(\vec{w}) = 0$ ?
- ▶ The gradient of the risk would be:

$$\begin{aligned}\vec{\nabla}R_{\text{tron}}(\vec{w}) &= \vec{\nabla} \frac{1}{n} \sum_{i=1}^n L_{\text{tron}}(H(\vec{x}_i), y_i) \\ &= \frac{1}{n} \sum_{i=1}^n \vec{\nabla}L_{\text{tron}}(H(\vec{x}_i), y_i)\end{aligned}$$

- ▶ However, the perceptron loss is **not differentiable**.

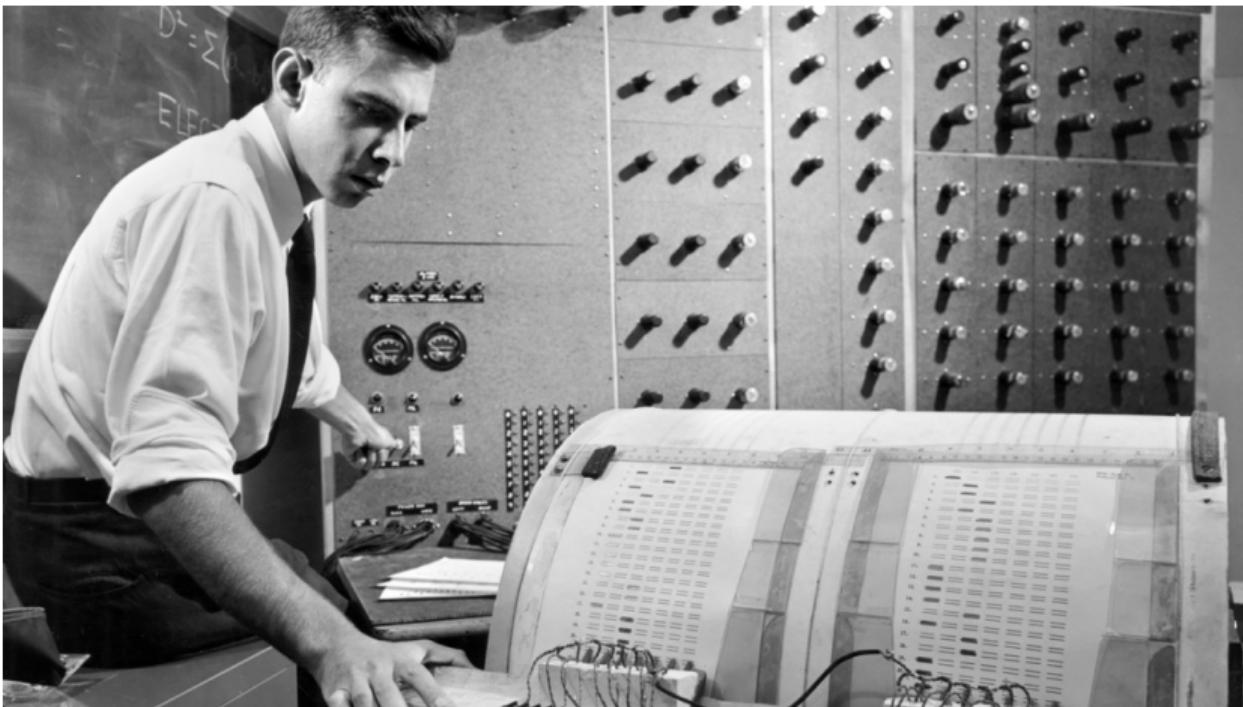
# Problem

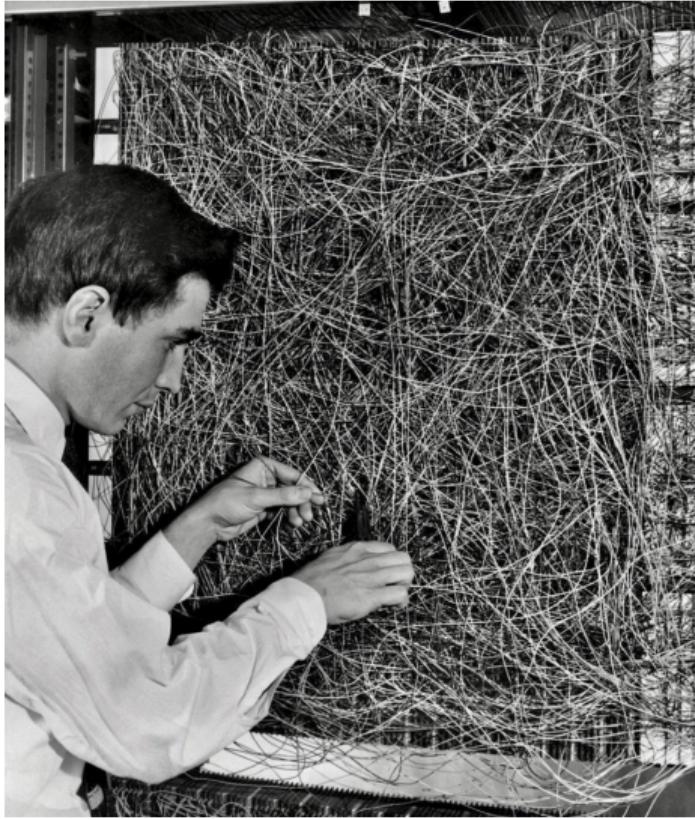
- ▶ The risk is **not differentiable**.
- ▶ However:
  - ▶ it is **not flat!**
  - ▶ its derivative exists almost everywhere.
- ▶ We can train iteratively using **subgradient descent** (as we'll see).

# Some History

- ▶ Perceptrons were one of the first “machine learning” models.
- ▶ The basis of modern neural networks.

# Rosenblatt's Perceptron





# DSC 140A

*Probabilistic Modeling & Machine Learning*

Lecture 4 | Part 2

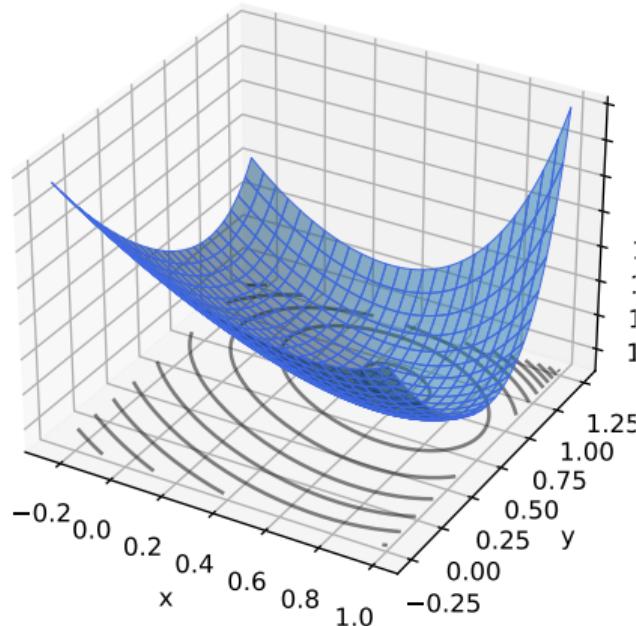
**Gradient Descent**

# Iterative Optimization

- ▶ To minimize a function  $f(\vec{x})$ , we may try to compute  $\vec{\nabla}f(\vec{x})$ ; set to 0; solve.
- ▶ Often, there is **no closed-form solution**.
- ▶ How do we minimize  $f$ ?

# Example

- ▶ Consider  $f(x, y) = e^{x^2+y^2} + (x - 2)^2 + (y - 3)^2$ .



# Example

► Try solving  $\vec{\nabla}f(x, y) = 0$ .

► The gradient is:

$$\vec{\nabla}f(x, y) = \begin{pmatrix} 2xe^{x^2+y^2} + 2(x - 2) \\ 2ye^{x^2+y^2} + 2(y - 3) \end{pmatrix}$$

► Can we solve the system?

$$2xe^{x^2+y^2} + 2(x - 2) = 0$$

$$2ye^{x^2+y^2} + 2(y - 3) = 0$$

# Example

► Try solving  $\vec{\nabla}f(x, y) = 0$ .

► The gradient is:

$$\vec{\nabla}f(x, y) = \begin{pmatrix} 2xe^{x^2+y^2} + 2(x - 2) \\ 2ye^{x^2+y^2} + 2(y - 3) \end{pmatrix}$$

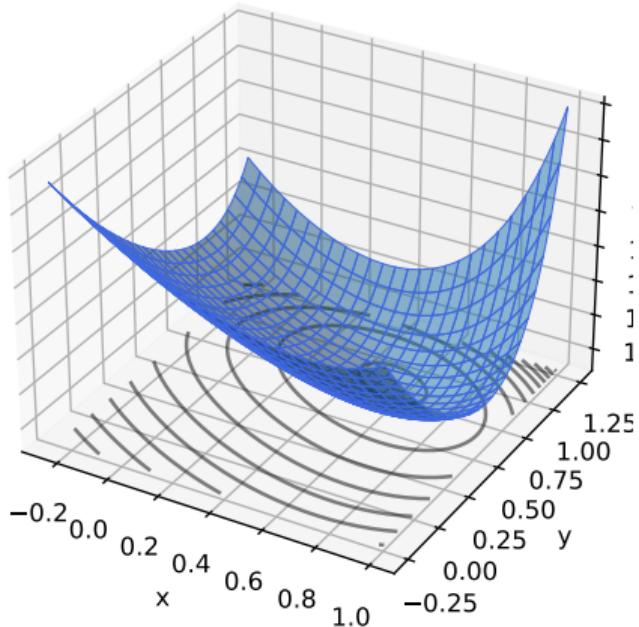
► Can we solve the system? **Not in closed form.**

$$2xe^{x^2+y^2} + 2(x - 2) = 0$$

$$2ye^{x^2+y^2} + 2(y - 3) = 0$$

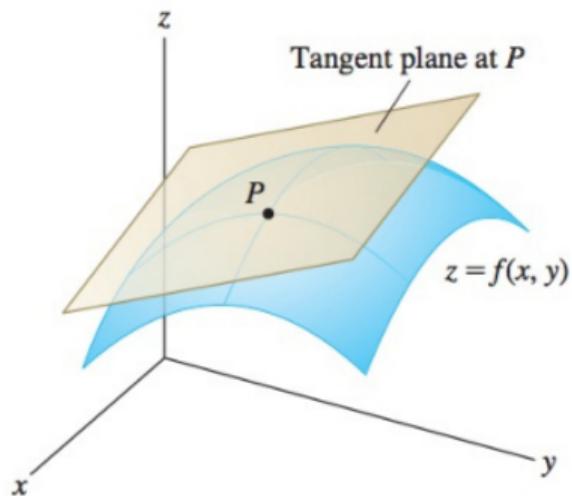
# Idea

- ▶ Apply an iterative approach.
- ▶ Start at an arbitrary location.
- ▶ “Walk downhill”, towards minimum.

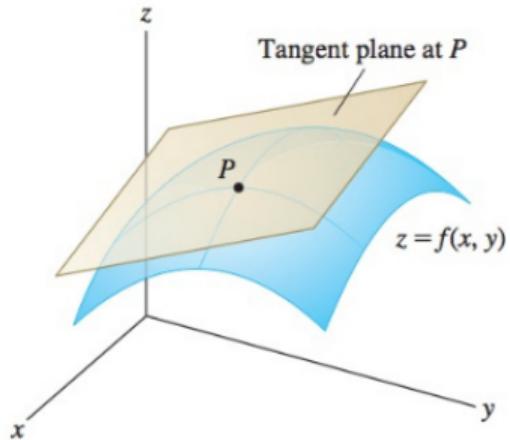


# Which way is down?

- ▶ Consider a differentiable function  $f(x, y)$ .
- ▶ We are standing at  $P = (x_0, y_0)$ .
- ▶ In a small region around  $P$ ,  $f$  looks like a plane.



# Which way is down?



- ▶ Linear approximation:

$$f(x_0 + \delta_x, y_0 + \delta_y) \approx f(x_0, y_0) +$$

# Which way is down?

- ▶ Define  $\vec{\delta} = (\delta_x, \delta_y)^T$
- ▶ Define the **gradient**:  $\vec{\nabla}f(x, y) = \left( \frac{\partial f}{\partial x}(x, y), \frac{\partial f}{\partial y}(x, y) \right)^T$
- ▶ Then the linear approximation becomes:

$$f(x_0 + \delta_x, y_0 + \delta_y) \approx f(x_0, y_0) + \vec{\delta} \cdot \vec{\nabla}f(x_0, y_0)$$

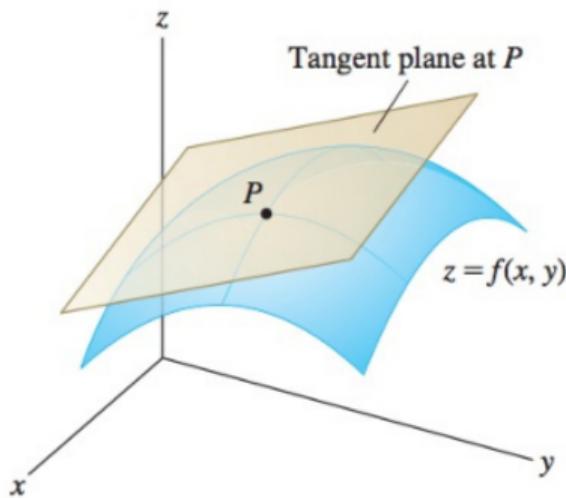
# Which way is up?

$$f(x_0 + \delta_x, y_0 + \delta_y) = f(x_0, y_0) + \vec{\delta} \cdot \vec{\nabla}f(x_0, y_0)$$

- ▶ Suppose  $\delta$  is a unit vector (a **direction**).
- ▶ Which direction results in the greatest **increase**?
- ▶ Recall  $\vec{\delta} \cdot \vec{\nabla}f(x_0, y_0) = \|\vec{\delta}\| \|\vec{\nabla}f(x_0, y_0)\| \cos \theta$
- ▶ Answer: choosing  $\delta$  in the direction of gradient.

# Which way is down?

- ▶  $\vec{\nabla}f(x_0, y_0)$  points in direction of steepest **ascent** at  $(x_0, y_0)$ .
- ▶  $-\vec{\nabla}f(x_0, y_0)$  points in direction of steepest **descent** at  $(x_0, y_0)$ .



# The Gradient

- ▶ Let  $f : \mathbb{R}^d \rightarrow \mathbb{R}$  be differentiable. The **gradient** of  $f$  at  $\vec{x}$  is defined:

$$\vec{\nabla}f(\vec{x}) = \left( \frac{\partial f}{\partial x_1}(\vec{x}), \frac{\partial f}{\partial x_2}(\vec{x}), \dots, \frac{\partial f}{\partial x_d}(\vec{x}) \right)^T$$

- ▶ **Note:**  $\vec{\nabla}f(\vec{x})$  is a **function** mapping  $\mathbb{R}^d \rightarrow \mathbb{R}^d$ .

# Gradient Properties

- ▶ The gradient is used in the linear approximation of  $f$ :

$$f(x_0 + \delta_x, y_0 + \delta_y) \approx f(x_0, y_0) + \vec{\delta} \cdot \vec{\nabla}f(x_0, y_0)$$

- ▶ Important properties:
  - ▶  $\vec{\nabla}f(\vec{x})$  points in direction of **steepest ascent** at  $\vec{x}$ .
  - ▶  $-\vec{\nabla}f(\vec{x})$  points in direction of **steepest descent** at  $\vec{x}$ .
  - ▶ In directions orthogonal to  $\vec{\nabla}f(\vec{x})$ ,  $f$  does not change!
  - ▶  $\|\vec{\nabla}f(\vec{x})\|$  measures steepness of ascent

# Gradient Descent

- ▶ Pick arbitrary starting point  $\vec{x}^{(0)}$ , learning rate parameter  $\eta > 0$ .
- ▶ Until convergence, repeat:
  - ▶ Compute gradient of  $f$  at  $\vec{x}^{(i)}$ ; that is, compute  $\vec{\nabla}f(\vec{x}^{(i)})$ .
  - ▶ Update  $\vec{x}^{(i+1)} = \vec{x}^{(i)} - \eta \vec{\nabla}f(\vec{x}^{(i)})$ .
- ▶ When do we stop?
  - ▶ When difference between  $\vec{x}^{(i)}$  and  $\vec{x}^{(i+1)}$  is negligible.
  - ▶ I.e., when  $\|\vec{x}^{(i)} - \vec{x}^{(i+1)}\|$  is small.

```
def gradient_descent(
    gradient, x, learning_rate=.01,
    threshold=.1e-4
):
    while True:
        x_new = x - learning_rate * gradient(x)
        if np.linalg.norm(x - x_new) < threshold:
            break
        x = x_new
    return x
```

# Example

- ▶ **Goal:** minimize  $f(x, y) = e^{x^2+y^2} + (x - 2)^2 + (y - 3)^2$  with gradient descent.
- ▶ Initial location  $\vec{x}^{(0)} = (\frac{1}{2}, 0)^T$ ; learning rate  $\eta = 0.05$
- ▶ Recall:

$$\vec{\nabla}f(x, y) = \begin{pmatrix} 2xe^{x^2+y^2} + 2(x - 2) \\ 2ye^{x^2+y^2} + 2(y - 3) \end{pmatrix}$$

# Example: First Step

- ▶ Compute gradient at  $\vec{x}^{(0)} = (\frac{1}{2}, 0)^T$ :

$$\vec{\nabla}f(\frac{1}{2}, 0) = \begin{pmatrix} 2 \cdot (1/2) \cdot e^{(1/2)^2+0^2} + 2((1/2) - 2) \\ 2 \cdot 0 \cdot e^{(1/2)^2+0^2} + 2(0 - 3) \end{pmatrix} \approx \begin{pmatrix} -1.71 \\ -6 \end{pmatrix}$$

- ▶ Update:

$$\begin{aligned}\vec{x}^{(1)} &= \vec{x}^{(0)} - \eta \vec{\nabla}f(\vec{x}^{(0)}) \\ &= (\frac{1}{2}, 0)^T - .05 \cdot (-1.71, -6)^T \\ &= (\frac{1}{2}, 0)^T - (-.08, -.3)^T \\ &= (.58, .3)^T\end{aligned}$$

# Example: Second Step

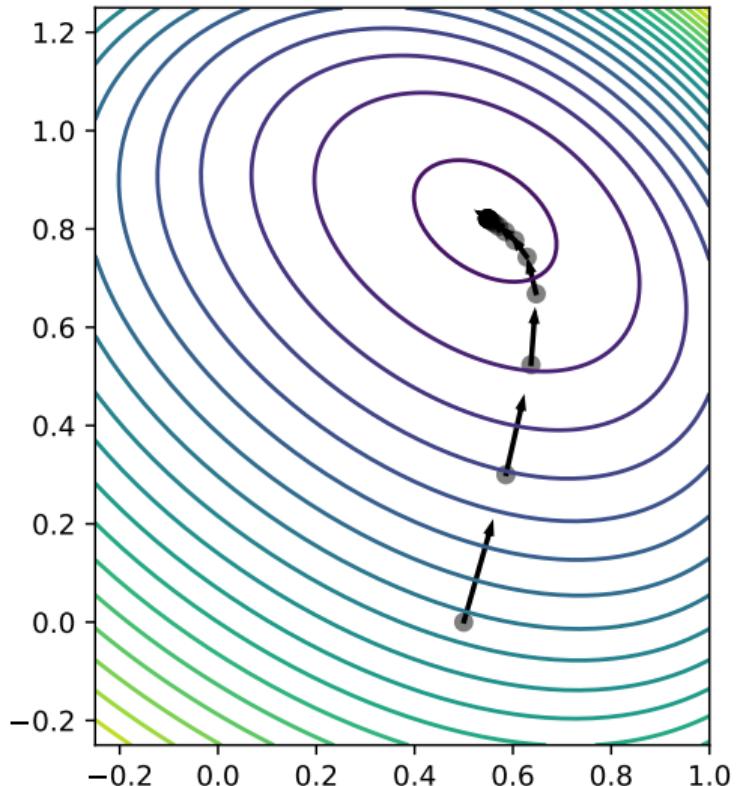
- ▶ Compute  $\vec{\nabla}f$  at  $\vec{x}^{(1)}$ :

$$\vec{\nabla}f(\vec{x}^{(1)}) = \vec{\nabla}f(.58, .3)^T \approx (-1.06, -4.48)^T$$

- ▶ Update:

$$\begin{aligned}\vec{x}^{(2)} &= \vec{x}^{(1)} - \eta \vec{\nabla}f(\vec{x}^{(1)}) \\ &= (.58, .3)^T - .05 \cdot (-1.06, -4.48)^T \\ &= (.63, .52)^T\end{aligned}$$

- ▶ Repeat...



# DSC 140A

*Probabilistic Modeling & Machine Learning*

Lecture 4 | Part 3

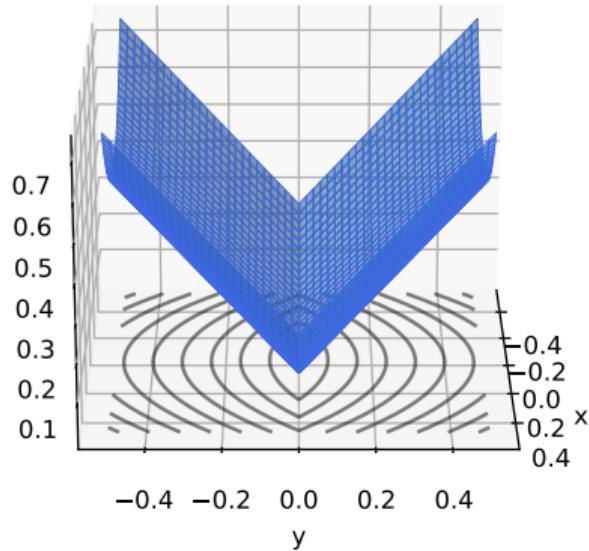
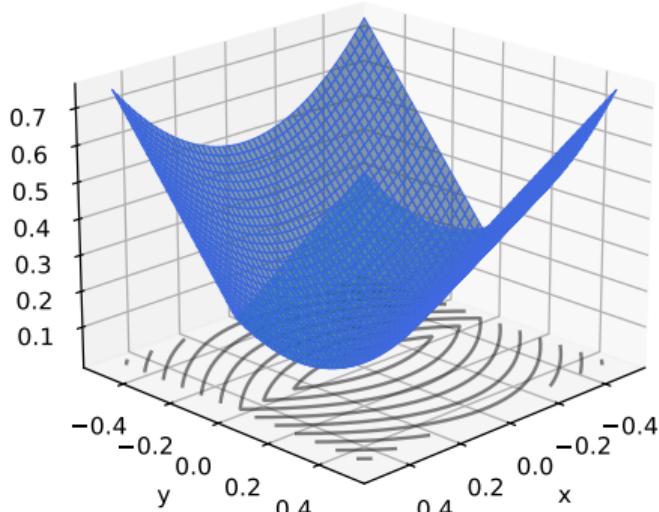
**Subgradient Descent**

# A Problem

- ▶ To perform gradient descent, function must be **differentiable**.
- ▶ What if it isn't?

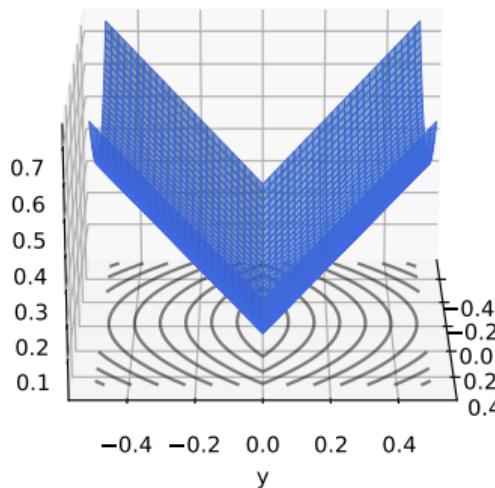
# Example

►  $f(x, y) = x^2 + |y|$



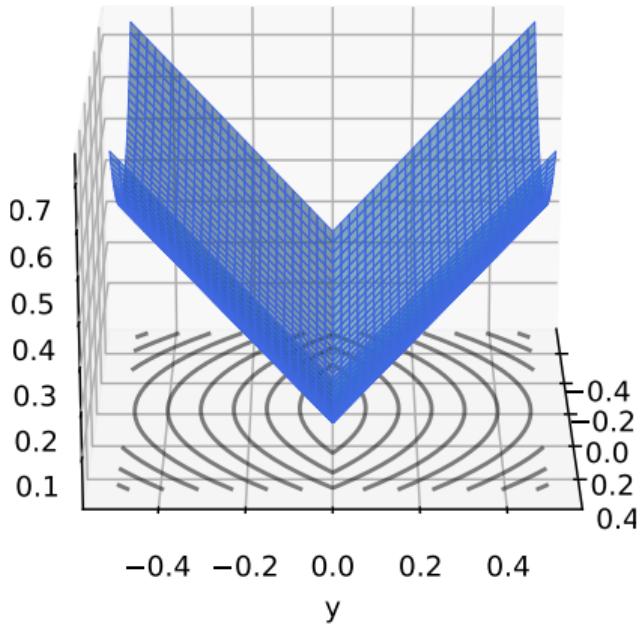
## Exercise

Where is the function **not** differentiable? That is, where is there not a well-defined tangent plane?



# Answer

- ▶  $\vec{\nabla}f(x, y)$  is defined everywhere except along  $y = 0$ .
  
- ▶ If  $y > 0$ ,  
 $f(x, y) = x^2 + |y| = x^2 + y.$ 
  - ▶  $\vec{\nabla}f(x, y) = (2x, 1)^T$
  
- ▶ If  $y < 0$ ,  
 $f(x, y) = x^2 + |y| = x^2 - y.$ 
  - ▶  $\vec{\nabla}f(x, y) = (2x, -1)^T$

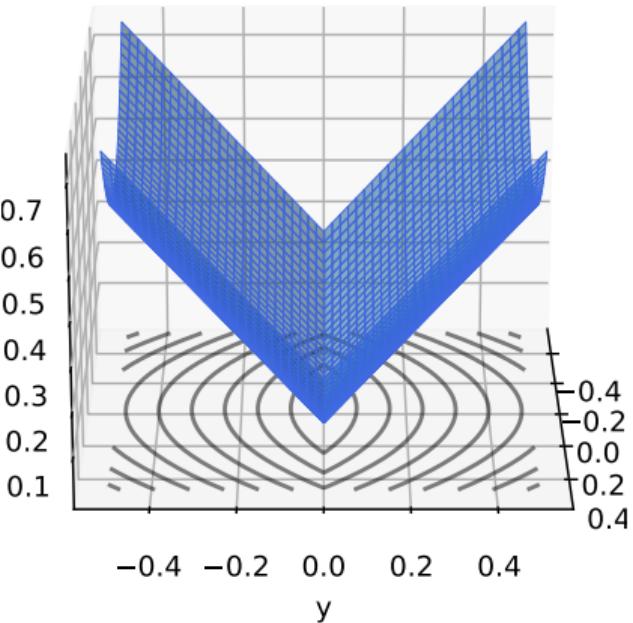


# Answer

- ▶ Piecewise gradient:

$$\vec{\nabla}f(x, y) = \begin{cases} (2x, 1)^T & , \text{if } y > 1, \\ (2x, -1)^T & , \text{if } y < 1. \end{cases}$$

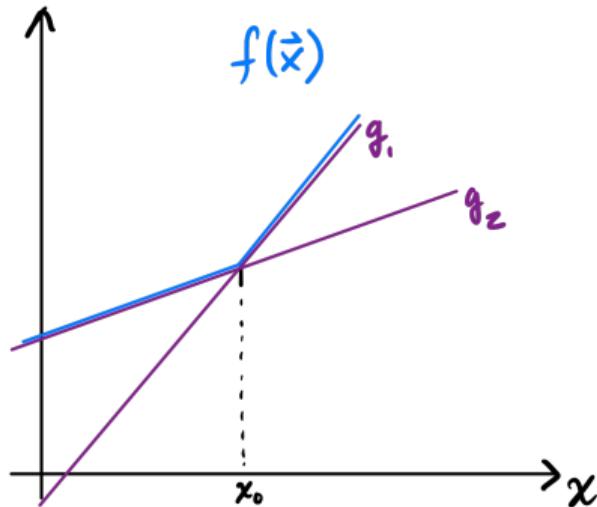
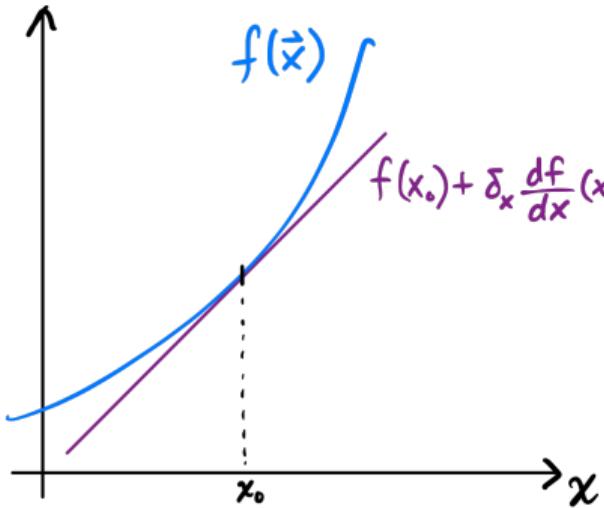
- ▶ **Not defined** along  $y = 0$ .



# Problem

- ▶ Gradient descent will *almost* work on  $f(x, y) = x^2 + |y|$ .
- ▶ What if it reaches a point where  $y = 0$ .
- ▶ Which direction should it go?

# Intuition



# Subgradient

- ▶ A **subgradient** of  $f$  at  $\vec{x}^{(0)}$  is a vector  $\vec{g}$  such that:

$$f(\vec{x}^{(0)}) + \vec{\delta} \cdot \vec{g}$$

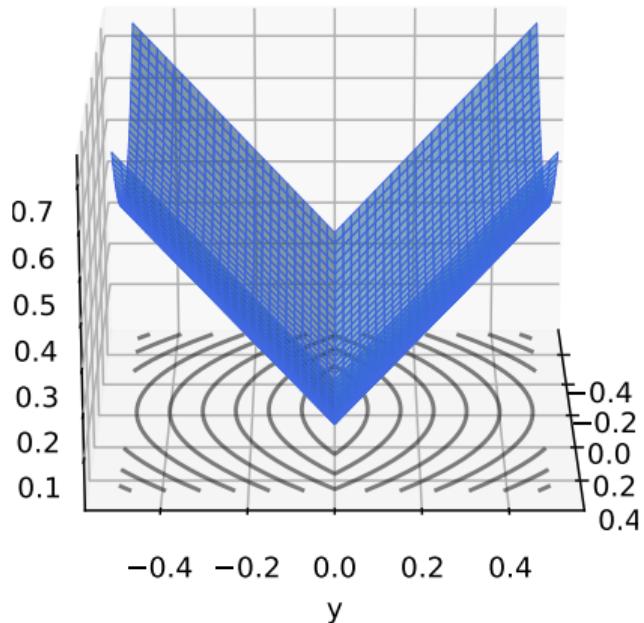
lies below  $f$  for any choice of  $\vec{\delta}$ .

- ▶ There are possibly (infinitely) many subgradients.

# Example

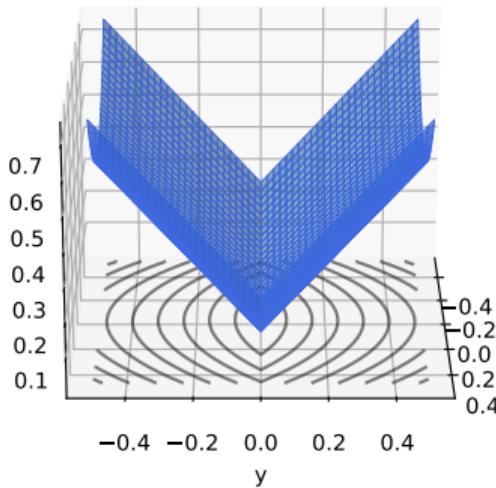
- ▶  $f(x, y) = x^2 + |y|$
- ▶ A subgradient:

$$\vec{g}(x, y) = \begin{cases} (2x, 1)^T & , \text{if } y > 1, \\ (2x, -1)^T & , \text{if } y < 1, \\ (2x, 0)^T & , \text{if } y = 0. \end{cases}$$



## Exercise

Find another subgradient of the function  $f(x, y) = x^2 + |y|$ .



# Subgradient Descent

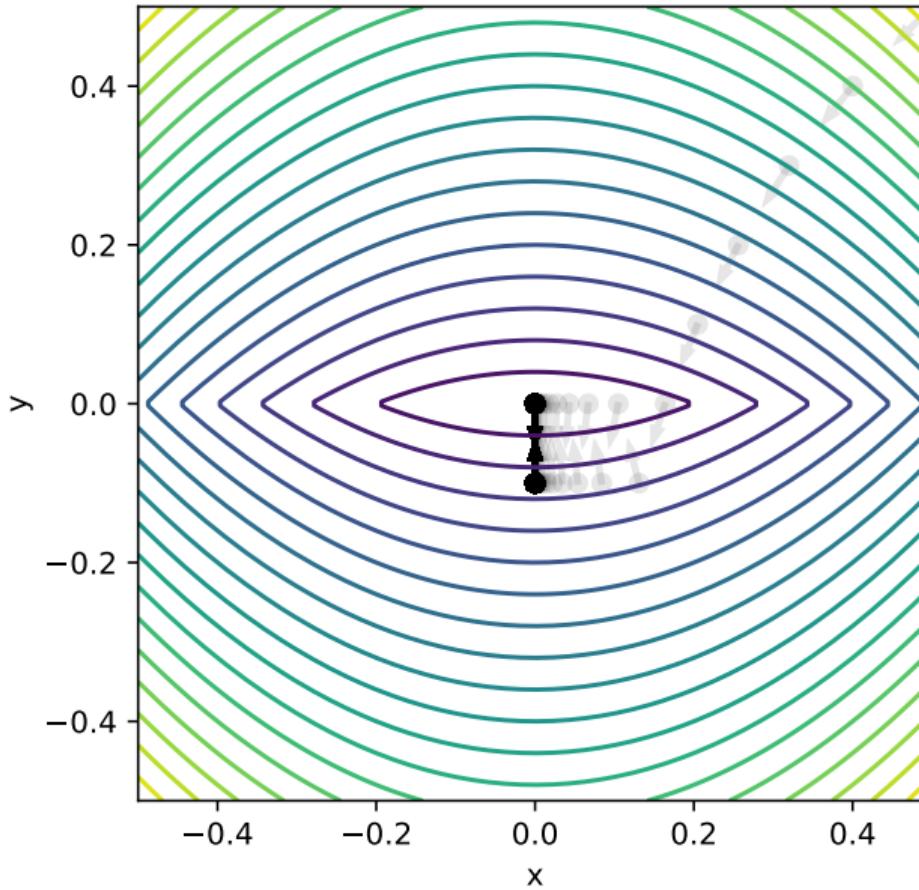
- ▶ **Idea:** use subgradient in place of gradient in gradient descent.
- ▶ Subgradient is not unique; choose an arbitrary one.

# Subgradient Descent

- ▶ Pick arbitrary starting point  $\vec{x}^{(0)}$ , **learning rate** parameter  $\eta > 0$ .
- ▶ Until convergence, repeat:
  - ▶ Compute a **subgradient** of  $f$  at  $\vec{x}^{(i)}$ .
  - ▶ Update  $\vec{x}^{(i+1)} = \vec{x}^{(i)} - \eta \vec{\nabla} f(\vec{x}^{(i)})$ .
- ▶ When do we stop?
  - ▶ When difference between  $\vec{x}^{(i)}$  and  $\vec{x}^{(i+1)}$  is negligible.
  - ▶ I.e., when  $\|\vec{x}^{(i)} - \vec{x}^{(i+1)}\|$  is small.

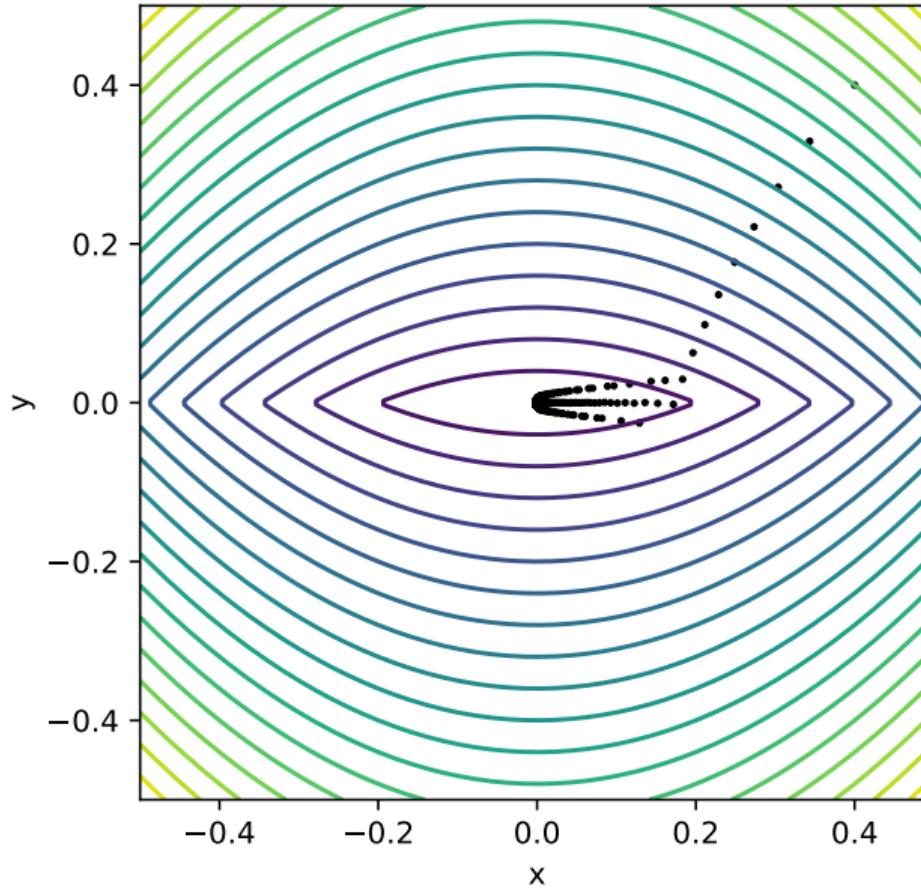
# Example

- ▶ Subgradient descent on  $f(x, y) = x^2 + |y|$
- ▶ Starting point:  $(1/2, 1/2)^T$
- ▶ Learning rate:  $\eta = 0.1$ .



# Problem

- ▶ Does not converge! Why?
- ▶ **Fix:** decrease learning rate with each iteration.
- ▶ Theory: choose  $\eta = O(1/\sqrt{i})$ , where  $i$  is iteration #.



# DSC 140A

*Probabilistic Modeling & Machine Learning*

Lecture 4 | Part 4

**(Sub)gradient Descent for ERM**

# ERM for the Perceptron

- ▶ **Goal:** minimize the empirical expected perceptron loss (risk):

$$L_{\text{tron}}(H(\vec{x}), y) = \begin{cases} 0, & \text{sign}(H(\vec{x})) = \text{sign}(y) \\ |H(\vec{x})|, & \text{sign}(H(\vec{x})) \neq \text{sign}(y) \end{cases}$$

# Optimization

- The perceptron risk is:

$$\begin{aligned} R_{\text{tron}}(\vec{w}) &= \frac{1}{n} \sum_{i=1}^n L_{\text{tron}}(H(\vec{x}_i), y_i) \\ &= \frac{1}{n} \sum_{i=1}^n \begin{cases} 0, & \text{sign}(H(\vec{x}_i)) = \text{sign}(y_i) \\ |H(\vec{x}_i)|, & \text{sign}(H(\vec{x}_i)) \neq \text{sign}(y_i) \end{cases} \end{aligned}$$

- To optimize, solve  $\vec{\nabla}R_{\text{tron}}(\vec{w}) = 0$ ?

# Optimization

- ▶ The gradient of the risk would be:

$$\begin{aligned}\vec{\nabla} R_{\text{tron}}(\vec{w}) &= \vec{\nabla} \frac{1}{n} \sum_{i=1}^n L_{\text{tron}}(H(\vec{x}), y) \\ &= \frac{1}{n} \sum_{i=1}^n \vec{\nabla} L_{\text{tron}}(H(\vec{x}), y)\end{aligned}$$

- ▶ However, the perceptron loss is **not differentiable**.

# Idea

- ▶ Instead of solving  $\vec{\nabla}R_{\text{tron}}(\vec{w}) = 0$ , use **subgradient descent**.
- ▶ We need to compute a **subgradient** of the perceptron loss.

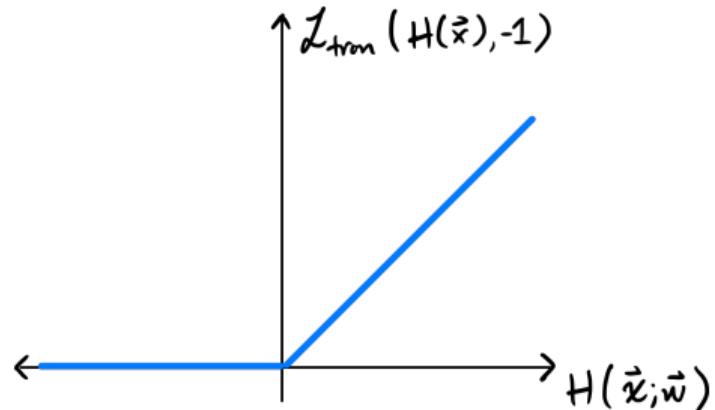
# Subgradient of Perceptron Loss

$$L_{\text{tron}}(H(\vec{x}; \vec{w}), y) = \begin{cases} 0, & \text{sign}(H(\vec{x})) = \text{sign}(y) \\ |\vec{x} \cdot \vec{w}|, & \text{sign}(H(\vec{x})) \neq \text{sign}(y) \end{cases}$$

- ▶ If prediction is **correct**,  $\vec{\nabla} L_{\text{tron}} = 0$ .
- ▶ If prediction is **incorrect** with  $\vec{x} \cdot \vec{w} > 0$ ,
  - ▶ then  $|\vec{x} \cdot \vec{w}| = \vec{x} \cdot \vec{w}$ .
  - ▶  $\vec{\nabla} L_{\text{tron}} = \frac{d}{d\vec{w}}(\vec{x} \cdot \vec{w} - y) = \vec{x}$
- ▶ If prediction is **incorrect** with  $\vec{x} \cdot \vec{w} < 0$ ,
  - ▶ then  $|\vec{x} \cdot \vec{w}| = -\vec{x} \cdot \vec{w}$ .
  - ▶  $\vec{\nabla} L_{\text{tron}} = \frac{d}{d\vec{w}}(-\vec{x} \cdot \vec{w} - y) = -\vec{x}$

# Subgradient of Perceptron Loss

- ▶ Gradient is not defined when  $H(\vec{x}; \vec{w}) = 0$ .
- ▶ Can choose any subgradient.
- ▶  $\vec{0}$  works.



# Subgradient

- ▶ A subgradient of the perceptron loss:

$$\vec{g}(\vec{w}, \vec{x}, y) = \begin{cases} 0, & \text{if } \text{sign}(\vec{w} \cdot \vec{x}) = \text{sign}(y) \text{ or } \vec{x} \cdot \vec{w} = 0, \\ \vec{x}, & \text{if } \text{sign}(\vec{w} \cdot \vec{x}) \neq \text{sign}(y) \text{ and } \vec{x} \cdot \vec{w} > 0, \\ -\vec{x}, & \text{if } \text{sign}(\vec{w} \cdot \vec{x}) \neq \text{sign}(y) \text{ and } \vec{x} \cdot \vec{w} < 0. \end{cases}$$

- ▶ Subgradient of the risk

$$\vec{g}_R(\vec{w}) = \frac{1}{n} \sum_{i=1}^n g(\vec{w}, \vec{x}^{(i)}, y_i)$$

# Training a Perceptron

- ▶ A perceptron can be trained with subgradient descent, using  $\vec{g}_R$  as the subgradient.
- ▶ If the data are linearly separable<sup>3</sup>, will obtain perfect accuracy on training data.

---

<sup>3</sup>and the learning rate is appropriately chosen

# Gradient Descent for ERM

- ▶ Suppose  $L$  is a **differentiable** loss function (e.g., square loss).
- ▶ Then the gradient of the risk,  $R(\vec{w})$ , is:

$$\vec{\nabla}R(\vec{w}) = \frac{1}{n} \sum_{i=1}^n \vec{\nabla}L(H(\vec{w}, \vec{x}^{(i)}), y_i)$$

# Subgradient Descent for ERM

- ▶ If  $L$  is not differentiable, we may still be able to use subgradient descent if a subgradient exists.
- ▶ If  $\vec{g}(\vec{w}, \vec{x}, y)$  is a subgradient of  $L$ , then a subgradient of the risk,  $R$ , is:

$$\frac{1}{n} \sum_{i=1}^n g(\vec{w}, \vec{x}^{(i)}, y_i)$$

## What's left?

- ▶ Does (sub)gradient descent always work? **No.**
  - ▶ When is it guaranteed to work?
- ▶ Computing the full gradient can be costly.

# DSC 140A

*Probabilistic Modeling & Machine Learning*

Lecture 4 | Part 5

**Perceptron Demo: MNIST**

# Demo: MNIST

- ▶ MNIST is a classic machine learning data set.
- ▶ Many images of handwritten digits, 0-9.
- ▶ Multiclass classification problem.
- ▶ But we can make it binary: 3 vs. 7.

# Example MNIST Digit



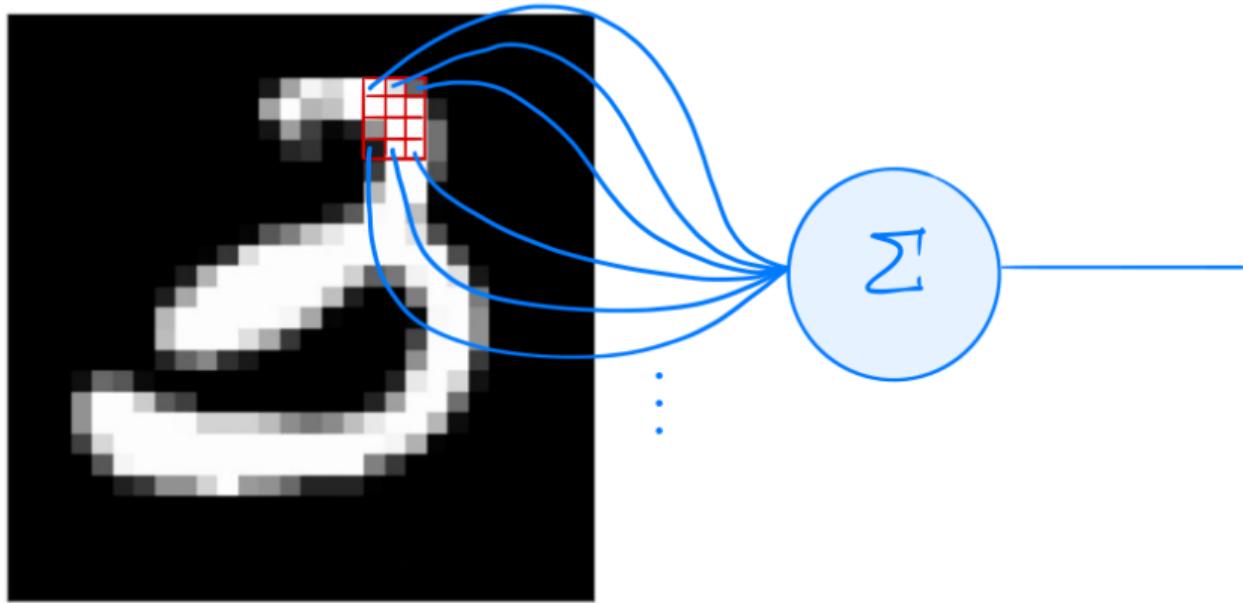
- ▶ Grayscale
- ▶ 28 x 28 pixels

# MNIST Feature Vectors

- ▶  $28 \times 28 = 784$  pixels
- ▶ Each image is a vector in  $\mathbb{R}^{784}$
- ▶ Each feature is intensity of single pixel
  - ▶ black → 0, white → 255
- ▶ A **very** simple representation.

# Demo: MNIST

- ▶ Use only images of 3s and 7s.
- ▶ 4132 training images.
- ▶ 680 testing images.
- ▶ Some minor tuning.
  - ▶ Added random noise for robustness.
  - ▶ Picked classification threshold automatically.



# Perceptron Learning

- ▶ Linear prediction function parameterized by  $\vec{w}$ .
- ▶ In this case, we can “reshape”  $\vec{w}$  to be same size as input image.

# Weight Vector

- ▶ Recall that the prediction is a **weighted vote**:

$$H(\vec{x}) = \text{sign}(w_0 + w_1x_1 + w_2x_2 + \dots + w_{784}x_{784})$$

- ▶ Positive → 7, Negative → 3
- ▶  $w_i$  is the weight of pixel  $i$ 
  - ▶ positive: if this pixel is bright, I think this is a 7
  - ▶ negative: if this pixel is bright, I think this is a 3
  - ▶ magnitude: confidence in prediction

# Perceptron Training



# Perceptron Training



# Perceptron Training



# Perceptron Training



# Perceptron Training

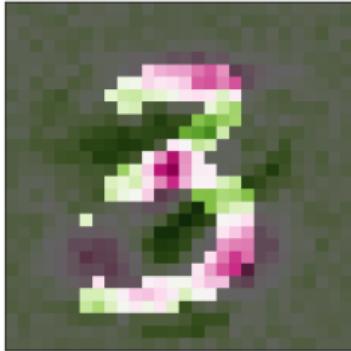


# Perceptron Training

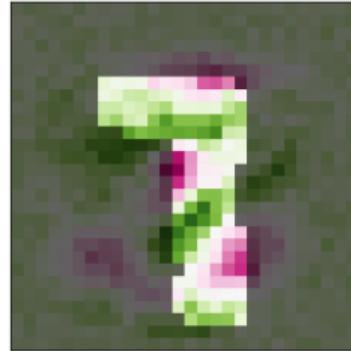


# Perceptron Weight Vector

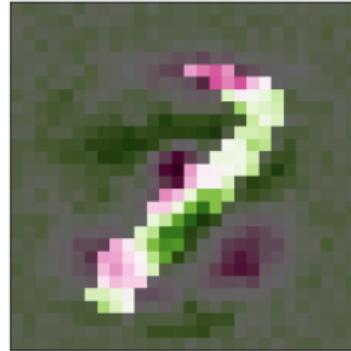
I predict that this is a 3!



I predict that this is a 7!



I predict that this is a 3!



# **Perceptron Results**

- ▶ Test accuracy: 97.3%

# **Square Loss for Classification**

- ▶ What if we use square loss for classification?
- ▶ We *can*, but will it work well?

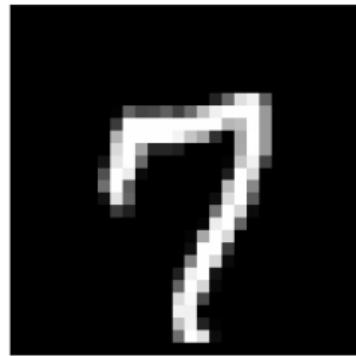
# Results: Least Squares

- ▶ Test Accuracy: 96.7% (marginally worse)

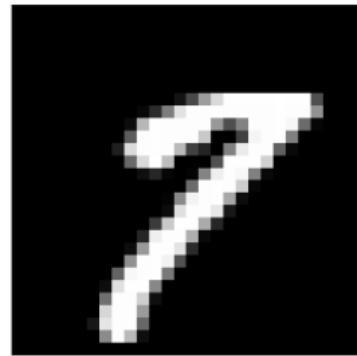
I think that this is a 3.



I think that this is a 7.



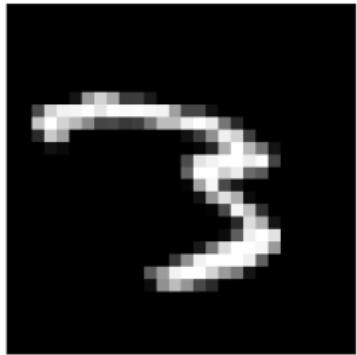
I think that this is a 7.



# Results: Least Squares

- ▶ Misclassifications are telling.

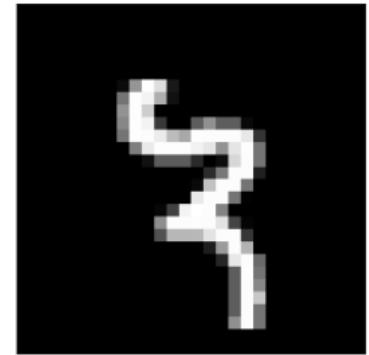
I think that this is a 7.



I think that this is a 7.

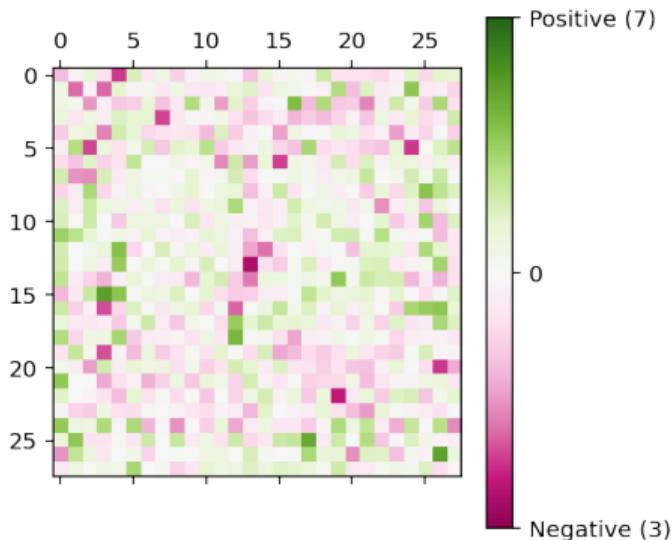


I think that this is a 7.



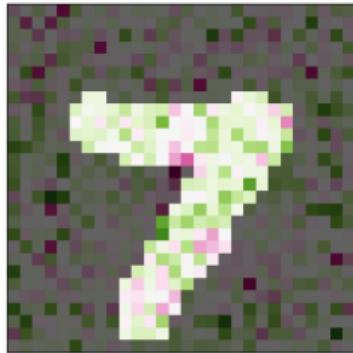
# Least Squares Weight Vector

- ▶ Can visualize weight of each pixel as an image.

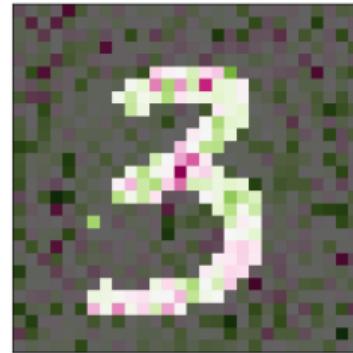


# Least Squares Weight Vector

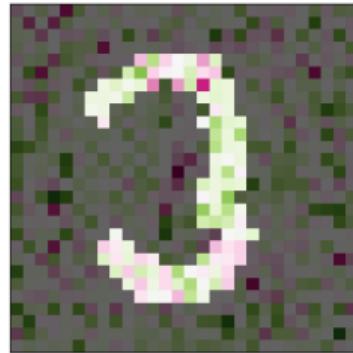
I predict that this is a 7!



I predict that this is a 3!



I predict that this is a 7!



# DSC 140A

*Probabilistic Modeling & Machine Learning*

Lecture 5 | Part 1

**Stochastic Gradient Descent**

# Recall: (Sub)gradient descent

- ▶ **Goal:** minimize function  $f(\vec{x})$ .
- ▶ Iterative procedure that takes small steps in direction of steepest descent.

# Recall: Gradient Descent

- ▶ Pick arbitrary starting point  $\vec{x}^{(0)}$ , learning rate parameter  $\eta > 0$ .
- ▶ Until convergence, repeat:
  - ▶ Compute gradient of  $f$  at  $\vec{x}^{(i)}$ ; that is, compute  $\vec{\nabla}f(\vec{x}^{(i)})$ .
  - ▶ Update  $\vec{x}^{(i+1)} = \vec{x}^{(i)} - \eta \vec{\nabla}f(\vec{x}^{(i)})$ .
- ▶ When do we stop?
  - ▶ When difference between  $\vec{x}^{(i)}$  and  $\vec{x}^{(i+1)}$  is negligible.
  - ▶ I.e., when  $\|\vec{x}^{(i)} - \vec{x}^{(i+1)}\|$  is small.

```
def gradient_descent(
    gradient, x, learning_rate=.01,
    threshold=.1e-4
):
    while True:
        x_new = x - learning_rate * gradient(x)
        if np.linalg.norm(x - x_new) < threshold:
            break
        x = x_new
    return x
```

# Gradient Descent for Minimizing Risk

- ▶ In ML, we often want to minimize a **risk function**:

$$R(\vec{w}) = \frac{1}{n} \sum_{i=1}^n L(H(\vec{x}^{(i)}; \vec{w}), y_i)$$

# Observation

- ▶ The gradient of the risk function is a sum of gradients:

$$\vec{\nabla}R(\vec{w}) = \frac{1}{n} \sum_{i=1}^n \vec{\nabla}L(H(\vec{x}^{(i)}; \vec{w}), y_i)$$

- ▶ One term for each point in training data.

# Problem

- ▶ In machine learning, the number of training points  $n$  can be **very large**.
- ▶ Computing the gradient can be **expensive** when  $n$  is large.
- ▶ Therefore, each step of gradient descent can be **expensive**.

# Idea

- ▶ The (full) gradient of the risk uses all of the training data:

$$\nabla R(\vec{w}) = \frac{1}{n} \sum_{i=1}^n \nabla L(H(\vec{x}^{(i)}; \vec{w}), y_i)$$

- ▶ It is an average of  $n$  gradients.
- ▶ **Idea:** instead of using all  $n$  points, randomly choose  $\ll n$ .

# Stochastic Gradient

- ▶ Choose a random subset (**mini-batch**)  $B$  of the training data.
- ▶ Compute a **stochastic gradient**:

$$\nabla R(\vec{w}) \approx \sum_{i \in B} \vec{\nabla} L(H(\vec{x}^{(i)}; \vec{w}), y_i)$$

# Stochastic Gradient

$$\nabla R(\vec{w}) \approx \sum_{i \in B} \vec{\nabla} L(H(\vec{x}^{(i)}; \vec{w}), y_i)$$

- ▶ **Good:** if  $|B| \ll n$ , this is much faster to compute.
- ▶ **Bad:** it is a (random) approximation of the full gradient, noisy.

# Stochastic Gradient Descent (SGD) for ERM

- ▶ Pick arbitrary starting point  $\vec{x}^{(0)}$ , learning rate parameter  $\eta > 0$ , batch size  $m \ll n$ .
- ▶ Until convergence, repeat:
  - ▶ Randomly sample a batch  $B$  of  $m$  training data points.
  - ▶ Compute stochastic gradient of  $f$  at  $\vec{x}^{(i)}$ :

$$\vec{g} = \sum_{i \in B} \vec{\nabla} L(H(\vec{x}^{(i)}; \vec{w}), y_i)$$

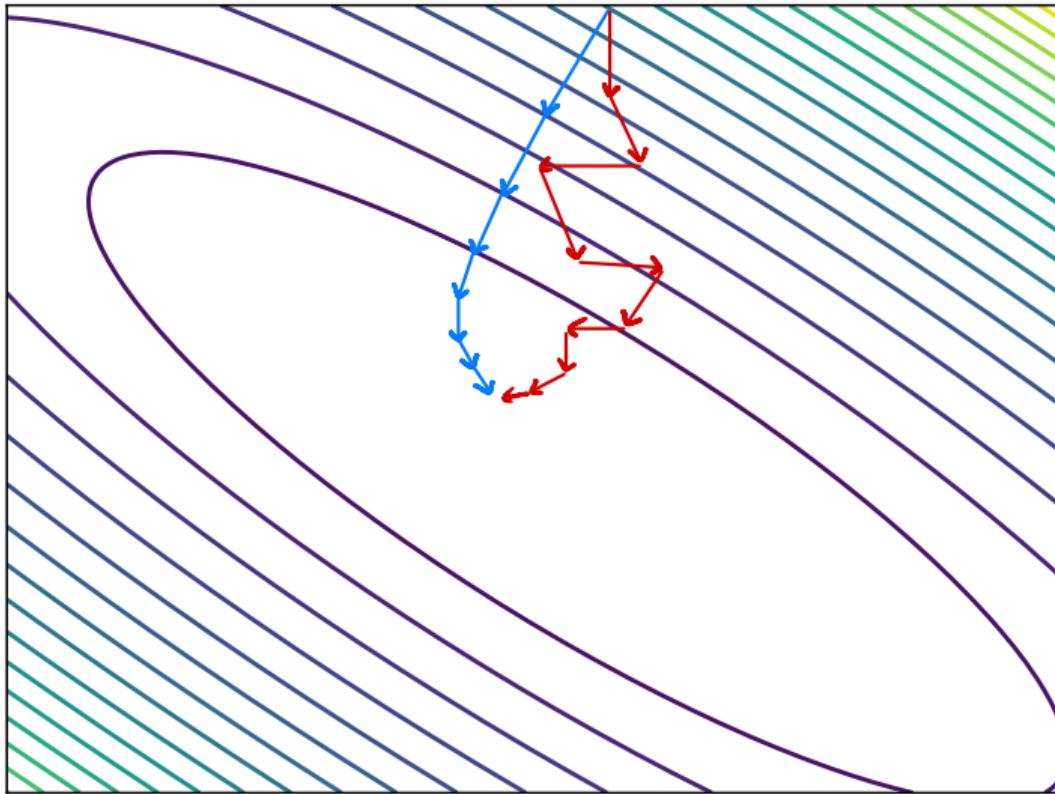
- ▶ Update  $\vec{x}^{(i+1)} = \vec{x}^{(i)} - \eta \vec{g}$

# Idea

- ▶ In practice, a stochastic gradient often works well enough.
- ▶ It is better to take many noisy steps quickly than few exact steps slowly.

# Batch Size

- ▶ Batch size  $m$  is a parameter of the algorithm.
- ▶ The larger  $m$ , the more reliable the stochastic gradient, but the more time it takes to compute.
- ▶ Extreme case when  $m = 1$  will still work.



# Usefulness of SGD

- ▶ SGD allows learning on **massive** data sets.
- ▶ Useful even when exactly solutions available.
  - ▶ E.g., least squares regression / classification.

# DSC 140A

*Probabilistic Modeling & Machine Learning*

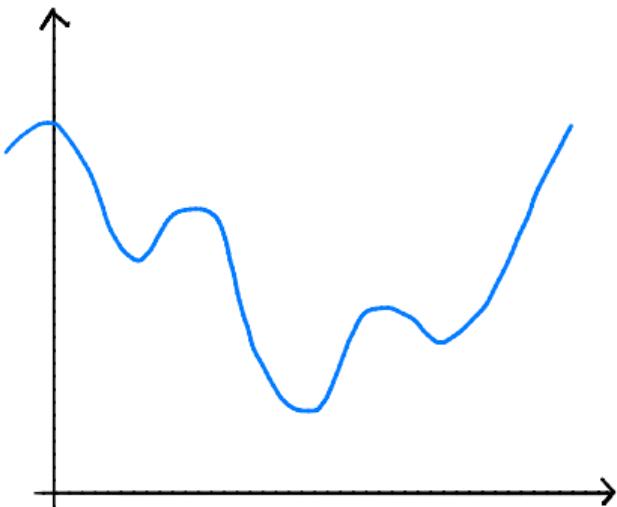
Lecture 5 | Part 2

**Convexity**

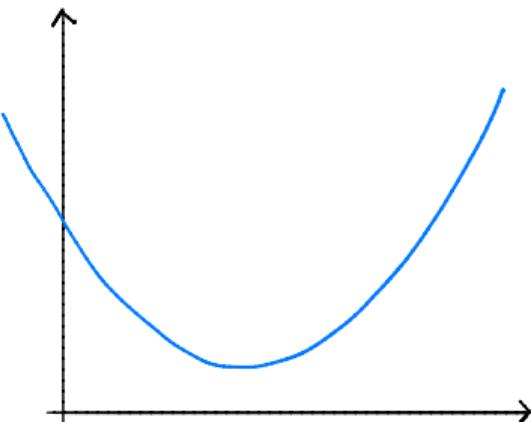
# Question

- ▶ When is gradient descent guaranteed to work?

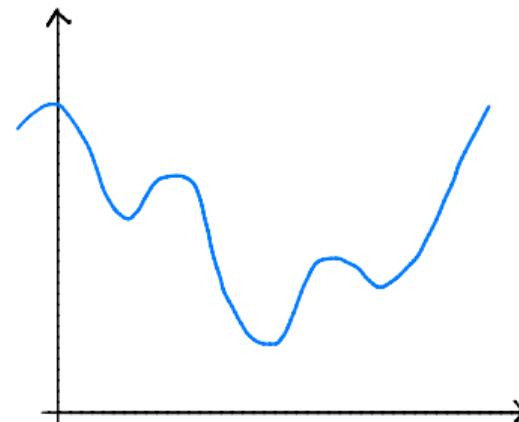
**Not here...**



# Convex Functions



Convex



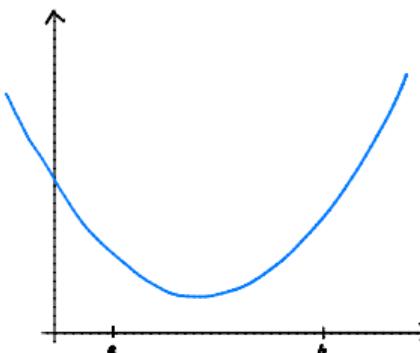
Non-convex

# Convexity: Definition

- ▶  $f$  is **convex** if for **every**  $a, b$  the line segment between

$$(a, f(a)) \quad \text{and} \quad (b, f(b))$$

does not go below the plot of  $f$ .

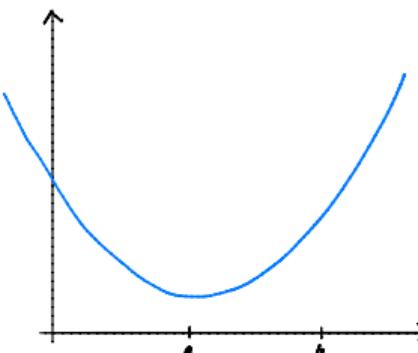


# Convexity: Definition

- ▶  $f$  is **convex** if for **every**  $a, b$  the line segment between

$$(a, f(a)) \quad \text{and} \quad (b, f(b))$$

does not go below the plot of  $f$ .

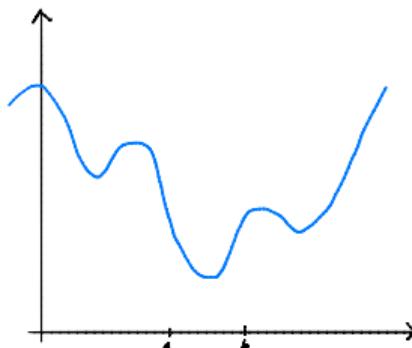


# Convexity: Definition

- ▶  $f$  is **convex** if for **every**  $a, b$  the line segment between

$$(a, f(a)) \quad \text{and} \quad (b, f(b))$$

does not go below the plot of  $f$ .

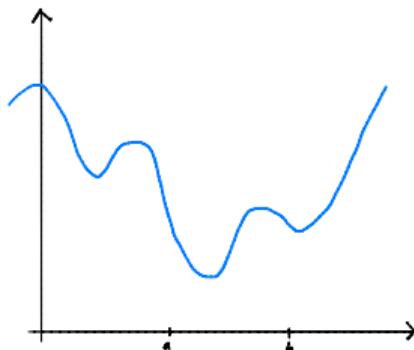


# Convexity: Definition

- ▶  $f$  is **convex** if for **every**  $a, b$  the line segment between

$$(a, f(a)) \quad \text{and} \quad (b, f(b))$$

does not go below the plot of  $f$ .



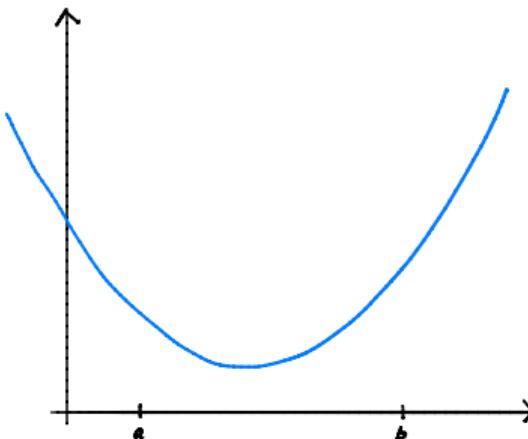
# Other Terms

- ▶ If a function is not convex, it is **non-convex**.
- ▶ **Strictly convex**: the line lies strictly above curve.
- ▶ **Concave**: the line lies on or below curve.

# Convexity: Formal Definition

- ▶ A function  $f : \mathbb{R} \rightarrow \mathbb{R}$  is **convex** if for every choice of  $a, b \in \mathbb{R}$  and  $t \in [0, 1]$ :

$$(1 - t)f(a) + tf(b) \geq f((1 - t)a + tb).$$

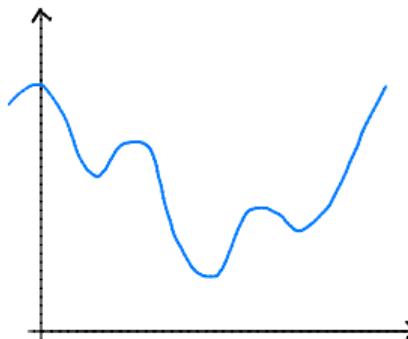
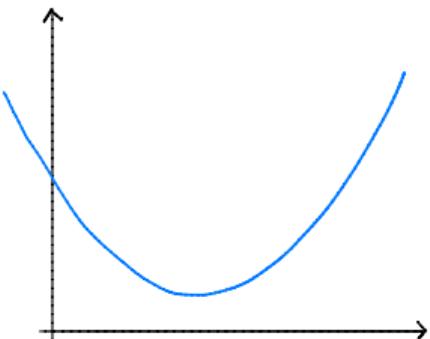


# Example

Is  $f(x) = |x|$  convex?

# Another View: Second Derivatives

- ▶ If  $\frac{d^2f}{dx^2}(x) \geq 0$  for all  $x$ , then  $f$  is convex.
- ▶ Example:  $f(x) = x^4$  is convex.
- ▶ **Warning!** Only works if  $f$  is twice differentiable!

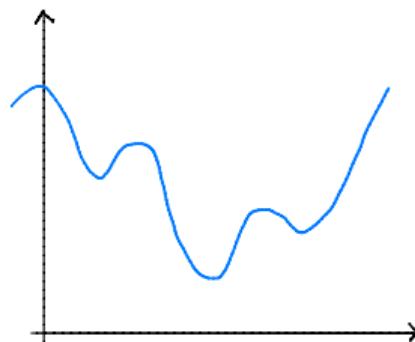
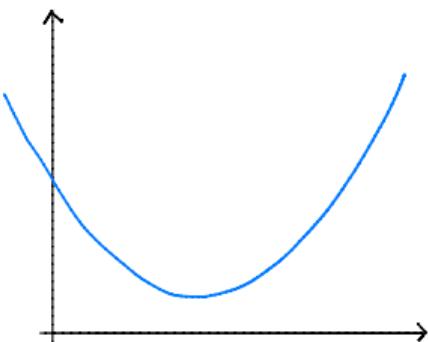


# Another View: Second Derivatives

- ▶ “Best” straight line at  $x_0$ :
  - ▶  $h_1(z) = f'(x_0) \cdot z + b$
- ▶ “Best” parabola at  $x_0$ :
  - ▶ At  $x_0$ ,  $f$  looks like  $h_2(z) = \frac{1}{2}f''(x_0) \cdot z^2 + f'(x_0)z + c$
  - ▶ Possibilities: upward-facing, downward-facing.

# Convexity and Parabolas

- ▶ Convex if for **every**  $x_0$ , parabola is upward-facing.
  - ▶ That is,  $f''(x_0) \geq 0$ .



# Proving Convexity Using Properties

Suppose that  $f(x)$  and  $g(x)$  are convex. Then:

- ▶  $w_1 f(x) + w_2 g(x)$  is convex, provided  $w_1, w_2 \geq 0$ 
  - ▶ Example:  $3x^2 + |x|$  is convex
- ▶  $g(f(x))$  is convex, provided  $g$  is non-decreasing.
  - ▶ Example:  $e^{x^2}$  is convex
- ▶  $\max\{f(x), g(x)\}$  is convex
  - ▶ Example:  $\begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases}$  is convex

# Convexity and Gradient Descent

- ▶ Convex functions are (relatively) easy to optimize.
- ▶ **Theorem:** if  $f(x)$  is convex and “not too steep”<sup>1</sup> then (stochastic) (sub)gradient descent converges to a **global optimum** of  $f$  provided that the step size is small enough<sup>2</sup>.

---

<sup>1</sup>Technically,  $c$ -Lipschitz

<sup>2</sup>step size related to steepness, should decrease like  $1/\sqrt{t}$ , where  $t$  is step number

# Nonconvexity and Gradient Descent

- ▶ Nonconvex functions are (relatively) hard to optimize.
- ▶ Gradient descent can still be useful.
- ▶ But not guaranteed to converge to a global minimum.

# DSC 140A

*Probabilistic Modeling & Machine Learning*

Lecture 5 | Part 3

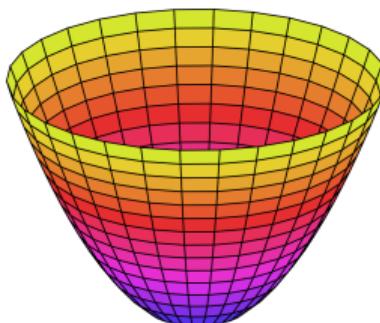
**Convexity in Many Dimensions**

# Convexity: Definition

- ▶  $f(\vec{x})$  is **convex** if for **every**  $\vec{a}, \vec{b}$  the line segment between

$$(\vec{a}, f(\vec{a})) \quad \text{and} \quad (\vec{b}, f(\vec{b}))$$

does not go below the plot of  $f$ .



# Convexity: Formal Definition

- ▶ A function  $f : \mathbb{R}^d \rightarrow \mathbb{R}$  is **convex** if for every choice of  $\vec{a}, \vec{b} \in \mathbb{R}^d$  and  $t \in [0, 1]$ :

$$(1 - t)f(\vec{a}) + tf(\vec{b}) \geq f((1 - t)\vec{a} + t\vec{b}).$$

# The Second Derivative Test

- ▶ For 1-d functions, convex if second derivative  $\geq 0$ .
- ▶ For 2-d functions, convex if ???

# Second Derivatives in 2-d

- ▶ In 2-d, there are 4 second derivatives of  $f(\vec{x})$ :
  - ▶  $\frac{\partial f^2}{\partial x_1^2}, \frac{\partial f^2}{\partial x_2^2}, \frac{\partial f^2}{\partial x_1 x_2}, \frac{\partial f^2}{\partial x_2 x_1}$

# Convexity in 2-d

- ▶ “Best” quadratic function approximating  $f$  at  $\vec{x}$ :

$$\begin{aligned} h_2(z_1, z_2) &= az_1^2 + bz_2^2 + cz_1z_2 + \dots \\ &= \frac{1}{2} \frac{\partial f^2}{\partial x_1^2}(\vec{x}) \cdot z_1 + \frac{1}{2} \frac{\partial f^2}{\partial x_2^2}(\vec{x}) \cdot z_2 + \frac{\partial f^2}{\partial x_1 x_2}(\vec{x}) \cdot z_1 z_2 + \dots \end{aligned}$$

- ▶  $a, b, c$  determine rough shape. Possibilities:
  - ▶ Upward-facing bowl.
  - ▶ Downward-facing bowl.
  - ▶ “Saddle”

# Convexity in 2-d

- Convex if at any  $\vec{x}$ , for any  $z_1, z_2$ :

$$\frac{1}{2} \frac{\partial f^2}{\partial x_1^2}(\vec{x}) \cdot z_1 + \frac{1}{2} \frac{\partial f^2}{\partial x_2^2}(\vec{x}) \cdot z_2 + \frac{\partial f^2}{\partial x_1 x_2}(\vec{x}) \cdot z_1 z_2 \geq 0$$

# The Hessian Matrix

- ▶ Create the **Hessian** matrix of second derivatives:

$$H(\vec{x}) = \begin{pmatrix} \frac{\partial f^2}{\partial x_1^2}(\vec{x}) & \frac{\partial f^2}{\partial x_1 x_2}(\vec{x}) \\ \frac{\partial f^2}{\partial x_2 x_1}(\vec{x}) & \frac{\partial f^2}{\partial x_2^2}(\vec{x}) \end{pmatrix}$$

# In General

- If  $f : \mathbb{R}^d \rightarrow \mathbb{R}$ , the **Hessian** at  $\vec{x}$  is:

$$H(\vec{x}) = \begin{pmatrix} \frac{\partial f^2}{\partial x_1^2}(\vec{x}) & \frac{\partial f^2}{\partial x_1 x_2}(\vec{x}) & \dots & \frac{\partial f^2}{\partial x_1 x_d}(\vec{x}) \\ \frac{\partial f^2}{\partial x_2 x_1}(\vec{x}) & \frac{\partial f^2}{\partial x_2^2}(\vec{x}) & \dots & \frac{\partial f^2}{\partial x_2 x_d}(\vec{x}) \\ \dots & \dots & \dots & \dots \\ \frac{\partial f^2}{\partial x_d x_1}(\vec{x}) & \frac{\partial f^2}{\partial x_d x_2}(\vec{x}) & \dots & \frac{\partial f^2}{\partial x_d^2}(\vec{x}) \end{pmatrix}$$

# Observations

- ▶  $H$  is square.
- ▶  $H$  is symmetric.

# Convexity in 2-d

- Convex if at any  $\vec{x}$ , for any  $z_1, z_2$ :

$$\frac{1}{2} \frac{\partial f^2}{\partial x_1^2}(\vec{x}) \cdot z_1 + \frac{1}{2} \frac{\partial f^2}{\partial x_2^2}(\vec{x}) \cdot z_2 + \frac{\partial f^2}{\partial x_1 x_2}(\vec{x}) \cdot z_1 z_2 \geq 0$$

- Equivalently, convex if for any  $\vec{x}$  and any  $\vec{z}$ :

$$\vec{z}^T H(\vec{x}) \vec{z} \geq 0$$

# Positive Semi-Definite

- ▶ A square,  $d \times d$  symmetric matrix  $X$  is **positive semi-definite** (PSD) if for any  $\vec{u}$ :

$$\vec{u}^T X \vec{u} \geq 0$$

## The Second Derivative Test

- ▶ A function  $f : \mathbb{R}^d \rightarrow \mathbb{R}$  is **convex** if for any  $\vec{x} \in \mathbb{R}^d$ , the Hessian matrix  $H(\vec{x})$  is positive semi-definite.

## **But wait...**

- ▶ How can we tell if a matrix is positive semi-definite?

# Example

$$M = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

# Example

$$M = \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}$$

# Example

Is  $f(x, y) = x^2 + 4xy + y^2$  convex?

## Sums of Convex Functions

- ▶ Suppose that  $f(\vec{x})$  and  $g(\vec{x})$  are convex. Then  $w_1 f(\vec{x}) + w_2 g(\vec{x})$  is convex, provided  $w_1, w_2 \geq 0$ .

# Affine Composition

- ▶ Suppose that  $f(x)$  is convex. Let  $A$  be a matrix, and  $\vec{x}$  and  $\vec{b}$  be vectors. Then

$$g(\vec{x}) = f(A\vec{x} + \vec{b})$$

is convex as a function of  $\vec{x}$ .

- ▶ Useful!

# DSC 140A

*Probabilistic Modeling & Machine Learning*

Lecture 5 | Part 4

## Convex Loss Functions

# Convexity and Gradient Descent

- ▶ Convex functions are (relatively) easy to optimize.
- ▶ **Theorem:** if  $f(x)$  is convex and “not too steep”<sup>3</sup> then (stochastic) (sub)gradient descent converges to a **global optimum** of  $f$  provided that the step size is small enough<sup>4</sup>.

---

<sup>3</sup>Technically,  $c$ -Lipschitz

<sup>4</sup>step size related to steepness, should decrease like  $1/\sqrt{t}$ , where  $t$  is step number

# Convex Loss

- ▶ **Recall:** sums of convex functions are convex.
- ▶ **Implication:** if loss function is convex as a function of  $\vec{w}$ , so is the risk.
- ▶ Convex losses are **nice**.

# Example

- ▶ Recall the square loss:

$$L(H(\vec{x}, \vec{w}), y) = (\vec{x} \cdot \vec{w} - y)^2$$

- ▶ Is this convex as a function of  $\vec{w}$ ?

# Mean Squared Error

- ▶ The square loss is a convex function of  $\vec{w}$ .
- ▶ We had an explicit solution for the best  $\vec{w}$ :

$$\vec{w} = (X^T X)^{-1} X^T \vec{y}$$

- ▶ But we could also have used gradient descent.

# Perceptron Loss

- ▶ The perceptron loss is:

$$L_{\text{perceptron}}(H(\vec{x}; \vec{w}), y) = \begin{cases} 0, & \text{sign}(\vec{w} \cdot \vec{x}) = \text{sign}(y) \\ |\vec{w} \cdot \vec{x}|, & \text{sign}(\vec{w} \cdot \vec{x}) \neq \text{sign}(y) \end{cases}$$

- ▶ Is it convex as a function of  $\vec{w}$ ?

# Summary

- ▶ We learned what it means for a function to be **convex**.
- ▶ Convex functions are (relatively) **easy** to optimize with gradient descent.
- ▶ We like **convex loss functions**, like the square loss.

# DSC 140A

*Probabilistic Modeling & Machine Learning*

Lecture 6 | Part 1

## Maximum Margin Classifiers

# Recall: Perceptrons

- ▶ Linear classifier fit using loss function:

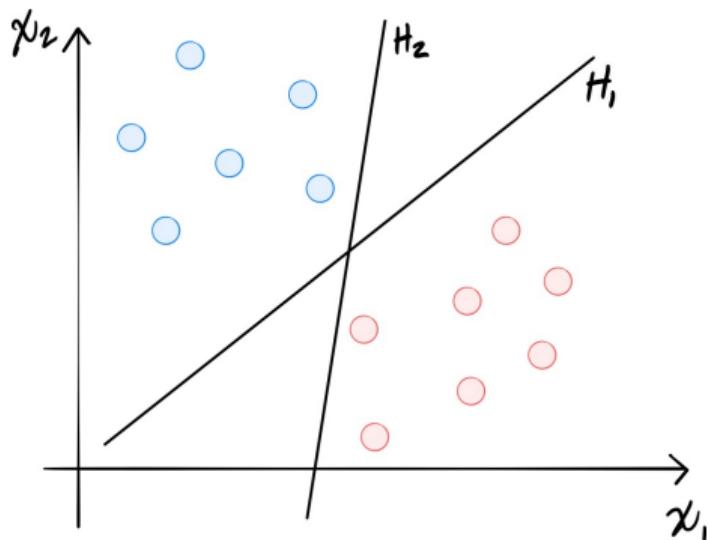
$$L_{\text{tron}}(H(\vec{x}), y) = \begin{cases} 0, & \text{sign}(H(\vec{x})) = \text{sign}(y) \\ |H(\vec{x})|, & \text{sign}(H(\vec{x})) \neq \text{sign}(y) \end{cases}$$

# A Problem with the Perceptron

- ▶ **Recall:** the perceptron loss assigns no penalty to points that are correctly classified.
- ▶ No matter how close the point is to the boundary.

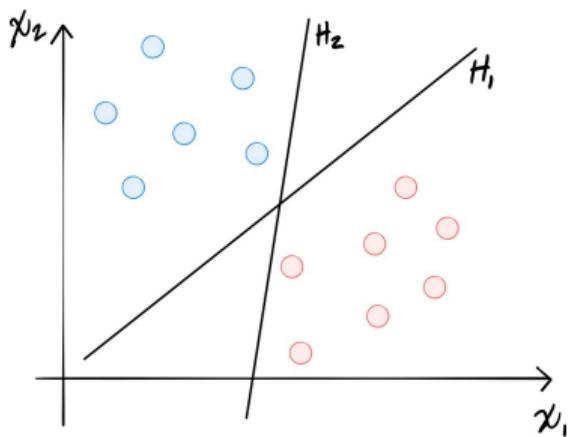
## Exercise

What is the empirical risk with respect to the perceptron loss of  $H_1$ ? What about  $H_2$ ?



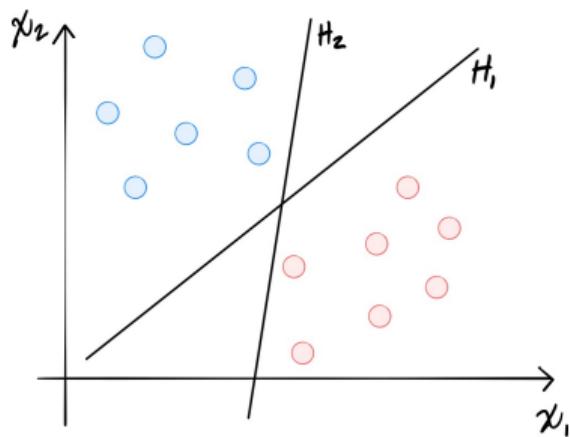
# Linear Separability

- ▶ Data are **linearly separable** if there exists a linear classifier which perfectly classifies the data.



# Margin

- ▶ The **margin** is the smallest distance between the decision boundary and a training point.



# Maximum Margin Classifier

- ▶ If training data are linearly separable, there are many classifiers with zero error.
- ▶ We prefer classifiers with larger margins.
  - ▶ Better generalization performance.
- ▶ Can we find the **maximum margin classifier**?
  - ▶ I.e., the classifier with the largest possible margin?

# Observation

- ▶ A point is classified correctly when:

$$\begin{cases} \vec{w} \cdot \text{Aug}(\vec{x}^{(i)}) > 0, & \text{if } y_i = 1 \\ \vec{w} \cdot \text{Aug}(\vec{x}^{(i)}) < 0, & \text{if } y_i = -1 \end{cases}$$

- ▶ Equivalently, classification is correct if:

$$y_i \vec{w} \cdot \text{Aug}(\vec{x}^{(i)}) > 0$$

## Attempt #1

- ▶ **Goal:** Assume linear separability. Find a  $\vec{w}$  so that  $y_i \vec{w} \cdot \text{Aug}(\vec{x}^{(i)}) > 0$  for all data points.
- ▶ That is, all points are correctly classified.
- ▶ Too easy!
  - ▶ Perceptron already does this.
  - ▶ Does not force margin to be maximized.

# Enforce a Margin

- ▶ **Recall:**  $|H(\vec{x})| = |\vec{w} \cdot \text{Aug}(\vec{x})|$  is **proportional** to distance from decision boundary.
  - ▶ Doesn't measure actual distance!
  - ▶ Scaled by a factor depending on  $1/\|\vec{w}\|$ .
- ▶ **Informal:**  $|H(\vec{x})|$  measures distance in “prediction units”
  - ▶ E.g., if  $H(\vec{x}) = -2$ ,  $\vec{x}$  is 2 “prediction units” away from boundary

# Enforce a Margin

- ▶ We can enforce a margin in “prediction units”.
- ▶ E.g., to require a margin of one prediction unit, we must have

$$y_i H(\vec{x}^{(i)}) = y_i \vec{w} \cdot \text{Aug } \vec{x}^{(i)} \geq 1$$

for each data point

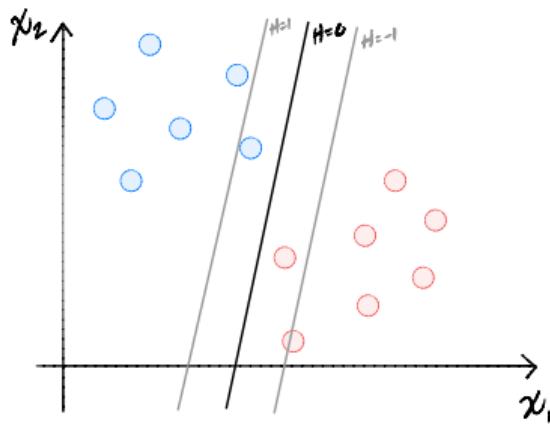
## Attempt #2

- ▶ **Goal:** Assume linear separability. Find a  $\vec{w}$  so that  $y_i \vec{w} \cdot \text{Aug}(\vec{x}^{(i)}) \geq 1$  for all data points.
- ▶ Still “too easy”.
  - ▶ Problem: prediction units aren’t actual distance.
  - ▶ We can artificially increase distance in “prediction units” by increasing  $\|\vec{w}\|$ .

## Exercise

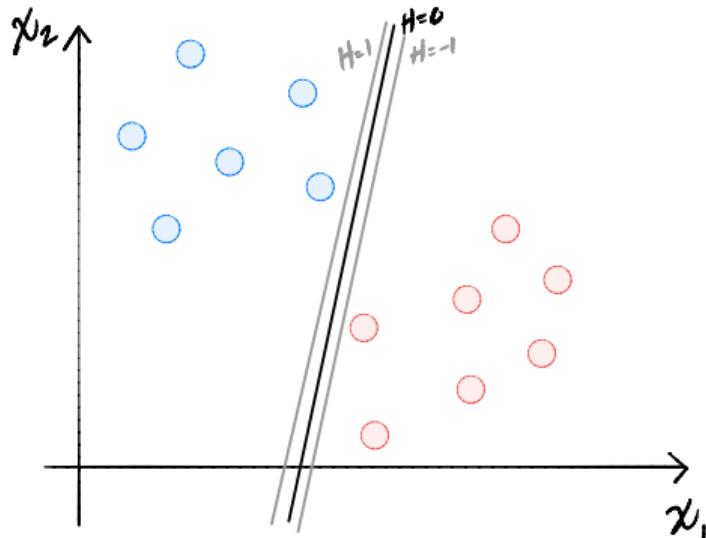
Suppose  $H$  is a linear predictor with parameter vector  $\vec{w}$ . Shown are the lines one “prediction unit” away from the decision boundary.

How will the decision boundary and these lines change if  $\vec{w}$  is doubled?



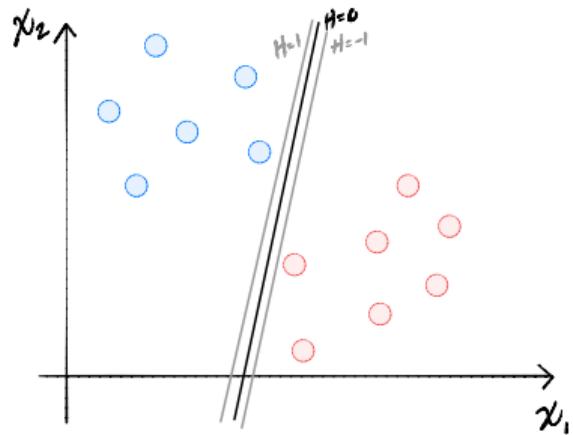
# Solution

- ▶ The decision boundary remains unchanged.
- ▶ The lines one “prediction unit” away move closer.



# Observe

- ▶  $H$  satisfies  $y_i \vec{w} \cdot \text{Aug}(\vec{x}^{(i)}) \geq 1$



# Observe

- ▶ Any vector  $\vec{w}$  satisfying  $y_i \vec{w} \cdot \text{Aug}(\vec{x}^{(i)}) > 0$  can be made to satisfy  $y_i \vec{w} \cdot \text{Aug}(\vec{x}^{(i)}) \geq 1$  by increasing  $\|\vec{w}\|$  appropriately.
- ▶ But this is **cheating!**
- ▶ Fix: search for a low-norm  $\vec{w}$  satisfying  $y_i \vec{w} \cdot \text{Aug}(\vec{x}^{(i)}) \geq 1$

## Attempt #3

- ▶ **Goal:** out of all  $\vec{w}$  satisfying  $y_i \vec{w} \cdot \text{Aug}(\vec{x}^{(i)}) \geq 1$  for all data points, find that with minimum  $\|\vec{w}\|$
- ▶ That is, find:

$$\vec{w}^* = \arg \min_{\vec{w}} \|\vec{w}\|$$

subject to:  $\forall i, y_i \vec{w} \cdot \text{Aug}(\vec{x}) \geq 1$

# Hard-SVM

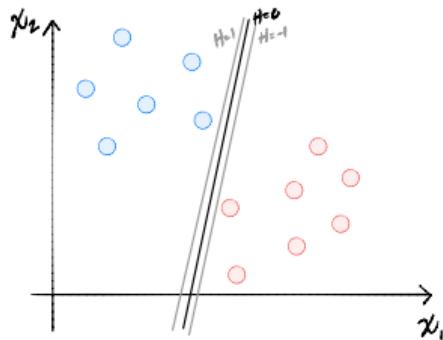
- ▶ This optimization problem is called the **Hard Support Vector Machine** classifier problem.
- ▶ Only makes sense if data are linearly separable.
- ▶ In a moment, we'll see the Soft-SVM.

# How?

- ▶ Turn it into a **convex quadratic** optimization problem:
  - ▶ Minimize  $\|\vec{w}\|^2$  subject to  $y_i \vec{w} \cdot \text{Aug}(\vec{x}^{(i)}) \geq 1$  for all  $i$ .
- ▶ Can be solved efficiently with **quadratic programming**.
  - ▶ But there is no exact general formula for the solution

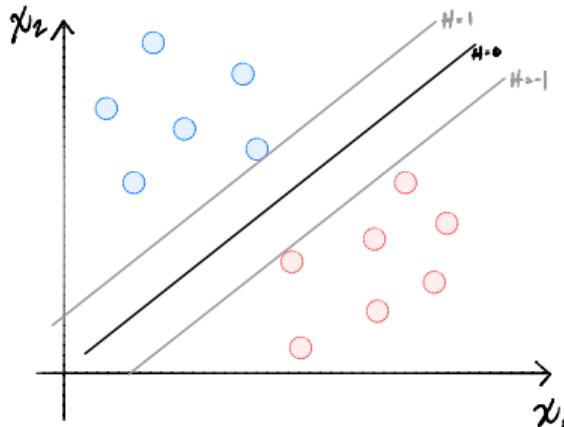
## Exercise

Can the below predictor be a solution of the Hard-SVM?



# SVMs are Maximum Margin Classifiers

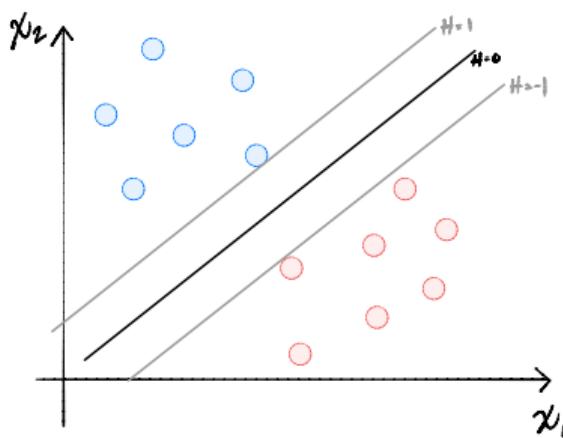
- ▶ Intuition says solutions of Hard-SVM will have large margins.
- ▶ Fact: they maximize the margin.



# Support Vectors

- ▶ A **support vector** is a training point  $\vec{x}^{(i)}$  such that

$$y_i \vec{w} \cdot \text{Aug}(\vec{x}^{(i)}) = 1$$



# Support Vectors

- ▶ **Fact:** the solution to Hard-SVM is always a linear combination of the support vectors.
- ▶ That is, let  $S$  be the set of support vectors. Then

$$\vec{w}^* = \sum_{i \in S} y_i \alpha_i \text{Aug}(\vec{x}^{(i)})$$

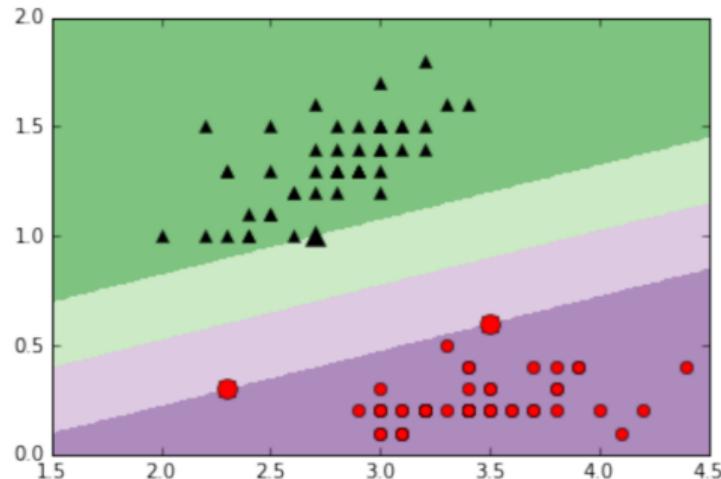
# Example: Irises



- ▶ 3 classes: *iris setosa*, *iris versicolor*, *iris virginica*
- ▶ 4 measurements: petal width/height, sepal width/height

# Example: Irises

- ▶ Using only sepal width/petal width
- ▶ Two classes: versicolor (black), setosa (red)



# DSC 140A

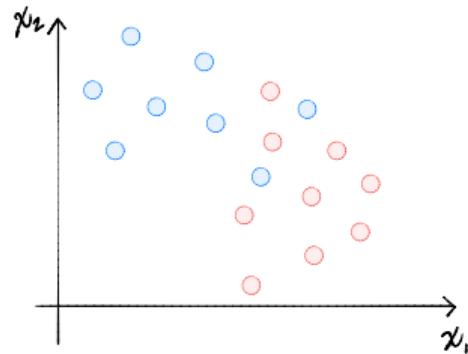
*Probabilistic Modeling & Machine Learning*

Lecture 6 | Part 2

**Soft-Margin SVMs**

# Non-Separability

- ▶ So far we've assumed data is linearly separable.
- ▶ What if it isn't?

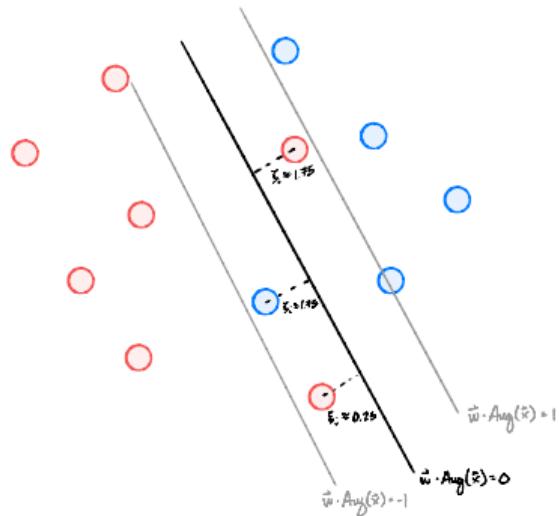


# The Problem

- ▶ **Old Goal:** Minimize  $\|\vec{w}\|^2$  subject to  $y_i \vec{w} \cdot \text{Aug}(\vec{x}^{(i)}) \geq 1$  for all  $i$ .
- ▶ This **no longer makes sense**.

# Cut Some Slack

- ▶ **Idea:** allow some classifications to be  $\xi_i$  wrong, but not too wrong.



# Cut Some Slack

- ▶ **New problem.** Fix some number  $C \geq 0$ .

$$\min_{\vec{w} \in \mathbb{R}^{d+1}, \vec{\xi} \in \mathbb{R}^n} \|\vec{w}\|^2 + C \sum_{i=1}^n \xi_i$$

subject to  $y_i \vec{w} \cdot \text{Aug}(\vec{x}^{(i)}) \geq 1 - \xi_i$  for all  $i$ ,  $\vec{\xi} \geq 0$ .

# The Slack Parameter, C

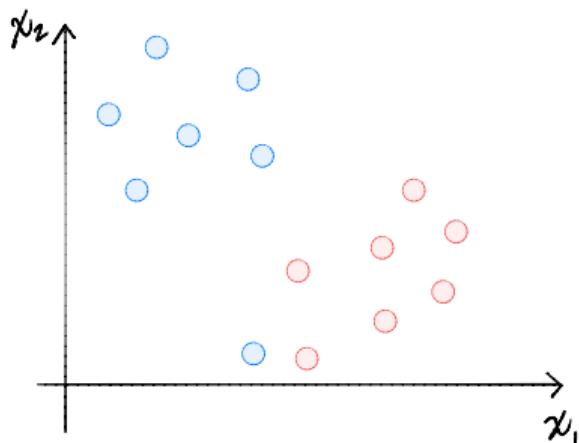
- ▶  $C$  controls how much slack is given.

$$\min_{\vec{w} \in \mathbb{R}^{d+1}, \vec{\xi} \in \mathbb{R}^n} \|\vec{w}\|^2 + C \sum_{i=1}^n \xi_i$$

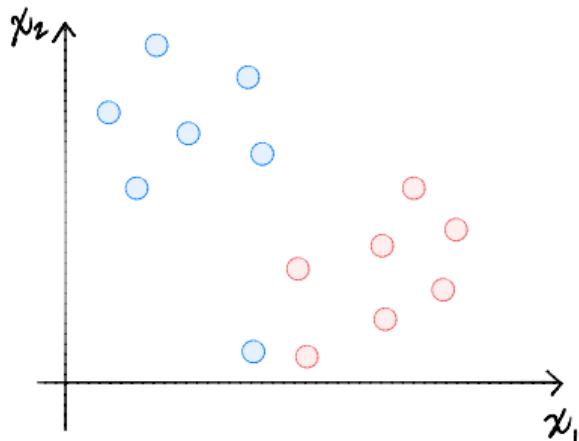
subject to  $y_i \vec{w} \cdot \text{Aug}(\vec{x}^{(i)}) \geq 1 - \xi_i$  for all  $i$ ,  $\vec{\xi} \geq 0$ .

- ▶ Large  $C$ : don't give much slack. Avoid misclassifications.
- ▶ Small  $C$ : allow more slack at the cost of misclassifications.

## Example: Small C



## Example: Large C



# Soft and Hard Margins

- ▶ Max-margin SVM from before has **hard margin**.
- ▶ Now: the **soft margin** SVM.
- ▶ As  $C \rightarrow \infty$ , the margin hardens.

# DSC 140A

*Probabilistic Modeling & Machine Learning*

Lecture 6 | Part 3

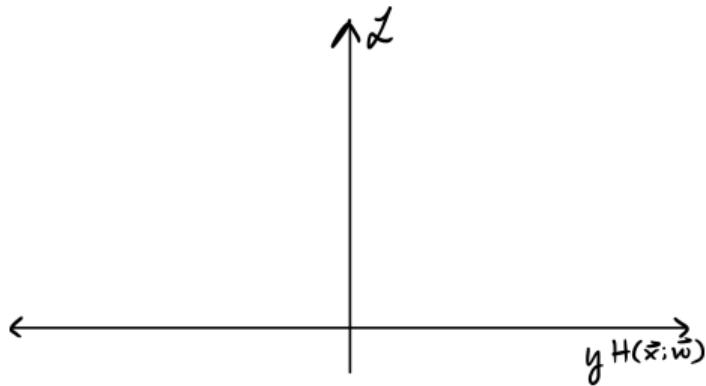
## Hinge Loss

# Loss Functions?

- ▶ So far, we've learned predictors by minimizing **expected loss** via ERM.
- ▶ But this isn't what we did with Hard-SVM and Soft-SVM.
- ▶ It turns out, we can frame Soft-SVM as an ERM problem.

# Recall: Perceptron Loss

$$L_{\text{tron}}(H(\vec{x}), y) = \begin{cases} 0, & \text{sign}(H(\vec{x})) = \text{sign}(y) \\ |H(\vec{x})|, & \text{sign}(H(\vec{x})) \neq \text{sign}(y) \end{cases}$$

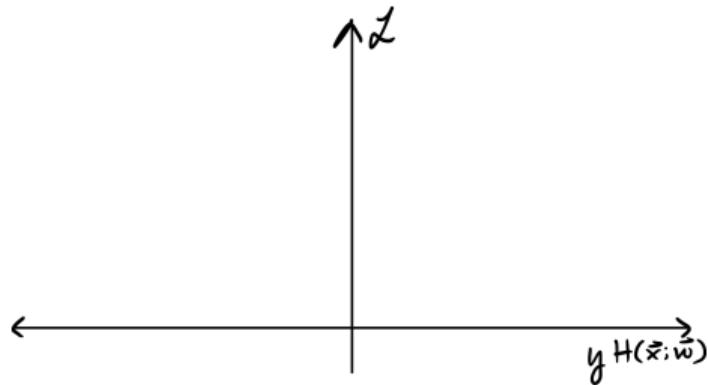


# Perceptron Loss

- ▶ Perceptron loss did not penalize correct classifications.
- ▶ Even if they were very close to boundary.
- ▶ **Idea:** penalize predictions that are close to the boundary, too.

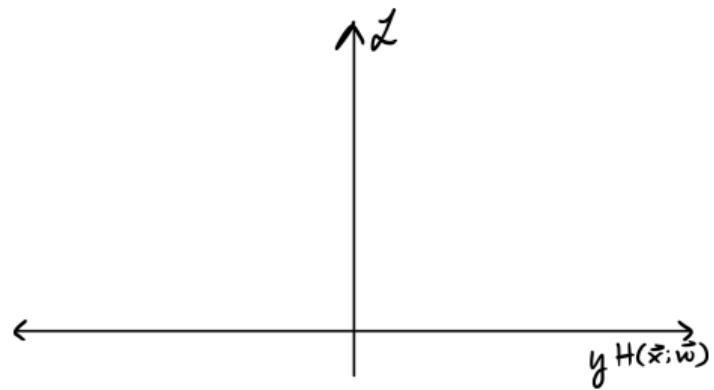
# The Hinge Loss

$$L_{\text{hinge}}(H(\vec{x}), y) = \begin{cases} 0, & yH(\vec{x}) \geq 1, \\ 1 - yH(\vec{x}), & yH(\vec{x}) < 1 \end{cases}$$



# The Hinge Loss

$$L_{\text{hinge}}(H(\vec{x}), y) = \max\{0, 1 - yH(\vec{x})\}$$



# Equivalence

- ▶ Recall the Soft-SVM problem:

$$\min_{\vec{w} \in \mathbb{R}^{d+1}, \vec{\xi} \in \mathbb{R}^n} \|\vec{w}\|^2 + C \sum_{i=1}^n \xi_i$$

subject to  $y_i \vec{w} \cdot \text{Aug}(\vec{x}^{(i)}) \geq 1 - \xi_i$  for all  $i$ ,  $\vec{\xi} \geq 0$ .

- ▶ **Note:** if  $\vec{x}^{(i)}$  is misclassified, then

$$\xi_i = 1 - y_i \vec{w} \cdot \text{Aug}(\vec{x}^{(i)})$$

# Equivalence

- ▶ The Soft-SVM problem is equivalent to finding  $\vec{w}$  that minimizes:

$$R_{\text{svm}}(\vec{w}) = \|\vec{w}\|^2 + C \sum_{i=1}^n \max\{0, 1 - y_i \vec{w} \cdot \vec{x}^{(i)}\}$$

- ▶  $R_{\text{svm}}$  is the **regularized** risk.
- ▶  $C$  is a parameter affecting “softness” of boundary; chosen by you.

## **Another Way to Optimize**

- ▶ In practice, SGD is often used to train soft SVMs.

# DSC 140A

*Probabilistic Modeling & Machine Learning*

Lecture 6 | Part 4

**Demo: Sentiment Analysis**

# Why use linear predictors?

- ▶ Linear classifiers look to be very simple.
- ▶ That can be both **good** and **bad**.
  - ▶ **Good**: the math is tractable, less likely to overfit
  - ▶ **Bad**: may be too simple, underfit
- ▶ They can work surprisingly well.

# Sentiment Analysis

- ▶ **Given:** a piece of text.
- ▶ **Determine:** if it is **positive** or **negative** in tone
- ▶ Example: “Needless to say, I wasted my money.”

# The Data

- ▶ Sentences from reviews on Amazon, Yelp, IMDB.
- ▶ Each labeled (by a human) **positive** or **negative**.
- ▶ Examples:
  - ▶ “**Needless to say, I wasted my money.**”
  - ▶ “**I have to jiggle the plug to get it to line up right.**”
  - ▶ “**Will order from them again!**”
  - ▶ “**He was very impressed when going from the original battery to the extended battery.**”

# The Plan

- ▶ We'll train a soft-margin SVM.
- ▶ **Problem:** SVMs take **fixed-length vectors** as inputs, not sentences.

# Bags of Words

To turn a document into a fixed-length vector:

- ▶ First, choose a **dictionary** of words:
  - ▶ E.g.: ["wasted", "impressed", "great", "bad", "again"]
- ▶ Count number of occurrences of each dictionary word in document.
  - ▶ "It was bad. So bad that I was impressed at how bad it was." →  $(0, 1, 0, 3, 0)^T$
- ▶ This is called a **bag of words** representation.

# Choosing the Dictionary

- ▶ Many ways of choosing the dictionary.
- ▶ Easiest: take all of the words in the training set.
  - ▶ Perhaps throw out **stop words** like “the”, “a”, etc.
- ▶ Resulting dimensionality of feature vectors: large.

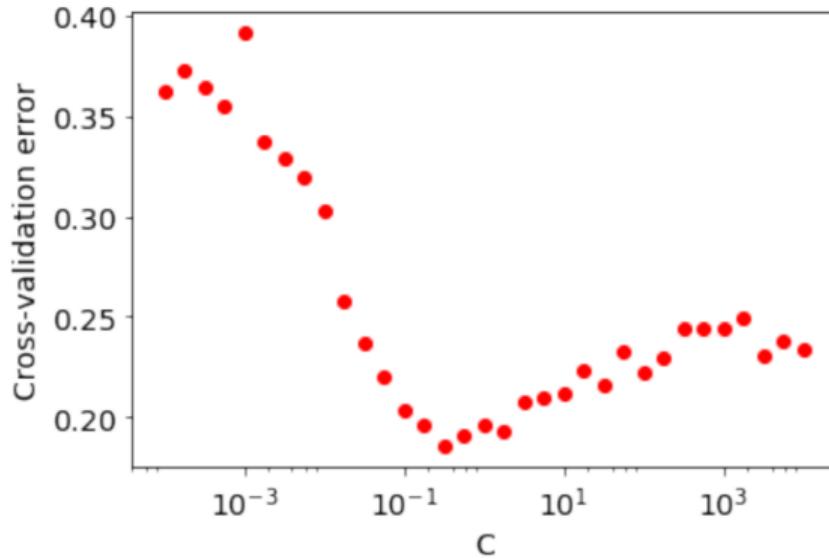
# Experiment

- ▶ Bag of words features with 4500 word dictionary.
- ▶ 2500 training sentences, 500 test sentences.
- ▶ Train a soft margin SVM.

# Choosing C

- ▶ We have to choose the slack parameter,  $C$ .
- ▶ Use **cross validation!**

# Cross Validation



# Results

- ▶ With  $C = 0.32$ , test error  $\approx 15.6\%$ .

$C$	training error (%)	test error (%)	# support vectors
0.01	23.72	28.4	2294
0.1	7.88	18.4	1766
1	1.12	16.8	1306
10	0.16	19.4	1105
100	0.08	19.4	1035
1000	0.08	19.4	950

# DSC 140A

*Probabilistic Modeling & Machine Learning*

Lecture 7 | Part 1

## Ridge Regression

# **News**

- ▶ Discussion worksheet solutions.

# Recall: Regression with Basis Functions

- ▶ We can fit any function of the form:

$$H(\vec{x}; \vec{w}) = w_0 + w_1\phi_1(\vec{x}) + w_2\phi_2(\vec{x}) + \dots + w_k\phi_k(\vec{x})$$

- ▶  $\phi_i(\vec{x}) : \mathbb{R}^d \rightarrow \mathbb{R}$  is called a **basis function**.

# Procedure

1. Define  $\vec{\phi}(\vec{x}) = (\phi_1(\vec{x}), \phi_2(\vec{x}), \dots, \phi_k(\vec{x}))^T$

2. Form  $n \times k$  **design matrix**:

$$\Phi = \begin{pmatrix} \text{Aug}(\phi(\vec{x}^{(1)})) & \longrightarrow \\ \text{Aug}(\phi(\vec{x}^{(2)})) & \longrightarrow \\ \vdots & \vdots \\ \text{Aug}(\phi(\vec{x}^{(n)})) & \longrightarrow \end{pmatrix} = \begin{pmatrix} \phi_1(\vec{x}^{(1)}) & \phi_2(\vec{x}^{(1)}) & \dots & \phi_k(\vec{x}^{(1)}) \\ \phi_1(\vec{x}^{(2)}) & \phi_2(\vec{x}^{(2)}) & \dots & \phi_k(\vec{x}^{(2)}) \\ \vdots & \vdots & \vdots & \vdots \\ \phi_1(\vec{x}^{(n)}) & \phi_2(\vec{x}^{(n)}) & \dots & \phi_k(\vec{x}^{(n)}) \end{pmatrix}$$

3. Solve the **normal equations**:

$$\vec{w}^* = (\Phi^T \Phi)^{-1} \Phi^T \vec{y}$$

# Example: Polynomial Curve Fitting

- ▶ Fit a function of the form:

$$H(x; \vec{w}) = w_0 + w_1x + w_2x^2 + w_3x^3$$

- ▶ Use basis functions:

$$\phi_0(x) = 1 \quad \phi_1(x) = x \quad \phi_2(x) = x^2 \quad \phi_3(x) = x^3$$

# Example: Polynomial Curve Fitting

- ▶ Design matrix becomes:

$$\Phi = \begin{pmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ \dots & \dots & \ddots & \vdots \\ 1 & x_n & x_n^2 & x_n^3 \end{pmatrix}$$

# Gaussian Basis Functions

- ▶ **Gaussians** make for useful basis functions.

$$\phi_i(x) = \exp\left(-\frac{(x - \mu_i)^2}{\sigma_i^2}\right)$$

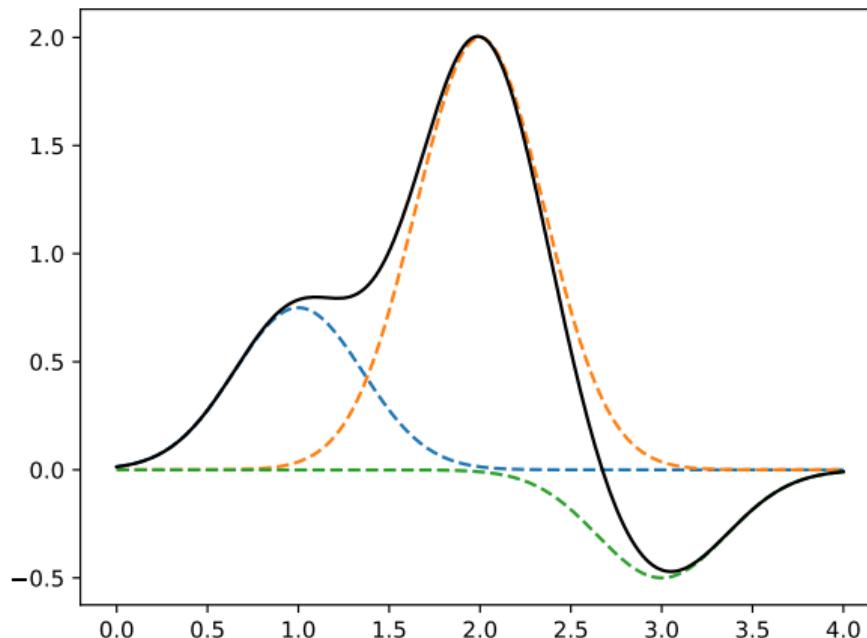
- ▶ Must specify<sup>1</sup> **center**  $\mu_i$  and **width**  $\sigma_i$  for each Gaussian basis function.

---

<sup>1</sup>You pick these; they are not learned!

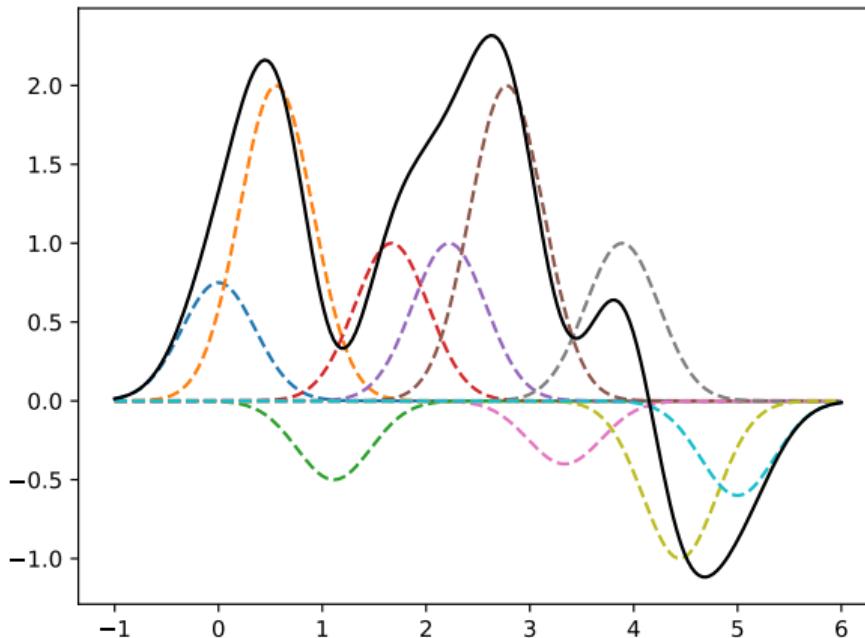
# Example: $k = 3$

- ▶ A function of the form:  $H(x) = w_1\phi_1(x) + w_2\phi_2(x) + w_3\phi_3(x)$ , using 3 Gaussian basis functions.



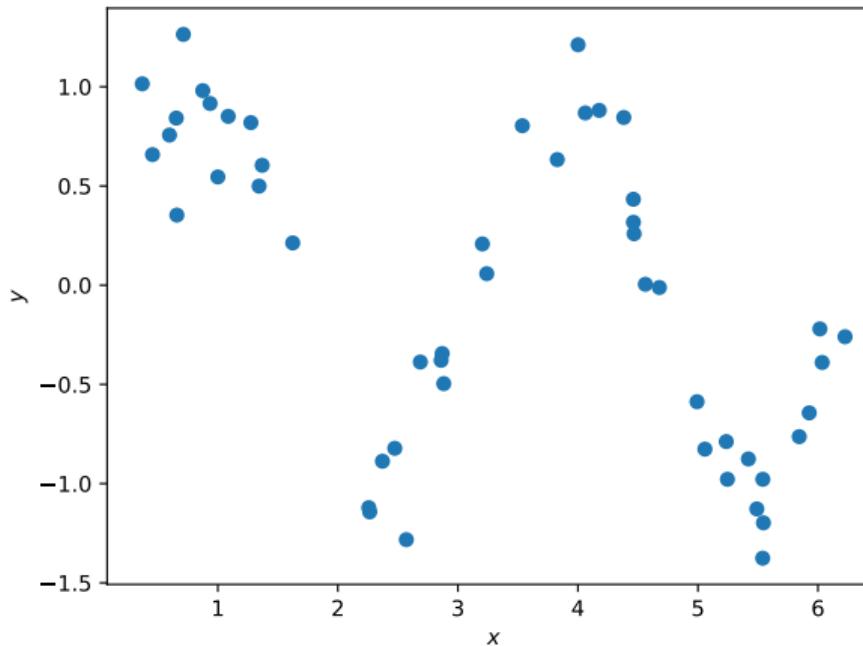
# Example: $k = 10$

- ▶ The more basis functions, the more complex  $H$  can be.



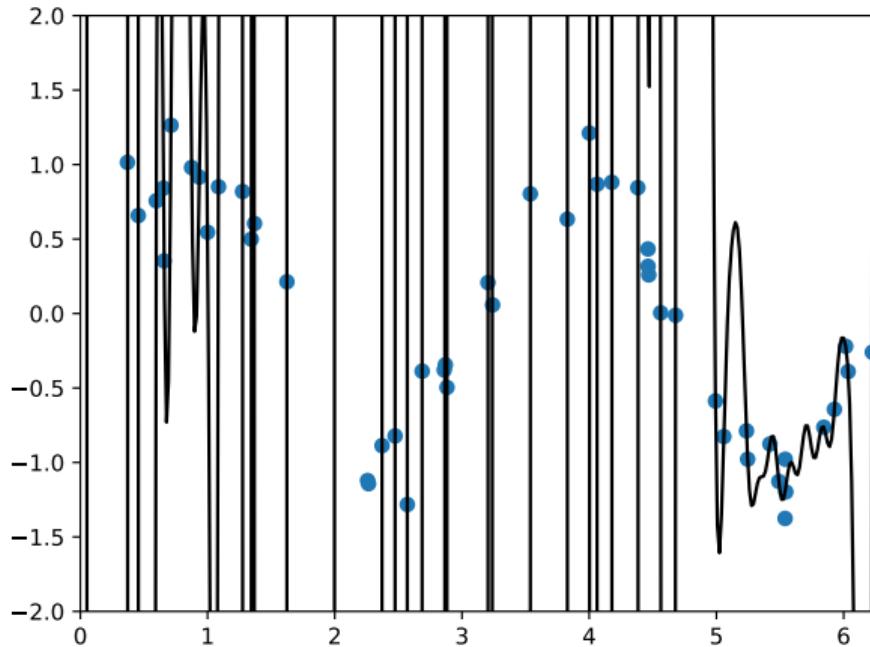
# Demo: Sinusoidal Data

- ▶ Fit curve to 50 noisy data points.
- ▶ Use  $k = 50$  Gaussian basis functions.



# Result

► Overfitting!



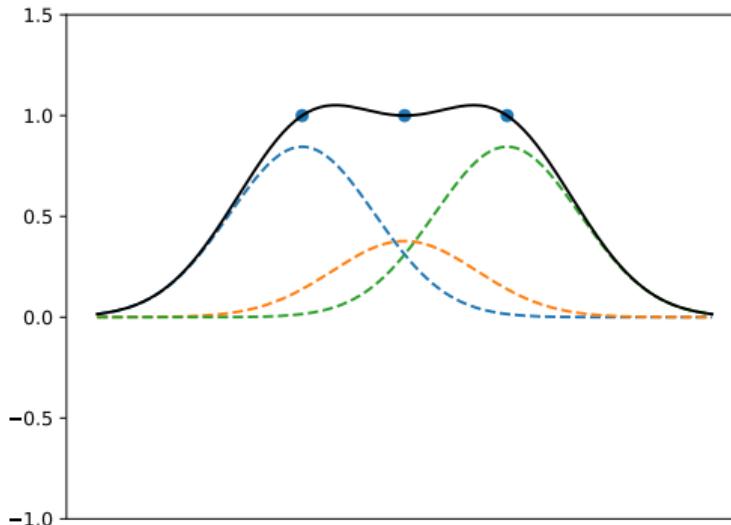
# Controlling Model Complexity

- ▶ Model is too complex.
- ▶ Can decrease complexity by reducing number of basis functions.
- ▶ Another way: **regularization**.

# Complexity and $\vec{w}$

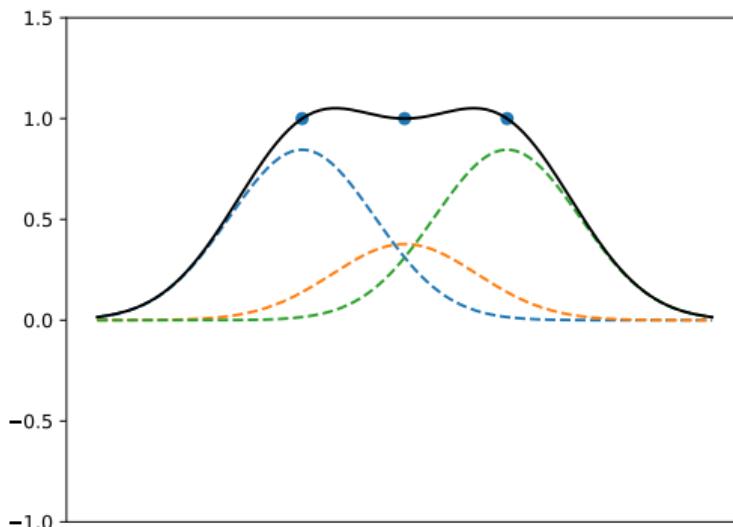
- ▶ Consider fitting 3 points with  $k = 3$ :

$$w_1\phi_1(\vec{x}) + w_2\phi_2(\vec{x}) + w_3\phi_3(\vec{x})$$

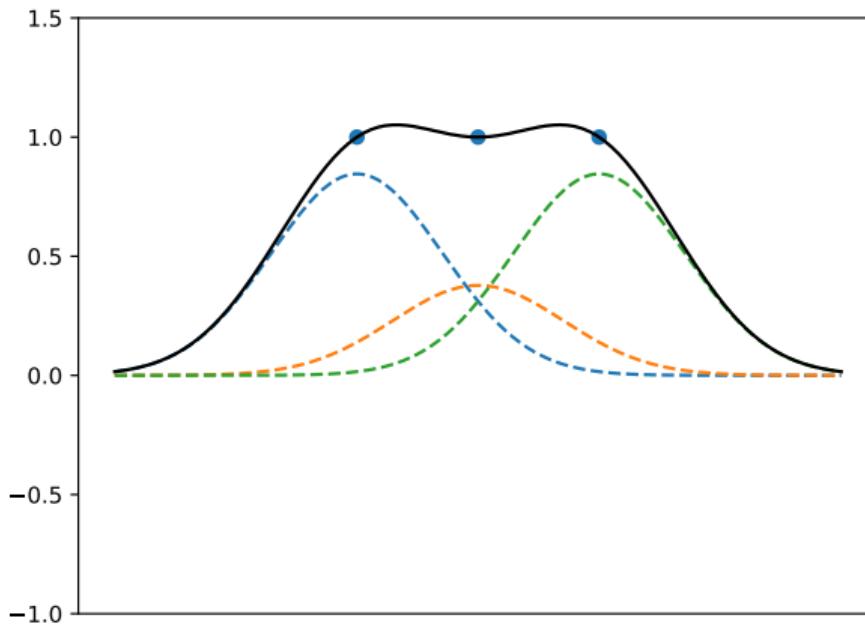


## Exercise

What will happen to  $w_1, w_2, w_3$  as the middle point is shifted down towards zero?

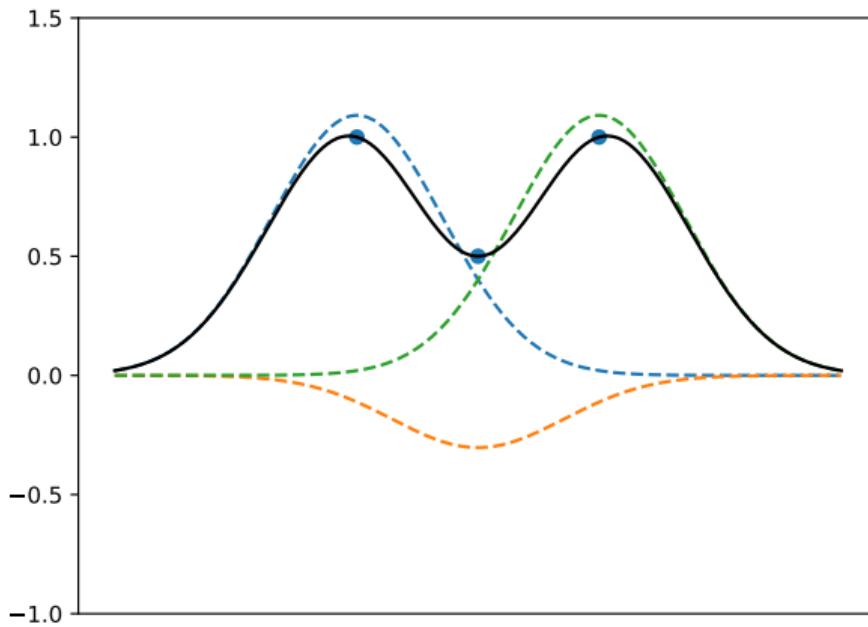


# Solution



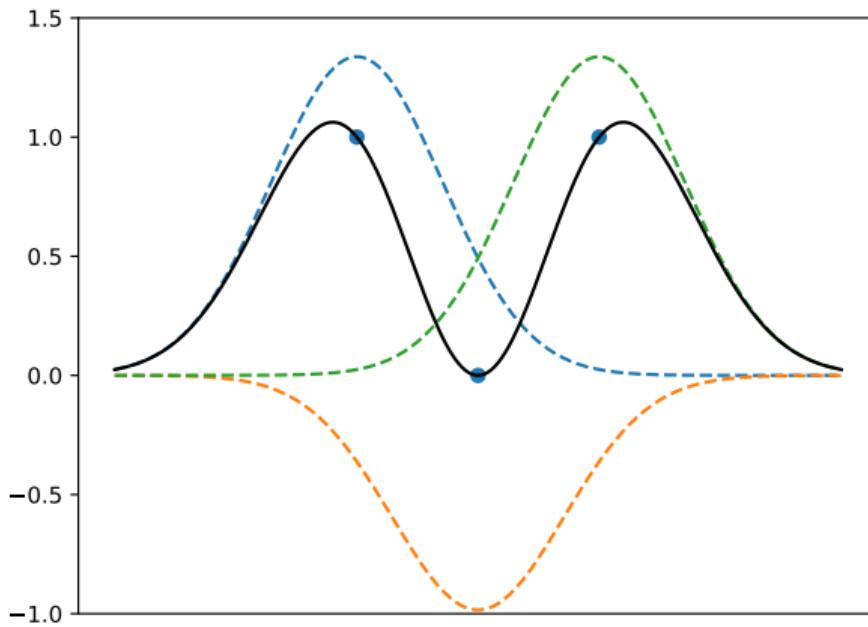
$$\begin{aligned} w &= [0.85 \ 0.38 \ 0.85] \\ \|\vec{w}\| &= 1.25 \end{aligned}$$

# Solution



$$\begin{aligned} \mathbf{w} &= [1.09 \ -0.3 \ 1.09] \\ \|\vec{w}\| &= 1.57 \end{aligned}$$

# Solution



$$\mathbf{w} = [1.34 \ -0.98 \ 1.34]$$
$$\|\vec{w}\| = 2.13$$

# Observations

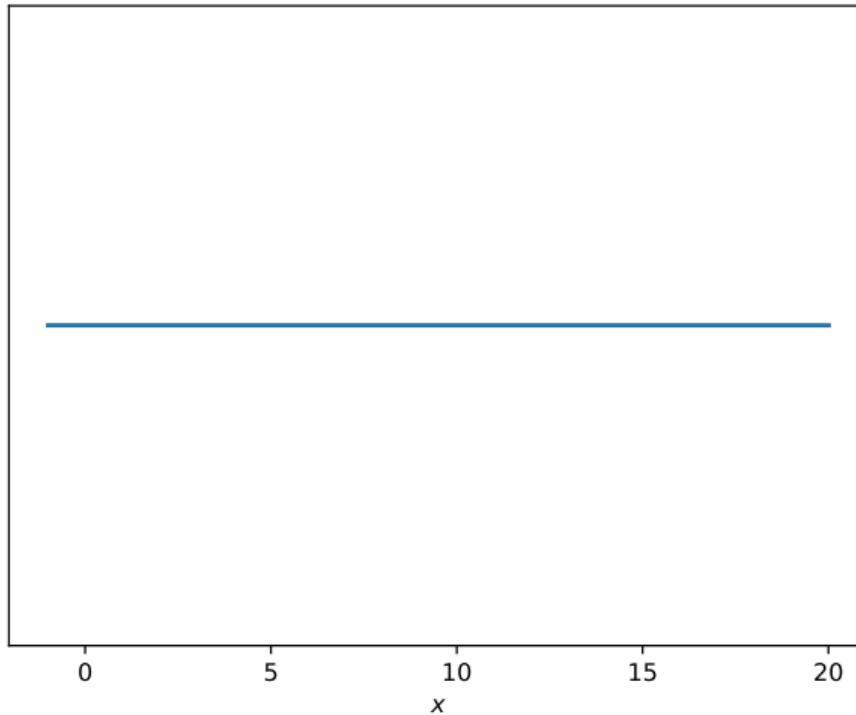
- ▶ As the middle point moves down,  $H$  becomes more complex.
- ▶ The weights grow in magnitude.
- ▶  $\|\vec{w}\|$  grows.
- ▶ **Idea:**  $\|\vec{w}\|$  measures **complexity** of  $H$ .

# Experiment

- ▶ Consider model with  $k = 20$  Gaussian basis functions.
- ▶ Generate 100 random parameter vectors  $\vec{w}$ .
- ▶ Plot overlapping; observe complexity.

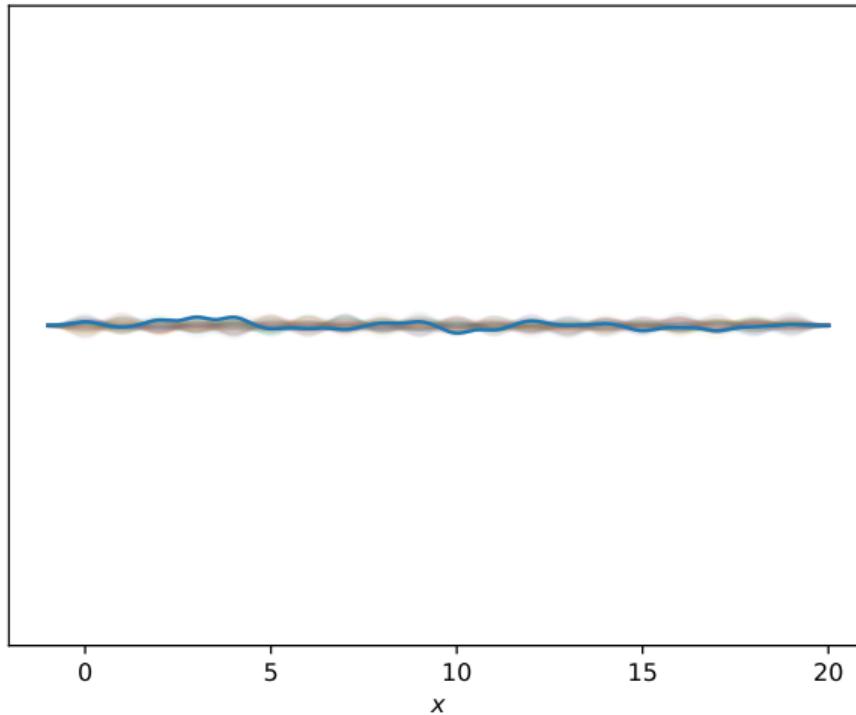
# Experiment

$$||\vec{w}|| = 0.0$$



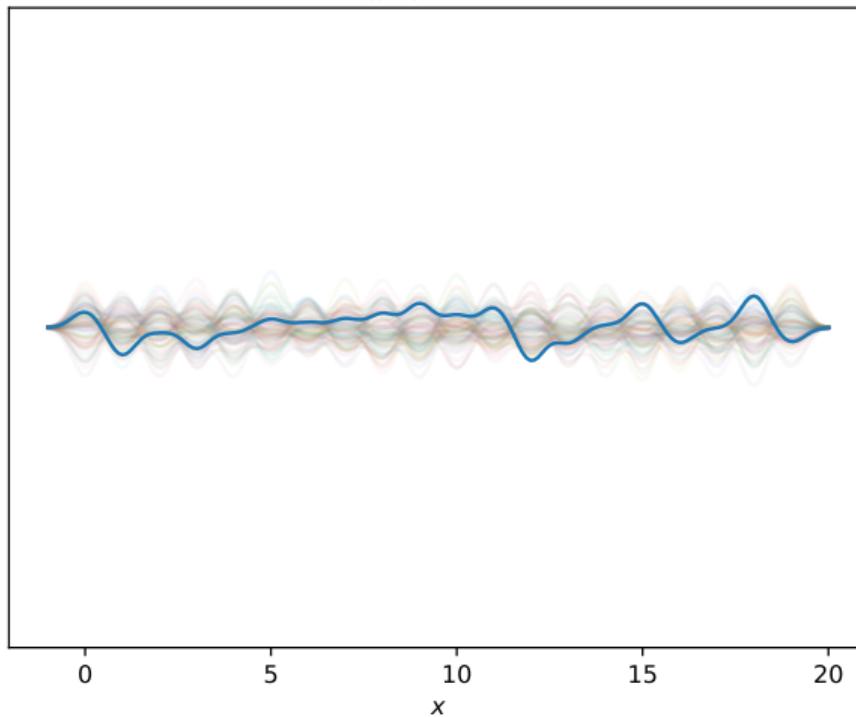
# Experiment

$$||\vec{w}|| = 0.5$$



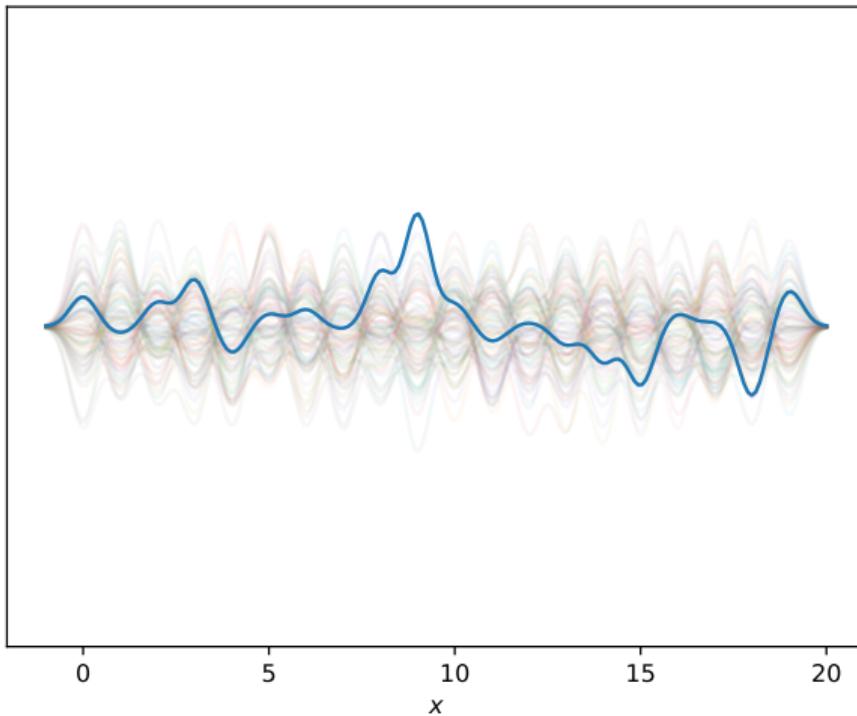
# Experiment

$$||\vec{w}|| = 1.0$$



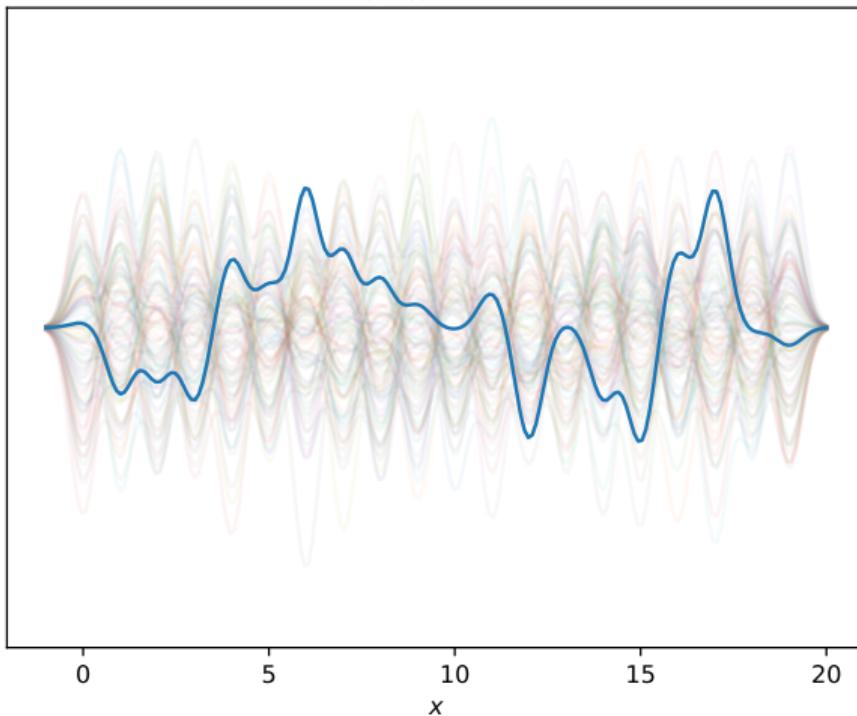
# Experiment

$$||\vec{w}|| = 1.5$$



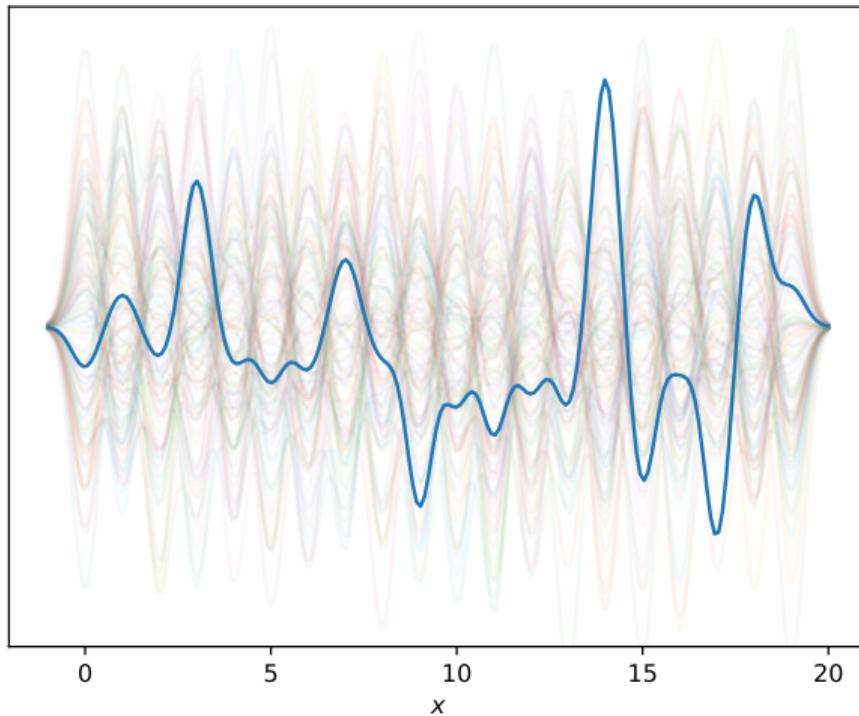
# Experiment

$$||\vec{w}|| = 2.0$$



# Experiment

$$||\vec{w}|| = 2.5$$



# Conclusion

- ▶  $\|\vec{w}\|$  is a proxy for model complexity.
  - ▶ The larger  $\|w\|$ , the more complex the model may be.
- ▶ **Idea:** find a model with
  - ▶ small mean squared error on the training data;
  - ▶ but also small  $\|\vec{w}\|$

# Recall: Least Squares Regression

- ▶ In **least squares regression**, we minimize the empirical **risk**:

$$\begin{aligned} R(\vec{w}) &= \frac{1}{n} \sum_{i=1}^n (H(\vec{x}^{(i)}) - y_i)^2 \\ &= \frac{1}{n} \sum_{i=1}^n (\vec{w} \cdot \phi(\vec{x}^{(i)}) - y_i)^2 \end{aligned}$$

# Regularized Least Squares

- ▶ **Idea:** penalize large  $\|\vec{w}\|$  to control overfitting.
- ▶ **Goal:** Minimize the **regularized risk**:

$$\tilde{R}(\vec{w}) = \frac{1}{n} \sum_{i=1}^n (\vec{w} \cdot \phi(\vec{x}^{(i)}) - y_i)^2 + \lambda \|\vec{w}\|^2$$

- ▶  $\lambda \|\vec{w}\|^2$  is a **regularization term**.
  - ▶ “Tikhonov regularization”
  - ▶  $\lambda$  controls “strength” of regularization.

# Ridge Regression

- Least squares with  $\|w\|^2$  regularization is also known as **ridge regression**.

## Why $\|\vec{w}\|^2$ ?

- ▶ We consider  $\|\vec{w}\|^2$  instead of  $\|\vec{w}\|$  because it will make the calculations cleaner.

# Ridge Regression Solution

- **Goal:** Find  $\vec{w}^*$  minimizing the **regularized risk**:

$$\tilde{R}(\vec{w}) = \frac{1}{n} \sum_{i=1}^n (\vec{w} \cdot \phi(\vec{x}^{(i)}) - y_i)^2 + \lambda \|\vec{w}\|^2$$

- Recall:

$$\frac{1}{n} \sum_{i=1}^n (\vec{w} \cdot \phi(\vec{x}^{(i)}) - y_i)^2 = \frac{1}{n} \|\Phi \vec{w} - \vec{y}\|^2$$

- So:

$$\tilde{R}(\vec{w}) = \frac{1}{n} \|\Phi \vec{w} - \vec{y}\|^2 + \lambda \|\vec{w}\|^2$$

# Ridge Regression Solution

- ▶ **Strategy:** calculate  $d\tilde{R}/d\vec{w}$ , set to  $\vec{0}$ , solve.
- ▶ **Solution:**  $\vec{w}^* = (\Phi^T \Phi + n\lambda I)^{-1} \Phi^T \vec{y}$
- ▶ Compare this to solution of unregularized problem:  $\vec{w}^* = (\Phi^T \Phi)^{-1} \Phi^T \vec{y}$

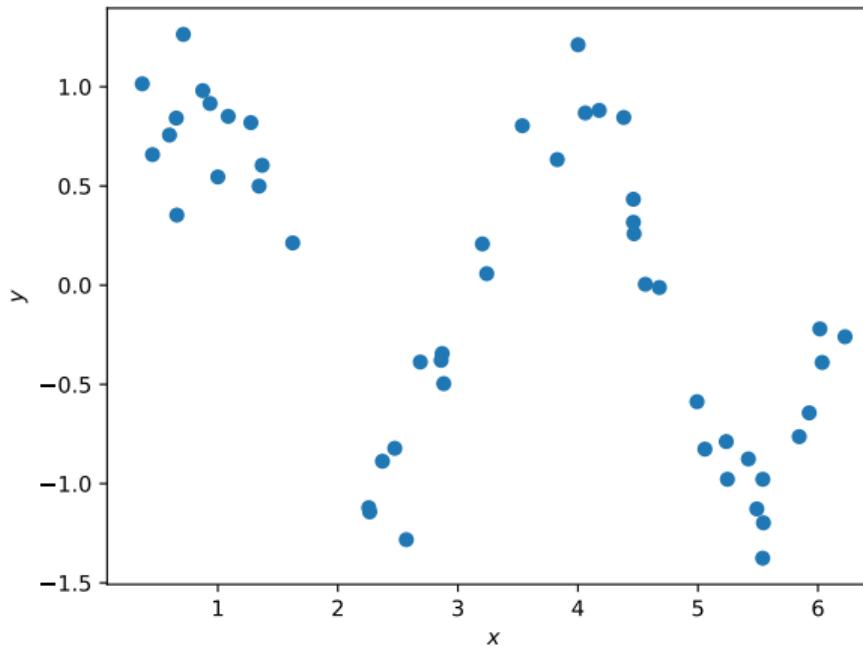
# Interpretation

$$\vec{w}^* = (\Phi^T \Phi + n\lambda I)^{-1} \Phi^T \vec{y}$$

- ▶ Adds small number  $\lambda$  to diagonal of  $\Phi^T \Phi$
- ▶ Improves condition number of  $\Phi^T \Phi + n\lambda I$ 
  - ▶ Helpful when multicollinearity exists

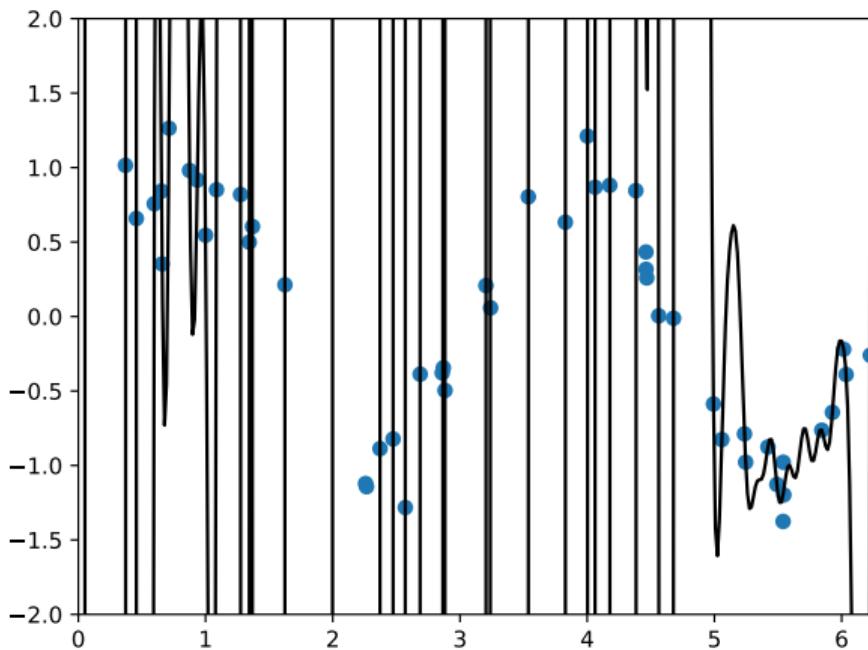
# Demo: Sinusoidal Data

- ▶ Fit curve to 50 noisy data points.
- ▶ Use  $k = 50$  Gaussian basis functions.

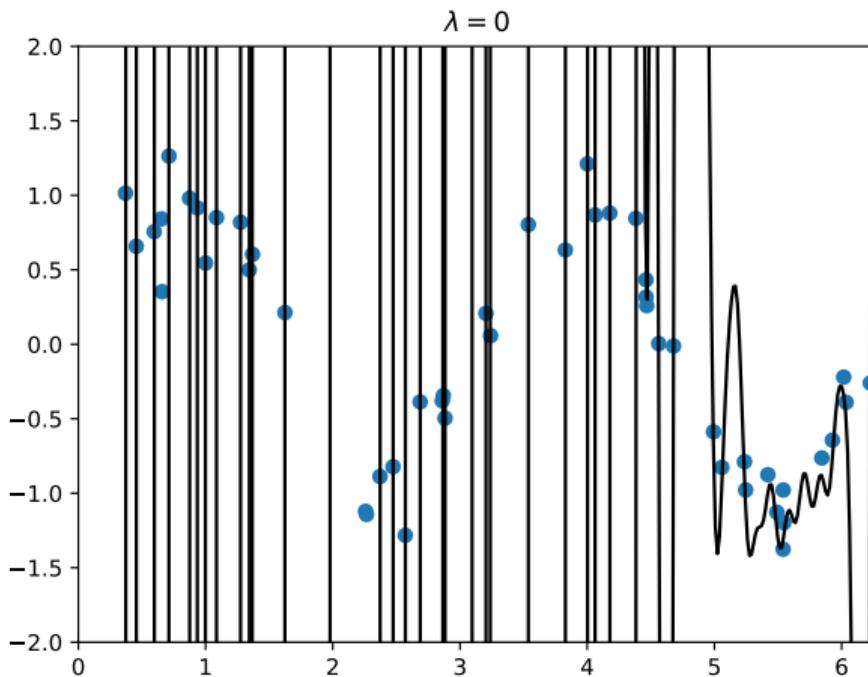


# Result: no regularization

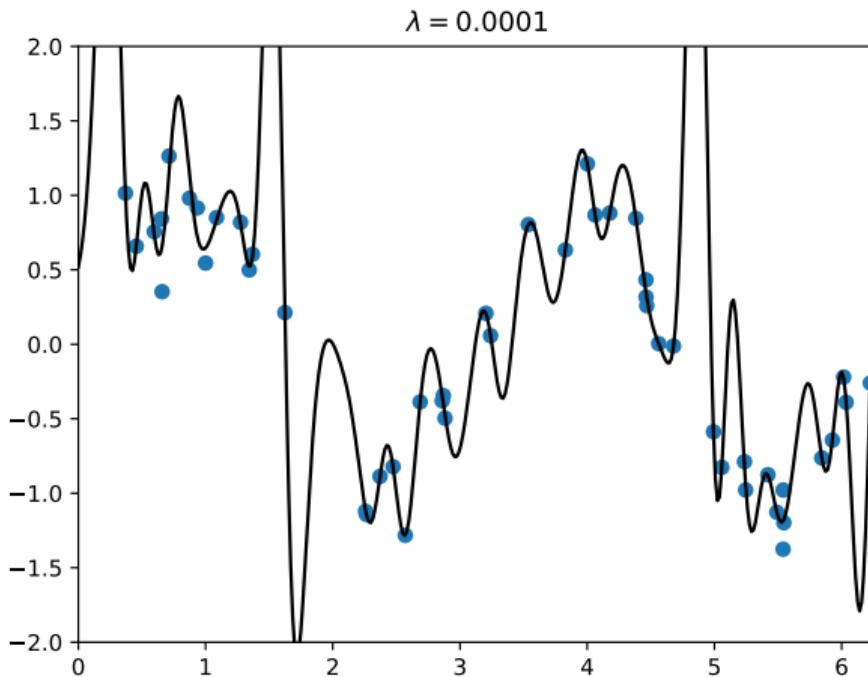
► Overfitting!



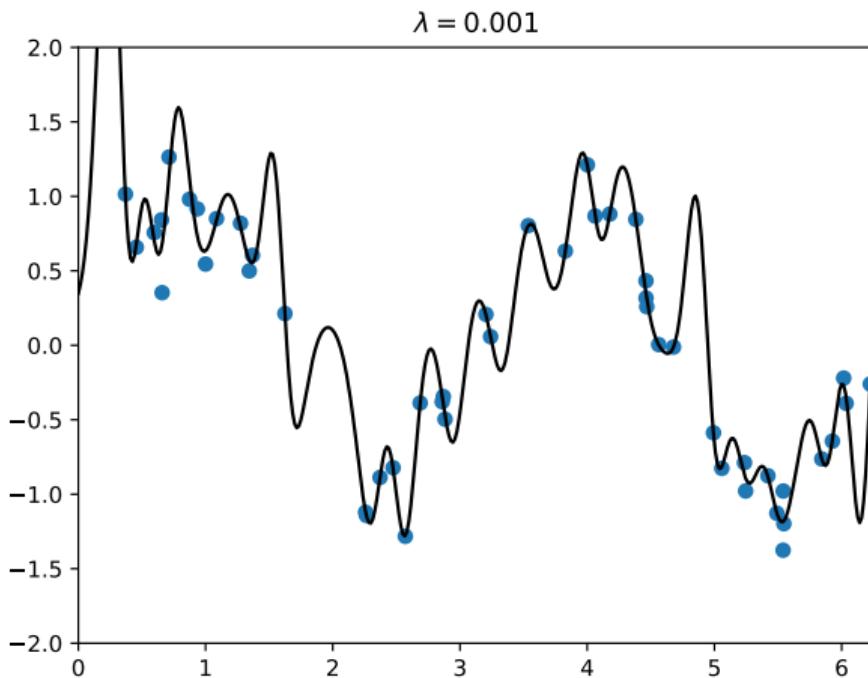
# Result: regularization



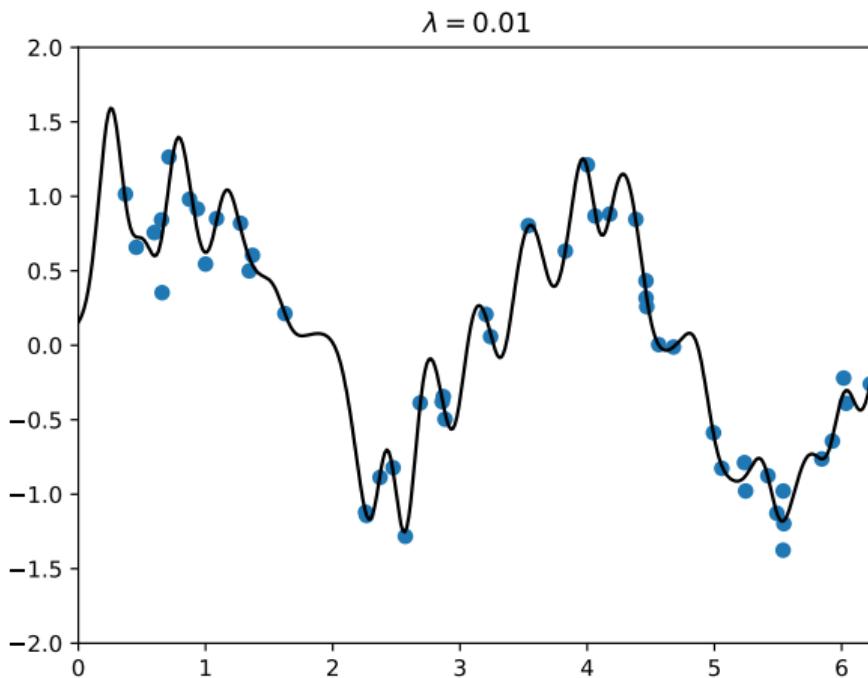
# Result: regularization



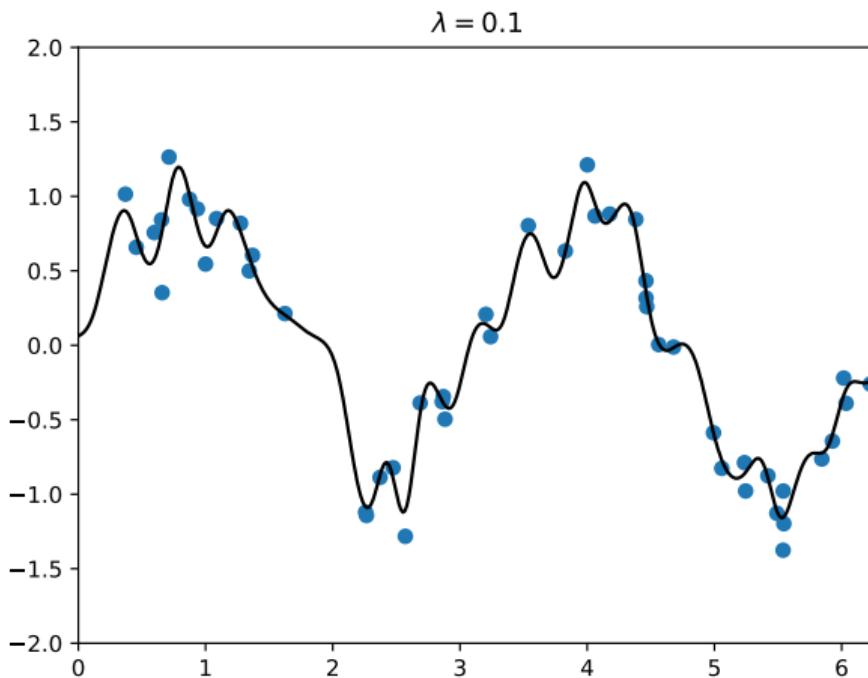
# Result: regularization



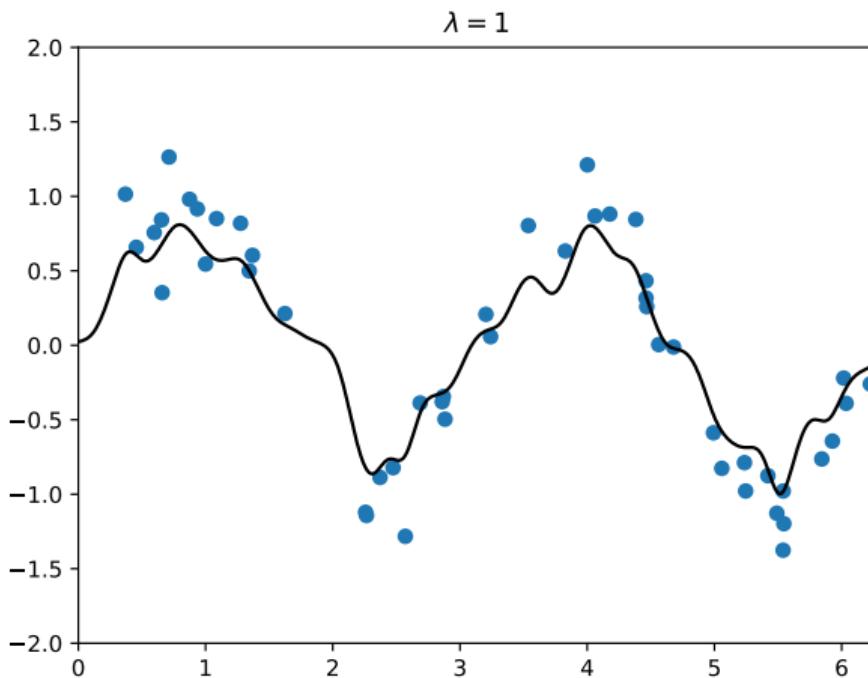
# Result: regularization



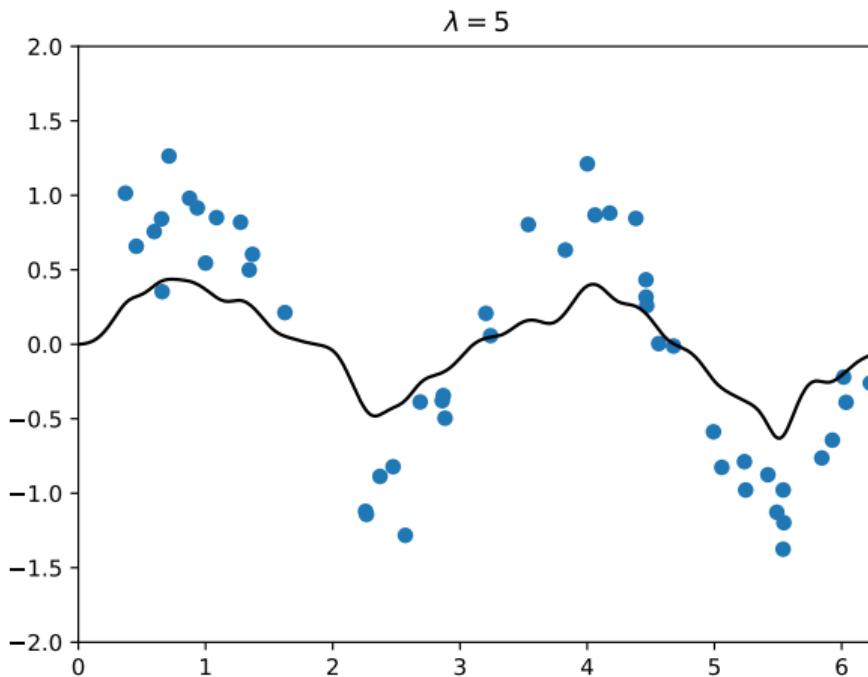
# Result: regularization



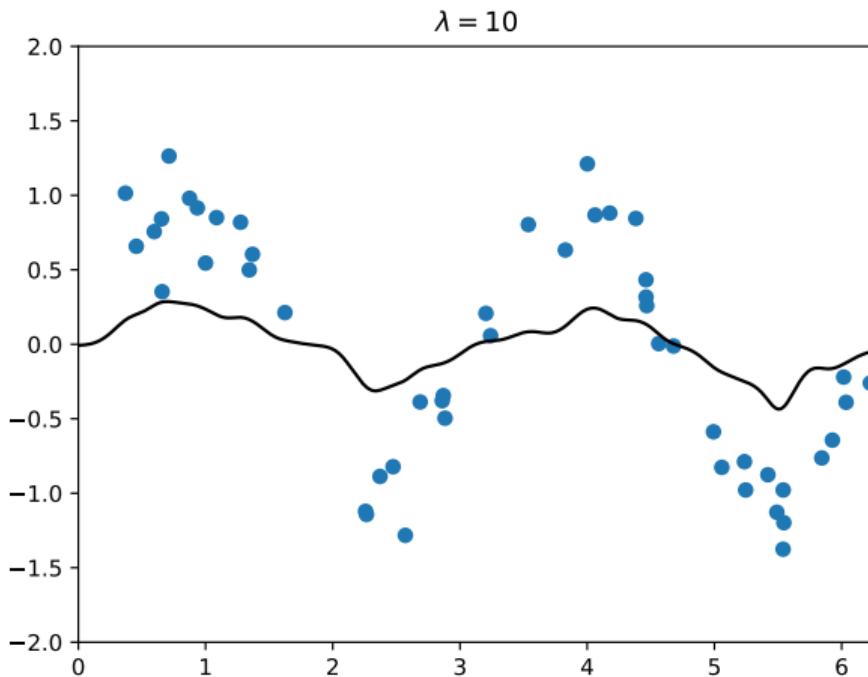
# Result: regularization



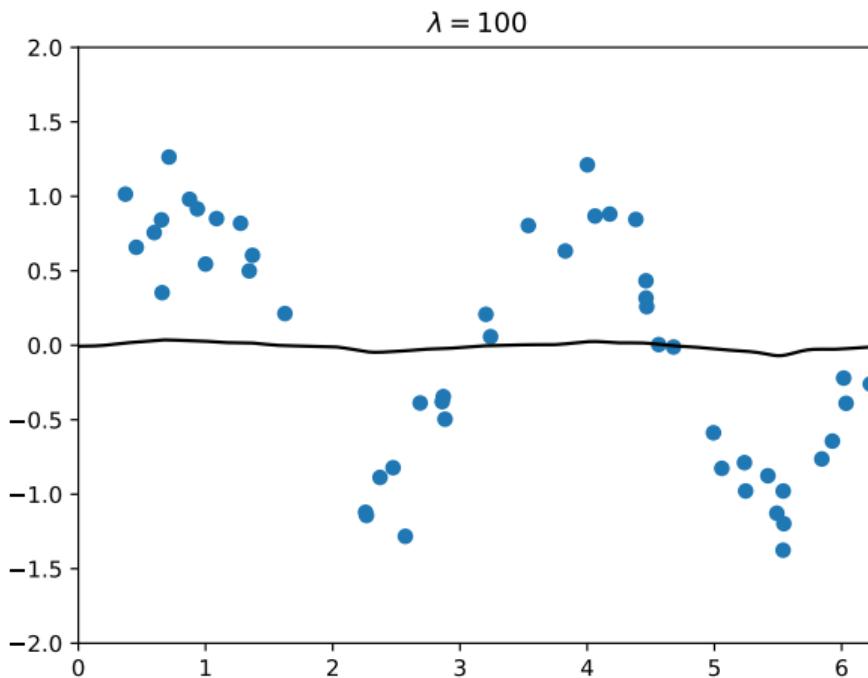
# Result: regularization



# Result: regularization

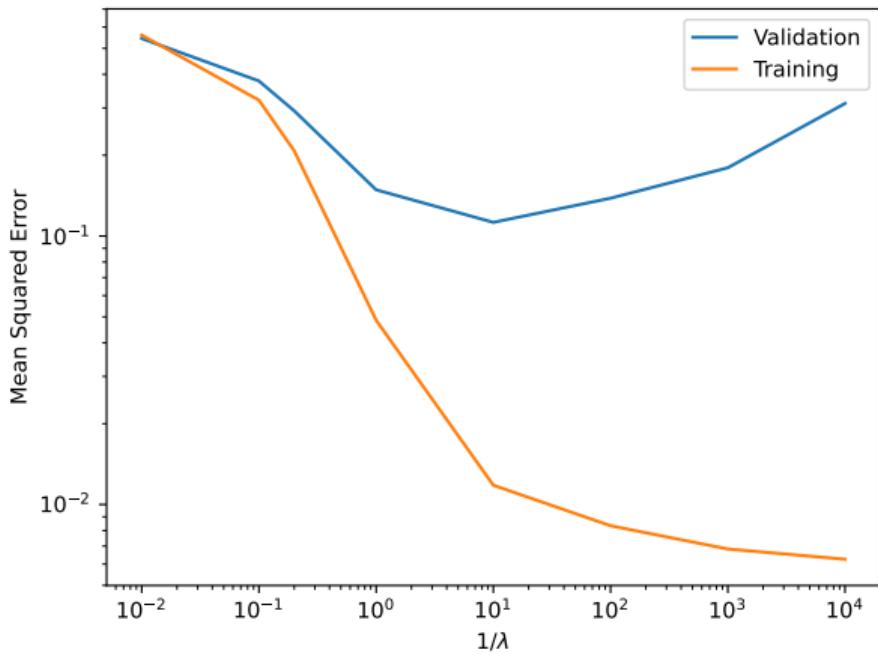


# Result: regularization



# Picking $\lambda$

- ▶  $\lambda$  controls strength of penalty
  - ▶ Larger  $\lambda$ : penalize complexity more
  - ▶ Smaller  $\lambda$ : allow more complexity
- ▶ To choose, use **cross-validation**.



# DSC 1410A

*Probabilistic Modeling & Machine Learning*

Lecture 7 | Part 2

The LASSO

# *p* norm regularization

- ▶ In the last section, we minimized:

$$\tilde{R}(\vec{w}) = \frac{1}{n} \sum_{i=1}^n (\vec{w} \cdot \phi(\vec{x}^{(i)}) - y_i)^2 + \lambda \|\vec{w}\|^2$$

- ▶ What is special about  $\|\vec{w}\|$ ?

## **$p$ norms**

- ▶ For any  $p \in [0, \infty)$ , the  **$p$  norm** of a vector  $\vec{u}$  is defined as

$$\|\vec{u}\|_p = \left( \sum_{i=1}^d |u_i|^p \right)^{1/p}$$

## Special Case: $p = 2$

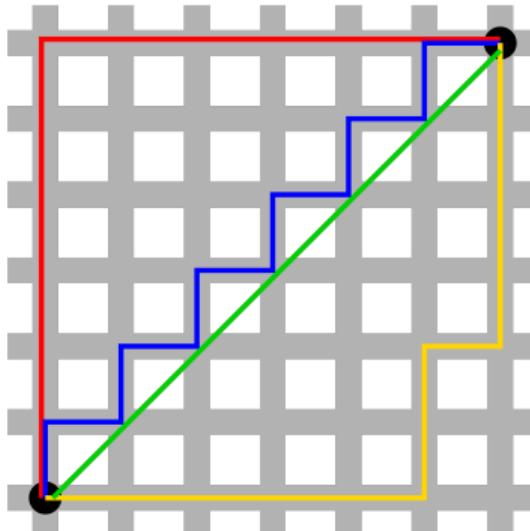
- When  $p = 2$ , we have the familiar **Euclidean norm**:

$$\|\vec{u}\|_2 = \left( \sum_{i=1}^d u_i^2 \right)^{1/2} = \|\vec{u}\|$$

# Special Case: $p = 1$

- When  $p = 1$ , we have the “taxicab norm”

$$\|\vec{u}\|_1 = \sum_{i=1}^d |u_i|$$



# 1-norm Regularization

- ▶ Consider the 1-norm regularized risk:

$$\tilde{R}(\vec{w}) = \frac{1}{n} \sum_{i=1}^n (\vec{w} \cdot \phi(\vec{x}^{(i)}) - y_i)^2 + \lambda \|\vec{w}\|_1$$

- ▶ Least squares regression with 1-norm regularization is called the **LASSO**.

# Solving the LASSO

- ▶ No longer differentiable.
- ▶ No exact solution, unlike ridge regression.<sup>2</sup>
- ▶ Can solve with subgradient descent.

---

<sup>2</sup>Except in special cases, such as orthonormal  $\Phi$

# 1-norm Regularization

- ▶ The 1-norm encourages **sparse** solutions.
  - ▶ That is, solutions where many entries of  $\vec{w}$  are zero.
- ▶ **Interpretation:** feature selection.

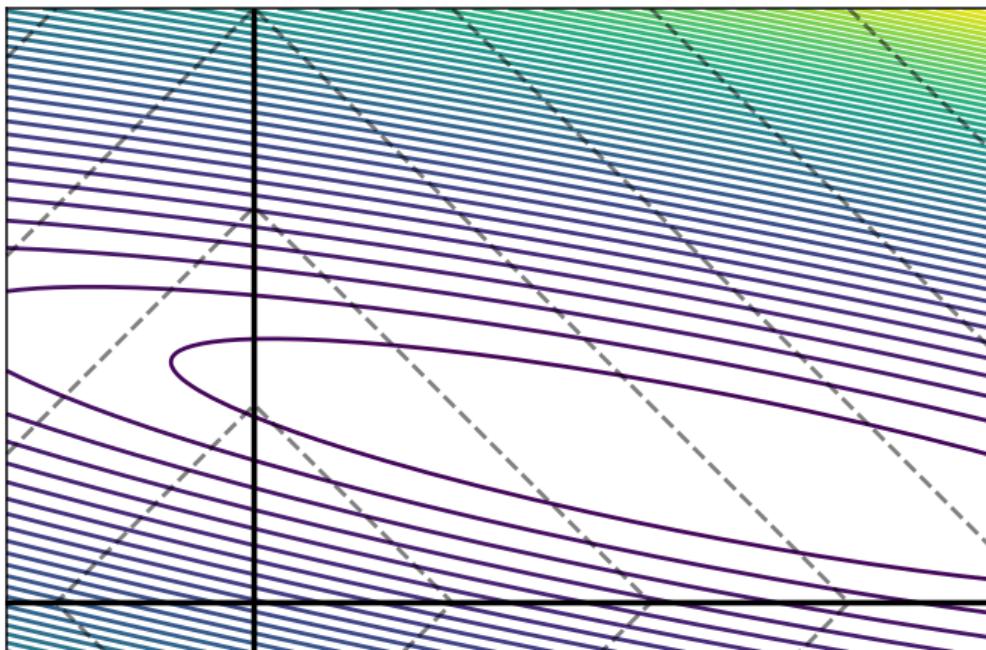
# Example

- ▶ Randomly-generated data:

$$y = 3x_1 + 0.2x_2 - 4x_3 + \mathcal{N}(0, .2)$$

	$w_1$	$w_2$	$w_3$
Unreg.	2.33	-0.08	-4.77
2-norm	2.29	-0.10	-4.73
1-norm	2.72	0	-3.76

# why?



# DSC 140A

*Probabilistic Modeling & Machine Learning*

Lecture 7 | Part 3

**Regularized Risk Minimization**

# Regularized ERM

- ▶ We have seen regularization in the context of least squares regression.
- ▶ However, it is generally useful with other risks.
- ▶ E.g., hinge loss + 2-norm regularization = soft-SVM

# General Regularization

- ▶ Let  $R(\vec{w})$  be a risk function.
- ▶ Let  $\rho(\vec{w})$  be a regularization function.
- ▶ The regularized risk is:

$$\tilde{R}(\vec{w}) = R(\vec{w}) + \rho(\vec{w})$$

- ▶ **Goal:** minimized regularized empirical risk.

# Regularized Linear Models

---

<b>Loss</b>	<b>Regularization</b>	<b>Name</b>
square	2-norm	ridge regression
square	1-norm	LASSO
square	1-norm + 2-norm	elastic net
hinge	2-norm	soft-SVM

# DSC 140A

*Probabilistic Modeling & Machine Learning*

Lecture 8 | Part 1

**High-Dimensional Feature Maps**

# Linear Prediction Rules

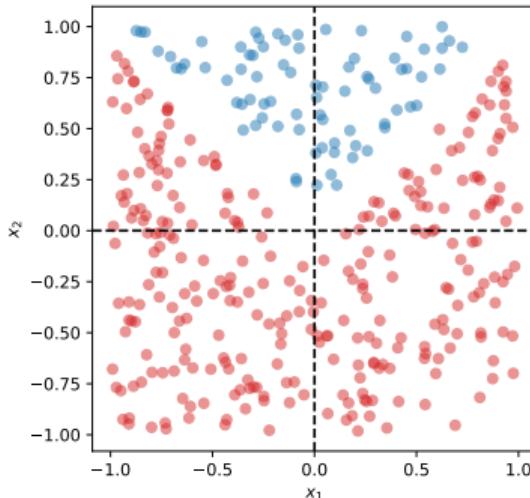
- ▶ We have seen how to fit linear functions:

$$H(\vec{x}) = w_0 + w_1x_1 + \dots + w_dx_d$$

- ▶ Used for both **regression** and **classification**
- ▶ **Limitation:** regression function / decision boundary is a straight line / plane / hyperplane

# Example

- ▶ The data below is not **linearly separable**
- ▶ No prediction function of the form  
 $H(x_1, x_2) = w_0 + w_1x_1 + x_2x_2$  will work well

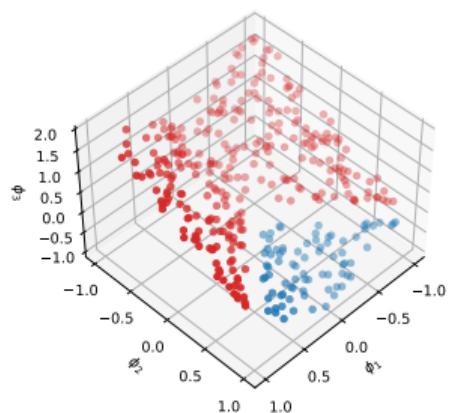
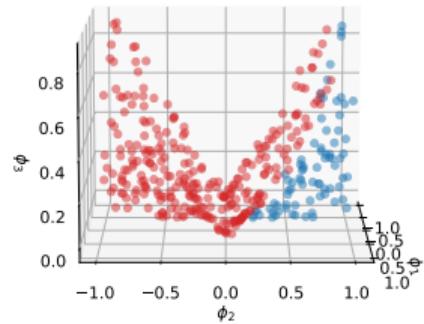
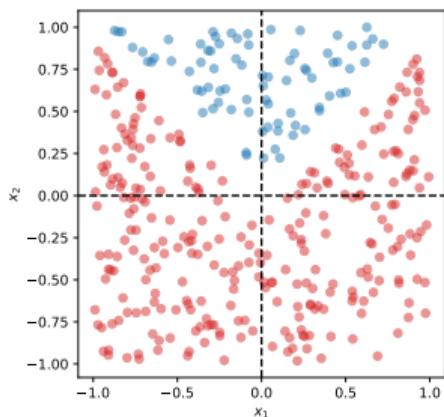


# However...

- ▶ We have seen a way around this limitation: **basis functions**.
- ▶ **Idea:** design a function  $\vec{\phi}(\vec{x})$  that maps data to a new space in which it is **linearly separable**.

# Example

- Consider the mapping  $\vec{\phi}(x_1, x_2) = (x_1, x_2, |x_1 x_2|)^T$

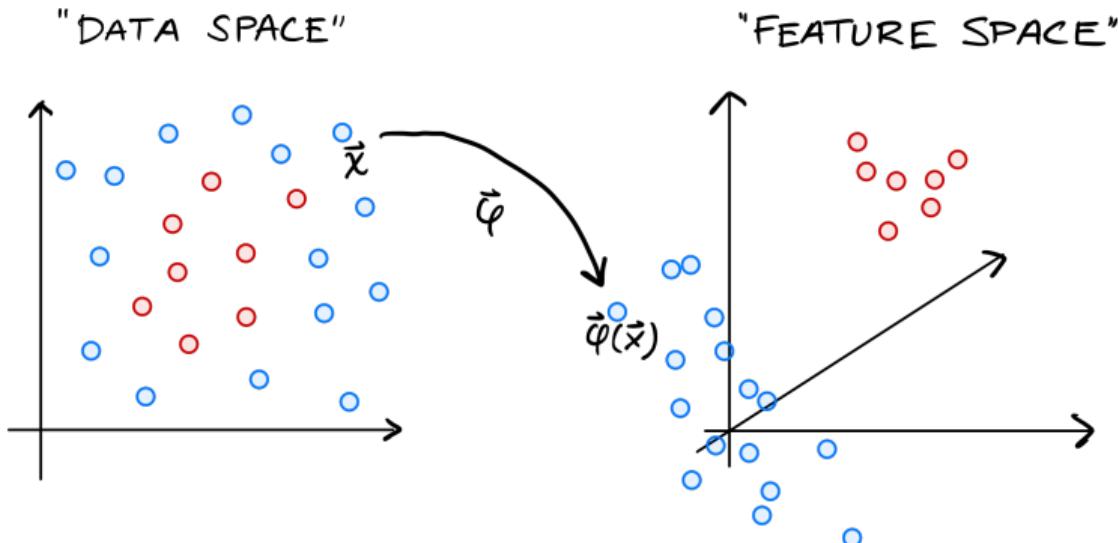


# Procedure

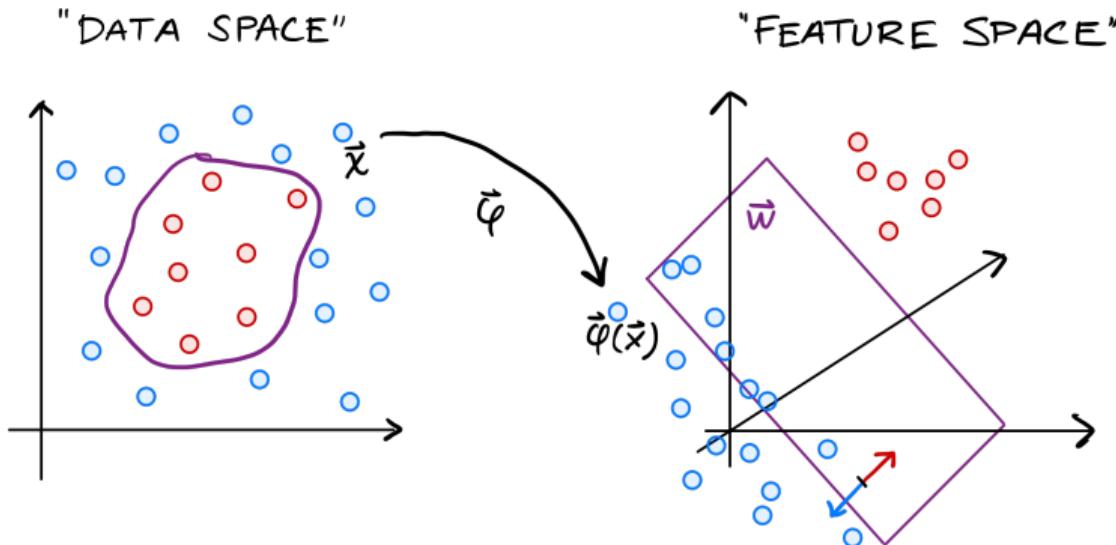
1. Define feature map  $\vec{\phi}(\vec{x}) : \mathbb{R}^d \rightarrow \mathbb{R}^k$ 
  - ▶  $\vec{\phi}(\vec{x}) = (\phi_1(\vec{x}), \dots, \phi_k(\vec{x}))^T$
  - ▶ Number of basis functions  $k$  can be  $>$  or  $\leq$  than  $d$
2. Map each training point to  $k$ -dimensional  
**feature space**:  $\vec{x}^{(i)} \mapsto \vec{\phi}(\vec{x}^{(i)})$
3. Learn a linear predictor in feature space:

$$H(\vec{x}) = w_0 + w_1 \phi_1(\vec{x}) + \dots + w_k \phi_k(\vec{x})$$

# Procedure

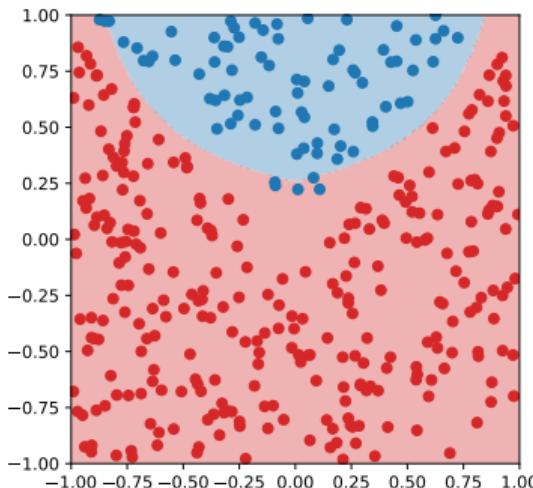


# Procedure



# Example

- ▶ Use mapping  $\vec{\phi}(\vec{x}) = (x_1, x_2, |x_1 x_2|)^T$
- ▶ Decision boundary in “data space” no longer a straight line.



## Exercise

Suppose  $\vec{w} = (3, -1, 2)^T$  defines a linear predictor in feature space and  $\vec{\phi} = (x_1, x_2, |x_1 x_2|)^T$  is the mapping from “data space” to “feature space”.

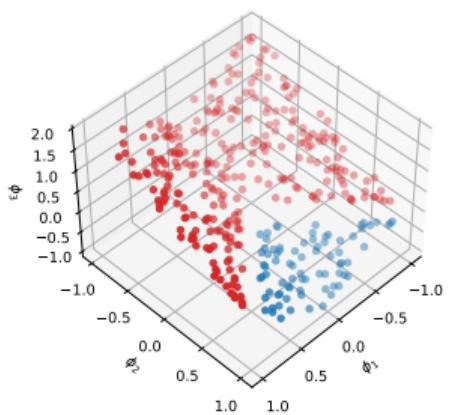
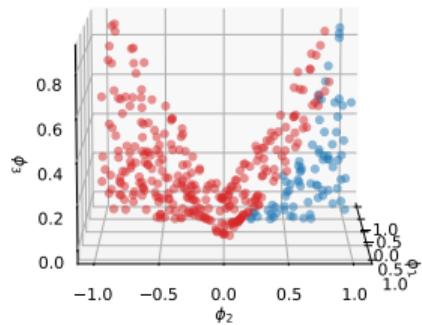
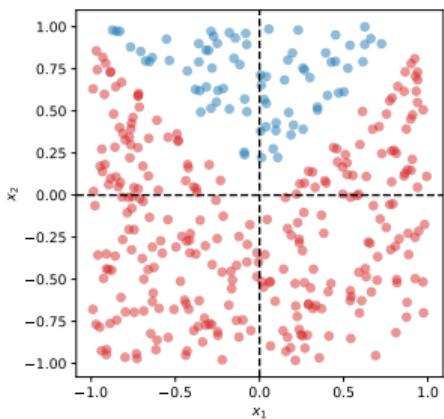
Let  $\vec{x} = (2, -3)^T$  be a new point that needs to be classified. What is the predicted label?

# Feature Maps

- ▶ How do we choose  $\vec{\phi}$ ?
- ▶ **Hope:** data is linearly separable in feature space
- ▶ Appears difficult to engineer  $\vec{\phi}$  to satisfy this.
  - ▶ Need to design  $\vec{\phi}$  for each new data set?
- ▶ **Goal:** design a general feature map that is likely to make any data set linearly separable

# High-Dimensional Feature Maps

- ▶ **Observe:** in our example,  $\vec{\phi}$  mapped to space of larger dimension



# High-Dimensional Feature Maps

- ▶ **Intuition:** each additional feature makes the data easier to classify.
- ▶ **Intuition:** a high-dimensional feature map is likely to make the data linearly separable.
- ▶ **Idea:** design very high-dimensional generic feature maps.

# Example: Monomials

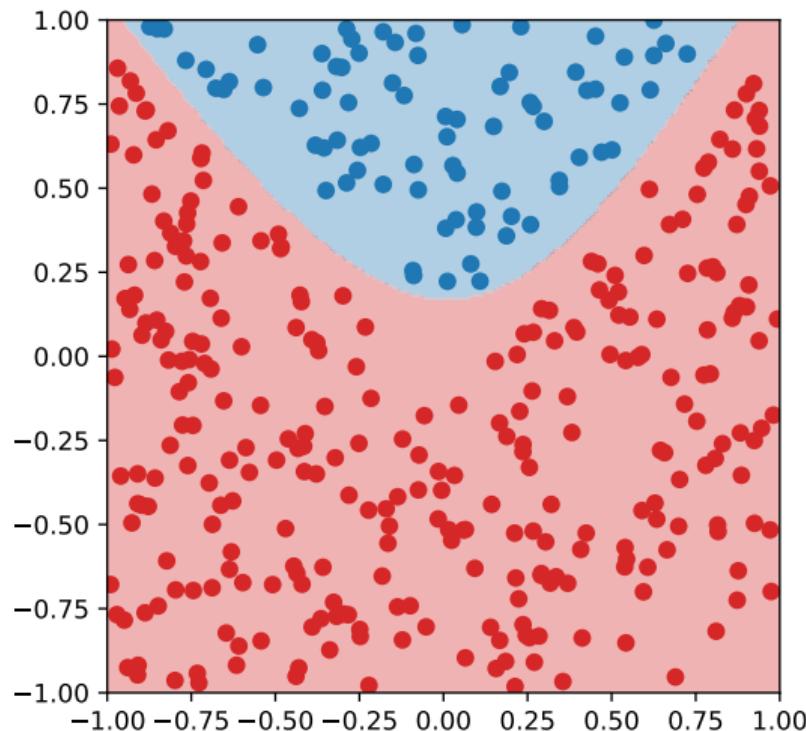
- ▶ Define a feature map  $\vec{\phi} : \mathbb{R}^2 \rightarrow \mathbb{R}^6$  as follows:

$$\vec{\phi}(\vec{x}) = (1, x_1, x_2, x_1x_2, x_1^2, x_2^2)^T$$

- ▶ We fit a prediction function of the form:

$$H(\vec{x}) = w_0 + w_1x_1 + w_2x_2 + w_3x_1x_2 + w_4x_1^2 + w_5x_2^2$$

# Example: Monomials



# Example: Monomials

- ▶ In general, define a feature map  $\vec{\phi}$  to contain all **monomials** of the form:

$$1, \quad x_i, \quad x_i x_j, \quad x_i^2$$

- ▶ If  $\vec{x} \in \mathbb{R}^d$ , then  $\vec{\phi}(\vec{x}) \in \mathbb{R}^{1+2d+\binom{d}{2}}$ .
- ▶ **Example:** if  $\vec{x} \in \mathbb{R}^{50}$ , then  $\vec{\phi}(\vec{x}) \in \mathbb{R}^{1,326}$ .

# Example: Monomials

- ▶ Why stop there? Design  $\vec{\phi}$  to contain all terms of form:

$$1, \quad x_i, \quad x_i x_j, \quad x_i^2, \quad x_i x_j x_k, \quad x_i^3$$

- ▶ If  $\vec{x} \in \mathbb{R}^d$ , then  $\vec{\phi}(\vec{x}) \in \mathbb{R}^{1+3d+\binom{d}{2}+\binom{d}{3}}$ .
- ▶ **Example:** if  $\vec{x} \in \mathbb{R}^{50}$ , then  $\vec{\phi}(\vec{x}) \in \mathbb{R}^{20,976}$ !
- ▶ And so on...

# Problem

- ▶ Mapping to very high dimensions is likely to make the data linearly separable.
- ▶ But fitting a linear prediction rule in high dimensions is **costly**.

# DSC 140A

*Probabilistic Modeling & Machine Learning*

Lecture 8 | Part 2

**The Kernel Trick**

# Recap

- We can learn non-linear patterns by:
  1. Defining a high-dimensional feature map,  
 $\vec{\phi} : \mathbb{R}^d \rightarrow \mathbb{R}^k$
  2. Mapping each training point to  $k$ -dimensional **feature space**:  $\vec{x}^{(i)} \mapsto \vec{\phi}(\vec{x}^{(i)})$
  3. Training a linear predictor in feature space.

# Problem

- ▶ Learning in a very high-dimensional space can be costly, or even infeasible.

# The Trick

- ▶ We can train many linear predictors *as if* we have mapped data to feature space, **without actually doing so.**

# Idea

- ▶ In many algorithms, when  $\vec{\phi}(\vec{x})$  appears, it always appears as part of a dot product:

$$\vec{\phi}(\vec{x}) \cdot \vec{\phi}(\vec{x}')$$

- ▶ To compute, we *could* map and do dot product in feature space.
- ▶ But this is **costly**!

# Kernels

- ▶ But some  $\vec{\phi}$  are special; for them, there is a function  $\kappa$  satisfying:

$$\kappa(\vec{x}, \vec{x}') = \vec{\phi}(\vec{x}) \cdot \vec{\phi}(\vec{x}')$$

- ▶ Crucially, computing  $\kappa$  does **not require mapping to feature space!**
- ▶  $\kappa$  is called a **kernel** function.

# The Kernel Trick

- ▶ In many algorithms, when  $\vec{\phi}(\vec{x})$  appears, it always appears as part of a dot product of the form:

$$\vec{\phi}(\vec{x}) \cdot \vec{\phi}(\vec{x}')$$

- ▶ By replacing all instances of  $\vec{\phi}(\vec{x}) \cdot \vec{\phi}(\vec{x}')$  with  $\kappa(\vec{x}, \vec{x}')$ , we **kernelize** the algorithm; avoid mapping to feature space.
- ▶ This is called the **kernel trick**.

# Example: Polynomial Kernel

- ▶ Define the feature map:

$$\vec{\phi}(\vec{x}) = (1, x_1^2, x_2^2, x_3^2, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_3, \sqrt{2}x_1x_2, \sqrt{2}x_1x_3, \sqrt{2}x_2x_3)^T$$

- ▶  $\kappa(\vec{x}, \vec{x}') = (1 + \vec{x} \cdot \vec{x}')^2$  is a kernel for this  $\vec{\phi}$ .
  - ▶ That is,  $\kappa(\vec{x}, \vec{x}') = \vec{\phi}(\vec{x}) \cdot \vec{\phi}(\vec{x}')$
- ▶ Called the **polynomial kernel**<sup>1</sup>

---

<sup>1</sup>In general,  $\kappa(\vec{x}, \vec{x}') = (1 + \vec{x} \cdot \vec{x}')^k$  is kernel for  $k$ -order monomial mappings

# Kernelized Algorithms

- ▶ Only certain mappings have efficiently-computed kernels.
- ▶ Only certain learning algorithms can be **kernelized**.
- ▶ **All** of the linear algorithms we've learned can.
  - ▶ Least squares, perceptron, SVMs, etc.

# Kernel Ridge Regression

- ▶ Let's kernelize **ridge regression**.
- ▶ First: verify that all instances of  $\vec{\phi}(\vec{x})$  appear as part of a dot product:  $\vec{\phi}(\vec{x}) \cdot \vec{\phi}(\vec{x}')$

# Kernel Ridge Regression

- ▶ Suppose  $\vec{\phi}(\vec{x})$  is a feature map with kernel  $k$ .
- ▶ To train a ridge regressor in feature space, we'd solve

$$\arg \min_{\vec{w}} \frac{1}{n} \sum_{i=1}^n (\vec{\phi}(\vec{x}^{(i)}) \cdot \vec{w} - y_i)^2 + \lambda \|\vec{w}\|^2$$

- ▶ In matrix-vector form, where  $\Phi$  is the design matrix:

$$\arg \min_{\vec{w}} \frac{1}{n} \|\Phi \vec{w} - \vec{y}\|^2 + \lambda \vec{w}^T \vec{w}$$

# Fact

- The solution  $w^*$  is a linear combination of  $\vec{\phi}(\vec{x}^{(i)})$ :

$$\vec{w}^* = \sum_{i=1}^n \alpha_i \vec{\phi}(\vec{x}^{(i)})$$

- Why? The gradient of the regularized risk is:

$$\frac{2}{n} \sum_{i=1}^n (\vec{\phi}(\vec{x}^{(i)}) \cdot \vec{w} - y_i) \vec{\phi}(\vec{x}^{(i)}) + 2\lambda \vec{w}$$

- Setting to zero, solving for  $\vec{w}$  gives:

$$\vec{w}^* = \sum_{i=1}^n \underbrace{\left( -\frac{1}{n\lambda} \vec{\phi}(\vec{x}^{(i)}) \cdot \vec{w}^* - y_i \right)}_{\alpha_i} \vec{\phi}(\vec{x}^{(i)})$$

# Fact

- ▶ The solution  $w^*$  is a linear combination of  $\vec{\phi}(\vec{x}^{(i)})$ :

$$\vec{w}^* = \sum_{i=1}^n \alpha_i \vec{\phi}(\vec{x}^{(i)})$$

- ▶ In matrix-vector form, where  $\vec{\alpha} = (\alpha_1, \dots, \alpha_n)^T$ :

$$\vec{w}^* = \Phi^T \vec{\alpha}$$

# Dual Problem

- ▶ Using the fact that  $\vec{w}^* = \sum_{i=1}^n \alpha_i \vec{\phi}(\vec{x}^{(i)}) = \Phi^T \vec{\alpha}$  for some  $\vec{\alpha}$ , the problem:

$$\arg \min_{\vec{w}} \frac{1}{n} \|\Phi \vec{w} - \vec{y}\|^2 + \lambda \vec{w}^T \vec{w}$$

is equivalent to the **dual** problem:

$$\arg \min_{\vec{\alpha}} \frac{1}{n} \|\Phi \Phi^T \vec{\alpha} - \vec{y}\|^2 + \lambda \vec{\alpha}^T \Phi \Phi^T \vec{\alpha}$$

# Kernelizing

- Where does  $\vec{\phi}(\vec{x})$  appear in this problem?

$$\arg \min_{\vec{a}} \frac{1}{n} \|\Phi \Phi^T \vec{a} - \vec{y}\|^2 + \lambda \vec{a}^T \Phi \Phi^T \vec{a}$$

- Inside  $\Phi$ :

$$\Phi = \begin{pmatrix} \vec{\phi}(\vec{x}^{(1)}) & \longrightarrow \\ \vec{\phi}(\vec{x}^{(2)}) & \longrightarrow \\ \vdots & \\ \vec{\phi}(\vec{x}^{(n)}) & \longrightarrow \end{pmatrix}$$

## Exercise

Argue that the (i, j) entry of  $\Phi\Phi^T$  is equal to  $\kappa(\vec{x}^{(i)}, \vec{x}^{(j)})$ .

$$\Phi = \begin{pmatrix} \vec{\phi}(\vec{x}^{(1)}) & \longrightarrow \\ \vec{\phi}(\vec{x}^{(2)}) & \longrightarrow \\ \vdots & \\ \vec{\phi}(\vec{x}^{(n)}) & \longrightarrow \end{pmatrix}$$

# Kernelizing

- The  $(i, j)$  entry of  $\Phi\Phi^T$  is  $\vec{\phi}(\vec{x}^{(i)}) \cdot \vec{\phi}(\vec{x}^{(j)}) = \kappa(\vec{x}^{(i)}, \vec{x}^{(j)})$

$$\Phi\Phi^T = \underbrace{\begin{pmatrix} \kappa(\vec{x}^{(1)}, \vec{x}^{(1)}) & \kappa(\vec{x}^{(1)}, \vec{x}^{(2)}) & \dots & \kappa(\vec{x}^{(1)}, \vec{x}^{(n)}) \\ \kappa(\vec{x}^{(2)}, \vec{x}^{(1)}) & \kappa(\vec{x}^{(2)}, \vec{x}^{(2)}) & \dots & \kappa(\vec{x}^{(2)}, \vec{x}^{(n)}) \\ \vdots & \vdots & \ddots & \vdots \\ \kappa(\vec{x}^{(n)}, \vec{x}^{(1)}) & \kappa(\vec{x}^{(n)}, \vec{x}^{(2)}) & \dots & \kappa(\vec{x}^{(n)}, \vec{x}^{(n)}) \end{pmatrix}}_K$$

- $K$  is called the **Kernel matrix** (or **Gram matrix**).

# Kernel Ridge Regression

- The dual problem becomes:

$$\arg \min_{\vec{\alpha}} \frac{1}{n} \|K\vec{\alpha} - \vec{y}\|^2 + \lambda \vec{\alpha}^T K \vec{\alpha}$$

- Exact solution:

$$\vec{\alpha}^* = (K + n\lambda I)^{-1} \vec{y}$$

- This is **kernel ridge regression**.

# Kernelization

- ▶ **Observe:** we train linear predictor in feature space without actually mapping to feature space:

$$\vec{\alpha}^* = (K + n\lambda I)^{-1} \vec{y}$$

# Making Predictions

- ▶ To predict on a new point  $\vec{x}$ , normally:  
 $H(\vec{x}) = \vec{w}^* \cdot \vec{\phi}(\vec{x}).$
- ▶ How to do this without actually mapping?
- ▶ Recall:  $w^* = \sum_{i=1}^n \alpha_i^* \vec{\phi}(\vec{x}^{(i)})$
- ▶ So:

$$H(\vec{x}) = \sum_{i=1}^n \alpha_i^* \vec{\phi}(\vec{x}^{(i)}) \cdot \vec{\phi}(\vec{x}) = \sum_{i=1}^n \alpha_i^* \kappa(\vec{x}^{(i)}, \vec{x})$$

# Making Predictions

- ▶ To make a prediction on a new point:

$$H(\vec{x}) = \sum_{i=1}^n \alpha_i^* \kappa(\vec{x}^{(i)}, \vec{x})$$

- ▶ No need to map to feature space.
- ▶ **Interpretation:** A weighted sum of kernel evaluations.

# Procedure: Kernel Ridge Regression

1. Pick a kernel function,  $\kappa$ .
2. Solve linear system:  $\vec{\alpha}^* = (K + n\lambda I)^{-1}\vec{y}$
3. To make new prediction,  $H(\vec{x}) = \sum_{i=1}^n \alpha_i^* \kappa(\vec{x}^{(i)}, \vec{x})$

# Kernel Soft-SVM

- ▶ Soft-SVM can also be **kernelized**.
- 1. Pick a kernel function,  $\kappa$ .
- 2. Solve dual problem (e.g., with SGD):
$$\arg \min_{\vec{\alpha}} \left( \lambda \vec{\alpha}^T K \vec{\alpha} + \frac{1}{n} \sum_{i=1}^n \max\{0, 1 - y_i (\vec{\alpha}^T K \vec{\alpha})_i\} \right)$$
- 3. To make new prediction,  $H(\vec{x}) = \sum_{i \in S} \alpha_i^* \kappa(\vec{x}^{(i)}, \vec{x})$ 
  - ▶ Where  $S$  is the set of indices of support vectors.

## Kernelization **Downsides**

- ▶ Often, training involves the  $n \times n$  kernel matrix.
  - ▶ Can be very large!
- ▶ There are ways to mitigate this:
  - ▶ Small-batch stochastic gradient descent.
  - ▶ Nyström method.

# DSC 140A

*Probabilistic Modeling & Machine Learning*

Lecture 8 | Part 3

**Kernel Functions**

# Valid Kernels

- ▶ The first step in kernel learning is to pick a **kernel function**,  $\kappa$ .
- ▶ To be a valid kernel, must compute the dot product w.r.t., some mapping,  $\vec{\phi}(\vec{x})$ .
- ▶ That is, it must be that

$$\kappa(\vec{x}, \vec{x}') = \vec{\phi}(\vec{x}) \cdot \vec{\phi}(\vec{x}')$$

for some  $\vec{\phi}$ .

# Constructing Kernels: Approach #1

- ▶ How do we come up with valid kernel functions?
- ▶ Approach #1:
  1. Start by picking  $\vec{\phi}$
  2. Find a function  $\kappa$  that efficiently computes  $\vec{\phi}(\vec{x}) \cdot \vec{\phi}(\vec{x}')$ , if one exists.

# Constructing Kernels: Approach #2

- ▶ New kernels can be constructed from other kernels.
- ▶ Suppose  $\kappa_1, \kappa_2, \kappa_3$  are kernels and  $f$  is any function. Then the below are kernels:
  - ▶  $\kappa(\vec{x}, \vec{x}') = \kappa_1(\vec{x}, \vec{x}') + \kappa_2(\vec{x}, \vec{x}')$
  - ▶  $\kappa(\vec{x}, \vec{x}') = \kappa_1(\vec{x}, \vec{x}') \times \kappa_2(\vec{x}, \vec{x}')$
  - ▶  $\kappa(\vec{x}, \vec{x}') = \kappa_3(\phi(\vec{x}), \phi(\vec{x}'))$
  - ▶  $\kappa(\vec{x}, \vec{x}') = f(\vec{x})\kappa_1(\vec{x}, \vec{x}')f(\vec{x}')$

# Verifying Kernels

## Theorem

A symmetric function  $\kappa$  is a valid kernel if and only if the kernel matrix,  $K$ , is positive semi-definite for any choice of data,  $\vec{x}^{(1)}, \dots, \vec{x}^{(n)}$ .

# Radial Basis Function Kernel

- ▶ Often, though, we don't design our own kernel.
- ▶ A very popular choice: the **radial basis function (RBF) kernel** (or **Gaussian kernel**):

$$\kappa(\vec{x}, \vec{x}') = e^{\frac{-\|\vec{x}-\vec{x}'\|^2}{2\sigma^2}} = e^{-\gamma\|\vec{x}-\vec{x}'\|^2} \quad \text{where } \gamma = 1/(2\sigma^2)$$

# RBF Kernel Interpretation

$$\kappa(\vec{x}, \vec{x}') = e^{\frac{-\|\vec{x}-\vec{x}'\|^2}{2\sigma^2}} = e^{-\gamma\|\vec{x}-\vec{x}'\|^2}$$

- ▶ **Interpretation:** RBF kernel measures similarity of  $\vec{x}$  and  $\vec{x}'$ 
  - ▶ Very similar:  $\kappa(\vec{x}, \vec{x}') \approx 1$ .
  - ▶ Very different:  $\kappa(\vec{x}, \vec{x}') \approx 0$ .
- ▶ Parameter  $\sigma$  (or  $\gamma$ ) controls the scale
  - ▶ The larger  $\sigma$  (smaller  $\gamma$ ), the wider the Gaussian

# RBF Kernel Interpretation

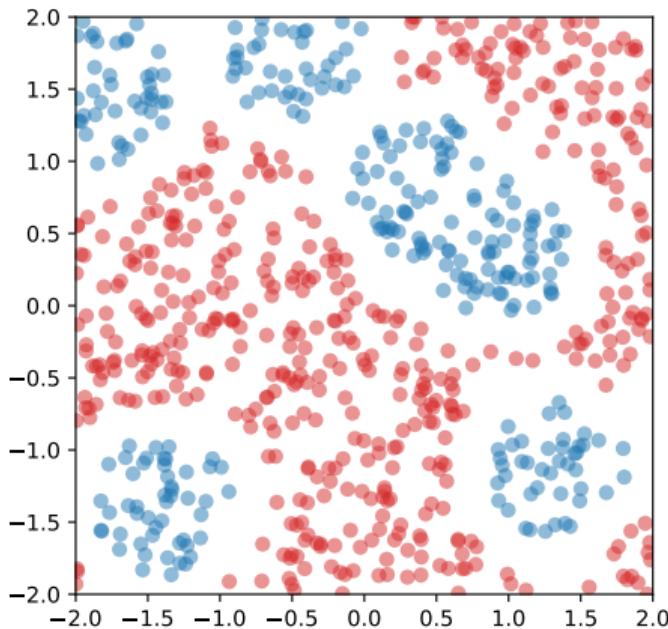
- ▶ Recall that in kernel ridge regression / SVM, the prediction is:

$$H(\vec{x}) = \sum_{i=1}^n \alpha_i \kappa(\vec{x}^{(i)}, \vec{x})$$

- ▶ **Observations:**
  - ▶ One parameter  $\alpha_i$  learned for **each** training point  $\vec{x}^{(i)}$
  - ▶  $\kappa(\vec{x}^{(i)}, \vec{x})$  will be  $\approx 0$  for any  $\vec{x}^{(i)}$  far from  $\vec{x}$
  - ▶  $H(\vec{x})$  is largely determined by the training points closest to  $\vec{x}$

# RBF Kernel Interpretation

- ▶ RBF function placed at each training point.
- ▶  $H(\vec{x})$  is largely determined by training points closest to  $\vec{x}$



## RBF Kernel Interpretation

- ▶ An RBF Kernel predictor can be seen as a generalization of the  $k$ -nearest neighbor rule

# RBF Kernel Map

- ▶ What  $\phi$  is the RBF kernel a kernel for?
- ▶ The mapping  $\vec{\phi}(\vec{x})$  with entries of the form:

$$e^{-\|\vec{x}\|^2/2}x_i, \quad \frac{1}{\sqrt{2!}}e^{-\|\vec{x}\|^2/2}x_i x_j, \quad \frac{1}{\sqrt{3!}}e^{-\|\vec{x}\|^2/2}x_i x_j x_k, \quad \dots$$

- ▶ This is a mapping to an **infinite dimensional Hilbert space!**

# Other Kernels

- ▶ There are other interesting kernels useful for specific domains.
- ▶ **Example:** string kernels for text classification.
  - ▶ Dot product in space generated by all substrings.

# DSC 1410A

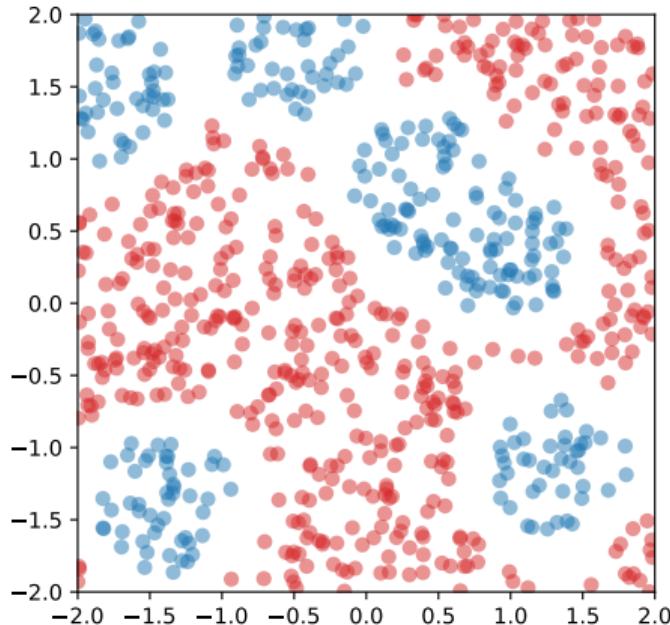
*Probabilistic Modeling & Machine Learning*

Lecture 8 | Part 4

**Demo: Kernel SVM**

# Demo

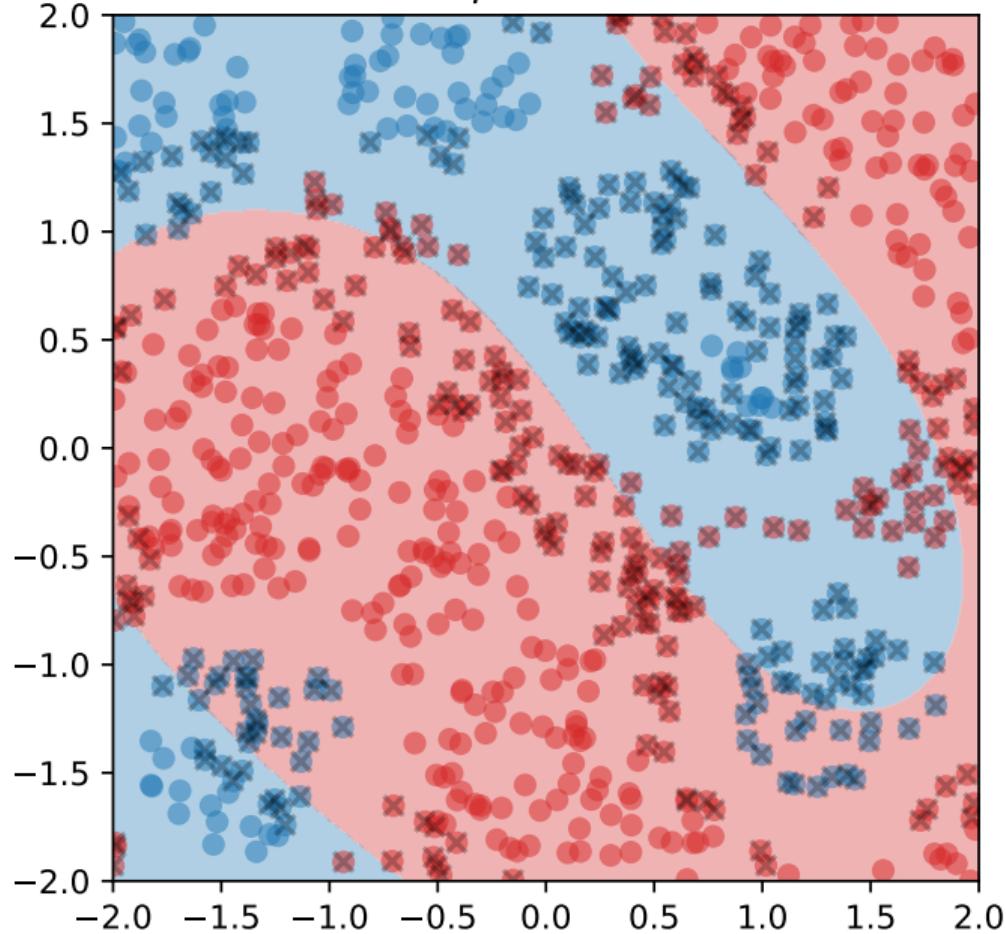
- ▶ Train an RBF kernel SVM on the data below.

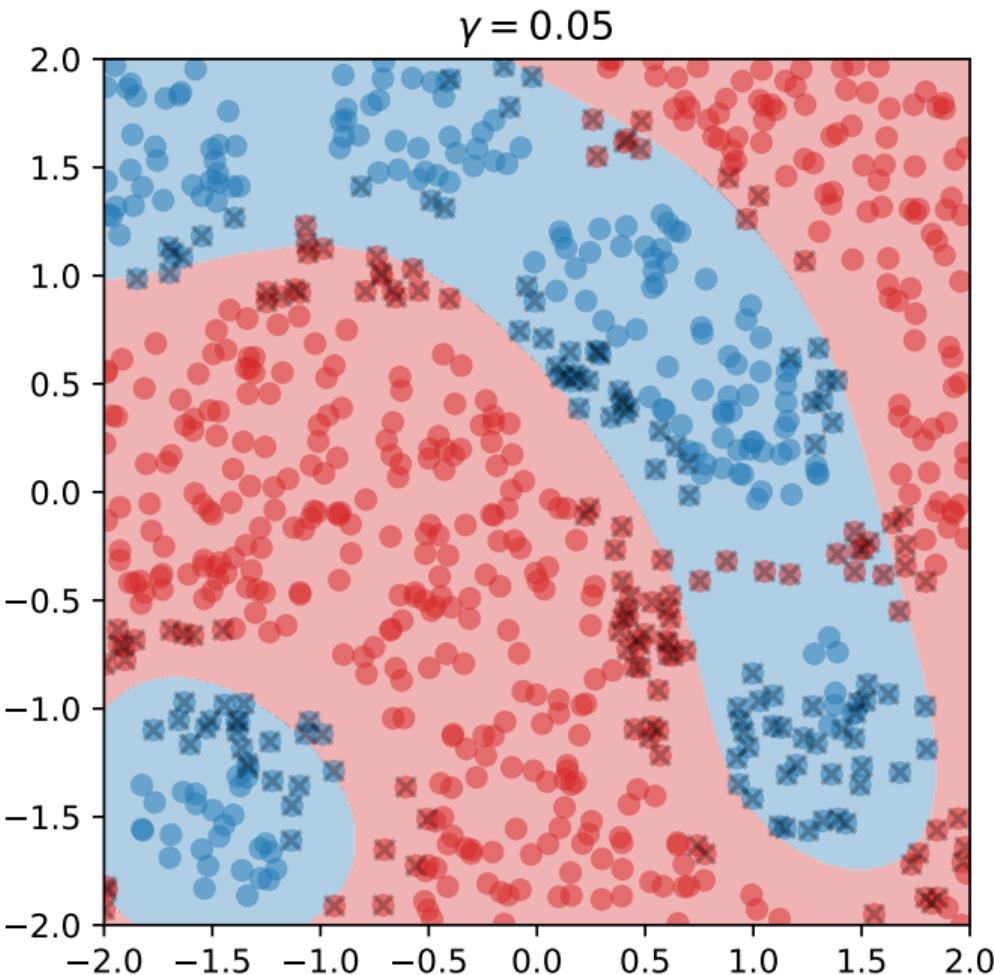


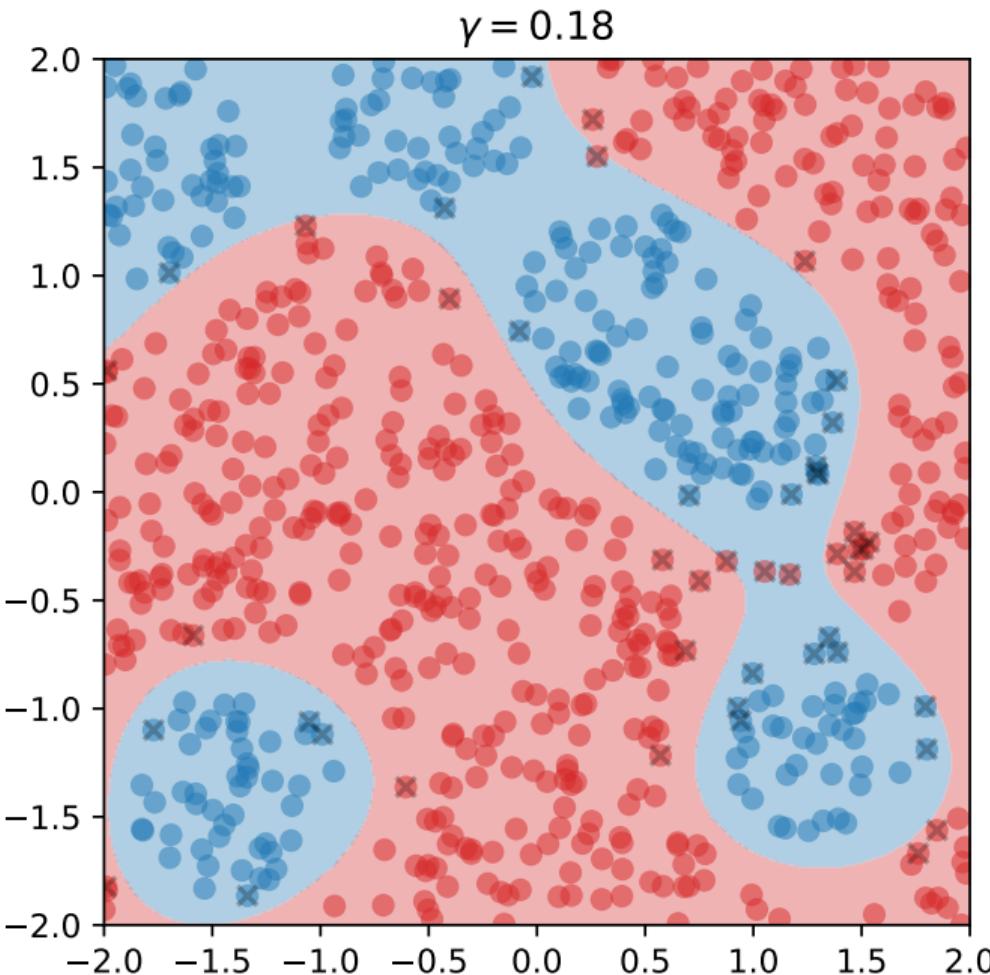
## Aside: Hyperparameter Selection

- ▶ Two hyperparameters to specify:
  - ▶ Slack:  $C$
  - ▶ Kernel width:  $\gamma$
- ▶ Choose with grid search cross-validation

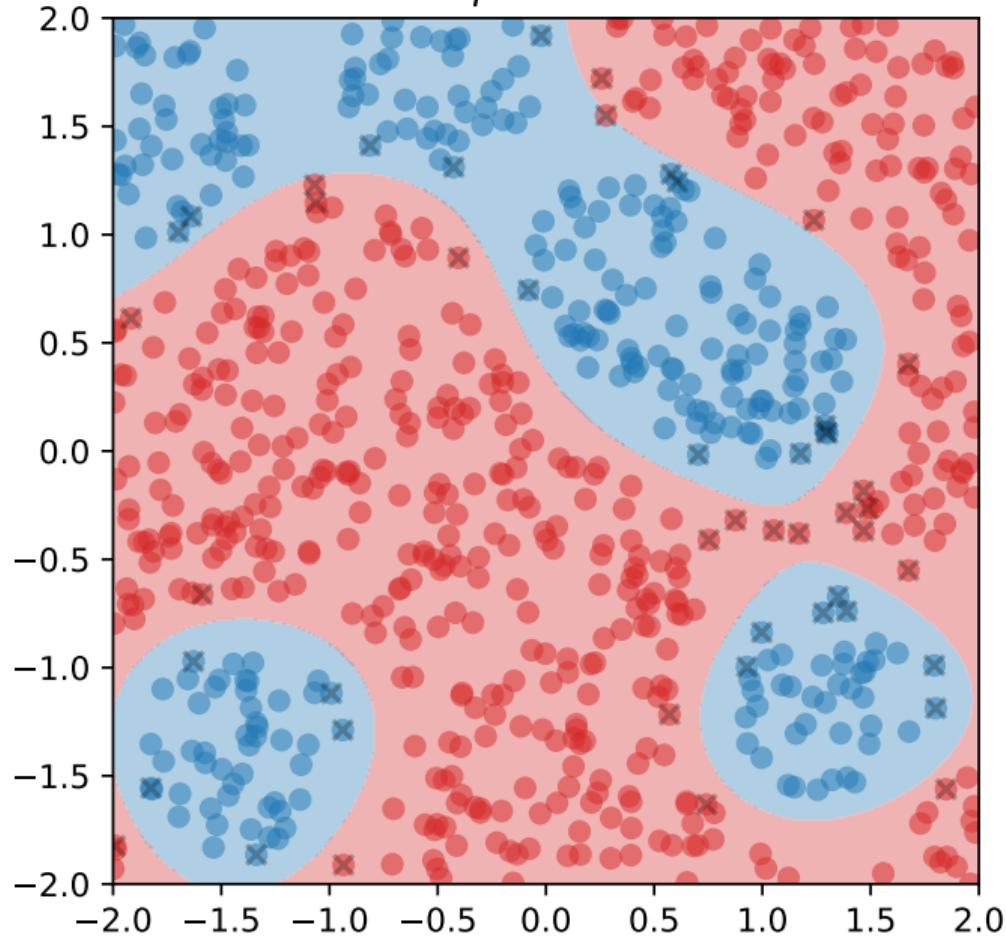
$$\gamma = 0.02$$



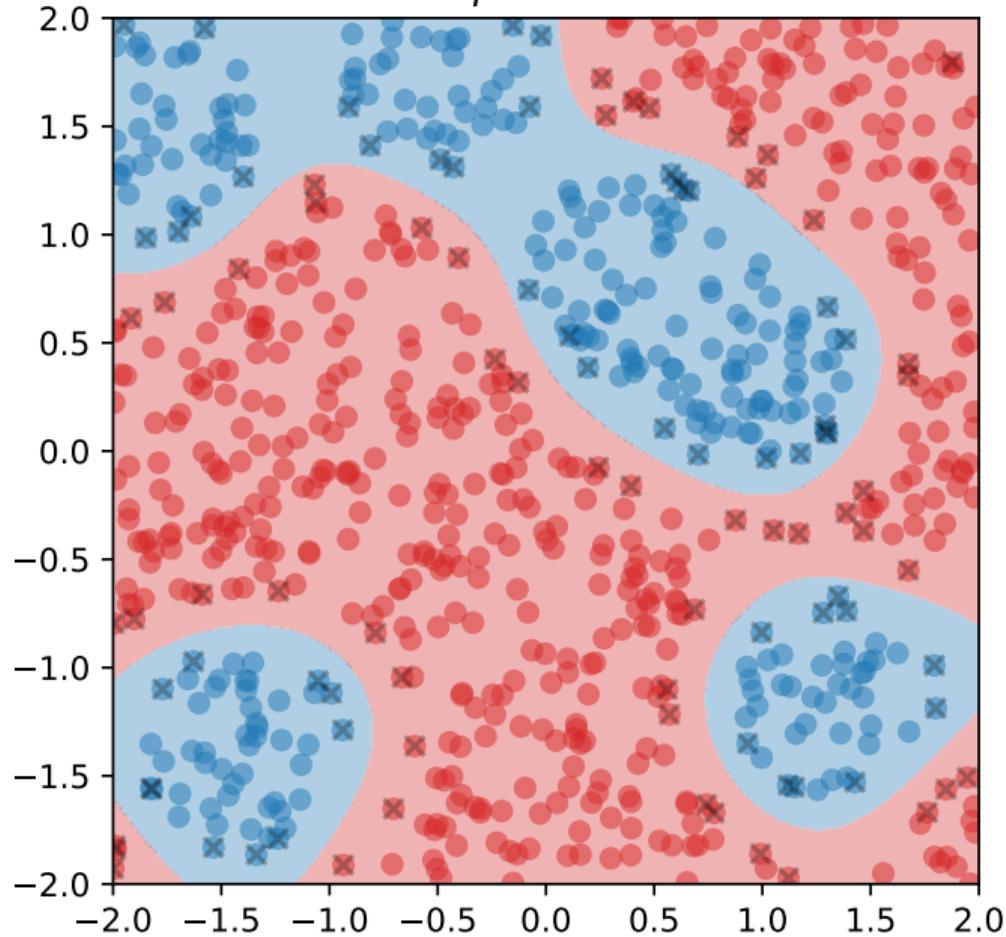




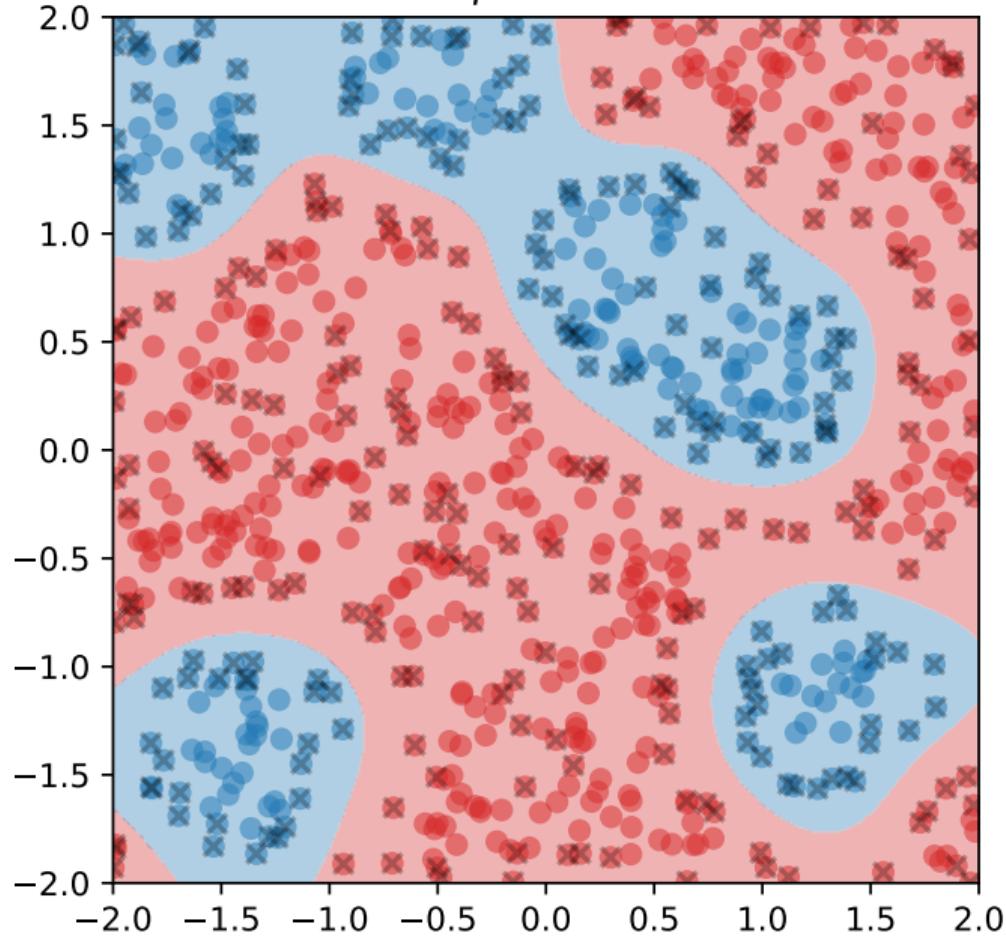
$$\gamma = 0.63$$



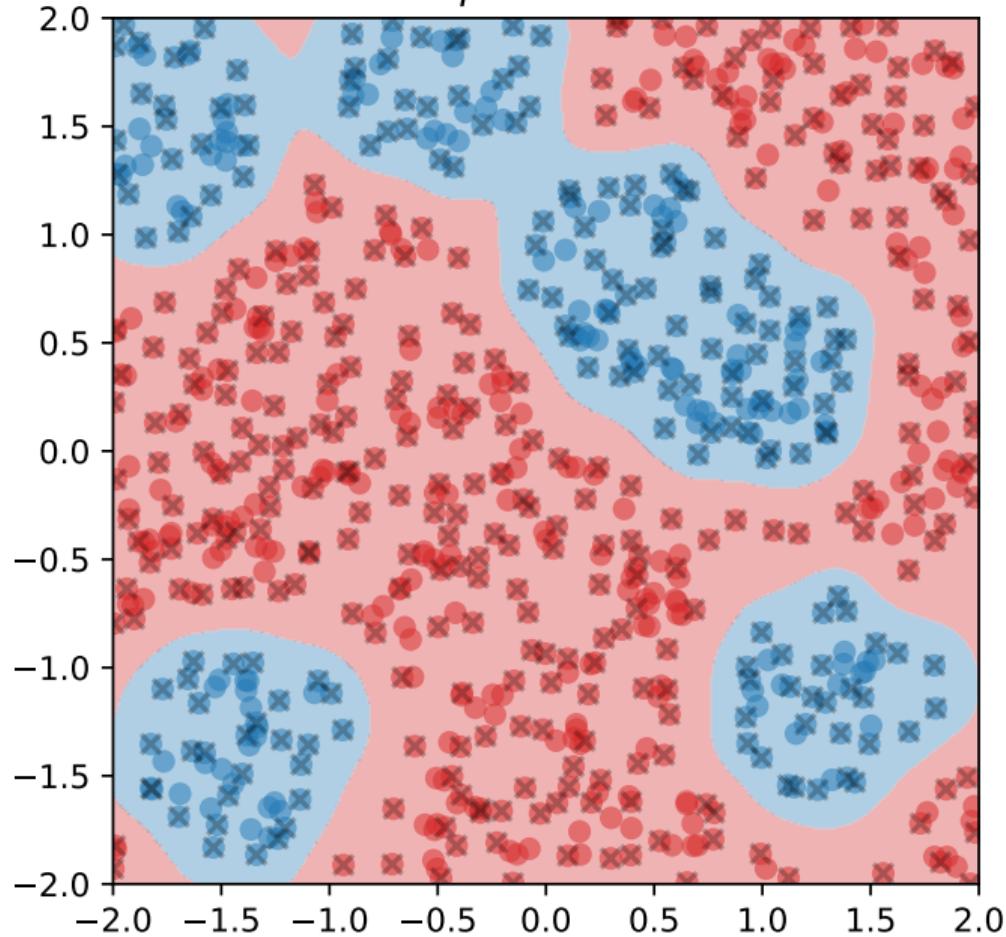
$$\gamma = 2.16$$



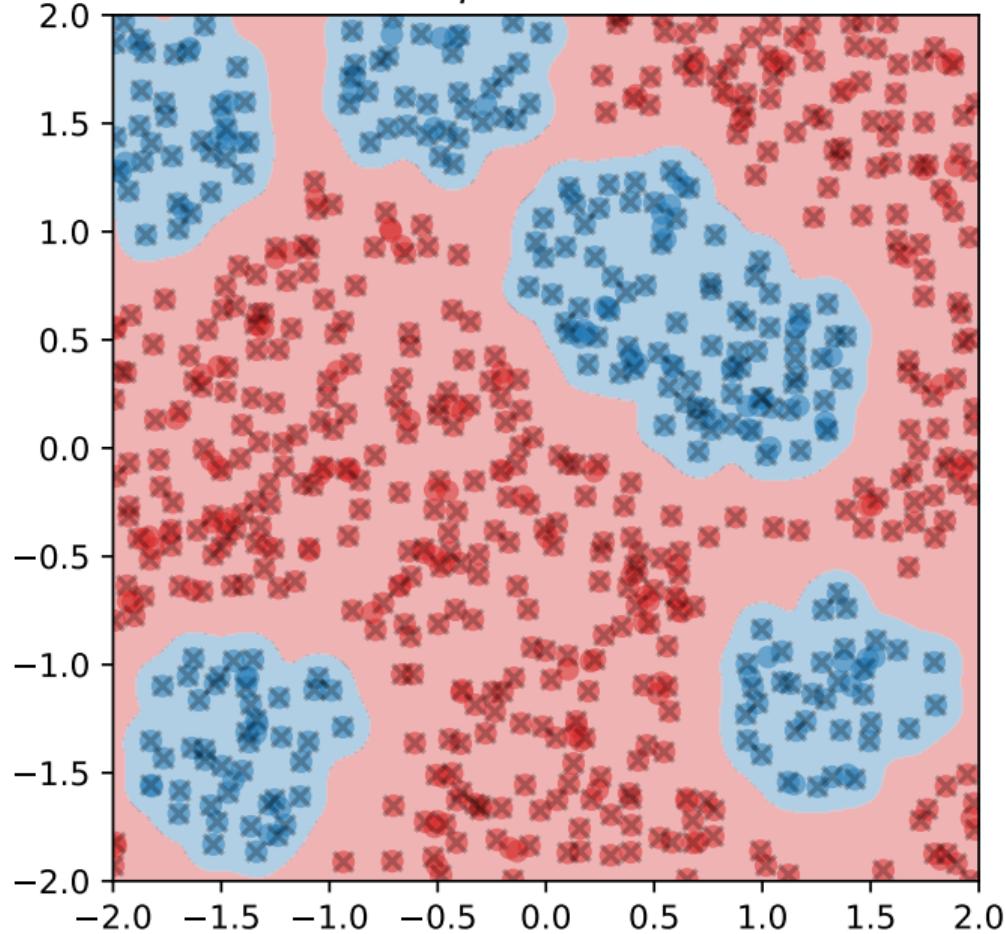
$$\gamma = 7.41$$



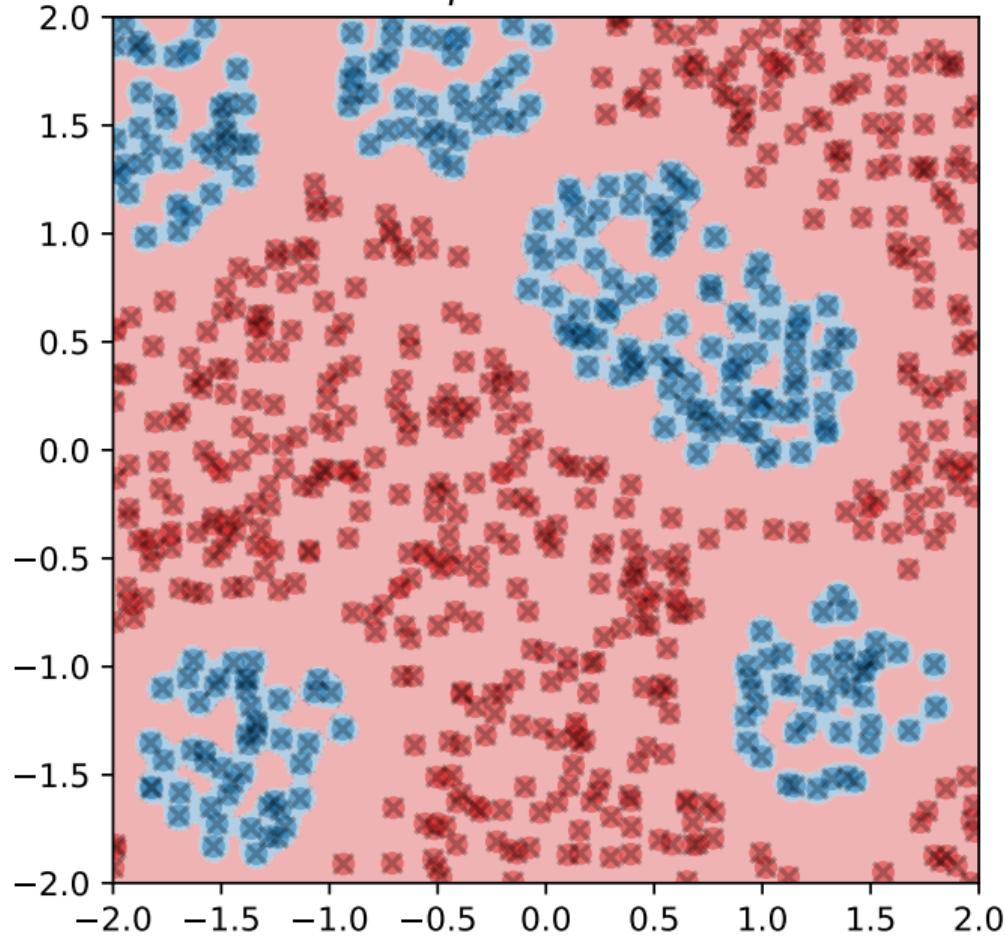
$$\gamma = 25.40$$



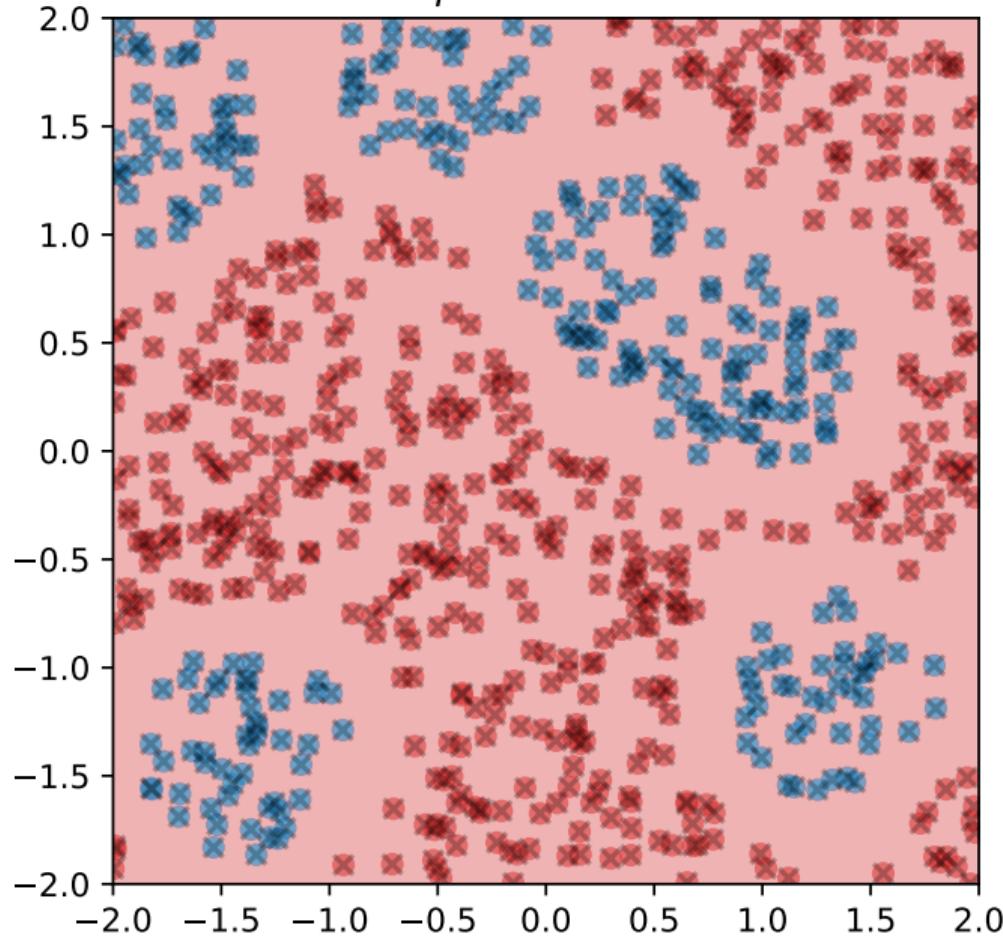
$$\gamma = 87.09$$



$$\gamma = 298.63$$



$$\gamma = 1024.00$$



# DSC 140A

*Probabilistic Modeling & Machine Learning*

Lecture 8 | Part 1

**Recap**

# Where have we been?

- ▶ We started with **nearest neighbor rules**.
  - ▶ Capable of learning **non-linear patterns**.
  - ▶ **Did not learn** feature importance.
  - ▶ Computationally-expensive.
    - ▶  $\Theta(n)$  memory and prediction time.

# Where have we been?

- ▶ In response, we developed **empirical risk minimization** (ERM).
- ▶ Step 1: choose a **hypothesis class**
- ▶ Step 2: choose a **loss function**
- ▶ Step 3: minimize **expected loss (empirical risk)**

# Where have we been?

- ▶ For first hypothesis class, we chose **linear models**.

$$H(\vec{x}) = w_0 + w_1x_1 + \dots + w_dx_d$$

- ▶  $\Theta(d)$  memory and prediction time.

# Where have we been?

- ▶ We saw different loss functions:
  - ▶ square, absolute, perceptron, hinge
- ▶ To train a linear model, pick loss  $L$  and minimize risk:

$$\arg \min_{\vec{w}} R(\vec{w}) = \arg \min_{\vec{w}} \frac{1}{n} \left[ \sum_{i=1}^n L(\vec{x}^{(i)}, y_i, \vec{w}) \right]$$

# Where have we been?

- ▶ We saw how to control the complexity of the learned model with **regularization**.

$$\arg \min_{\vec{w}} \tilde{R}(\vec{w}) = \arg \min_{\vec{w}} \frac{1}{n} \left[ \sum_{i=1}^n L(\vec{x}^{(i)}, y_i, \vec{w}) \right] + \rho(\vec{w})$$

# Where have we been?

- ▶ Some ERM problems have direct solutions.
  - ▶ Least squares, ridge regression.
- ▶ We saw most others do not, and must be solved iteratively with, e.g., **(stochastic) (sub)gradient descent**.

# Linear Model Zoo

Name	Loss Function	Regularizer	Direct Solution
Least Squares	square	-	yes
Ridge Regression	square	$\ \vec{w}\ ^2$	yes
LASSO	square	$\ \vec{w}\ _1$	no
Perceptron	perceptron	-	no
Soft-SVM	hinge	$\ \vec{w}\ ^2$	no

# Non-Linear Patterns

- ▶ We saw two ways of learning non-linear patterns with linear models:
- 1. Explicit mapping to feature space with **basis functions**.
  - ▶ E.g., learn  $H(\vec{x}) = w_0 + w_1\phi_1(\vec{x}) + \dots + w_k\phi_k(\vec{x})$
- 2. Implicit mapping with **kernel methods**.
- ▶ Each has downsides.

# Basis Functions

- ▶ **Idea:** choose a mapping  $\vec{\phi}$  that transforms data; train linear model in feature space.
- ▶ Downsides:
  - ▶ Must choose a good mapping. How?
  - ▶ Feature space is often very high-dimensional (**costly**).

# Kernels

- ▶ **Idea:** implicitly map to high-dimensional space with **kernel trick**.
- ▶ Downsides:
  - ▶ Since prediction is sum over training points,  $\Theta(n)$  in memory and time

# **Where are we now?**

- ▶ A new hypothesis class, beyond linear models.

# DSC 140A

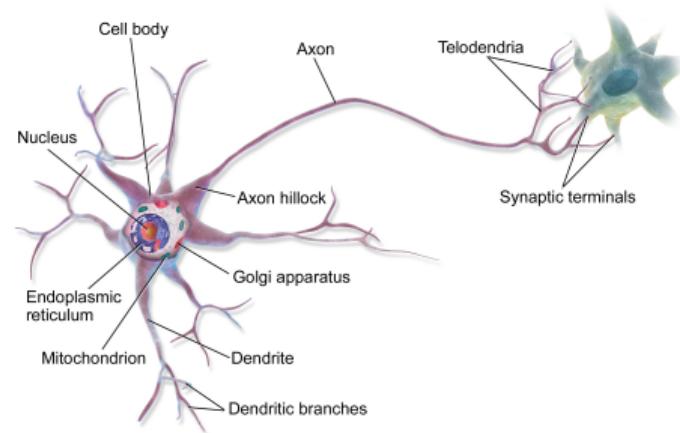
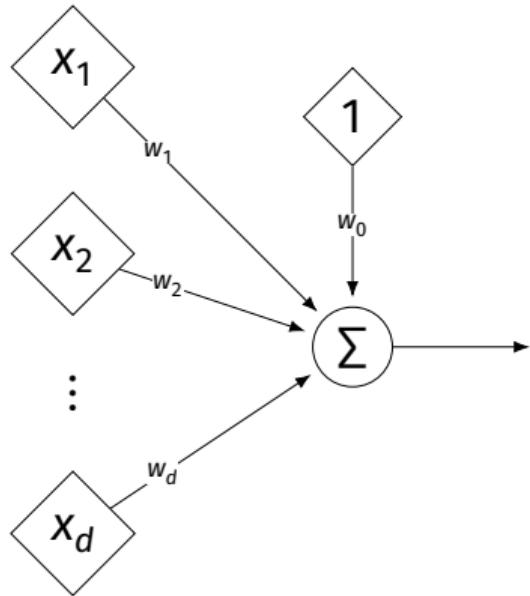
*Probabilistic Modeling & Machine Learning*

Lecture 8 | Part 2

**Neural Networks**

# Linear Models

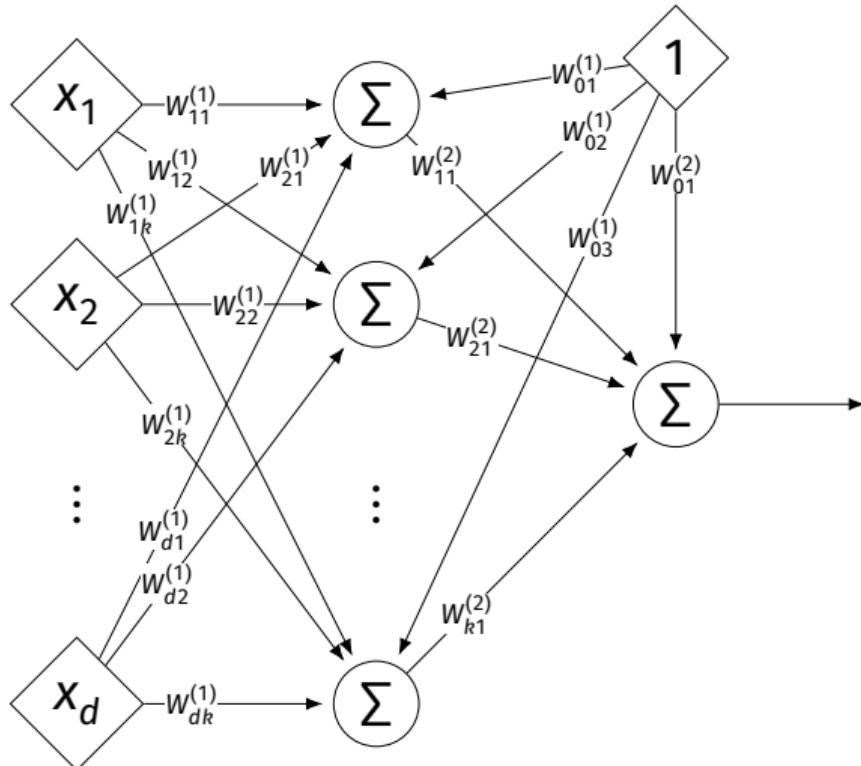
$$H(\vec{x}) = w_0 + w_1x_1 + \dots + w_dx_d$$



# Generalizing Linear Models

- ▶ The brain is a **network** of neurons.
- ▶ The output of a neuron is used as an input to another.
- ▶ **Idea:** chain together multiple “neurons” into a **neural network**.

# Neural Network<sup>1</sup> (One Hidden Layer)



<sup>1</sup>Specifically, a fully-connected, feed-forward neural network

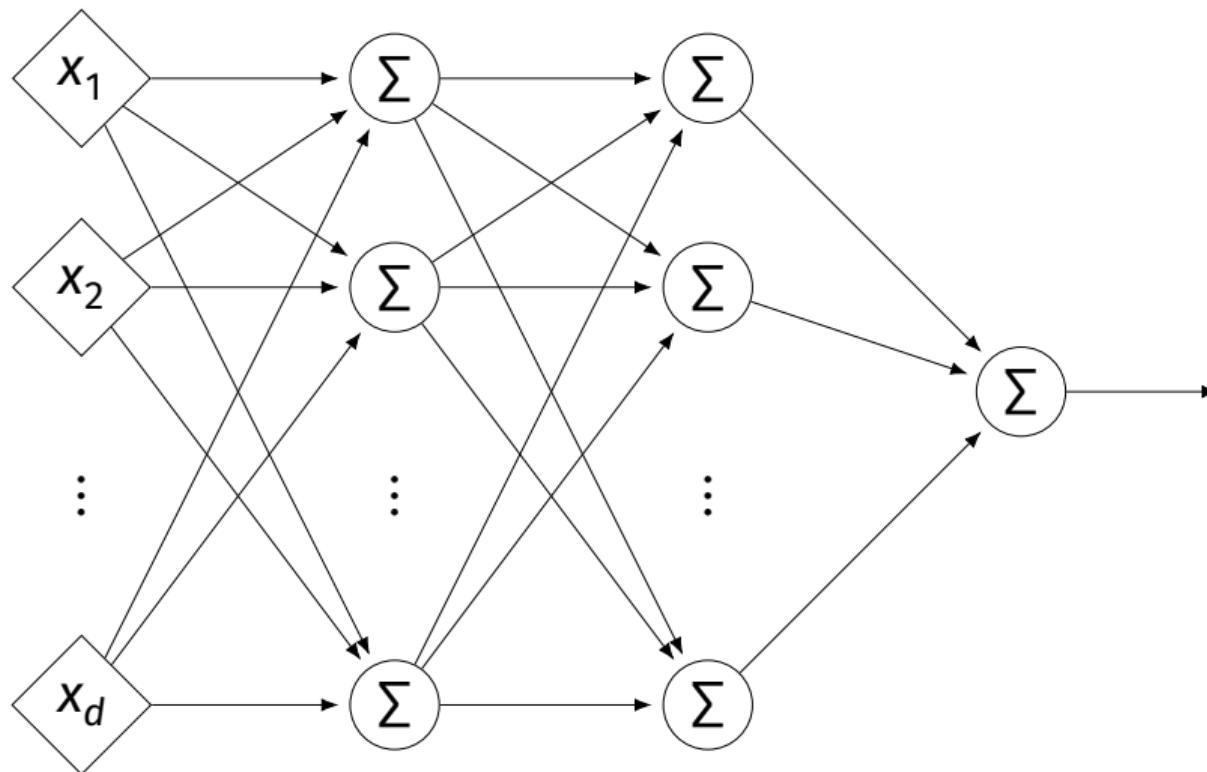
# Architecture

- ▶ Neurons are organized into **layers**.
  - ▶ **Input layer**, **output layer**, and **hidden layers**.
- ▶ Number of cells in input layer determined by dimensionality of input feature vectors.
- ▶ Number of cells in hidden layer(s) is determined by you.
- ▶ Output layer can have  $>1$  neuron.

# Architecture

- ▶ Can have more than one hidden layer.
  - ▶ A network is “**deep**” if it has  $>1$  hidden layer.
- ▶ Hidden layers can have different number of neurons.

# Neural Network (Two Hidden Layers)

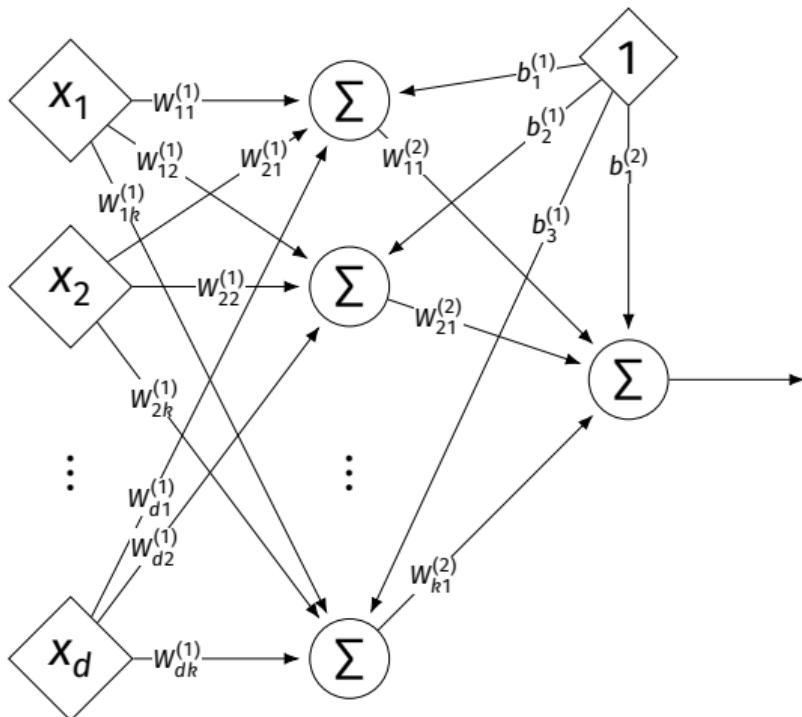


# Network Weights

- ▶ A neural network is a type of function.
- ▶ Like a linear model, a NN is **totally determined** by its weights.
- ▶ But there are often many more weights to learn!

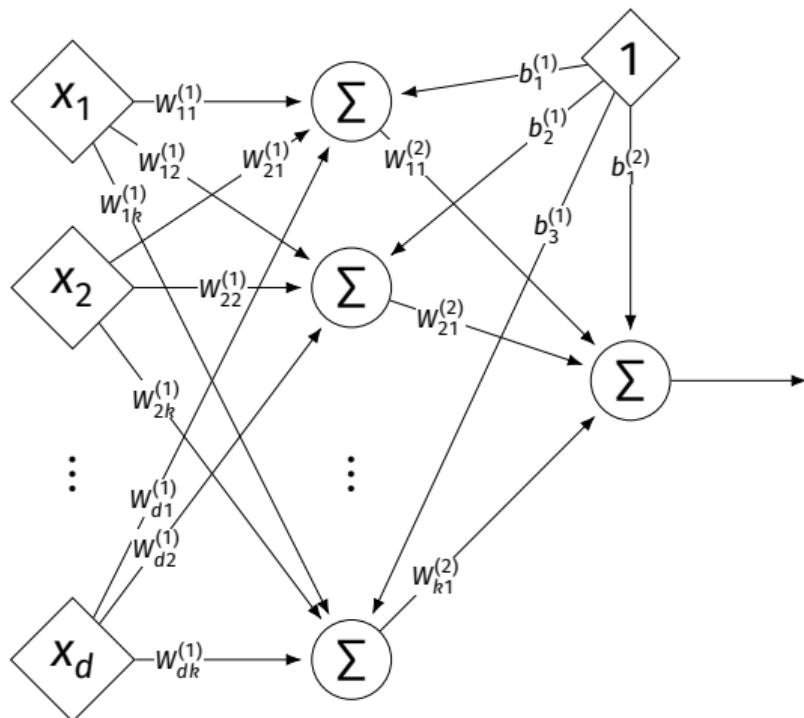
# Notation

- ▶ Input is layer #0.
- ▶  $W_{jk}^{(i)}$  denotes weight of connection between neuron  $j$  in layer  $(i - 1)$  and neuron  $k$  in layer  $i$
- ▶ Layer weights are 2-d arrays.



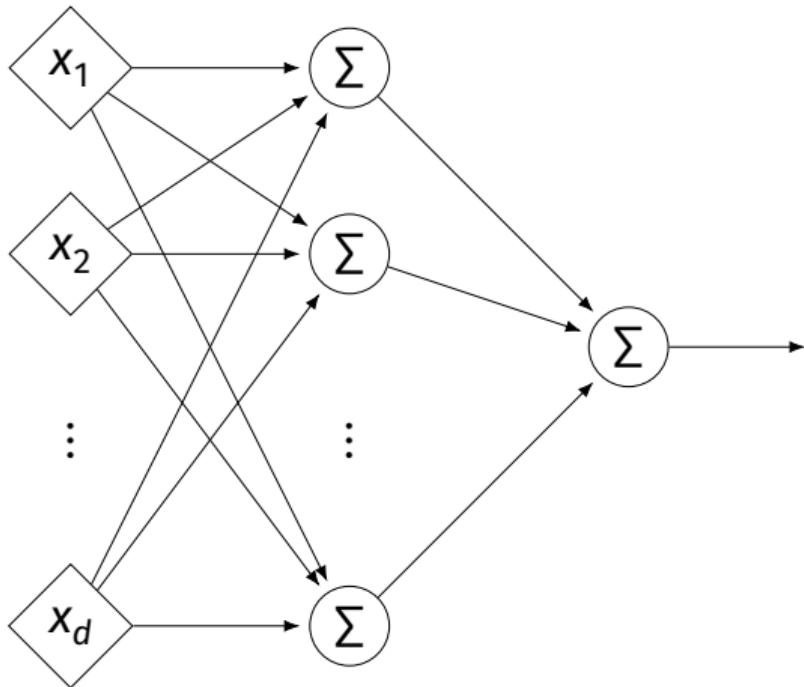
# Notation

- ▶ Each hidden/output neuron gets a “dummy” input of 1.
- ▶  $j$ th node in  $i$ th layer assigned a bias weight of  $b_j^{(i)}$
- ▶ Biases for layer are a vector:  $\vec{b}^{(i)}$

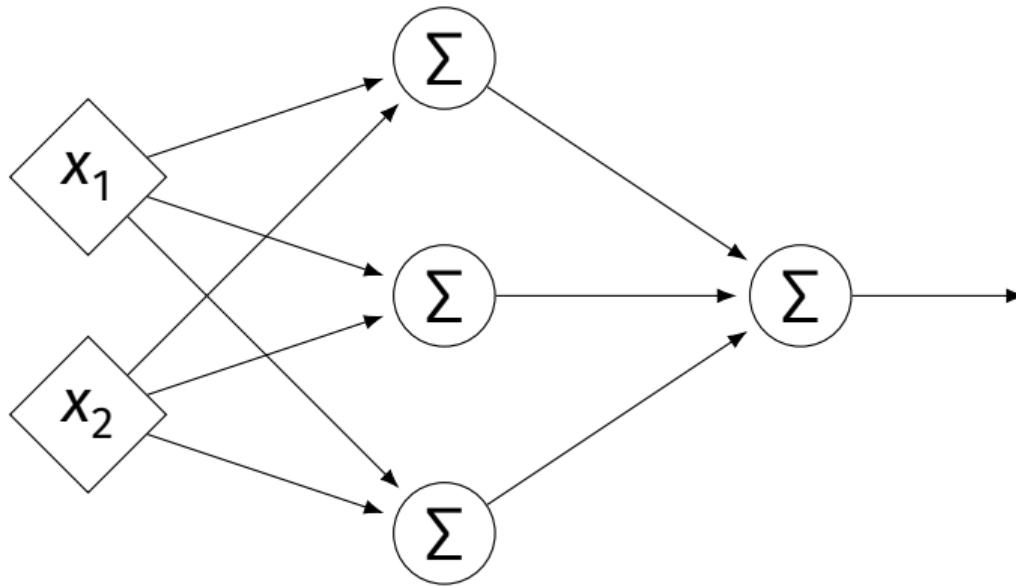


# Notation

- ▶ Typically, we will not draw the weights.
- ▶ We will not draw the dummy input, too, but it is there.



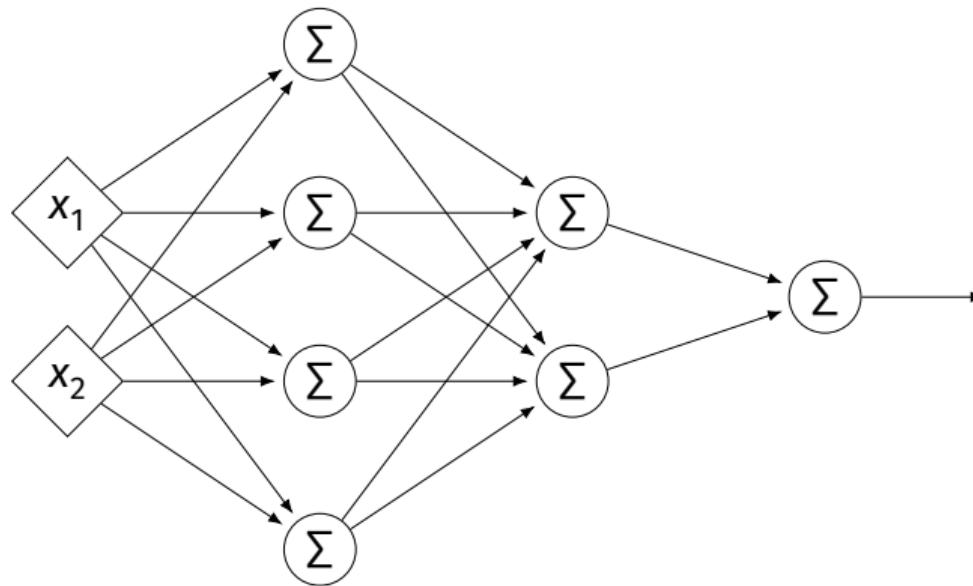
# Example



$$W^{(1)} = \begin{pmatrix} 2 & -1 & 0 \\ 4 & 5 & 2 \end{pmatrix} \quad W^{(2)} = \begin{pmatrix} 3 \\ 2 \\ -4 \end{pmatrix}$$

$$\vec{b}^{(1)} = (3, -2, -2)^T \quad \vec{b}^{(2)} = (-4)^T$$

# Example



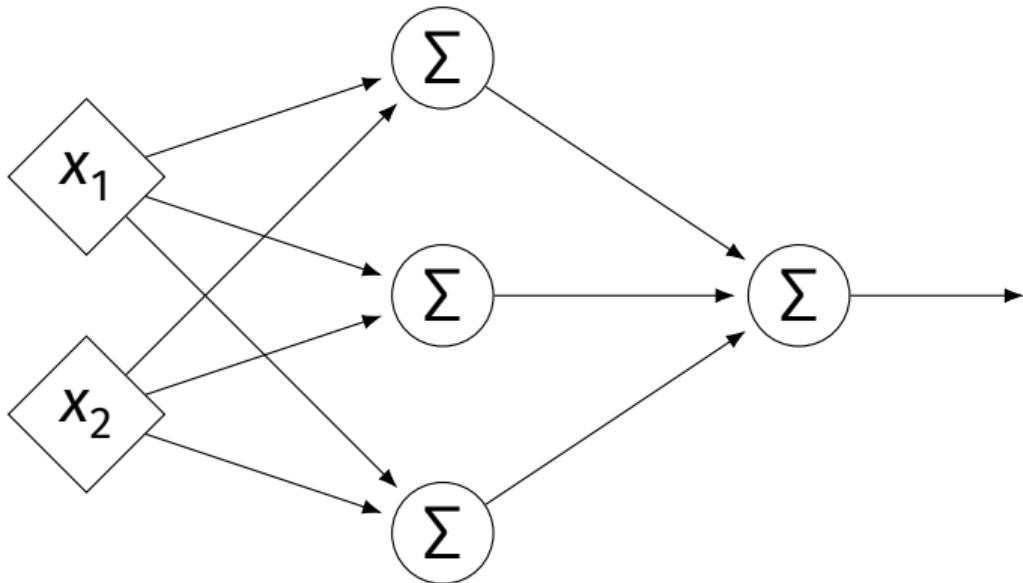
$$W^{(1)} = \begin{pmatrix} 2 & -1 & -3 & 0 \\ 4 & 5 & -7 & 2 \end{pmatrix} \quad W^{(2)} = \begin{pmatrix} 1 & 2 \\ -4 & 3 \\ -6 & -2 \\ 3 & 4 \end{pmatrix} \quad W^{(3)} = \begin{pmatrix} -1 & 5 \end{pmatrix}$$

$$\vec{b}^{(1)} = (3, 6, -2, -2)^T \quad \vec{b}^{(2)} = (-4, 0)^T \quad \vec{b}^{(3)} = (1)^T$$

# Evaluation

- ▶ These are “**fully-connected, feed-forward**” networks with one output.
- ▶ They are functions  $H(\vec{x}) : \mathbb{R}^d \rightarrow \mathbb{R}^1$
- ▶ To evaluate  $H(\vec{x})$ , compute result of layer  $i$ , use as inputs for layer  $i + 1$ .

# Example



►  $\vec{x} = (3, -1)^T$

►  $z_1^{(1)} =$

►  $z_2^{(1)} =$

►  $z_3^{(1)} =$

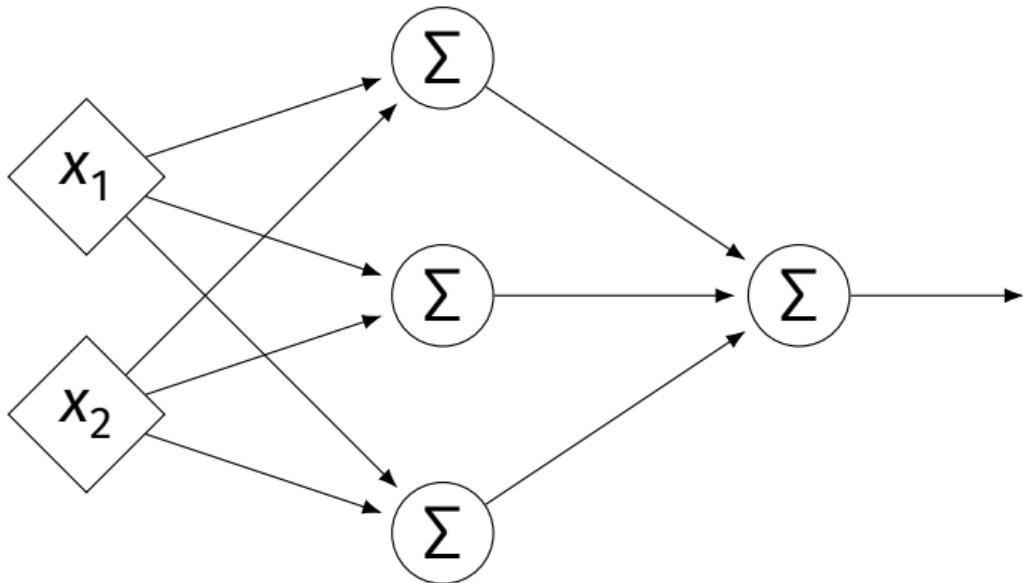
►  $z_1^{(2)} =$

$$W^{(1)} = \begin{pmatrix} 2 & -1 & 0 \\ 4 & 5 & 2 \end{pmatrix} \quad W^{(2)} = \begin{pmatrix} 3 \\ 2 \\ -4 \end{pmatrix} \quad \vec{b}^{(1)} = (3, -2, -2)^T \quad \vec{b}^{(2)} = (-4)^T$$

# Evaluation as Matrix Multiplication

- ▶ Let  $z_j^{(i)}$  be the output of node  $j$  in layer  $i$ .
- ▶ Make a vector of these outputs:  $\vec{z}^{(i)} = (z_1^{(i)}, z_2^{(i)}, \dots)^T$
- ▶ Observe that  $\vec{z}^{(i)} = [W^{(i)}]^T \vec{z}^{(i-1)} + \vec{b}^{(i)}$

# Example



►  $\vec{x} = (3, -1)^T$

►  $z_1^{(1)} =$

►  $z_2^{(1)} =$

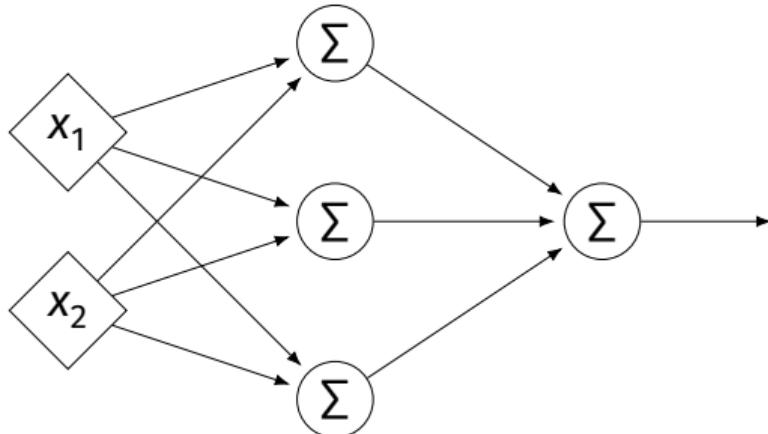
►  $z_3^{(1)} =$

►  $z_1^{(2)} =$

$$W^{(1)} = \begin{pmatrix} 2 & -1 & 0 \\ 4 & 5 & 2 \end{pmatrix} \quad W^{(2)} = \begin{pmatrix} 3 \\ 2 \\ -4 \end{pmatrix} \quad \vec{b}^{(1)} = (3, -2, -2)^T \quad \vec{b}^{(2)} = (-4)^T$$

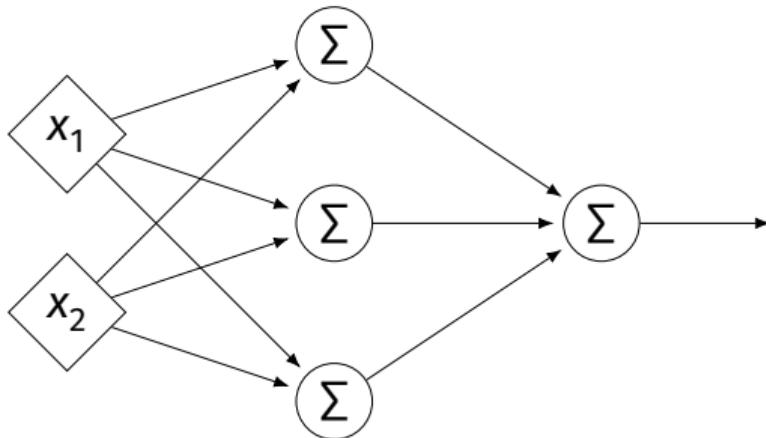
# Each Layer is a Function

- ▶ We can think of each layer as a function mapping a vector to a vector.
- ▶  $H^{(1)}(\vec{z}) = [W^{(1)}]^T \vec{z} + \vec{b}^{(1)}$ 
  - ▶  $H^{(1)} : \mathbb{R}^2 \rightarrow \mathbb{R}^3$
- ▶  $H^{(2)}(\vec{z}) = [W^{(2)}]^T \vec{z} + \vec{b}^{(2)}$ 
  - ▶  $H^{(2)} : \mathbb{R}^3 \rightarrow \mathbb{R}^1$



# NNs as Function Composition

- The full NN is a composition of layer functions.



$$H(\vec{x}) = H^{(2)}(H^{(1)}(\vec{x})) = \underbrace{\left[ W^{(2)} \right]^T \left( \left[ W^{(1)} \right]^T \vec{x} + \vec{b}^{(1)} \right) + \vec{b}^{(2)}}_{\vec{z}^{(1)}}$$

# NNs as Function Composition

- ▶ In general, if there  $k$  hidden layers:

$$H(\vec{x}) = H^{(k+1)} \left( \dots H^{(3)} \left( H^{(2)} \left( H^{(1)}(\vec{x}) \right) \right) \dots \right)$$

## Exercise

Show that:

$$H(\vec{x}) = [W^{(2)}]^T \left( [W^{(1)}]^T \vec{x} + \vec{b}^{(1)} \right) + \vec{b}^{(2)} = \vec{w} \cdot \text{Aug}(\vec{x})$$

for some appropriately-defined vector  $\vec{w}$ .

# Result

- ▶ The composition of linear functions is again a linear function.
- ▶ The NNs we have seen so far are all equivalent to linear models!
- ▶ For NNs to be more useful, we will need to add **non-linearity**.

# Activations

- ▶ So far, the output of a neuron has been a linear function of its inputs:

$$w_0 + w_1 x_1 + w_2 x_2 + \dots$$

- ▶ Can be arbitrarily large or small.
- ▶ But real neurons are **activated** non-linearly.
  - ▶ E.g., saturation.

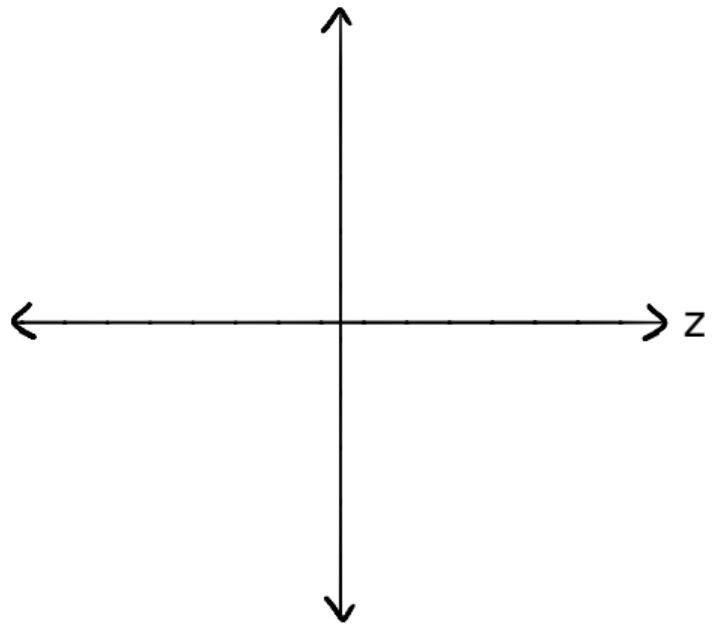
# Idea

- ▶ To add nonlinearity, we will apply a non-linear **activation function**  $g$  to the output of **each** hidden neuron (and sometimes the output neuron).

# Linear Activation

- ▶ The **linear** activation is what we've been using.

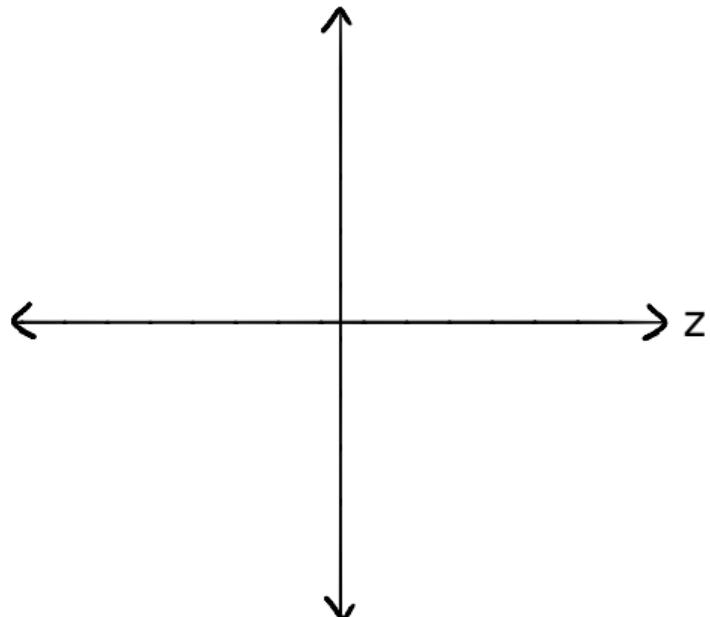
$$\sigma(z) = z$$



# Sigmoid Activation

- The **sigmoid** models saturation in many natural processes.

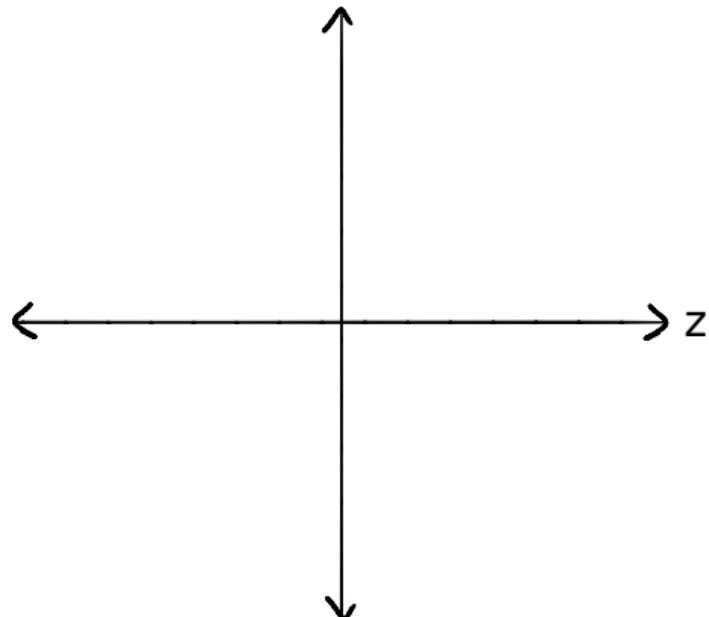
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



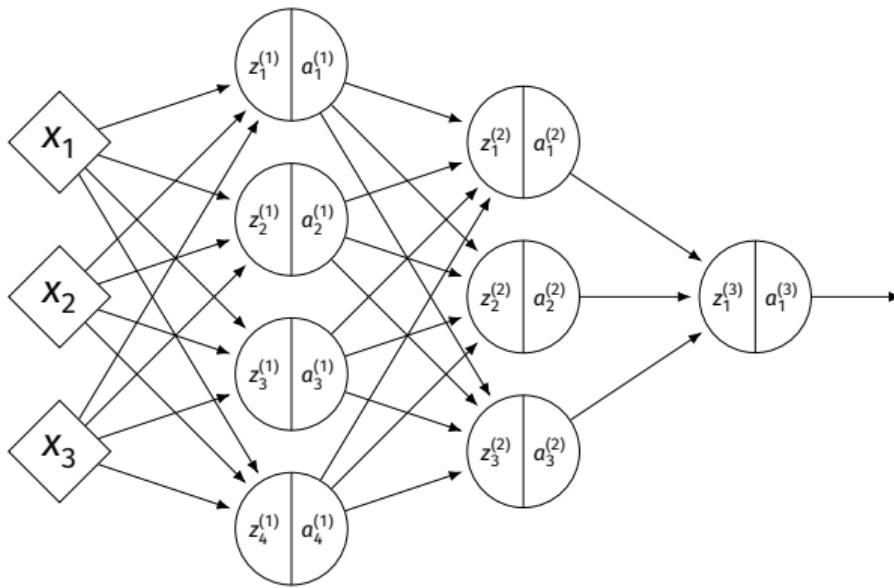
# ReLU Activation

- The **Rectified Linear Unit (ReLU)** tends to work better in practice.

$$g(z) = \max\{0, z\}$$

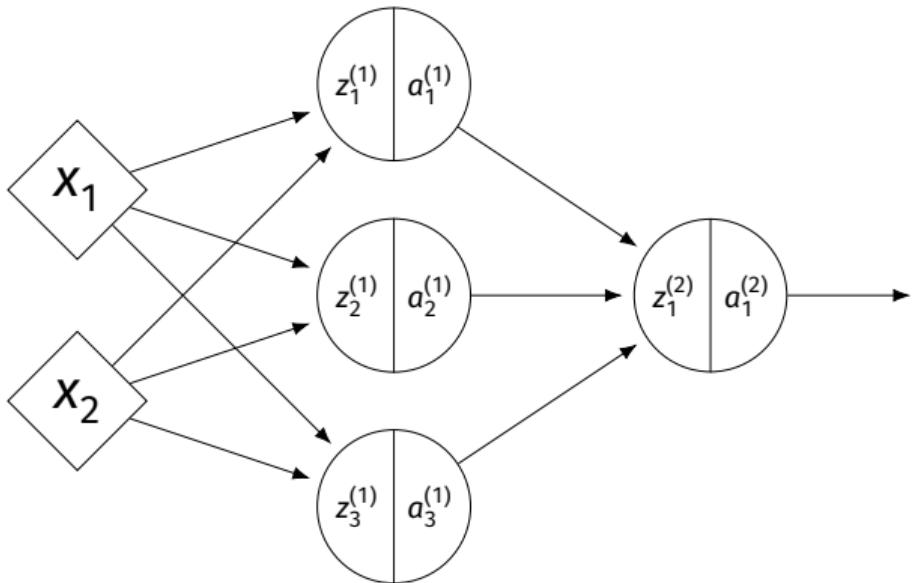


# Notation



- ▶  $z_j^{(i)}$  is the linear activation before  $g$  is applied.
- ▶  $a_j^{(i)} = g(z_j^{(i)})$  is the actual output of the neuron.

# Example



- ▶  $g = \text{ReLU}$
  - ▶ Linear output
  - ▶  $\vec{x} = (3, -1)^T$
  - ▶  $z_1^{(1)} =$
  - ▶  $a_1^{(1)} =$
  - ▶  $z_2^{(1)} =$
  - ▶  $a_2^{(1)} =$
  - ▶  $z_3^{(1)} =$
  - ▶  $a_3^{(1)} =$
  - ▶  $z_1^{(2)} =$
- $$W^{(1)} = \begin{pmatrix} 2 & -1 & 0 \\ 4 & 5 & 2 \end{pmatrix} \quad W^{(2)} = \begin{pmatrix} 3 \\ 2 \\ -4 \end{pmatrix} \quad \vec{b}^{(1)} = (3, -2, -2)^T \quad \vec{b}^{(2)} = (-4)^T$$

# Output Activations

- ▶ The activation of the output neuron(s) can be different than the activation of the hidden neurons.
- ▶ In classification, **sigmoid** activation makes sense.
- ▶ In regression, **linear** activation makes sense.

## Main Idea

A neural network with linear activations is a linear model. If non-linear activations are used, the model is made non-linear.

# DSC 140A

*Probabilistic Modeling & Machine Learning*

Lecture 8 | Part 3

**Training Neural Networks**

# Training Neural Networks

- ▶ As with linear models, we can use ERM.
- ▶ Step 1: choose a **hypothesis class**
  - ▶ Choose a neural network architecture (depth / width), activation.
- ▶ Step 2: choose a **loss function**
- ▶ Step 3: minimize **expected loss (empirical risk)**

# Parameter Vectors

- ▶ A neural network is totally determined by its parameters.
- ▶ We can package all of the parameter arrays  $W^{(1)}, W^{(2)}, \dots$ , as well as the biases  $\vec{b}^{(1)}, \vec{b}^{(2)}, \dots$  into a single **parameter vector**  $\vec{w}$ .

# Square Loss for NNs

- ▶ The square loss can be used to train a neural network.

$$L(\vec{x}, y, \vec{w}) = (H(\vec{x}; \vec{w}) - y)^2$$

# Cross-Entropy Loss for NNs

- ▶ When using sigmoid output activation for classification, we often use the **cross-entropy**.
- ▶ Assume labels are 1 and 0. Then:

$$L(\vec{x}, y, \vec{w}) = - \begin{cases} \log H(\vec{x}; \vec{w}), & \text{if } y = 1 \\ \log[1 - H(\vec{x}; \vec{w})], & \text{if } y = 0 \end{cases}$$

# Minimizing Risk

- ▶ Having chosen a loss, we next minimize empirical risk:<sup>2</sup>

$$\arg \min_{\vec{w}} R(\vec{w}) = \arg \min_{\vec{w}} \frac{1}{n} \left[ \sum_{i=1}^n L(\vec{x}^{(i)}, y_i, \vec{w}) \right]$$

- ▶ Except for special cases, there is no direct solution for the minimizer.
- ▶ Use iterative methods: e.g., SGD.

---

<sup>2</sup>Can also add a regularizer.

# Gradient Descent for NNs

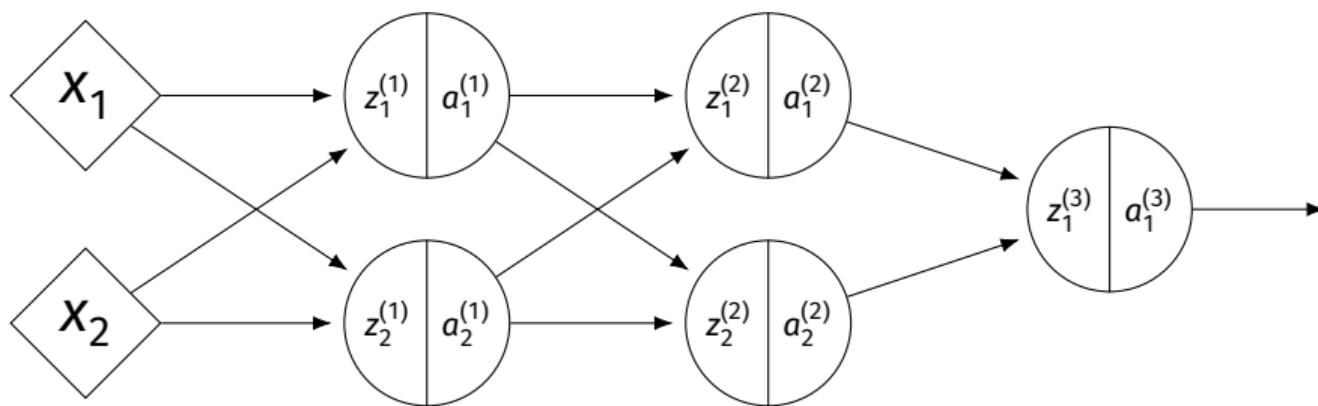
- ▶ To perform SGD, we must compute  $\nabla R$ .
- ▶ E.g., using square loss:

$$\nabla R = \frac{2}{n} \sum_{i=1}^n (H(\vec{x}^{(i)}; \vec{w}) - y_i) \nabla_{\vec{w}} H(\vec{x}^{(i)}; \vec{w})$$

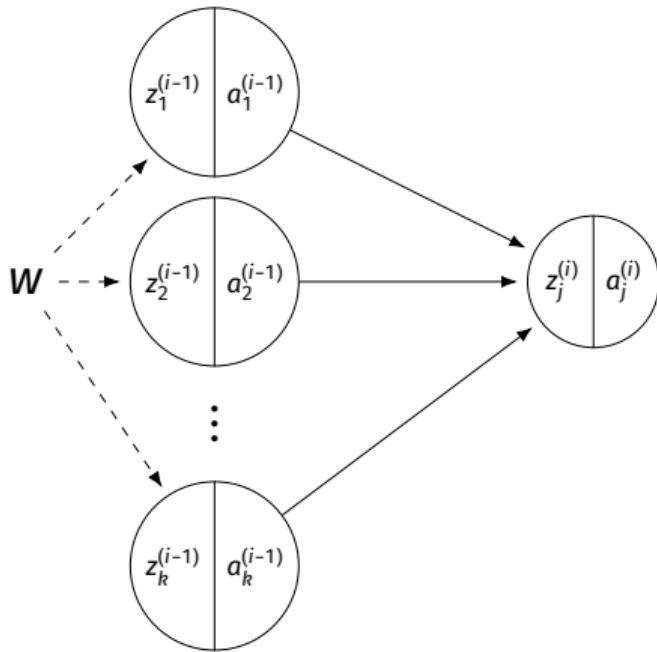
- ▶ We must compute  $\nabla_{\vec{w}} H$ ; the gradient of the network with respect to the parameter vector,  $\vec{w}$ .

# Gradient of a Network

- ▶ The gradient of  $H$  w.r.t., parameter vector  $\vec{w}$  can be computed using the chain rule.



# Example



$$z_j^{(i)} = b_j^{(i)} + \sum_{\ell=1}^k W_{\ell j}^{(i)} a_\ell^{(i-1)}$$

$$\frac{\partial a_j^{(i)}}{\partial w} = \frac{\partial a_j^{(i)}}{\partial z_j^{(i)}} \frac{\partial z_j^{(i)}}{\partial w}$$

$$= \frac{\partial a_j^{(i)}}{\partial z_j^{(i)}} \left[ \sum_{\ell=1}^k \frac{\partial z_j}{\partial a_\ell^{(i-1)}} \right]$$

$$= \frac{\partial a_j^{(i)}}{\partial z_j^{(i)}} \left[ \sum_{\ell=1}^k W_{\ell j}^{(i)} \frac{\partial a_\ell^{(i-1)}}{\partial w} \right]$$

# Gradient of a Network

- We found:

$$\frac{\partial a_j^{(i)}}{\partial w} = \frac{\partial a_j^{(i)}}{\partial z_j^{(i)}} \left[ \sum_{\ell=1}^k W_{\ell j}^{(i)} \frac{\partial a_\ell^{(i-1)}}{\partial w} \right]$$

- Recalling that  $a_j^{(i)} = g(z_j^{(i)})$ , we can simplify a little:

$$\frac{\partial a_j^{(i)}}{\partial w} = g'(z_j^{(i)}) \left[ \sum_{\ell=1}^k W_{\ell j}^{(i)} \frac{\partial a_\ell^{(i-1)}}{\partial w} \right]$$

# Gradient of a Network

$$\frac{\partial a_j^{(i)}}{\partial w} = g'(z_j^{(i)}) \left[ \sum_{\ell=1}^k W_{\ell j}^{(i)} \frac{\partial a_\ell^{(i-1)}}{\partial w} \right]$$

- ▶ We can apply the above formula recursively to compute  $\partial H / \partial w$  for any parameter  $w$ .
- ▶ Efficient algorithm: **backpropagation**.
- ▶ Outside of the scope of DSC 140A (take DSC 140B).

# Implications

$$\frac{\partial a_j^{(i)}}{\partial w} = g'(z_j^{(i)}) \left[ \sum_{\ell=1}^k W_{\ell j}^{(i)} \frac{\partial a_\ell^{(i-1)}}{\partial w} \right]$$

- ▶ **Vanishing gradients**: the deeper the network, the “weaker” the gradients; harder to train.
- ▶ Prefer activations with stronger gradients.
  - ▶ ReLU > sigmoid

# Convex Risk?

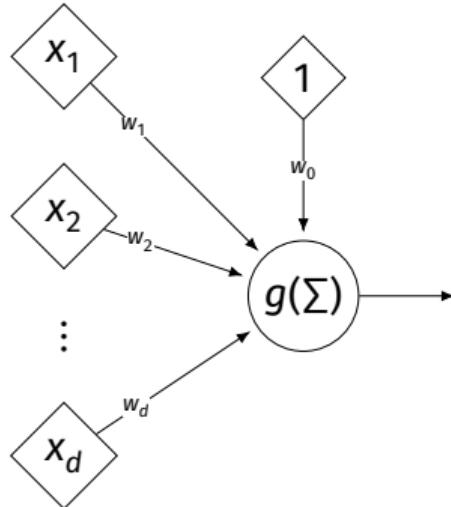
- ▶ When training linear models with convex losses, the risk was a convex function of  $\vec{w}$ .
  - ▶ Because it is composed of convex functions.
- ▶ E.g., with the square loss:

$$\frac{1}{n} \sum_{i=1}^n (\vec{w} \cdot \vec{x}^{(i)} - y_i)^2$$

- ▶ We like this because it was **easy** to optimize.

# Convex Risk?

- ▶ What about with NNs? Is the risk still convex?
- ▶ Consider a very simple network with zero hidden layers, representing  $H(\vec{x}; \vec{w}) = g(\vec{w} \cdot \text{Aug}(\vec{x}))$ .



# Convex Risk?

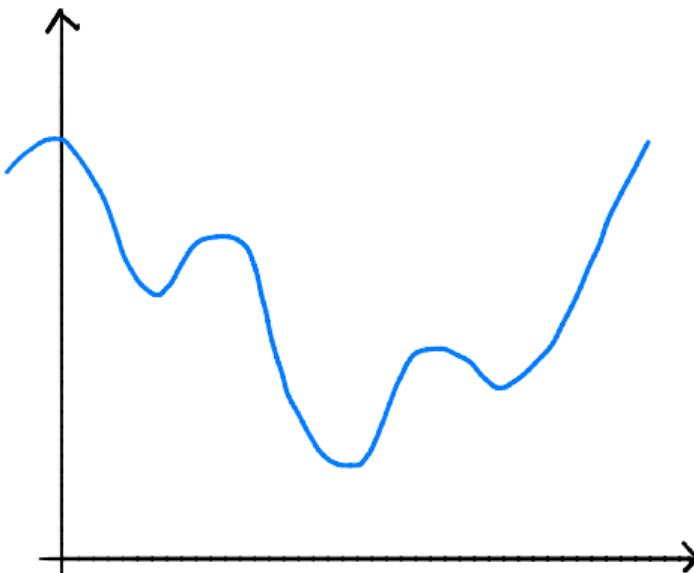
- ▶ The risk w.r.t. the square loss is:

$$\frac{1}{n} \sum_{i=1}^n (g(\vec{w} \cdot \text{Aug}(\vec{x}^{(i)})) - y_i)^2$$

- ▶ If  $g$  is non-convex, the risk is in general no longer convex!

# Non-Convexity

- ▶ In general, NNs with non-linear activations have (highly) **non-convex** risk functions.



# Training NNs

- ▶ SGD is still used to train neural networks, even though the risk is **non-convex**.
- ▶ May get stuck in local minima; depends heavily on starting position.

# Downsides of NNs

- ▶ NNs tend to be harder to train than linear models.
- ▶ How do we choose the architecture?
- ▶ Engineering challenges with large networks.

# DSC 1410A

*Probabilistic Modeling & Machine Learning*

Lecture 8 | Part 4

**Demo**

# Feature Map

- ▶ We have seen how to fit non-linear patterns with linear models via **basis functions** (i.e., a feature map).

$$H(\vec{x}) = w_0 + w_1\phi_1(\vec{x}) + \dots + w_k\phi_k(\vec{x})$$

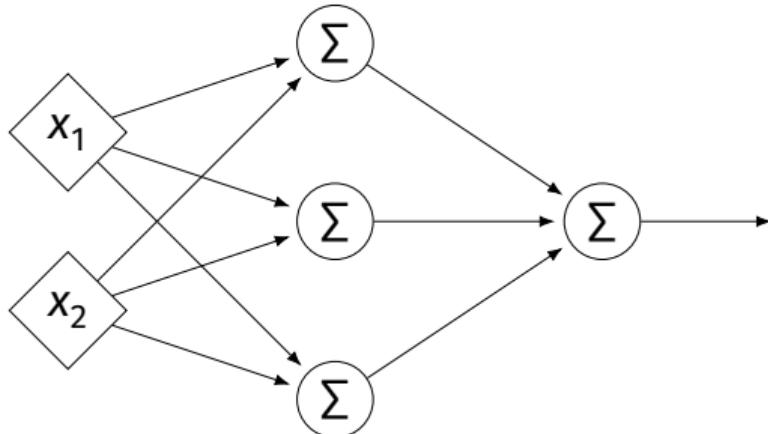
- ▶ These basis functions are fixed **before** learning.
- ▶ **Downside:** we have to choose  $\vec{\phi}$  somehow.

# Learning a Feature Map

- ▶ **Interpretation:** The hidden layers of a neural network **learn** a feature map.

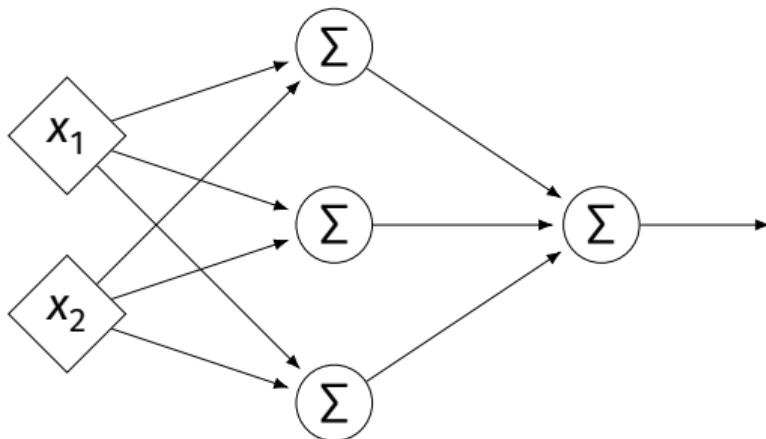
# Each Layer is a Function

- ▶ We can think of each layer as a function mapping a vector to a vector.
- ▶  $H^{(1)}(\vec{z}) = [W^{(1)}]^T \vec{z} + \vec{b}^{(1)}$ 
  - ▶  $H^{(1)} : \mathbb{R}^2 \rightarrow \mathbb{R}^3$
- ▶  $H^{(2)}(\vec{z}) = [W^{(2)}]^T \vec{z} + \vec{b}^{(2)}$ 
  - ▶  $H^{(2)} : \mathbb{R}^3 \rightarrow \mathbb{R}^1$



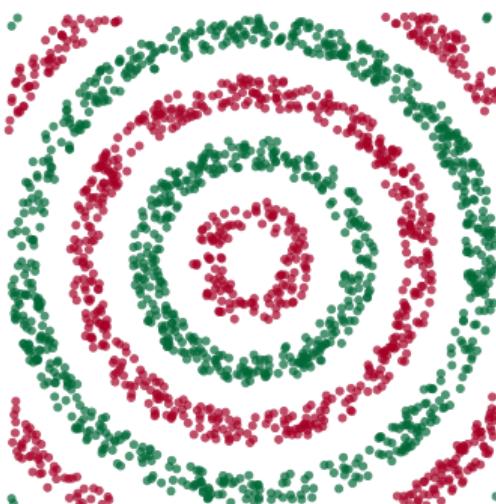
# Each Layer is a Function

- ▶ The hidden layer performs a feature map from  $\mathbb{R}^2$  to  $\mathbb{R}^3$ .
- ▶ The output layer makes a prediction in  $\mathbb{R}^3$ .
- ▶ **Intuition:** The feature map is learned so as to make the output layer's job “easier”.



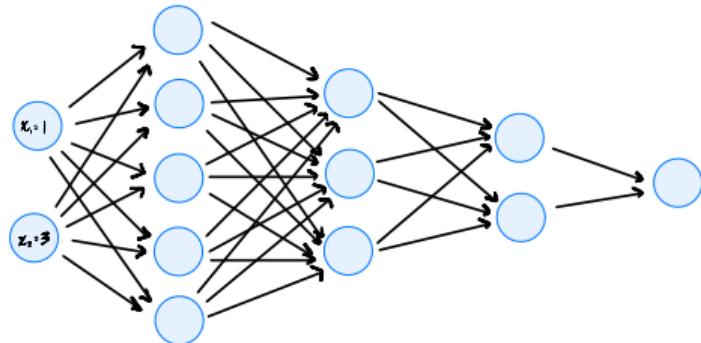
# Demo

- ▶ Train a deep network to classify the data below.
- ▶ Hidden layers will learn a new feature map that makes the data linearly separable.

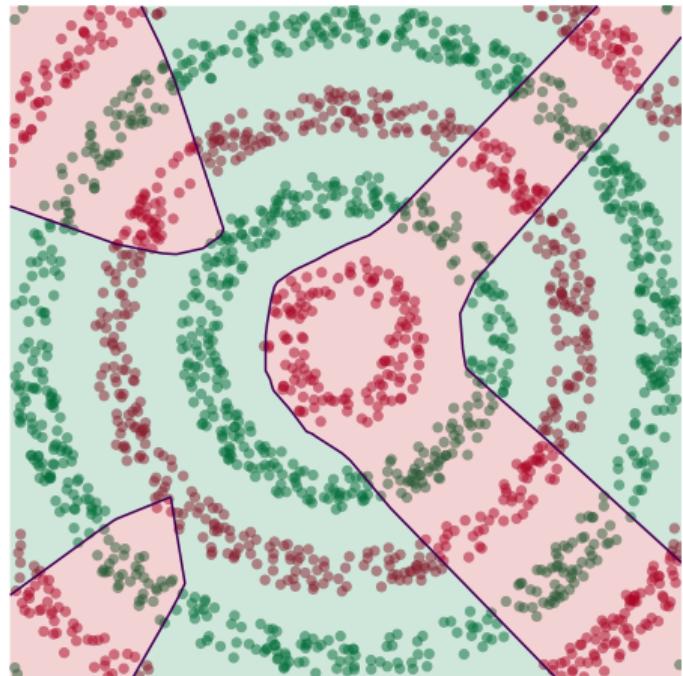


# Demo

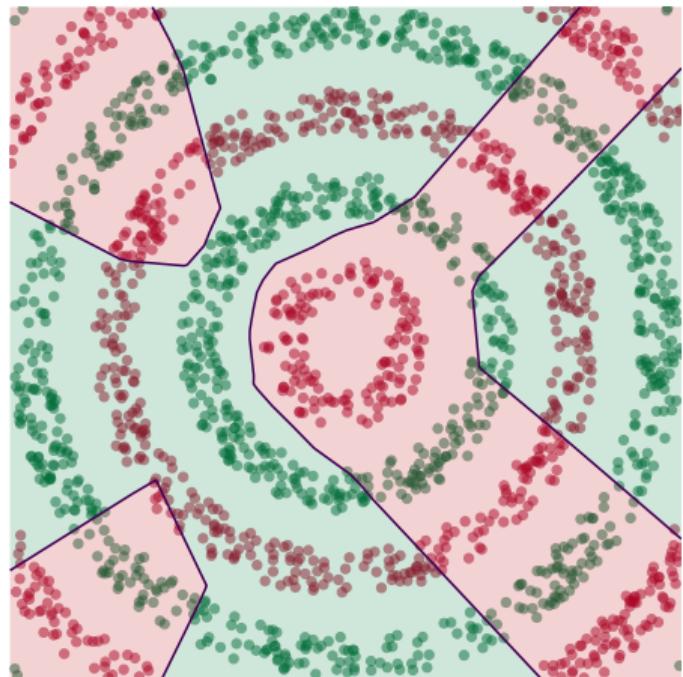
- ▶ We'll use three hidden layers, with last having two neurons.
- ▶ We can see this new representation!
- ▶ Plug in  $\vec{x}$  and see activations of last hidden layer.



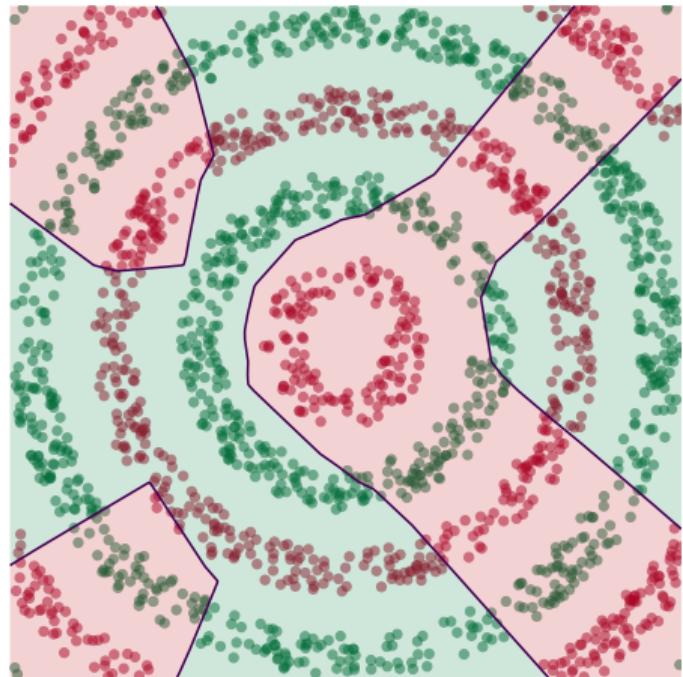
# Learning a New Representation



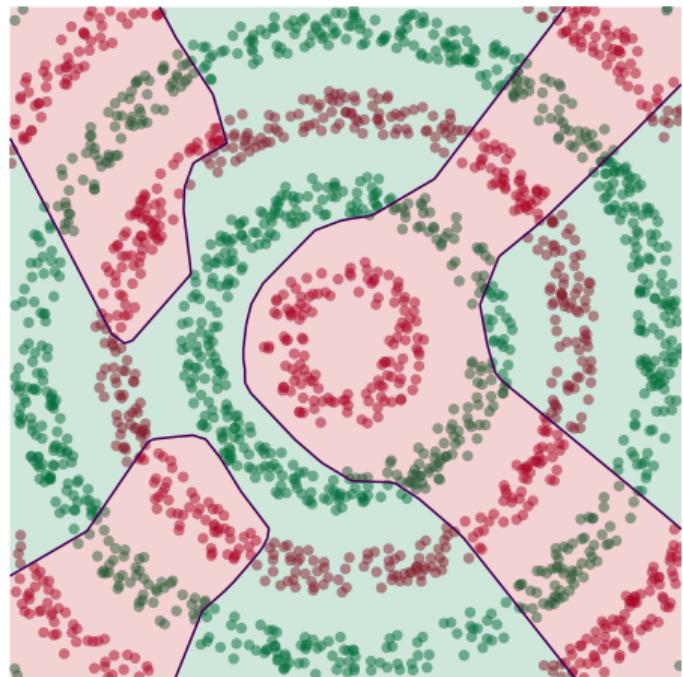
# Learning a New Representation



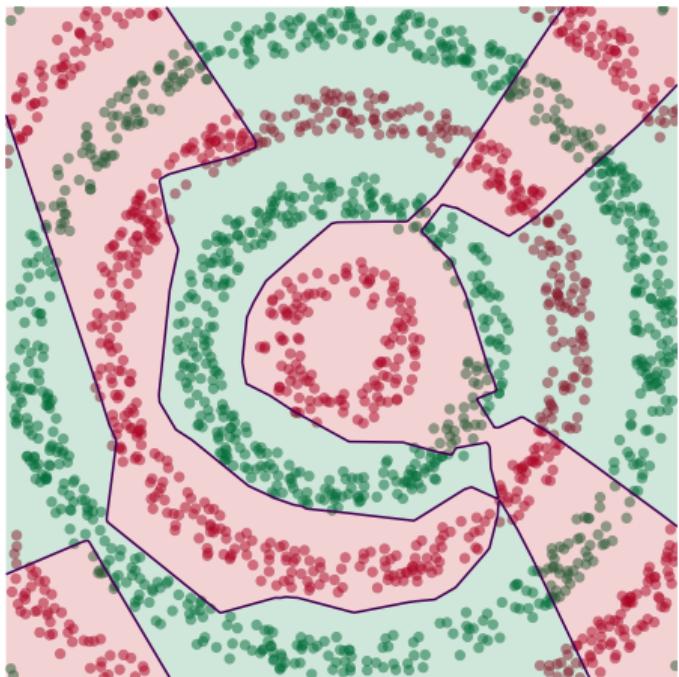
# Learning a New Representation



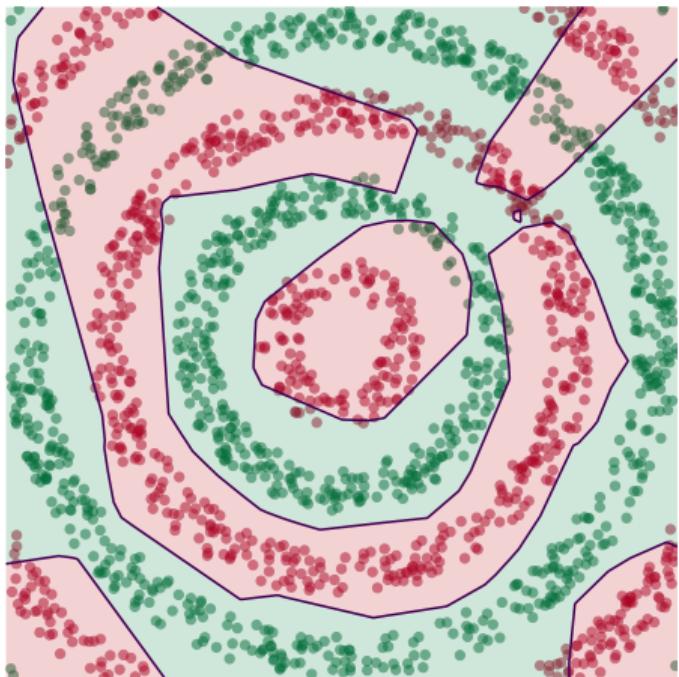
# Learning a New Representation



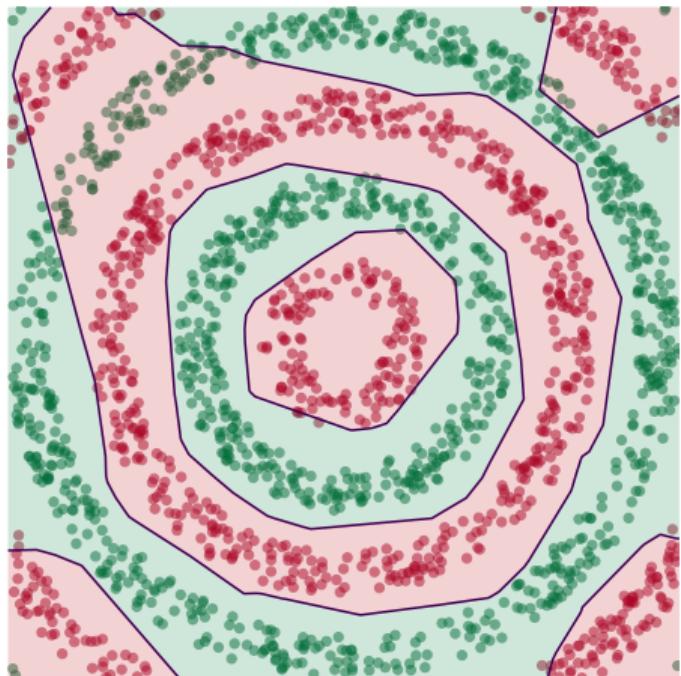
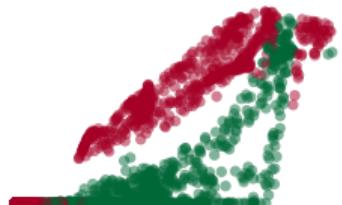
# Learning a New Representation



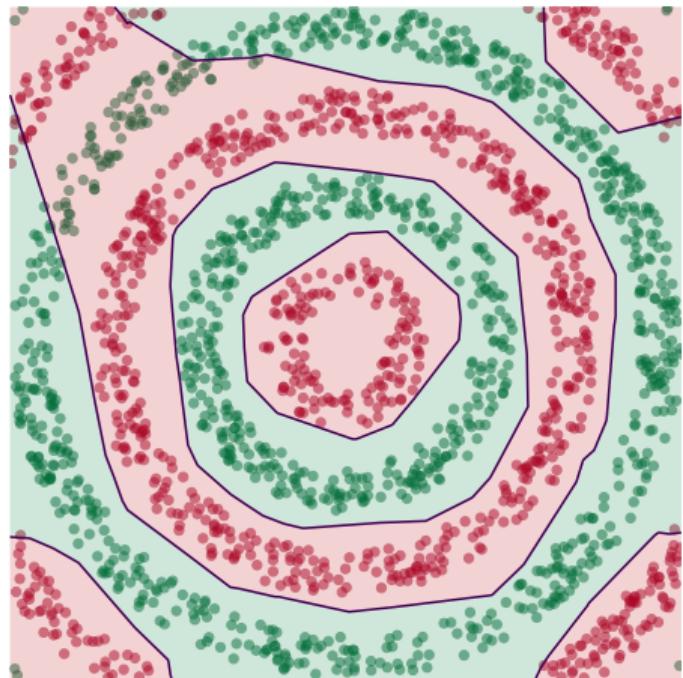
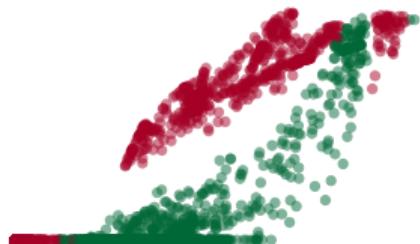
# Learning a New Representation



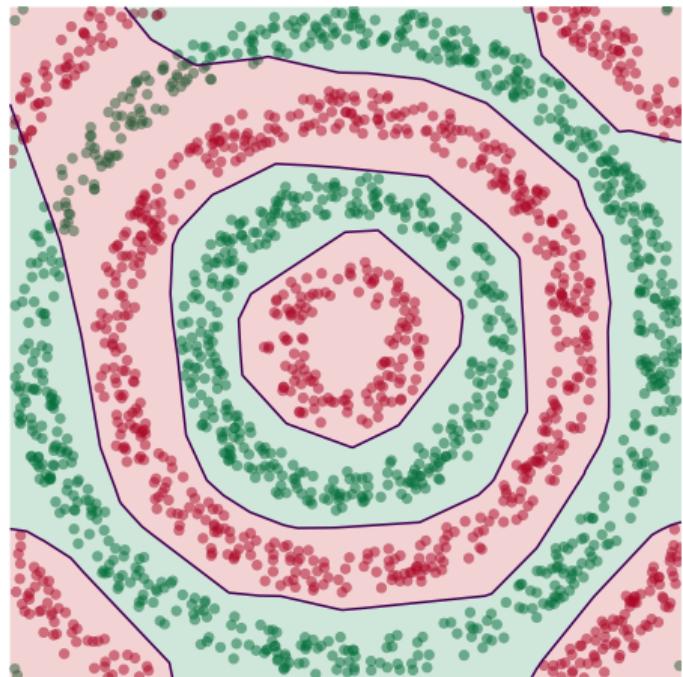
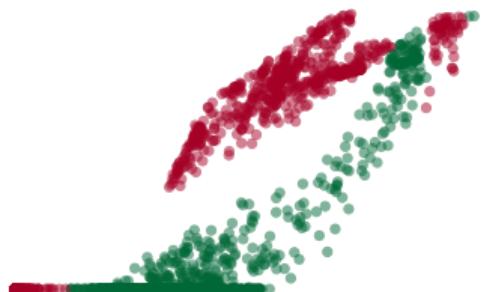
# Learning a New Representation



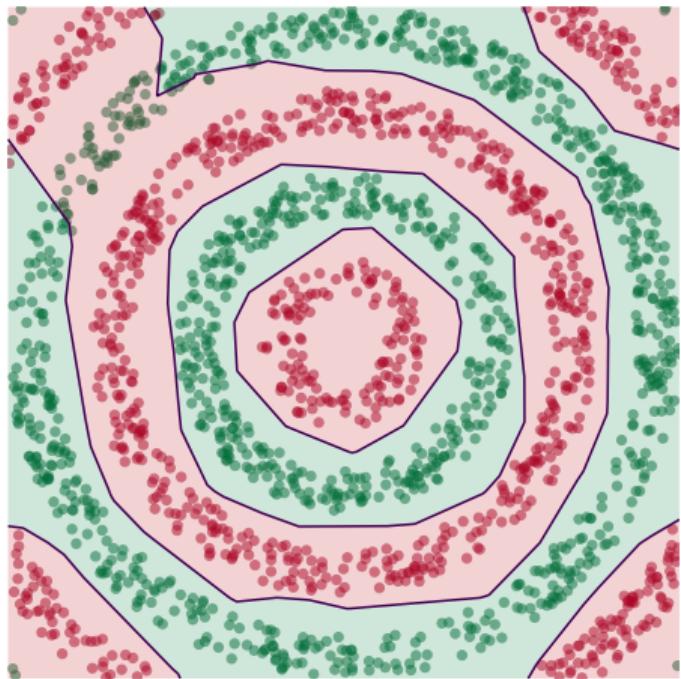
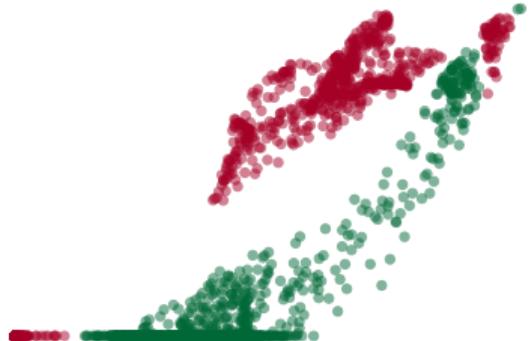
# Learning a New Representation



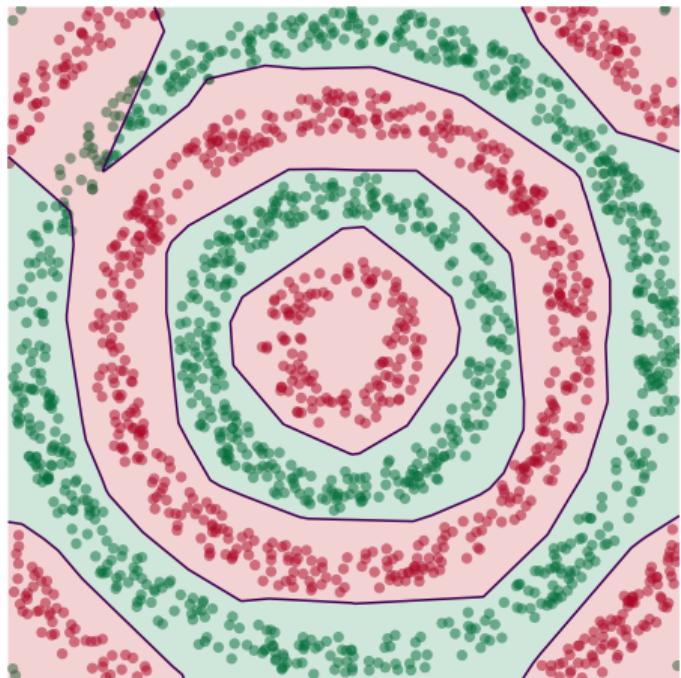
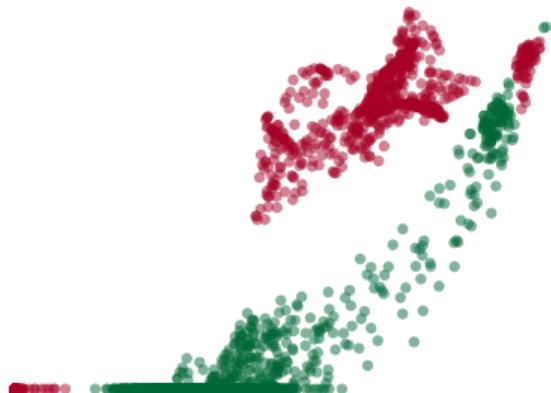
# Learning a New Representation



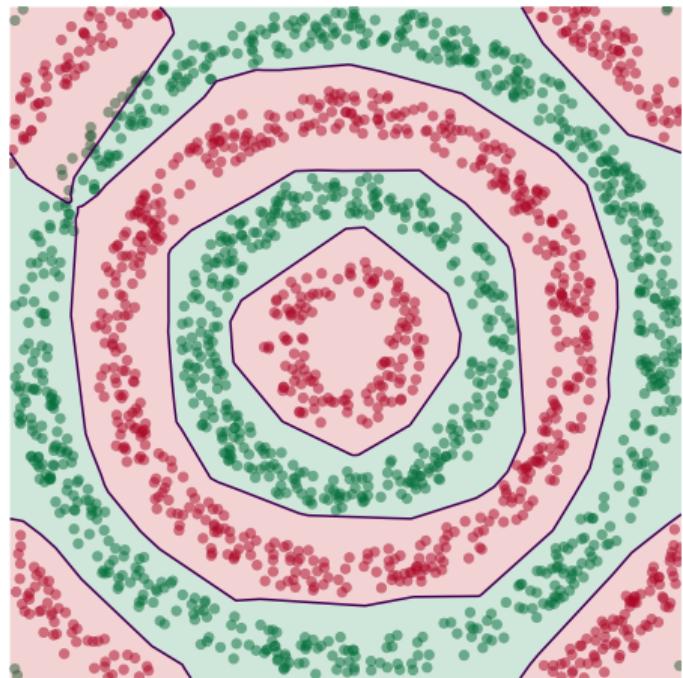
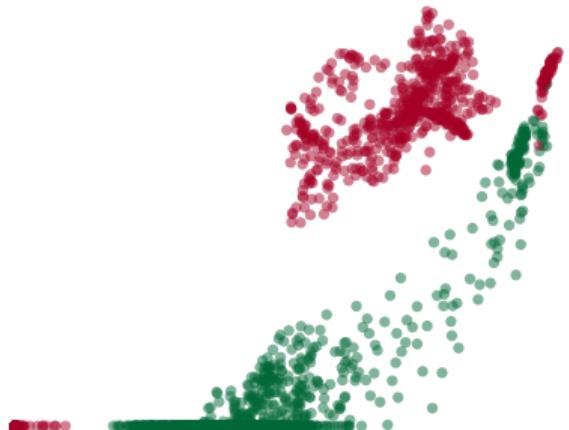
# Learning a New Representation



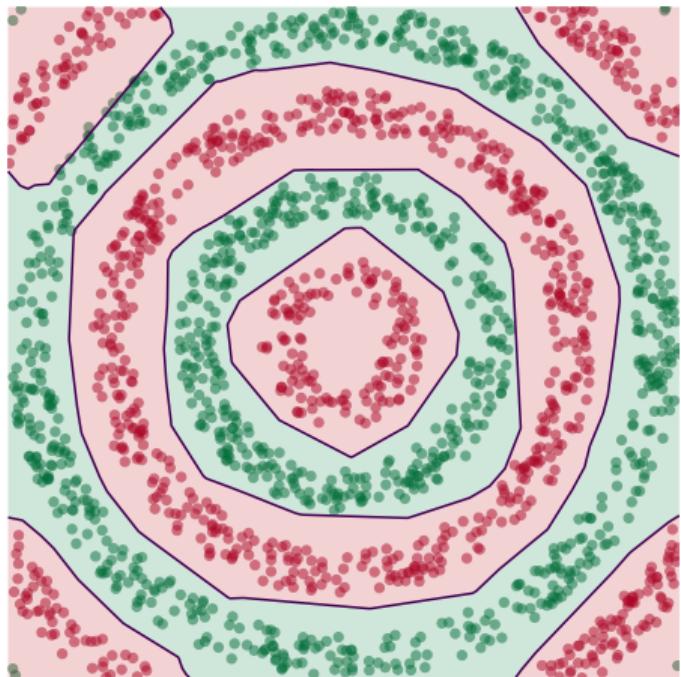
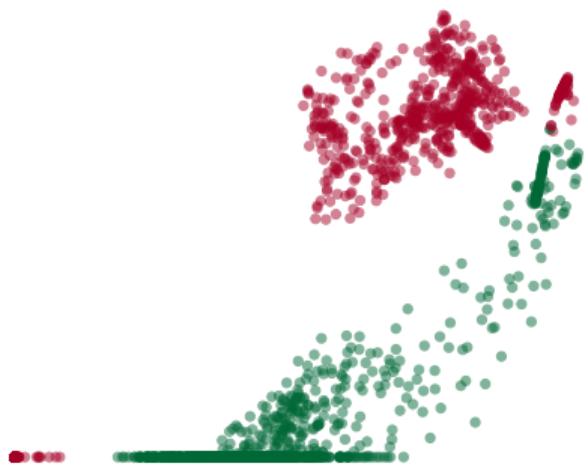
# Learning a New Representation



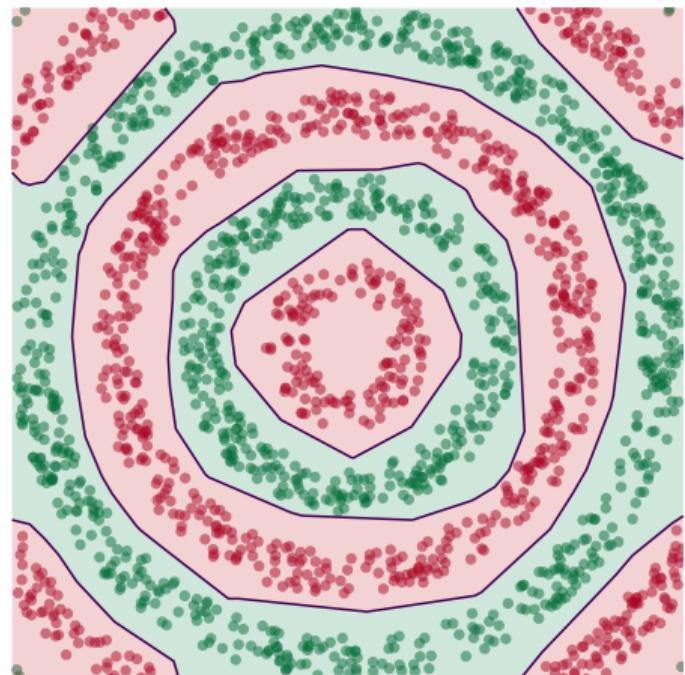
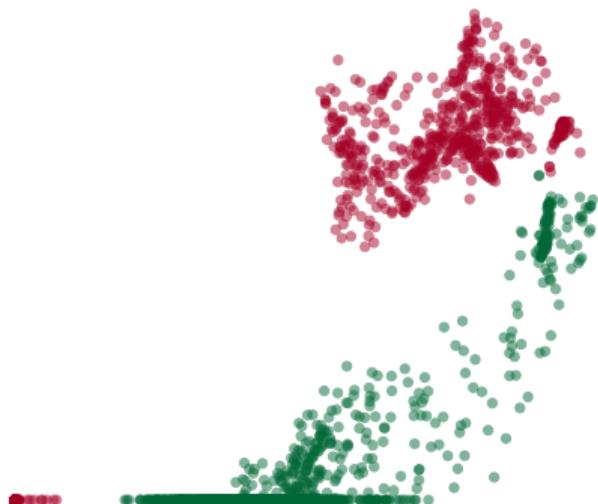
# Learning a New Representation



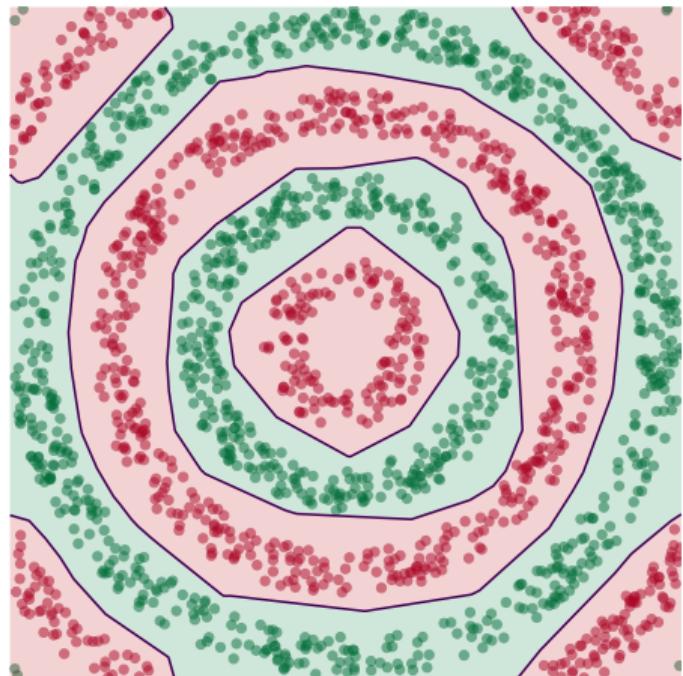
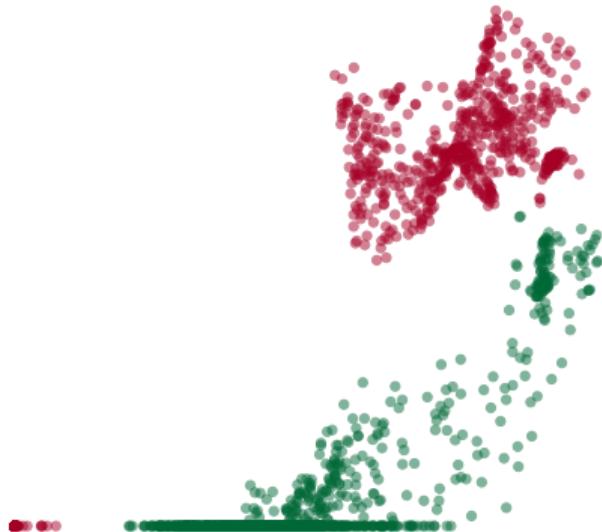
# Learning a New Representation



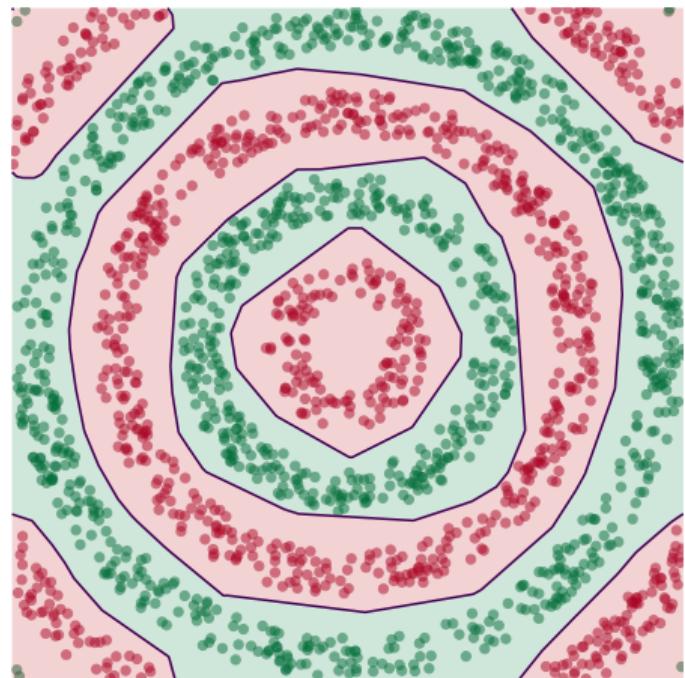
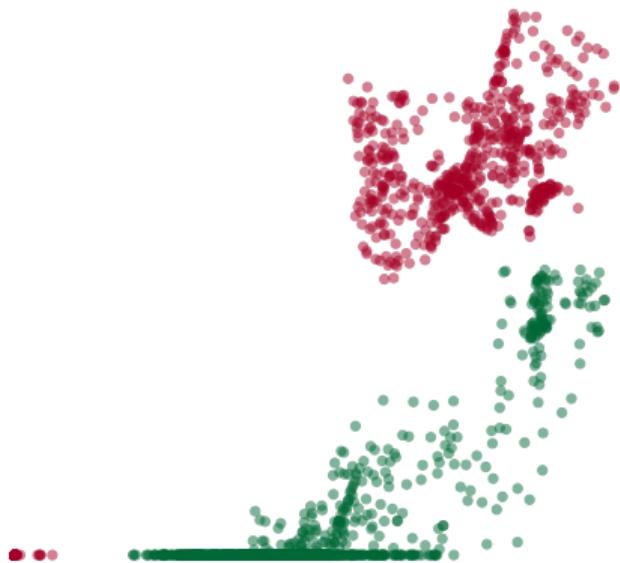
# Learning a New Representation



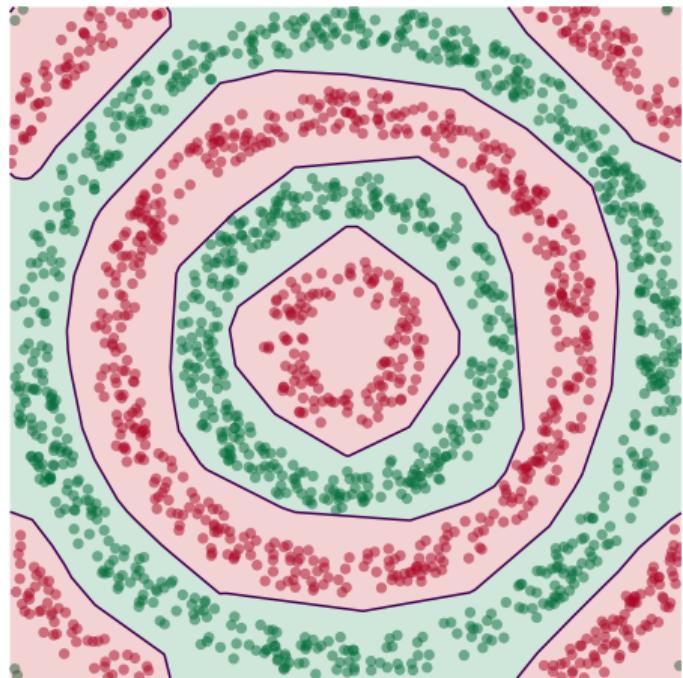
# Learning a New Representation



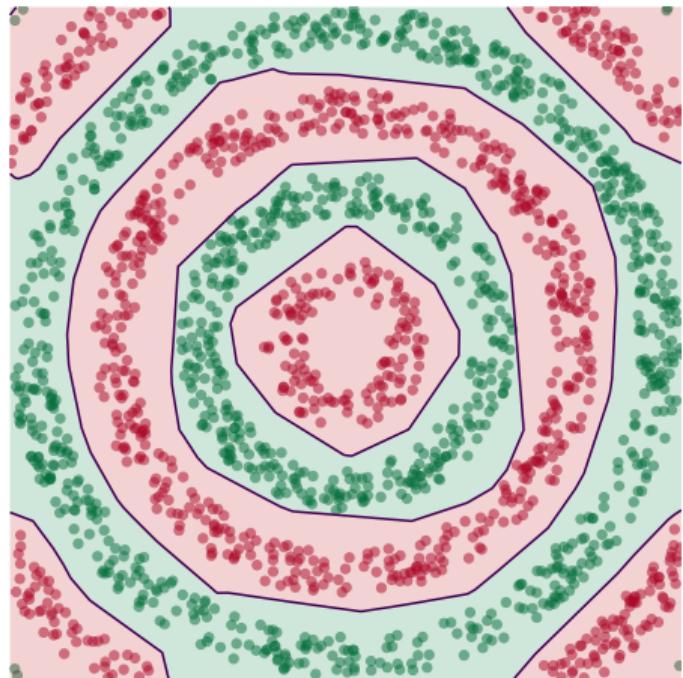
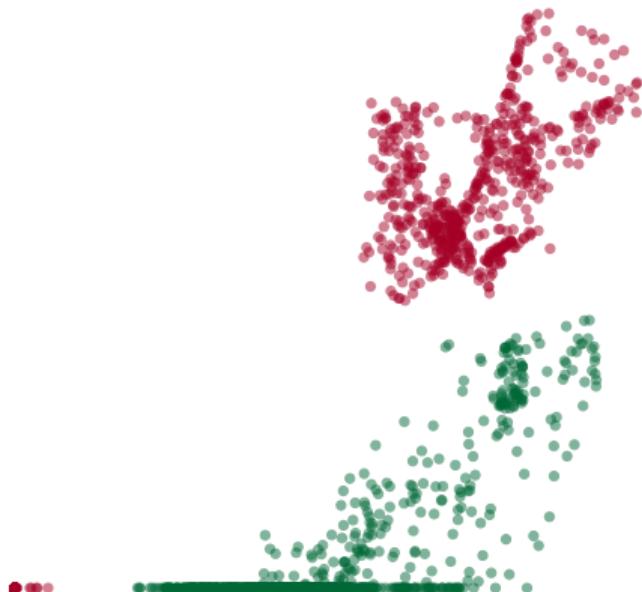
# Learning a New Representation



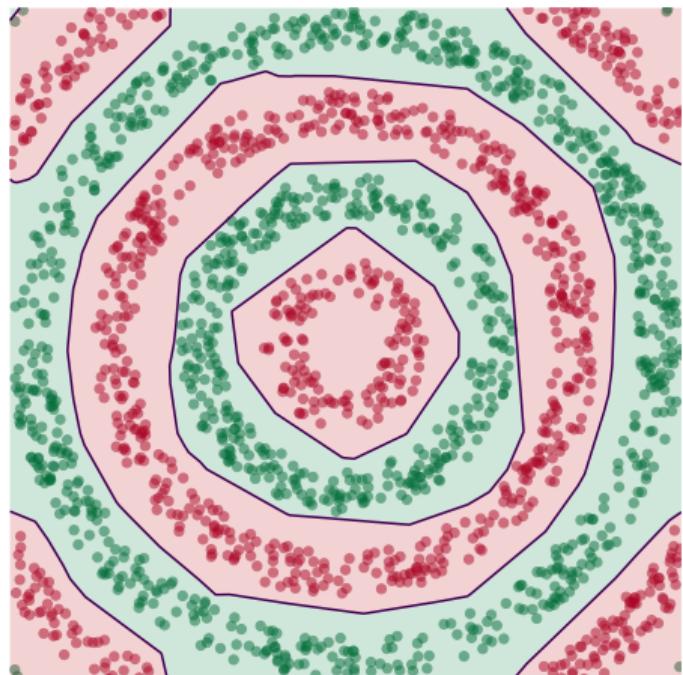
# Learning a New Representation



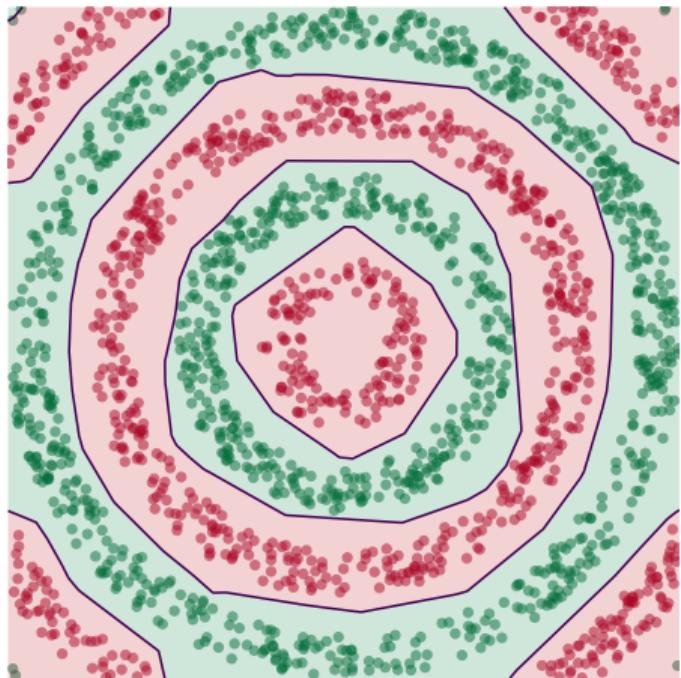
# Learning a New Representation



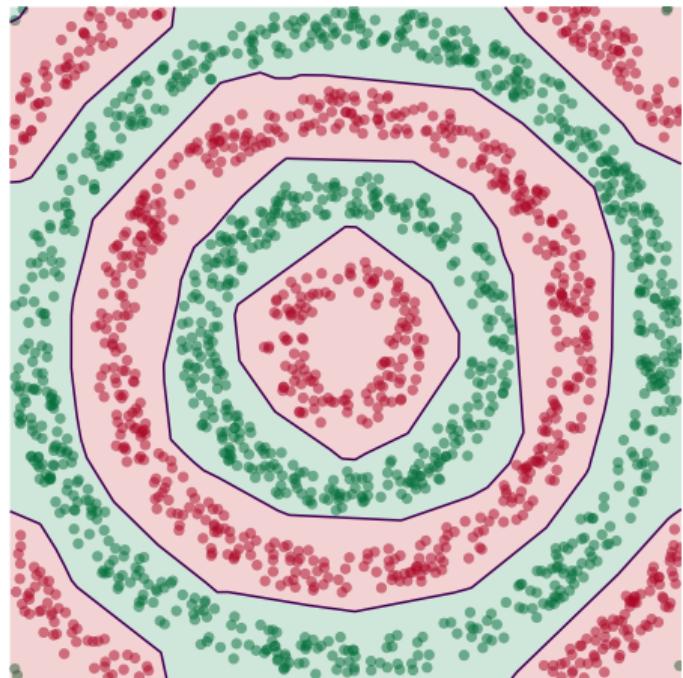
# Learning a New Representation



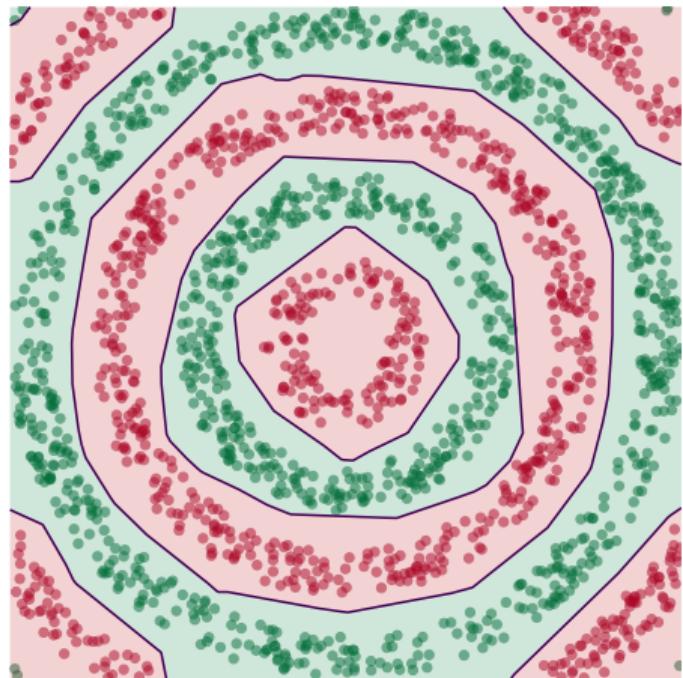
# Learning a New Representation



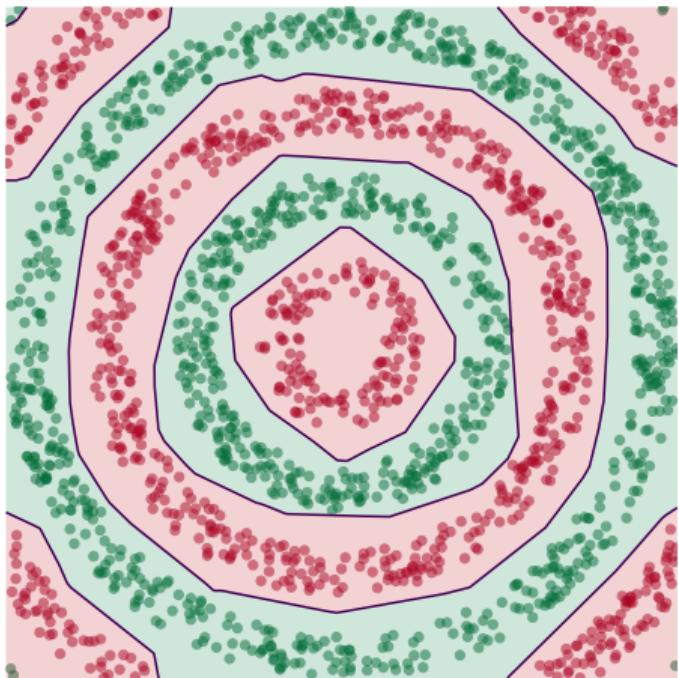
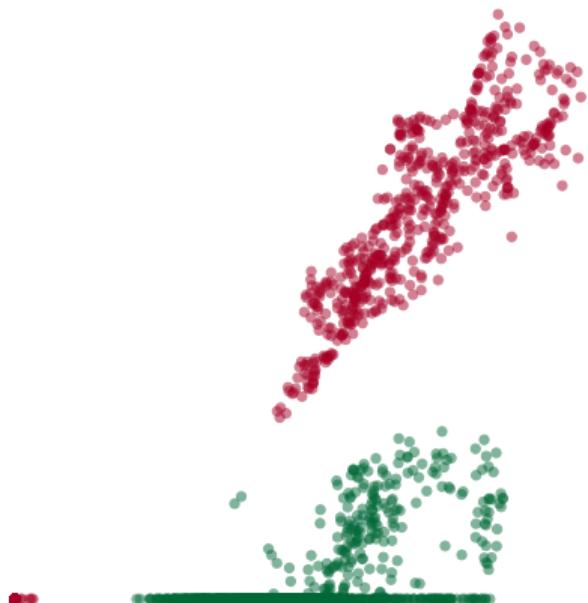
# Learning a New Representation



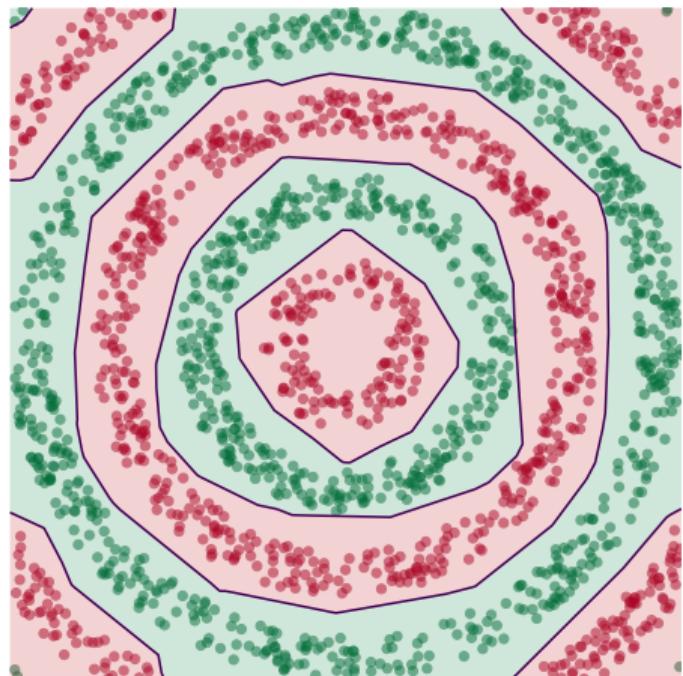
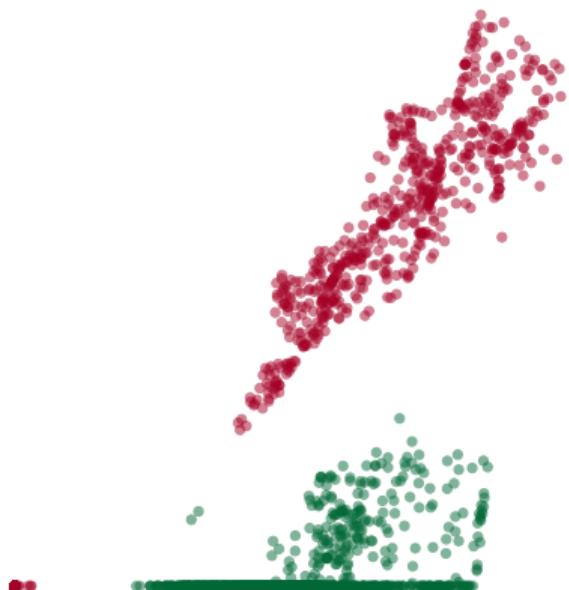
# Learning a New Representation



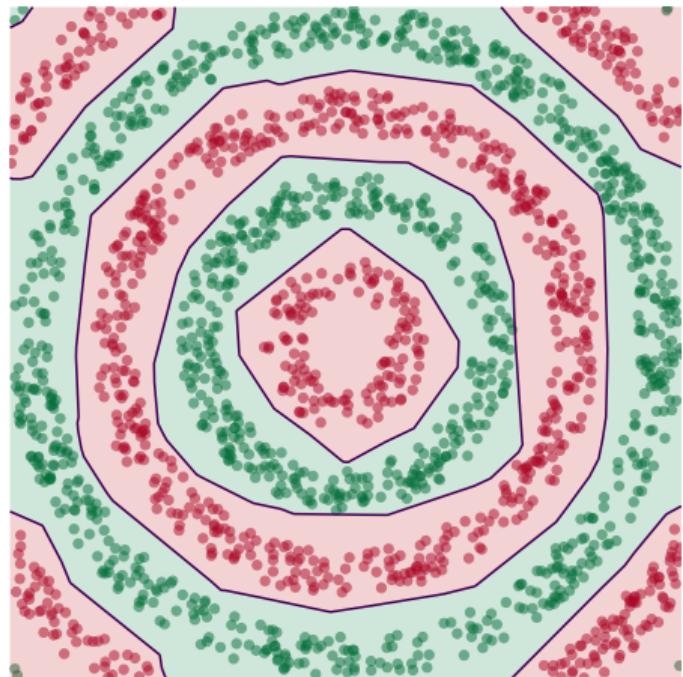
# Learning a New Representation



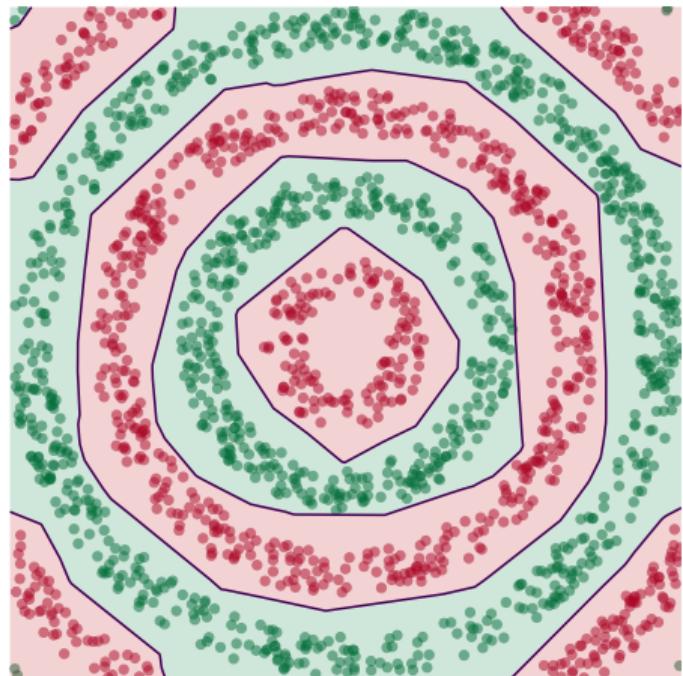
# Learning a New Representation



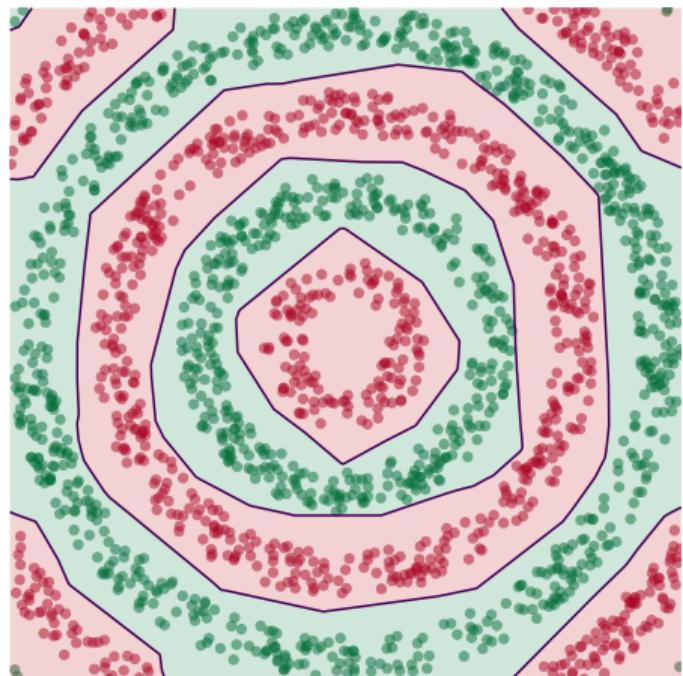
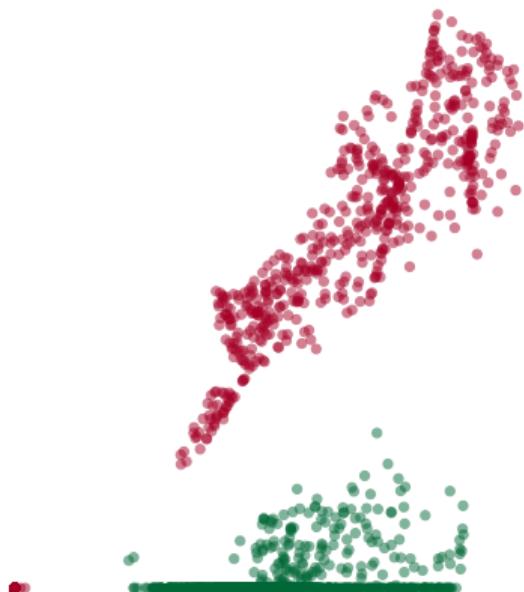
# Learning a New Representation



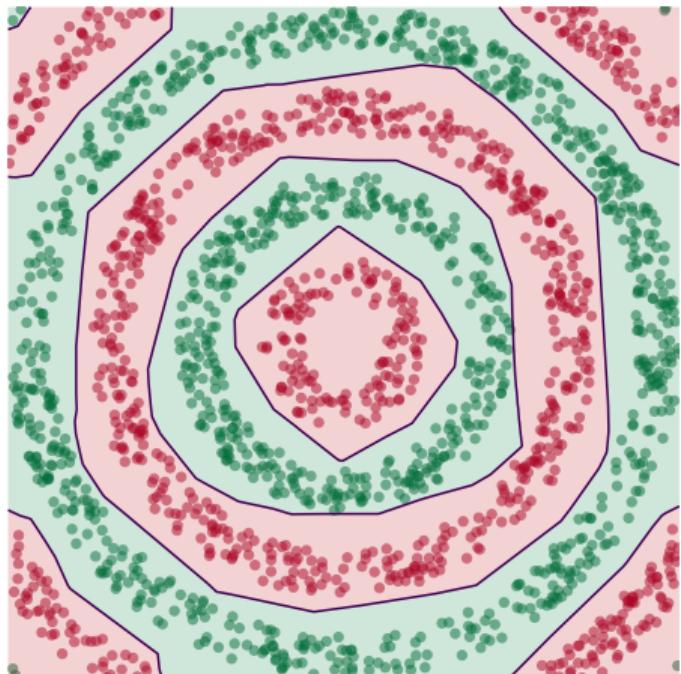
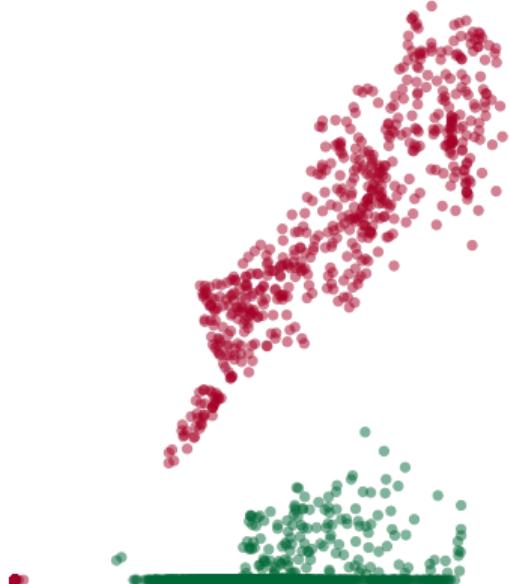
# Learning a New Representation



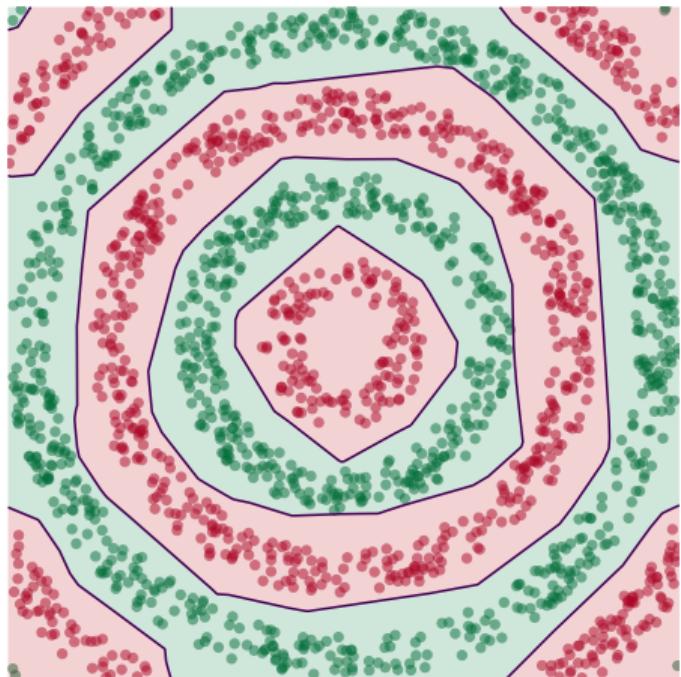
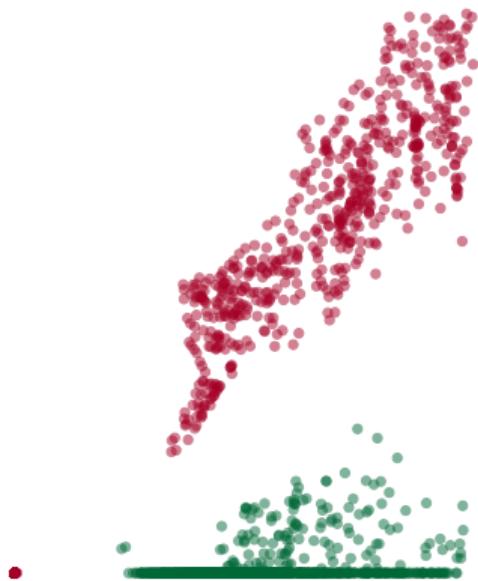
# Learning a New Representation



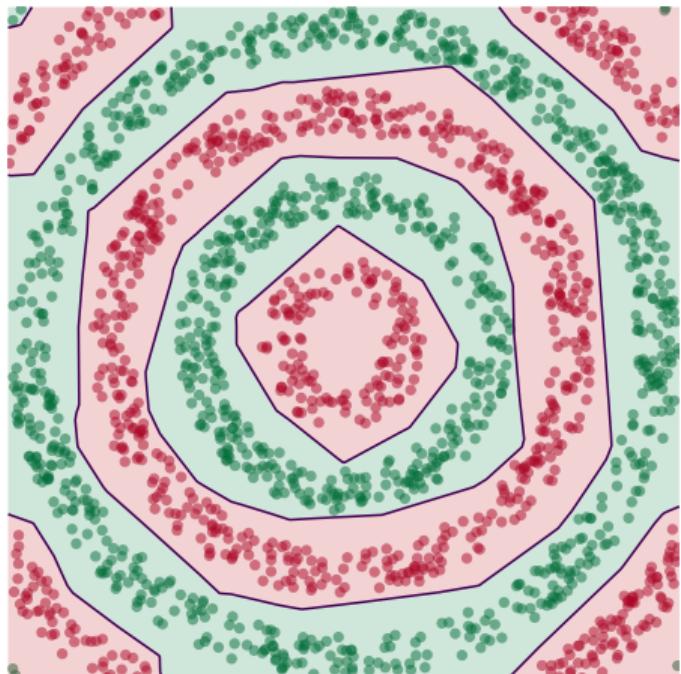
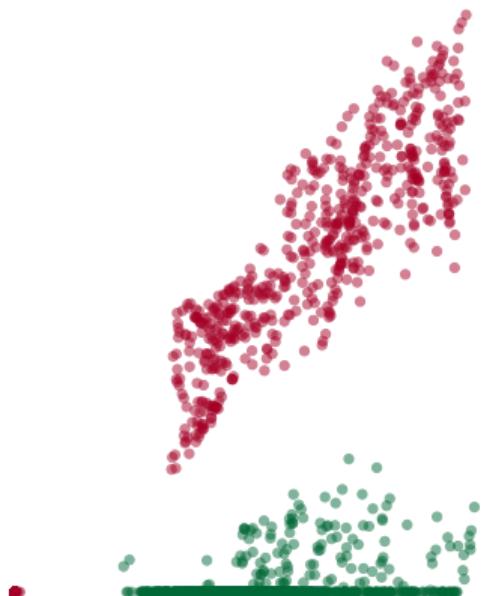
# Learning a New Representation



# Learning a New Representation



# Learning a New Representation



# Deep Learning

- ▶ The NN has learned a new **representation** in which the data is easily classified.
- ▶ For more: **DSC 140B - Representation Learning.**

# DSC 140A

*Probabilistic Modeling & Machine Learning*

Lecture 8 | Part 1

## Probabilistic Modeling

# Probabilistic Modeling

- ▶ Where does data come from?
- ▶ We might imagine that “Nature” generates it using some random (i.e., **probabilistic**) process.
- ▶ Maybe modeling this probabilistic process will suggest new ways of making predictions?

# Example: Flowers

- ▶ Suppose there are two species of flower.
- ▶ One species tends to have more petals.
- ▶ **Goal:** Given a new flower with  $X = x$  petals predict its species,  $Y$ .



# Example: Flowers

- ▶ **Idea:** The number of petals,  $X$ , and the species,  $Y$ , are **random variables**.
- ▶ **Assumption:** When Nature generates a new flower, it picks  $X$  and  $Y$  from some **probability distribution**.
- ▶ Let's imagine (for now) that we know this distribution.

# The Joint Distribution

- ▶ The **joint distribution**  $\mathbb{P}(X = x, Y = y)$  gives us full information.

	$Y = 0$	$Y = 1$
$X = 0$	0%	0%
$X = 1$	5%	0%
$X = 2$	10%	5%
$X = 3$	15%	15%
$X = 4$	5%	20%
$X = 5$	0%	15%
$X = 6$	0%	10%

# Observation

- ▶ The entries of the joint distribution table sum to 100%.
- ▶ Mathematically:  $\sum_{x \in \{0,1,\dots,6\}} \sum_{y \in \{0,1\}} \mathbb{P}(X = x, Y = y) = 1$ .

	$Y = 0$	$Y = 1$
$X = 0$	0%	0%
$X = 1$	5%	0%
$X = 2$	10%	5%
$X = 3$	15%	15%
$X = 4$	5%	20%
$X = 5$	0%	15%
$X = 6$	0%	10%

# Marginal Distributions

- ▶ What is the probability that a new flower has  $X = 4$  petals (regardless of species)?

	$Y = 0$	$Y = 1$
$X = 0$	0%	0%
$X = 1$	5%	0%
$X = 2$	10%	5%
$X = 3$	15%	15%
$X = 4$	5%	20%
$X = 5$	0%	15%
$X = 6$	0%	10%

# Marginal Distributions

- ▶ The **marginal distribution** for  $X$  is found by summing over values of  $Y$ .
- ▶ That is:  $\mathbb{P}(X = x) = \sum_{y \in \{0,1\}} P(X = x, Y = y)$

	$Y = 0$	$Y = 1$
$X = 0$	0%	0%
$X = 1$	5%	0%
$X = 2$	10%	5%
$X = 3$	15%	15%
$X = 4$	5%	20%
$X = 5$	0%	15%
$X = 6$	0%	10%

joint

$X = 0$	0%
$X = 1$	5%
$X = 2$	15%
$X = 3$	30%
$X = 4$	25%
$X = 5$	15%
$X = 6$	10%

marginal in X

# Marginal Distributions

- ▶ What is the probability that a new flower is species 1 (regardless of number of petals)?

	$Y = 0$	$Y = 1$
$X = 0$	0%	0%
$X = 1$	5%	0%
$X = 2$	10%	5%
$X = 3$	15%	15%
$X = 4$	5%	20%
$X = 5$	0%	15%
$X = 6$	0%	10%

# Marginal Distributions

- ▶ The **marginal distribution** for  $Y$  is found by summing over values of  $X$ .
- ▶ That is:  $\mathbb{P}(Y = y) = \sum_{x \in \{0, \dots, 6\}} P(X = x, Y = y)$

	$Y = 0$	$Y = 1$
$X = 0$	0%	0%
$X = 1$	5%	0%
$X = 2$	10%	5%
$X = 3$	15%	15%
$X = 4$	5%	20%
$X = 5$	0%	15%
$X = 6$	0%	10%

joint

	$Y = 0$	$Y = 1$
	35%	65%

marginal in  $Y$

# **Observation**

- ▶ The probabilities in the marginal distributions also sum to 1.

## Exercise

Suppose flower A has 4 petals. What do you predict its species to be?

	$Y = 0$	$Y = 1$
$X = 0$	0%	0%
$X = 1$	5%	0%
$X = 2$	10%	5%
$X = 3$	15%	15%
$X = 4$	5%	20%
$X = 5$	0%	15%
$X = 6$	0%	10%

# Intuition

- ▶ It seems **more likely** that a petal with 4 flowers is from species 1.

	$Y = 0$	$Y = 1$
$X = 0$	0%	0%
$X = 1$	5%	0%
$X = 2$	10%	5%
$X = 3$	15%	15%
$X = 4$	5%	20%
$X = 5$	0%	15%
$X = 6$	0%	10%

# Conditional Probabilities

- This is captured by the **conditional probability**  
 $\mathbb{P}(Y = y | X = x) = \mathbb{P}(X = x, Y = y) / \mathbb{P}(X = x).$

	$Y = 0$	$Y = 1$
$X = 0$	0%	0%
$X = 1$	5%	0%
$X = 2$	10%	5%
$X = 3$	15%	15%
$X = 4$	5%	20%
$X = 5$	0%	15%
$X = 6$	0%	10%

joint

$\mathbb{P}(Y = y   X = 1)$	
$Y = 0$	100%
$Y = 1$	0%

$\mathbb{P}(Y = y   X = 2)$	
$Y = 0$	66.5%
$Y = 1$	33.3%

$\mathbb{P}(Y = y   X = 4)$	
$Y = 0$	20%
$Y = 1$	80%

# Conditional Probabilities

- The **conditional probability**

$$\mathbb{P}(X = x | Y = y) = \mathbb{P}(X = x, Y = y) / \mathbb{P}(Y = y).$$

	$Y = 0$	$Y = 1$
$X = 0$	0%	0%
$X = 1$	5%	0%
$X = 2$	10%	5%
$X = 3$	15%	15%
$X = 4$	5%	20%
$X = 5$	0%	15%
$X = 6$	0%	10%

	$\mathbb{P}(X = x   Y = 0)$
$X = 0$	0%
$X = 1$	14.2%
$X = 2$	28.5%
$X = 3$	42.8%
$X = 4$	14.2%
$X = 5$	0%
$X = 6$	0%

joint

# Observation

- ▶ Conditional probabilities sum to 1 as well.
- ▶ For any fixed  $x$ :

$$\sum_y \mathbb{P}(Y = y | X = x) = 1$$

- ▶ For any fixed  $y$ :

$$\sum_x \mathbb{P}(X = x | Y = y) = 1$$

# Five Distributions

- ▶ We've seen five distributions:
  - ▶ **Joint**:  $\mathbb{P}(X = x, Y = y)$
  - ▶ **Marginal in X**:  $\mathbb{P}(X = x)$
  - ▶ **Marginal in Y**:  $\mathbb{P}(Y = y)$
  - ▶ **Conditional on X**:  $\mathbb{P}(Y = y | X = x)$
  - ▶ **Conditional on Y**:  $\mathbb{P}(X = x | Y = y)$
- ▶ If we know the **joint** distribution, we can compute any of the others.

# Bayes' Theorem

- ▶ **Bayes' Theorem** relates conditional probabilities and provides another way of computing them:

$$\mathbb{P}(Y = y | X = x) = \frac{\mathbb{P}(X = x | Y = y)\mathbb{P}(Y = y)}{\mathbb{P}(X = x)}$$

# Bayes' Theorem

- ▶ Derivation:

# Bayes Decision Theory

- ▶ **Goal:** Given a new flower with  $X = x$  petals, predict its species,  $Y$ .
- ▶ **Idea:** Predict species 1 if  $\mathbb{P}(Y = 1 | X = x) > \mathbb{P}(Y = 0 | X = x)$ ; otherwise predict species 0.
- ▶ That is, pick whichever species is more likely.

# Bayes Classification Rule

- ▶ This is the **Bayes classification rule**:
  - ▶ Predict class 1 if  $\mathbb{P}(Y = 1 | X = x) > \mathbb{P}(Y = 0 | X = x)$ ;
  - ▶ Otherwise, predict class 0.

# Bayes Decision Theory

- ▶ Using Bayes' rule,

$$\mathbb{P}(Y = y | X = x) = \mathbb{P}(X = x | Y = y)\mathbb{P}(Y = y) / \mathbb{P}(X = x)$$

- ▶ **Bayes classification rule** (original form):

- ▶ Predict class 1 if  $\mathbb{P}(Y = 1 | X = x) > \mathbb{P}(Y = 0 | X = x)$ ;
- ▶ Otherwise, predict class 0.

- ▶ **Bayes classification rule** (alternative form):

- ▶ Predict class 1 if  
 $\mathbb{P}(X = x | Y = 1)\mathbb{P}(Y = 1) > \mathbb{P}(X = x | Y = 0)\mathbb{P}(Y = 0)$
- ▶ Otherwise, predict class 0.

## Main Idea

If we know the conditional probability of the label  $Y$  given feature  $X$ , the Bayes classification rule is a natural way to make predictions.

# DSC 140A

*Probabilistic Modeling & Machine Learning*

Lecture 8 | Part 2

**Continuous Distributions**

## Example: Penguins

- ▶ Suppose there are two species of penguin.
- ▶ One species tends to have longer flippers.
- ▶ **Goal:** given a new penguin with flipper length  $X = x$ , predict its species,  $Y$ .

# Five Distributions

- ▶ In this situation, what do the five distributions look like?
  - ▶ Joint distribution of  $X$  and  $Y$
  - ▶ Marginal distribution in  $X$
  - ▶ Marginal distribution in  $Y$
  - ▶ Conditional on  $X$
  - ▶ Conditional on  $Y$

# Marginal in $Y$

- ▶ What is the probability that Nature generates a penguin from species  $Y$ ?
  - ▶ Marginal distribution:  $\mathbb{P}(Y = y)$ .
- ▶ This is a **discrete** distribution, as before.
- ▶ Example:

$Y = 0$	$ $	30%
$Y = 1$	$ $	70%

# Marginal in X

- ▶ What is the probability that Nature generates a flipper length of  $x$ , without regard to species?
- ▶ Flipper length is a **continuous** random variable.
- ▶ Distribution is described by a **probability density function (pdf)**,  $p : \mathbb{R} \rightarrow \mathbb{R}^+$ .

# Recall: Density Functions

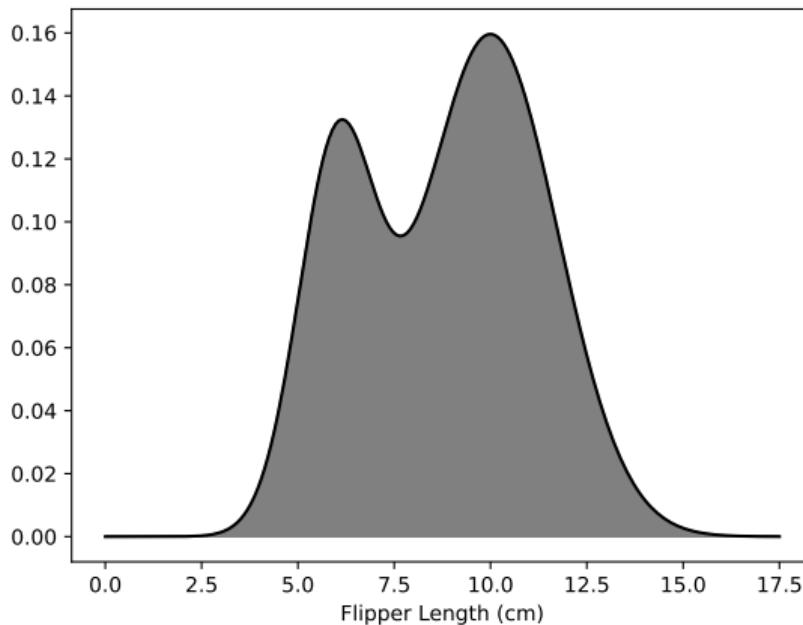
- ▶ A **probability density function (pdf)** for a random variable  $X$  is a function  $p : \mathbb{R} \rightarrow \mathbb{R}^+$  satisfying:

$$\mathbb{P}(a < X < b) = \int_a^b p_X(x) dx$$

- ▶ That is, the pdf  $p$  describes how likely it is to get a value of  $X$  in any interval  $[a, b]$ .
- ▶ Note:  $\int_{-\infty}^{\infty} p_X(x) dx = 1$ , but  $p(x)$  can be larger than one.

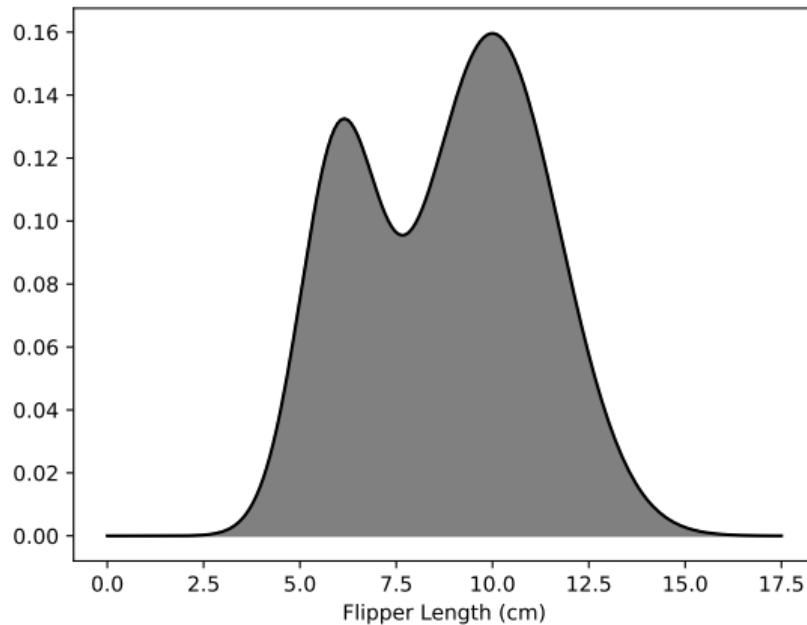
# Marginal in $X$

- ▶ The distribution of flipper lengths is described by a density function,  $p_X(x)$ .



## Exercise

What is the probability that Nature generates a penguin with flipper length equal to 10 cm?

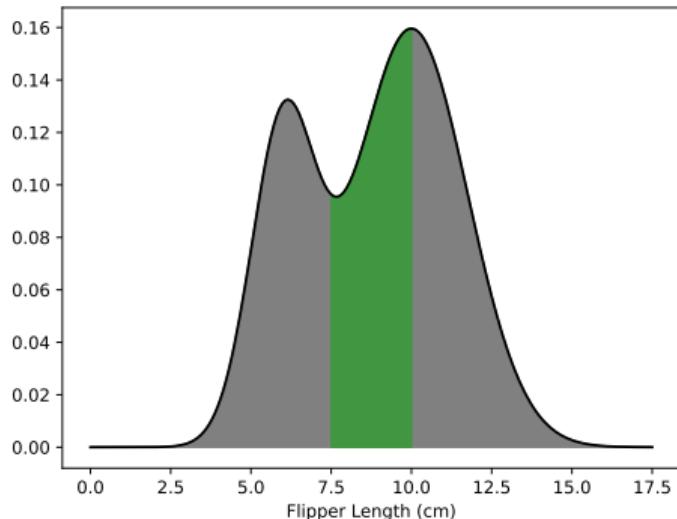


# Solution

- ▶ **Zero!**
- ▶  $p_X(x)$  is **not** the probability that  $X = x$ .
- ▶ Instead,  $\mathbb{P}(X = x) = \mathbb{P}(x < X < x) = \int_x^x p_X(x) dx = 0$
- ▶ The **probability** of a continuous random variable being *exactly* a certain value is zero.

# Example

- ▶ What is the probability that Nature generates a penguin whose flipper length is between 7.5 and 10 cm?

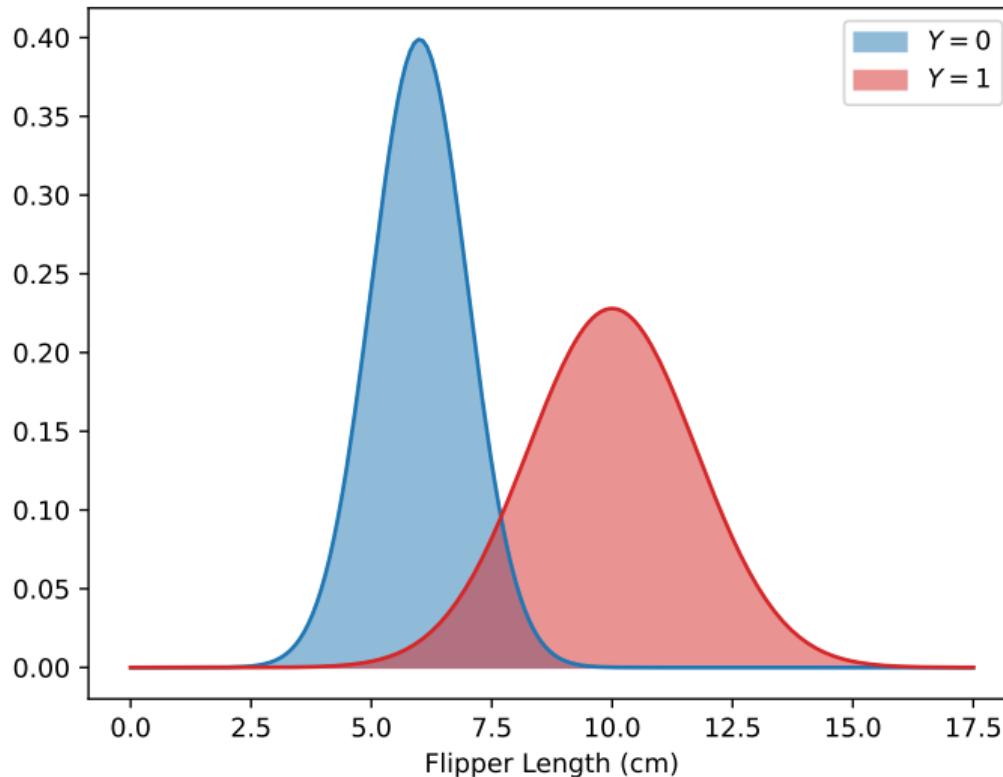


$$\mathbb{P}(7.5 < X < 10) = \int_{7.5}^{10} p_X(x) dx$$

# Conditional on Y

- ▶ What is the probability of a certain flipper length, given that the species is  $y$ ?
- ▶ Also a continuous distribution, described by **conditional density**  $p(x | Y = y)$ .
- ▶ Two conditional density functions: one for  $Y = 0$  and one for  $Y = 1$ .
  - ▶ Each integrates to one.

# Conditional on $Y$



# Conditional on X

- ▶ What is the probability that the species is  $y$  given a flipper length of  $x$ ?
- ▶ The conditional distribution of  $Y$  given  $X$ .

## Exercise

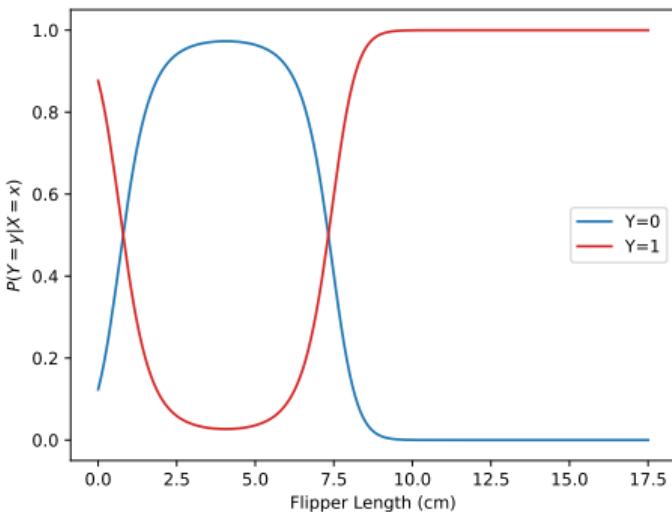
Is this distribution **continuous** or **discrete**?

## Conditional on $X$

- ▶ Answer: **discrete**, because  $Y$  is discrete.
- ▶ One distribution  $P(Y = y | X = x)$  for each possible value of  $X$  (infinitely many).

# Conditional on $X$

- ▶ Although for any fixed  $x$ ,  $\mathbb{P}(Y = y | X = x)$  is discrete, we can plot the functions  $f_0(x) = \mathbb{P}(Y = 0 | X = x)$  and  $f_1(x) = \mathbb{P}(Y = 1 | X = x)$



# Bayes' Rule

- ▶ Bayes' Rule applies to densities, too:

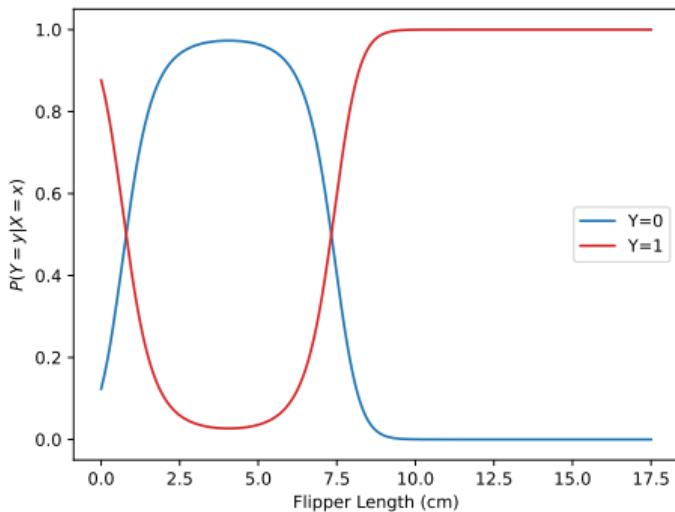
$$\mathbb{P}(Y = y | X = x) = \frac{p(x | Y = y)\mathbb{P}(Y = y)}{p_X(x)}$$

# Bayes Decision Theory

- ▶ **Bayes classification rule:**
  - ▶ Predict class 1 if  $\mathbb{P}(Y = 1 | X = x) > \mathbb{P}(Y = 0 | X = x)$ ;
  - ▶ Otherwise, predict class 0.
- ▶ **Bayes classification rule** (alternative form):
  - ▶ Predict class 1 if  
 $p(x | Y = 1)\mathbb{P}(Y = 1) > p(X = x | Y = 0)\mathbb{P}(Y = 0)$
  - ▶ Otherwise, predict class 0.

## Exercise

Penguins with flippers of length 0, 3, and 12 are observed. What are their predicted species according to the Bayes' classification rule?



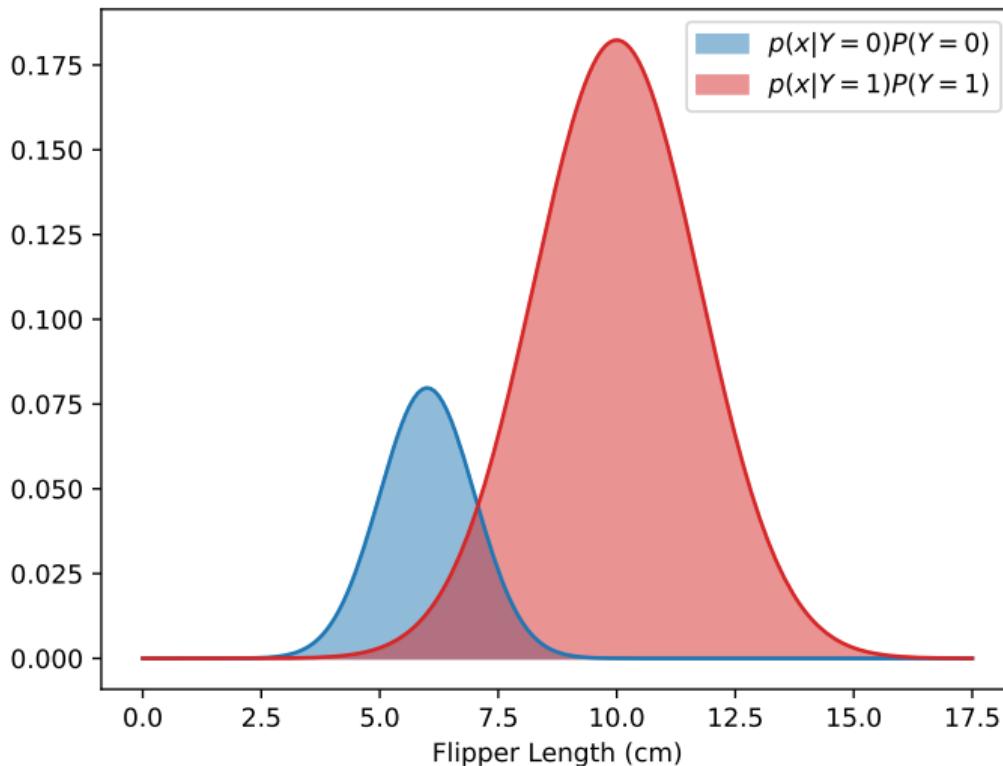
# Joint

- ▶ The **joint** distribution in this case is neither totally continuous nor totally discrete.
- ▶ From Bayes' rule:

$$p(x, 0) = p(x | Y = 0)\mathbb{P}(Y = 0)$$

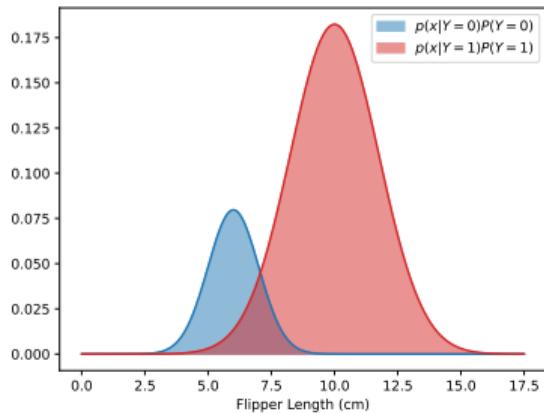
$$p(x, 1) = p(x | Y = 1)\mathbb{P}(Y = 1)$$

# Joint Distribution



## Exercise

Where does the Bayes decision rule make a prediction for class 1?



- ▶ Predict class 1 if  $p(x | Y = 1)\mathbb{P}(Y = 1) > p(X = x | Y = 0)\mathbb{P}(Y = 0)$
- ▶ Otherwise, predict class 0.

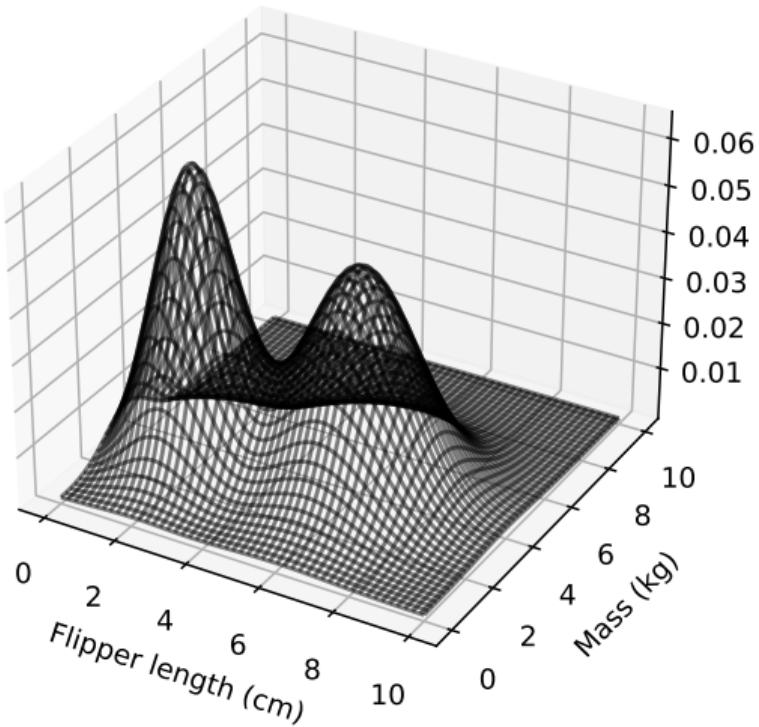
# Multivariate Distributions

- ▶ In binary classification,  $y \in \{0, 1\}$ .
- ▶ But we usually deal with feature vectors,  $\vec{x}$ .
- ▶ The previous applies with straightforward changes.

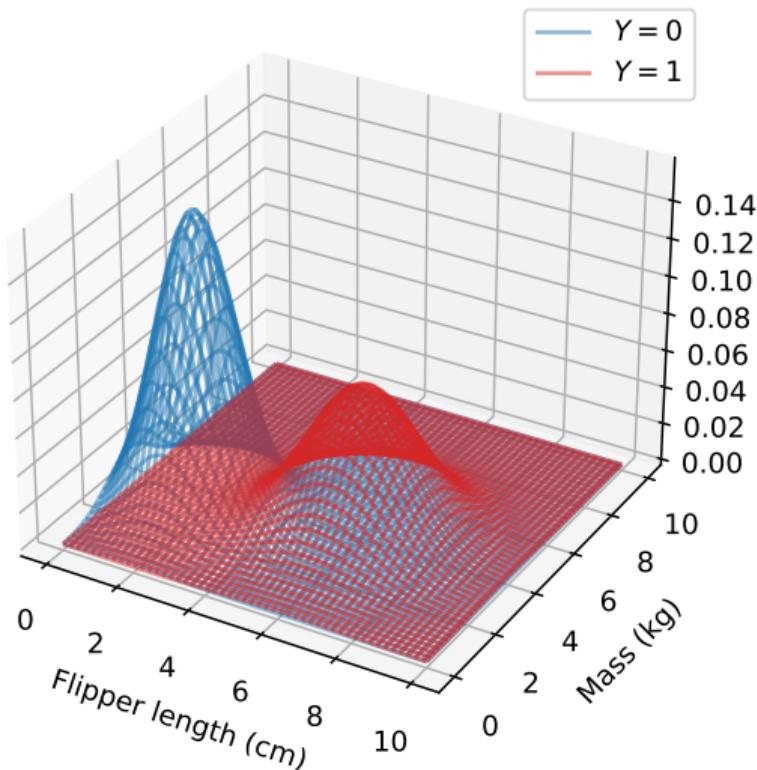
# Example: Penguins

- ▶ Again consider penguins of two species, but now consider both flipper length and body mass.
- ▶ Each penguin's measurements are a **random vector**:  $\vec{X}$ .
- ▶ Densities are now functions of a vector.
  - ▶ E.g., marginal:  $p_X(\vec{x}) : \mathbb{R}^2 \rightarrow \mathbb{R}^+$

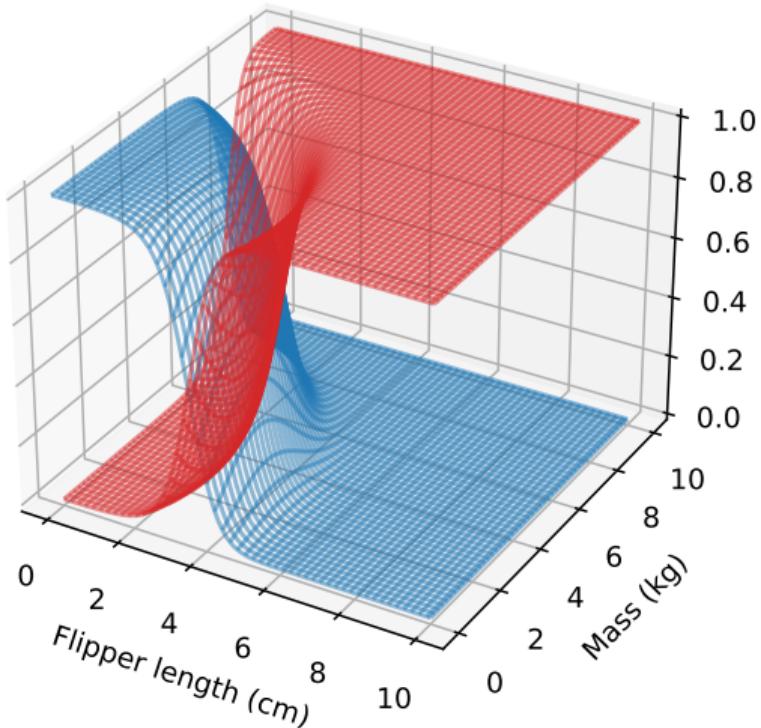
# Marginal in $\vec{X}$



# Conditional on $Y$



# Conditional on X



# DSC 140A

*Probabilistic Modeling & Machine Learning*

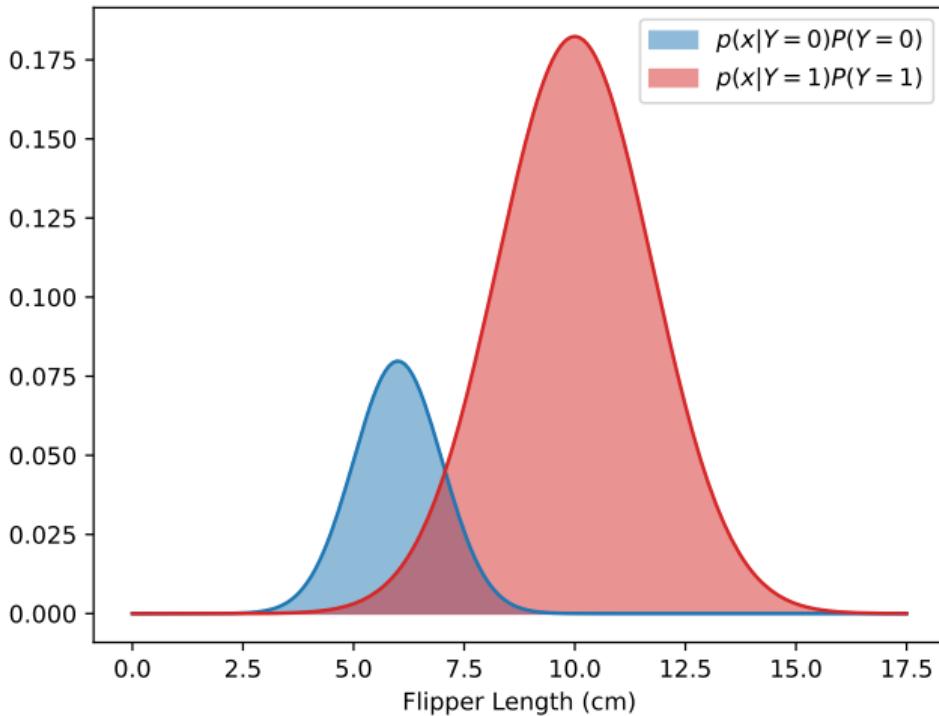
Lecture 8 | Part 3

**Bayes Error**

# Bayes Error

- ▶ If we know the joint distribution, the **Bayes classification rule** is a natural approach to making predictions.
- ▶ It is also the **best you can do**, in a sense.

# Intuition



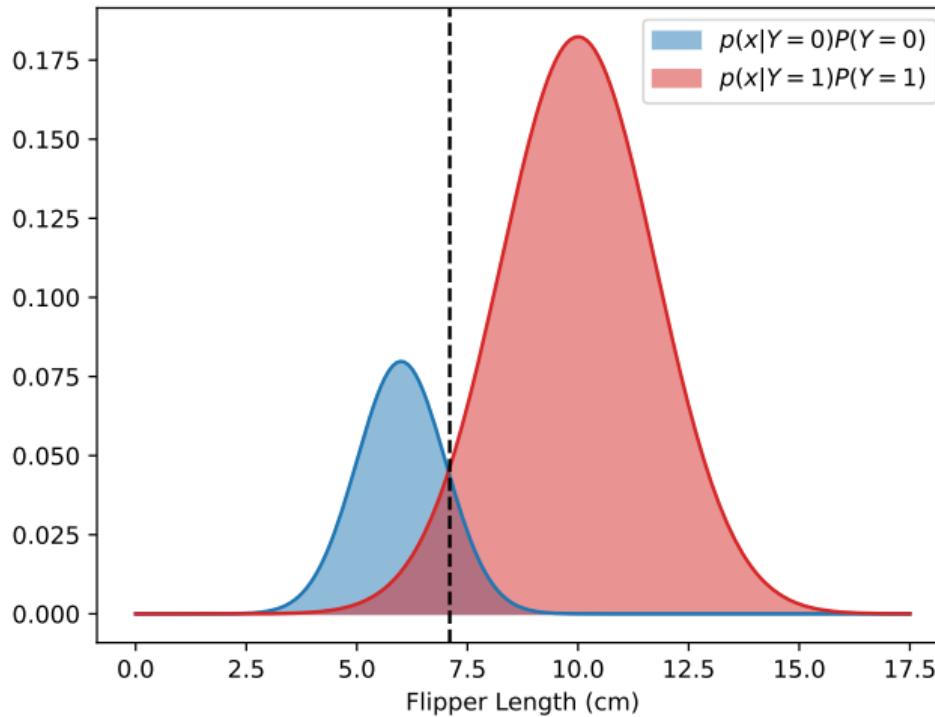
# Error Probability

- ▶ In binary classification, there are two kinds of errors:
  - ▶ Predicted 0, but the right answer is 1 (Case 1).
  - ▶ Predicted 1, but the right answer is 0 (Case 2).
- ▶ The probability of an error is:

$$\mathbb{P}(\text{error}) = \mathbb{P}(\text{Case 1}) + \mathbb{P}(\text{Case 2})$$

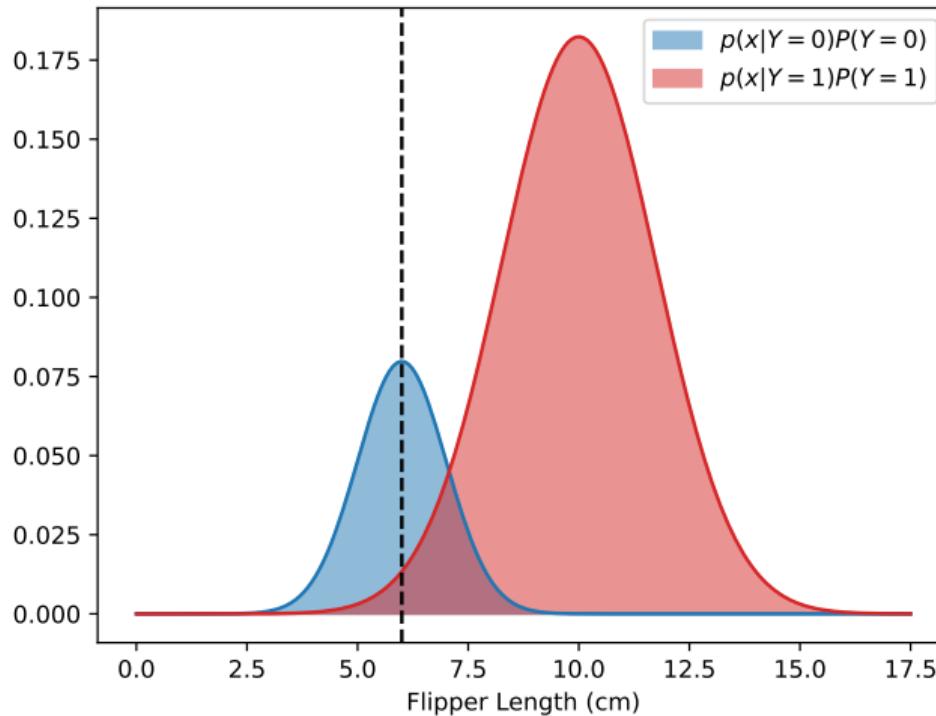
# Example

- ▶ Case 1: Predicted 0, but the right answer is 1.
- ▶ Case 2: Predicted 1, but the right answer is 0.



# Example

- ▶ Case 1: Predicted 0, but the right answer is 1.
- ▶ Case 2: Predicted 1, but the right answer is 0.



# Optimality

- ▶ The Bayes decision rule achieves the **minimum possible** error probability.
  - ▶ Sometimes called the **Bayes classifier**.
- ▶ In most cases, the minimum possible error probability is  $>0$ .

# What's next?

- ▶ The Bayes classifier is optimal.
- ▶ But it requires knowing the joint distribution; we almost never know this.
- ▶ Next time: **estimating** probability distributions from data.

# DSC 140A

*Probabilistic Modeling & Machine Learning*

Lecture 8 | Part 1

## Estimating Discrete Probabilities

# Recap: Bayes Classifier

- ▶ Bayes classifier: given a new point  $\vec{x}$ , predict:
  - ▶ Class 1 if  $\mathbb{P}(Y = 1 | \vec{X} = \vec{x}) > \mathbb{P}(Y = 0 | \vec{X} = \vec{x})$
  - ▶ Class 0 otherwise.
- ▶ Alternative form:
  - ▶ Class 1 if
$$\mathbb{P}(\vec{X} = \vec{x} | Y = 1)\mathbb{P}(Y = 1) > \mathbb{P}(\vec{X} = \vec{x} | Y = 0)\mathbb{P}(Y = 0)$$
  - ▶ Class 0 otherwise.
- ▶ **Optimal:** smallest possible probability of error.

# Problem

- ▶ This assumed that we **know** the true probabilities used by Nature.
- ▶ Typically, we do not.
- ▶ But we can **estimate** them from data.

# Example: Flowers

- ▶ **Example:** two species of flower (1 and 0); one species tends to have more petals than the other.
- ▶ **Goal:** given new flower with  $X$  petals, predict species,  $Y$ .
- ▶ Both  $X$  and  $Y$  are **discrete**.

# Before: Joint Distribution

- Before: we somehow knew the joint distribution:

	$Y = 0$	$Y = 1$
$X = 0$	0%	0%
$X = 1$	5%	0%
$X = 2$	10%	5%
$X = 3$	15%	15%
$X = 4$	5%	20%
$X = 5$	0%	15%
$X = 6$	0%	10%

# Now

- ▶ In practice, we do not know the joint distribution:

	$Y = 0$	$Y = 1$
$X = 0$	?	?
$X = 1$	?	?
$X = 2$	?	?
$X = 3$	?	?
$X = 4$	?	?
$X = 5$	?	?
$X = 6$	?	?

# Data

- ▶ Suppose we observe 10 flowers.
- ▶ We can use this data to estimate probabilities.
- ▶ E.g., what is  $\mathbb{P}(X = 4, Y = 1)$ ?

X	Y
5	0
3	0
4	1
4	1
2	0
5	1
2	1
5	1
4	1
3	0

# Estimating Joint Probabilities

- ▶ We estimate  $\mathbb{P}(X = x, Y = y)$  with:

$$\mathbb{P}(X = x, Y = y) \approx \frac{\#(X = x \text{ and } Y = y)}{n}$$

- ▶ E.g., estimate  $\mathbb{P}(X = 4, Y = 1)$ :
- ▶ E.g., estimate  $\mathbb{P}(X = 3, Y = 0)$ :
- ▶ E.g., estimate  $\mathbb{P}(X = 3, Y = 1)$ :

X	Y
5	0
3	0
4	1
4	1
2	0
5	1
2	1
5	1
4	1
3	0

# Estimating Other Probabilities

- ▶ Recall the other probabilities:
  - ▶ **Marginals:**  $\mathbb{P}(X = x)$  and  $\mathbb{P}(Y = y)$ .
  - ▶ **Conditionals:**  $\mathbb{P}(X = x | Y = y)$  and  $\mathbb{P}(Y = y | X = x)$ .
- ▶ Can be calculated from the joint distribution.
  - ▶ Or an estimate of the joint distribution.
- ▶ Can also estimate more directly.

# Estimating Marginals

- ▶ We estimate  $\mathbb{P}(Y = y)$  with:

$$\mathbb{P}(Y = y) \approx \frac{\#(Y = y)}{n}$$

- ▶ E.g., estimate  $\mathbb{P}(Y = 1)$ :

- ▶ E.g., estimate  $\mathbb{P}(Y = 0)$ :

X	Y
5	0
3	0
4	1
4	1
2	0
5	1
2	1
5	1
4	1
3	0

# Estimating Marginals

- ▶ We estimate  $\mathbb{P}(X = x)$  with:

$$\mathbb{P}(X = x) \approx \frac{\#(X = x)}{n}$$

- ▶ E.g., estimate  $\mathbb{P}(X = 4)$ :

- ▶ E.g., estimate  $\mathbb{P}(X = 3)$ :

X	Y
5	0
3	0
4	1
4	1
2	0
5	1
2	1
5	1
4	1
3	0

# Estimating Conditionals

- We estimate  $\mathbb{P}(X = x | Y = y)$  with:

$$\mathbb{P}(X = x | Y = y) \approx \frac{\#(X = x \text{ and } Y = y)}{\#(Y = y)}$$

- E.g., estimate  $\mathbb{P}(X = 4 | Y = 1)$ :
- E.g., estimate  $\mathbb{P}(X = 2 | Y = 0)$ :

X	Y
5	0
3	0
4	1
4	1
2	0
5	1
2	1
5	1
4	1
3	0

# Estimating Conditionals

- ▶ We estimate  $\mathbb{P}(Y = y | X = x)$  with:

$$\mathbb{P}(Y = y | X = x) \approx \frac{\#(X = x \text{ and } Y = y)}{\#(X = x)}$$

- ▶ E.g., estimate  $\mathbb{P}(Y = 1 | X = 4)$ :
- ▶ E.g., estimate  $\mathbb{P}(Y = 0 | X = 2)$ :
- ▶ E.g., estimate  $\mathbb{P}(Y = 0 | X = 6)$ :

X	Y
5	0
3	0
4	1
4	1
2	0
5	1
2	1
5	1
4	1
3	0

# Law of Large Numbers

- ▶ As data size  $n \rightarrow \infty$ , these esimated probabilities converge to their true values.<sup>1</sup>

---

<sup>1</sup>Assuming the data was sampled iid from the true distribution.

# Bayes Classifier

- ▶ The Bayes classifier assumed we knew the true probabilities.
- ▶ But we can still use it if we replace the true probabilities with estimated probabilities.
- ▶ No longer guaranteed to be optimal!

# Bayes Classifier

- ▶ Given a new flower with 5 petals, what is its class?
- ▶ Idea: estimate  $\mathbb{P}(Y = 1 | X = 5)$ .

X	Y
5	0
3	0
4	1
4	1
2	0
5	1
2	1
5	1
4	1
3	0

# Multivariate Distributions

- ▶ We can also estimate when there are more variables in the same way.
- ▶ E.g., estimate  $\mathbb{P}(Y = 1 | X_1 = 4, X_2 = 2)$ :
- ▶ E.g., estimate  $\mathbb{P}(X_1 = 2)$ :
- ▶ E.g., estimate  $\mathbb{P}(X_1 = 5, X_2 = 1 | Y = 1)$ :

$X_1$	$X_2$	Y
5	1	0
3	3	0
4	2	1
4	5	1
2	3	0
5	2	1
2	1	1
5	1	1
4	2	0
3	6	0

# DSC 140A

*Probabilistic Modeling & Machine Learning*

Lecture 8 | Part 2

## Histogram Density Estimators

# Continuous Variables

- ▶ We have seen how to estimate **discrete** probabilities. What about **continuous** variables?
- ▶ Suppose there are two species of penguin; one species tends to have longer flippers.
- ▶ **Goal:** given a new penguin with flipper length  $X = x$ , predict its species,  $Y$ .

# Data

- ▶ Recall: The distribution of a **continuous** random variable is described by a **density**.
- ▶ Can we estimate a density from data in the same way?
- ▶ E.g.: marginal density for  $x$ ,  $p_X(x)$ .  
What is  $p_X(7)$ ?

$$p_X(7) \stackrel{?}{\approx} \frac{\#(X = 7)}{n}$$

X	Y
7.2	0
11.3	1
8.0	1
5.1	0
5.6	1
12.3	1
13.1	1
10.9	0
12.0	1
5.0	0

# Estimating Density

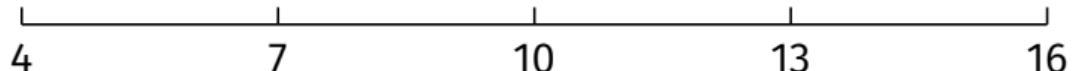
- ▶ Since  $X$  is continuous, most values of  $X$  are **never seen** in the data.
- ▶ We need to do some **smoothing**.
- ▶ One approach: **histogram estimators**.

# Histogram Estimators

- ▶ Suppose data  $x_1, \dots, x_n$  came from density  $f$
- ▶ Divide domain into  $k$  **bins**:  $[a_i, b_i)$ .
  - ▶ Often equal-sized grid, though not necessary.
- ▶ Within each bin  $i$ , estimate density:

$$f(x) \text{ within bin } i \approx \frac{\# \text{ data points } \in [a_i, b_i)}{n \times \underbrace{(b_i - a_i)}_{\text{"bin width"}}$$

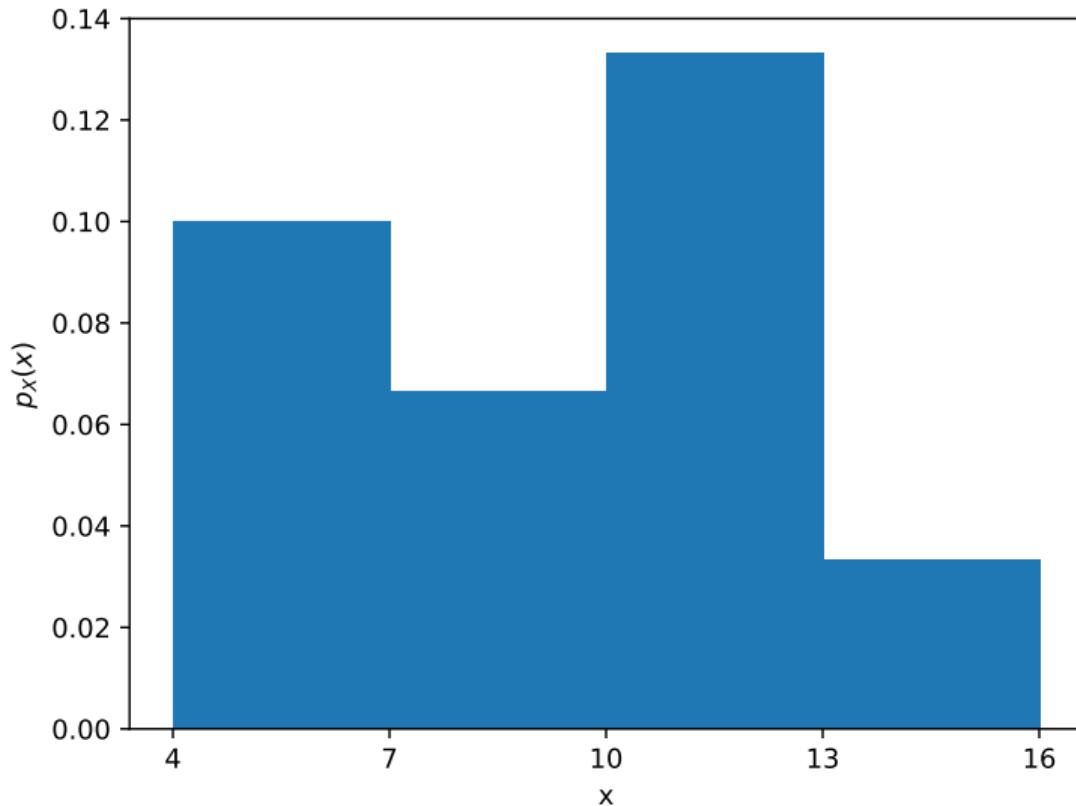
# Example



$$[a_1, b_1) = [4, 7) \quad [a_2, b_2) = [7, 10) \quad [a_3, b_3) = [10, 13) \quad [a_4, b_4) = [13, 16)$$

X	Y
7.2	0
11.3	1
8.0	1
5.1	0
5.6	1
12.3	1
13.1	1
10.9	0
12.0	1
5.0	0

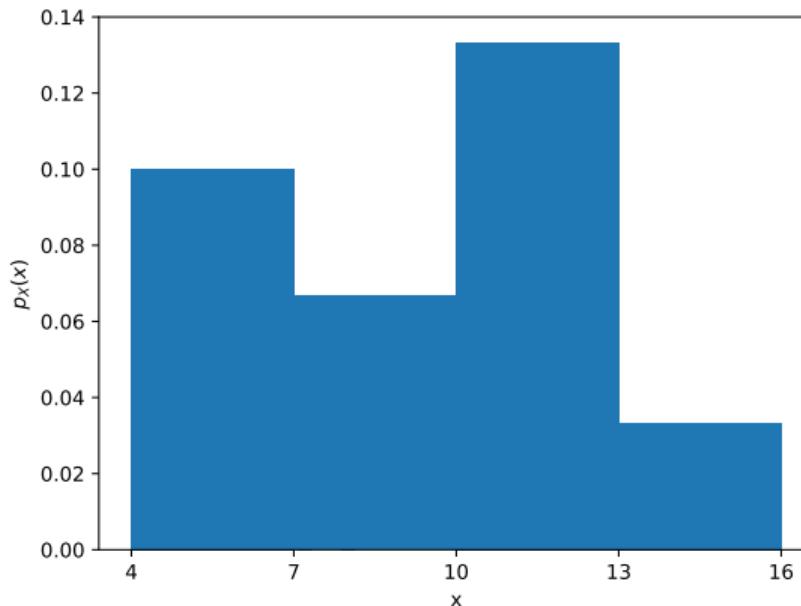
# Histogram Estimator



X	Y
7.2	0
11.3	1
8.0	1
5.1	0
5.6	1
12.3	1
13.1	1
10.9	0
12.0	1
5.0	0

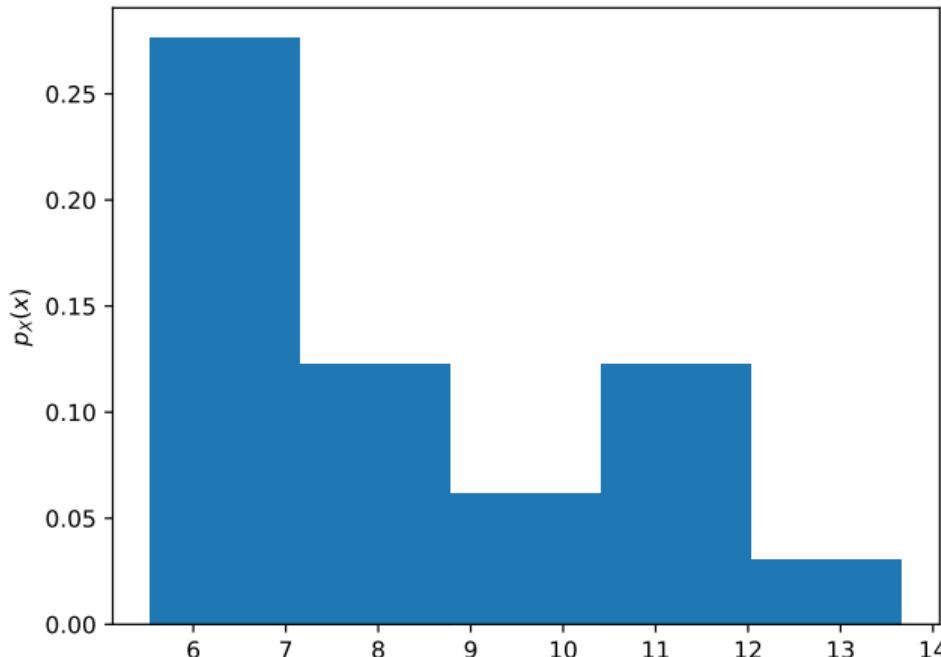
# Histogram Estimator

- ▶ Histogram estimators produce density functions.
  - ▶ E.g., what is the estimated  $p_x(4.7)$ ?
  - ▶ integrates (sums) to 1.



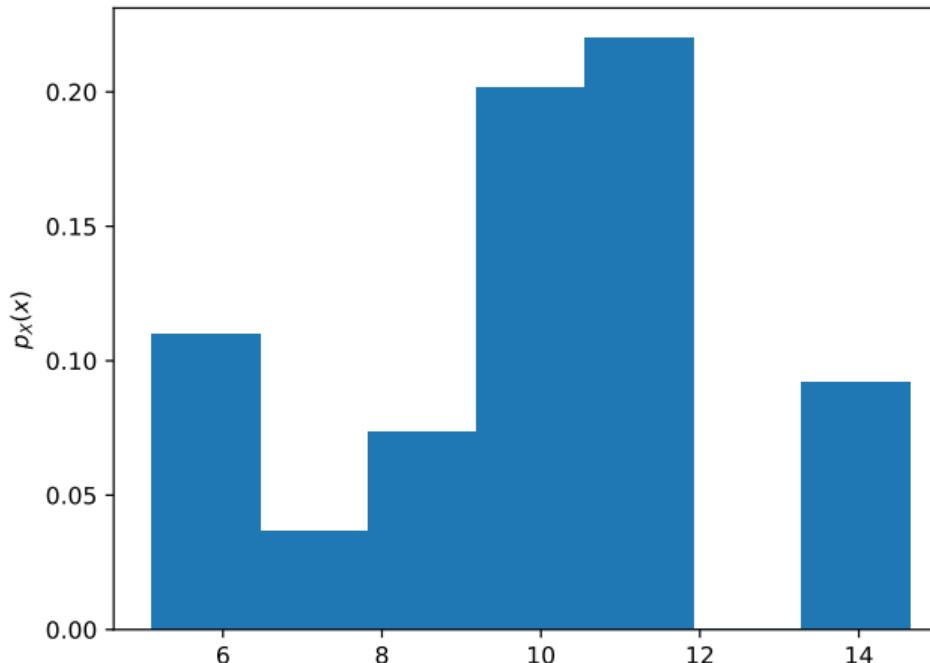
# Bin Number and Sizes

- ▶ As we get more data, we can:
  - ▶ Decrease bin width.
  - ▶ Increase number of bins.



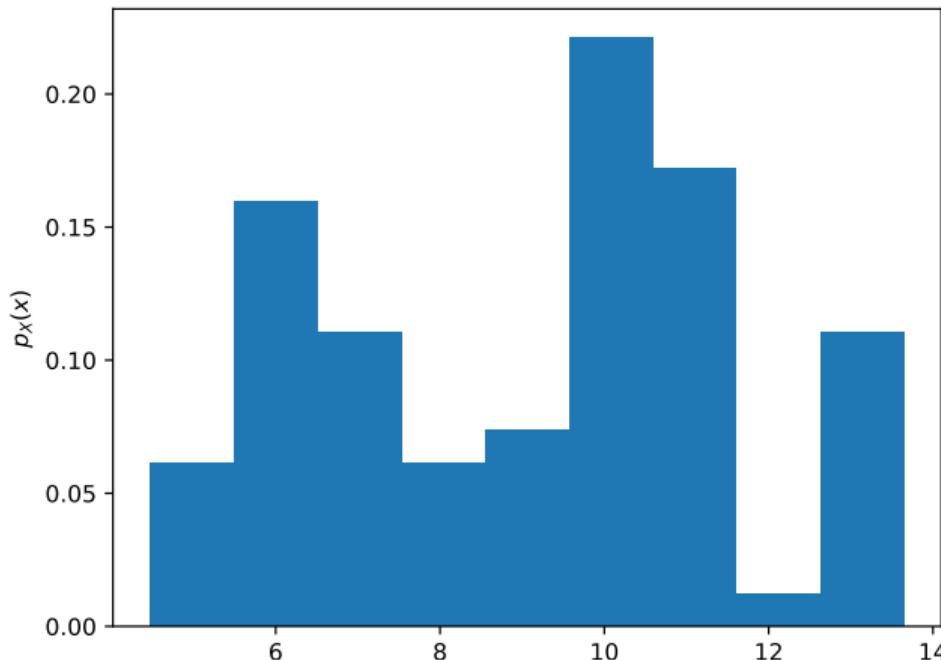
# Bin Number and Sizes

- ▶ As we get more data, we can:
  - ▶ Decrease bin width.
  - ▶ Increase number of bins.



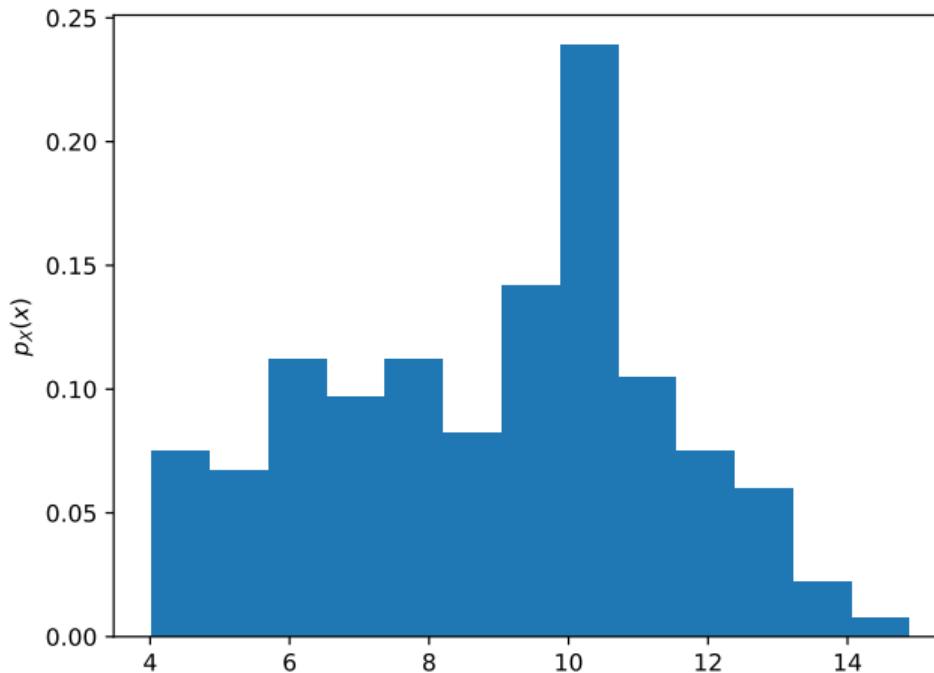
# Bin Number and Sizes

- ▶ As we get more data, we can:
  - ▶ Decrease bin width.
  - ▶ Increase number of bins.



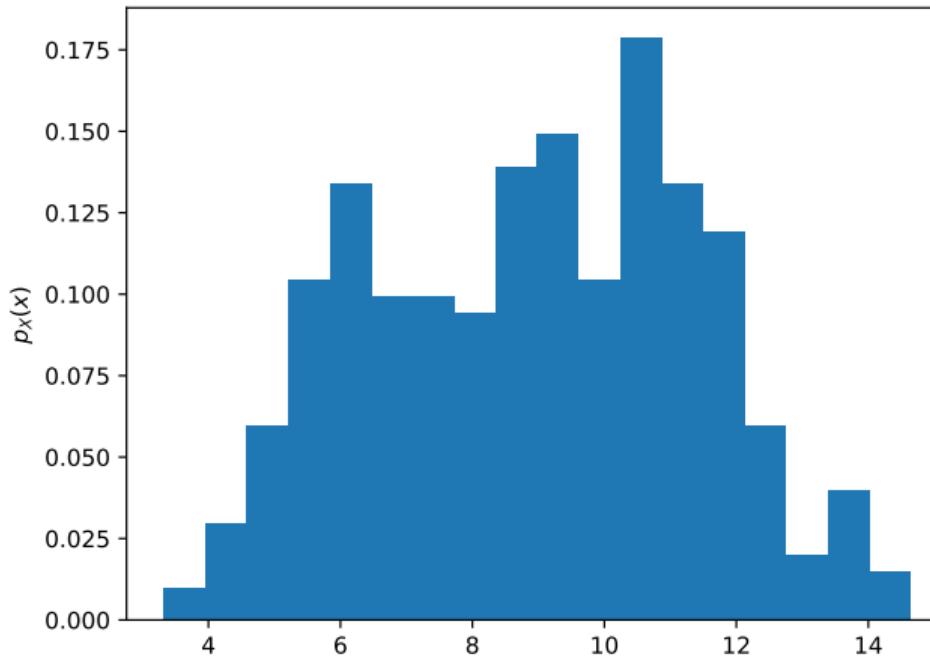
# Bin Number and Sizes

- ▶ As we get more data, we can:
  - ▶ Decrease bin width.
  - ▶ Increase number of bins.



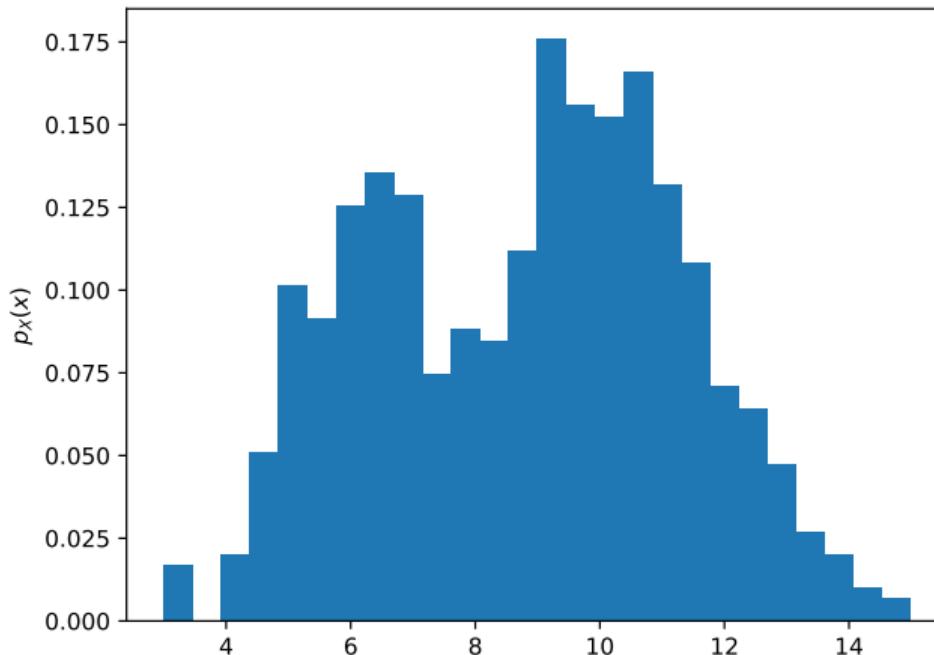
# Bin Number and Sizes

- ▶ As we get more data, we can:
  - ▶ Decrease bin width.
  - ▶ Increase number of bins.



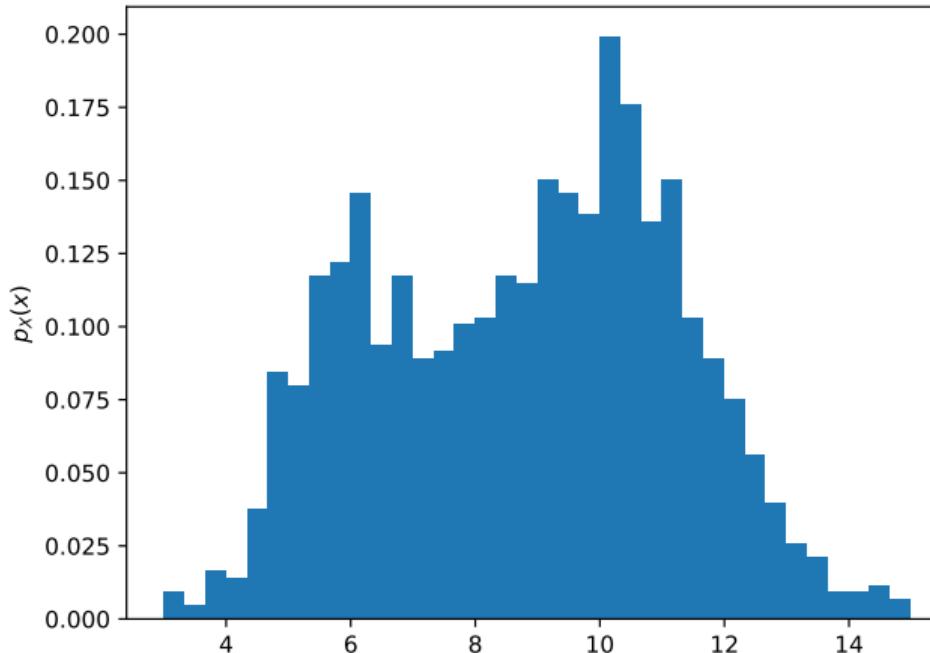
# Bin Number and Sizes

- ▶ As we get more data, we can:
  - ▶ Decrease bin width.
  - ▶ Increase number of bins.



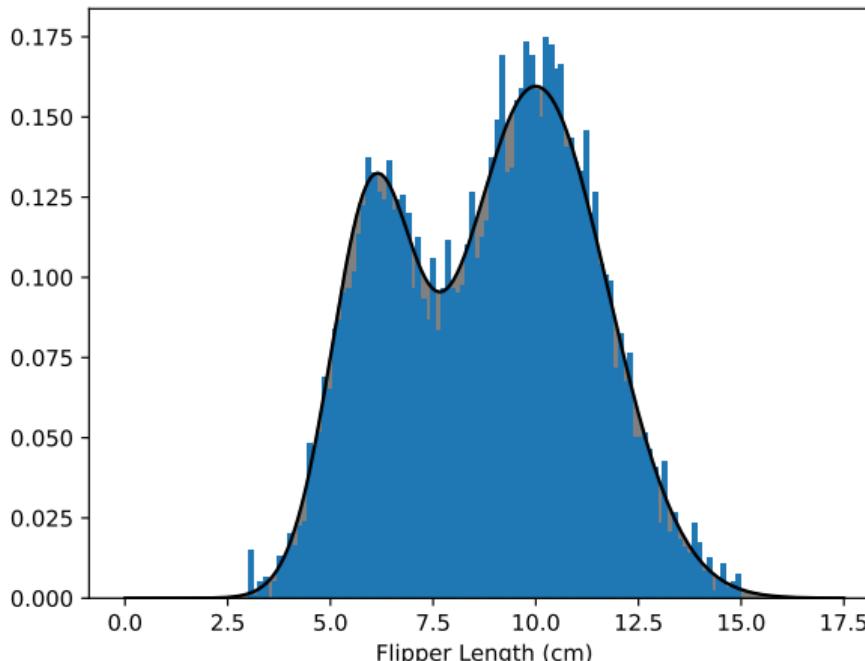
# Bin Number and Sizes

- ▶ As we get more data, we can:
  - ▶ Decrease bin width.
  - ▶ Increase number of bins.



# Law of Large Numbers

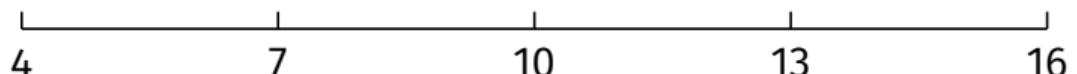
- ▶ Eventually, as  $n$  and # of bins  $\rightarrow \infty$ , the histogram estimator approaches the true density:



# Estimating Conditional Distributions

- ▶ How do we estimate  $p(x | Y = 1)$  and  $p(x | Y = 0)$ ?
  - ▶ The flipper length densities for species 1 and 0.
- ▶ Restrict to data where  $Y = 1$  (or  $Y = 0$ ) and use histogram estimator.

# Estimating $p(x | Y = y)$

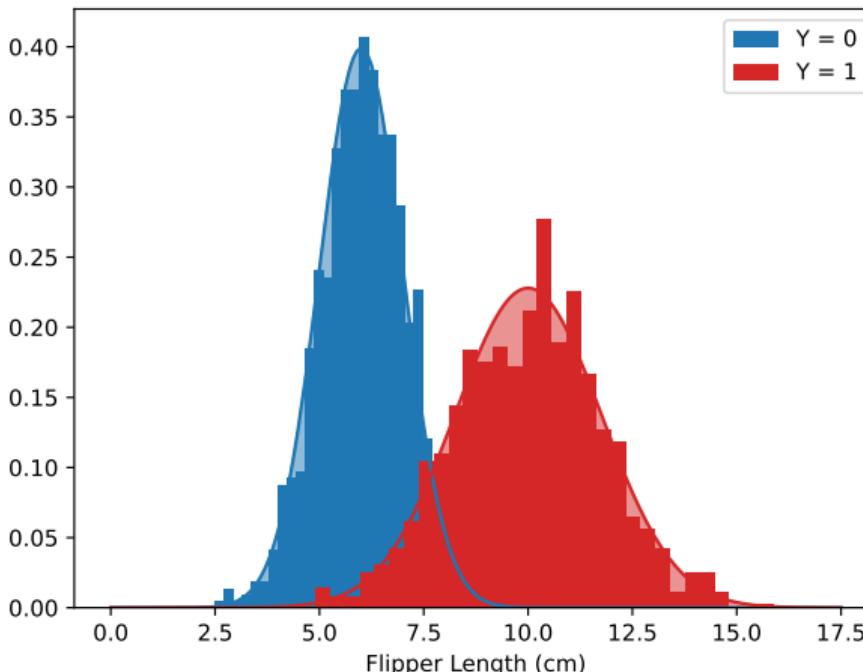


Estimate  $p(x | Y = 0)$

X	Y
7.2	0
11.3	1
8.0	1
5.1	0
5.6	1
12.3	1
13.1	1
10.9	0
12.0	1
5.0	0

# Law of Large Numbers

- ▶ Eventually, as  $n$  and # of bins  $\rightarrow \infty$ , the histogram estimators approach the true densities:



# Estimating $\mathbb{P}(Y = y | X = x)$

- ▶ How do we estimate  $\mathbb{P}(Y = y | X = x)$  with histograms?
- ▶ **Recall:** useful for making predictions.
- ▶ A discrete distribution, but conditioned on continuous variable.
  - ▶ Particular  $x$  may not be seen in data.

# Estimating $\mathbb{P}(Y = y | X = x)$

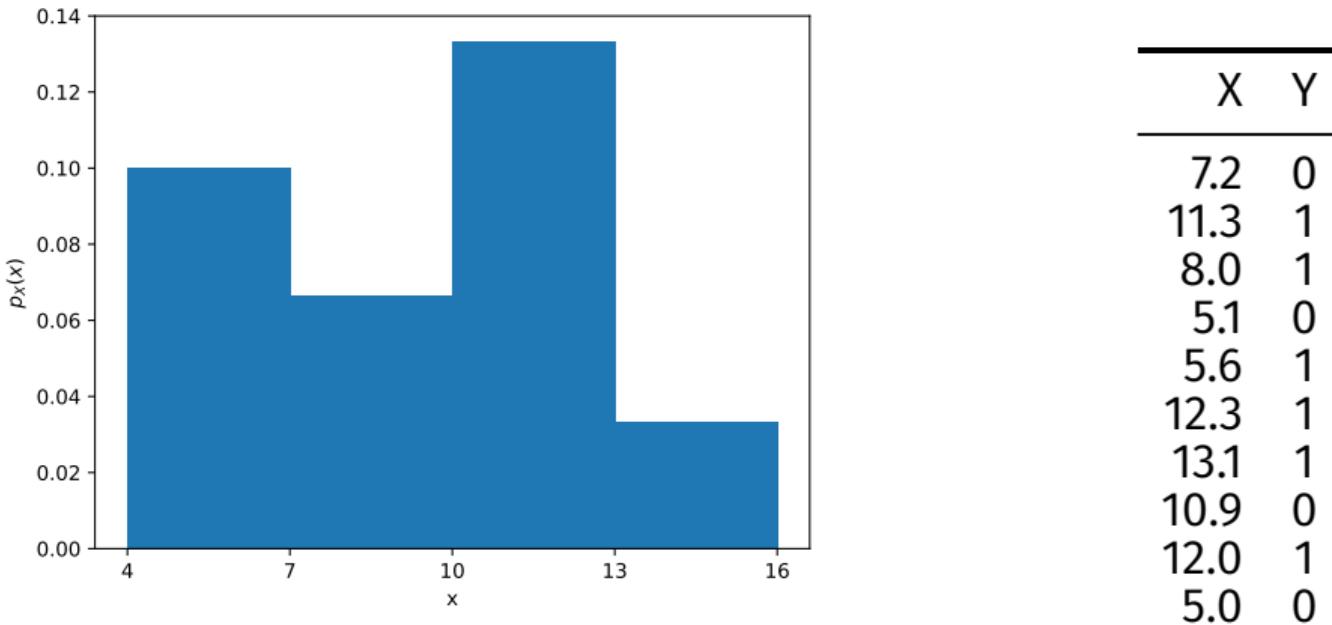
- ▶ Two equivalent approaches:
  1. Count  $\#(Y = 1)$  and  $\#(Y = 0)$  within bin containing  $x$ .
  2. Compute from Bayes' rule and other estimates.

# Approach #1: Directly

- ▶ To estimate  $\mathbb{P}(Y = y | X = x)$  with histograms when  $Y$  is discrete and  $X$  is continuous:
  1. Find the bin containing  $x$ .
  2. Estimate:

$$\mathbb{P}(Y = y | X = x) \approx \frac{\#(Y = y \text{ within this bin })}{\#(\text{points within this bin})}$$

# Approach #1: Directly



Example: estimate  $\mathbb{P}(Y = 1 | X = 4.3)$ .

## Approach #2: Bayes' Rule

1. Estimate other densities / probabilities:

$$p(x | Y = y) \quad \mathbb{P}(Y = y) \quad p_X(x)$$

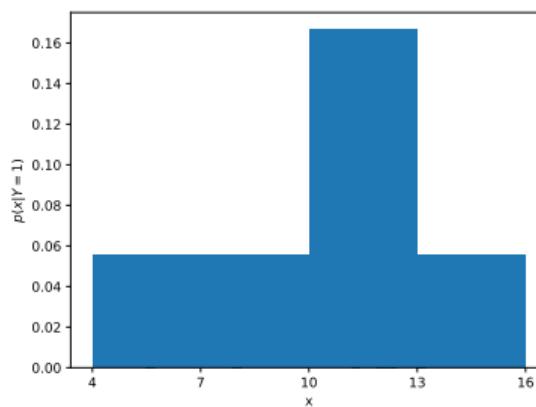
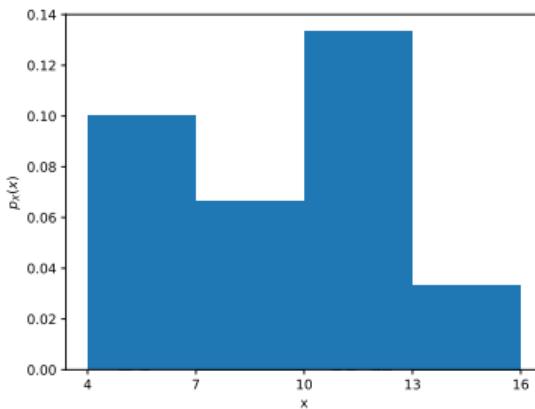
2. Use Bayes' rule to combine them:

$$\mathbb{P}(Y = y | X = x) = \frac{p(x | Y = y) \mathbb{P}(Y = y)}{p_X(x)}$$

# Approach #2: Bayes' Rule

- ▶ Using Bayes' rule:

$$\mathbb{P}(Y = y | X = x) = \frac{p(x | Y = y)\mathbb{P}(Y = y)}{p_X(x)}$$



X	Y
7.2	0
11.3	1
8.0	1
5.1	0
5.6	1
12.3	1
13.1	1
10.9	0
12.0	1
5.0	0

Example: estimate  $\mathbb{P}(Y = 1 | X = 4.3)$ .

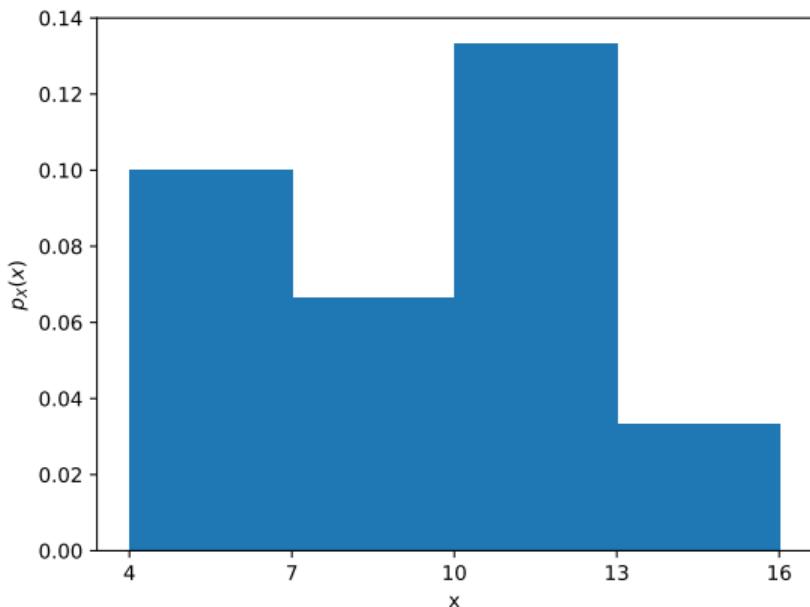
# Equivalence

- ▶ Both approaches produce the same answer if same bins used to estimate all densities.
- ▶ Related via Bayes' rule.

# Prediction

- ▶ Suppose there are two species of penguin; one species tends to have longer flippers.
- ▶ **Goal:** given a new penguin with flipper length  $X = x$ , predict its species,  $Y$ .

# Example

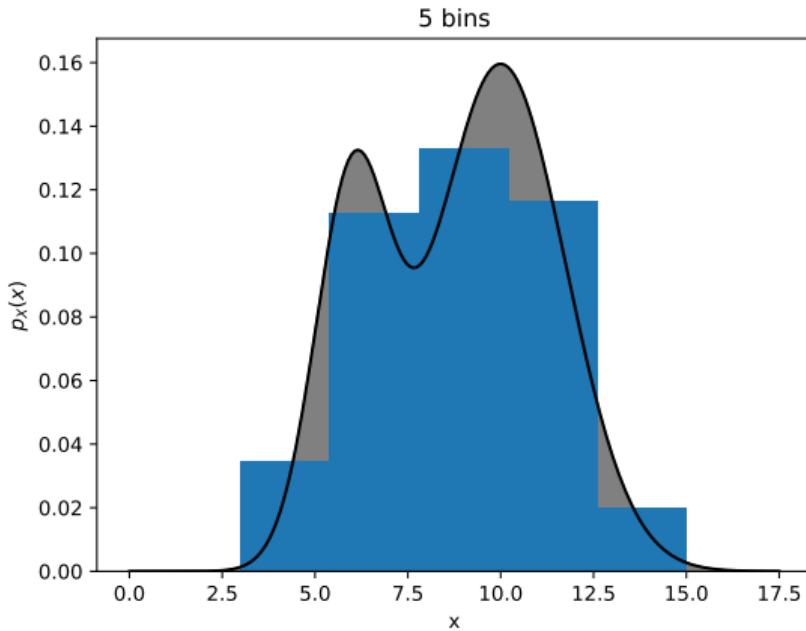


X	Y
7.2	0
11.3	1
8.0	1
5.1	0
5.6	1
12.3	1
13.1	1
10.9	0
12.0	1
5.0	0

Example: what is predicted species when  $X = 10.8$ ?

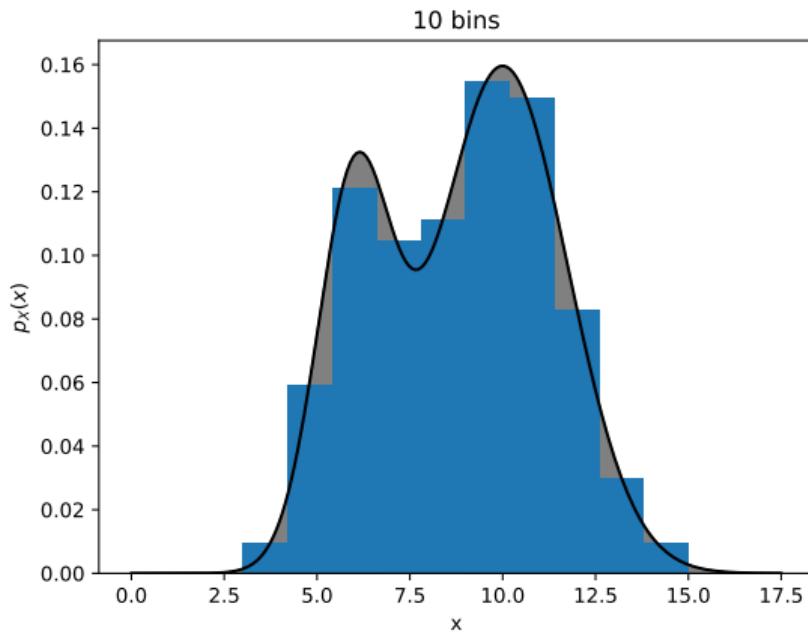
# Over- and Under-fitting

- ▶ The number of bins must be chosen appropriately to avoid over- or under-fitting.



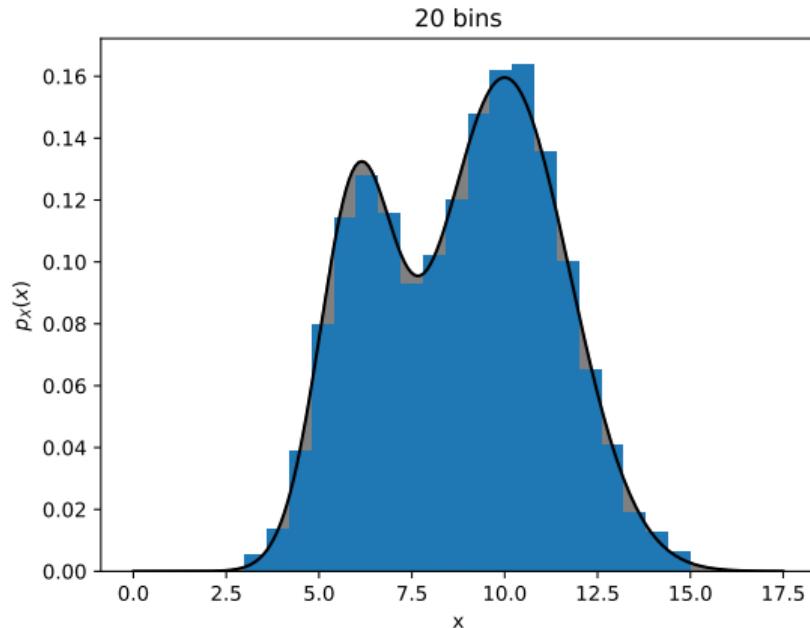
# Over- and Under-fitting

- ▶ The number of bins must be chosen appropriately to avoid over- or under-fitting.



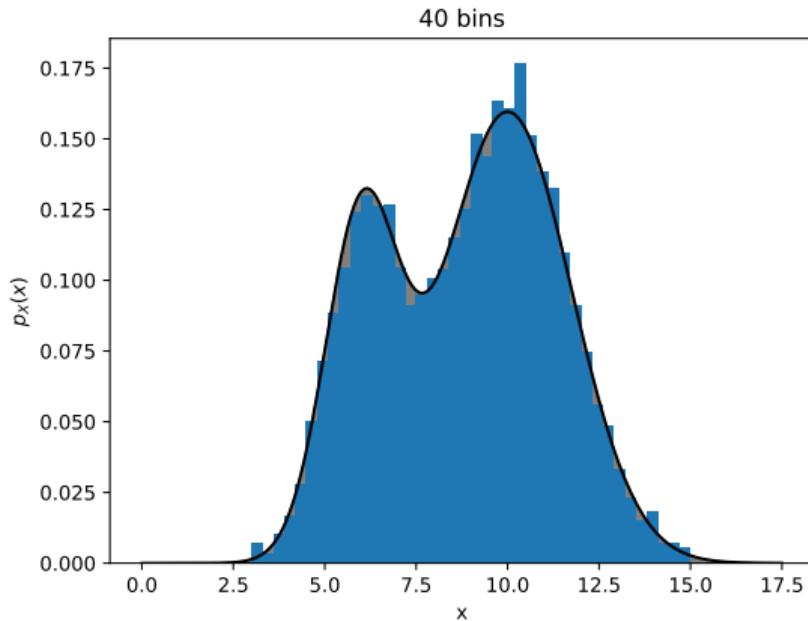
# Over- and Under-fitting

- The number of bins must be chosen appropriately to avoid over- or under-fitting.



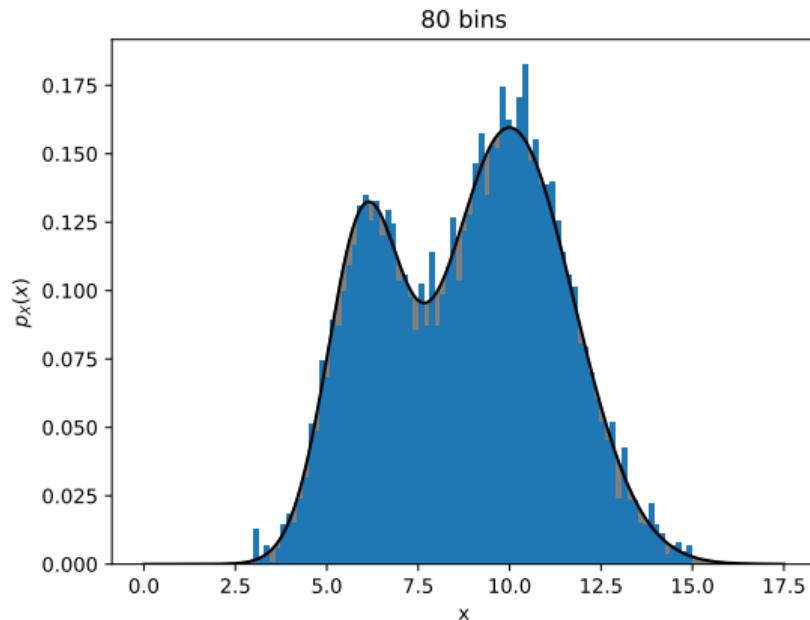
# Over- and Under-fitting

- ▶ The number of bins must be chosen appropriately to avoid over- or under-fitting.



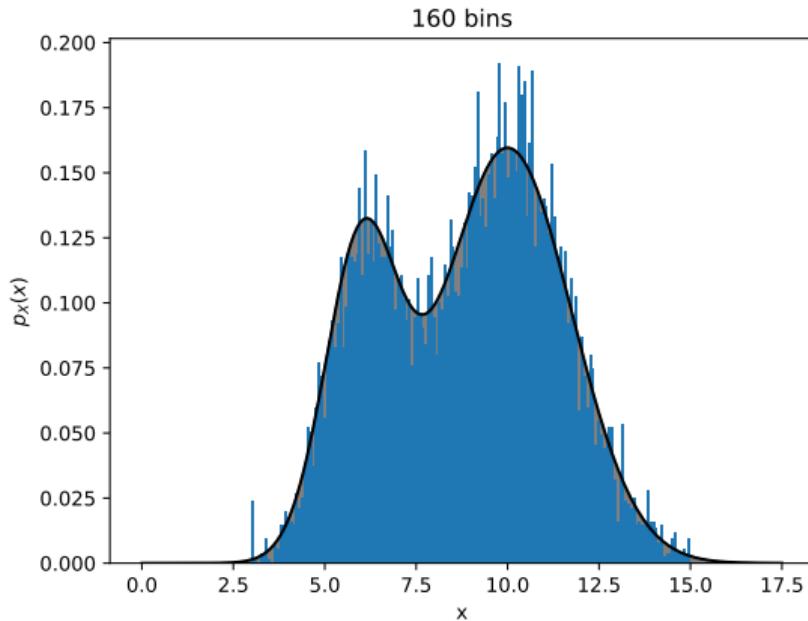
# Over- and Under-fitting

- ▶ The number of bins must be chosen appropriately to avoid over- or under-fitting.



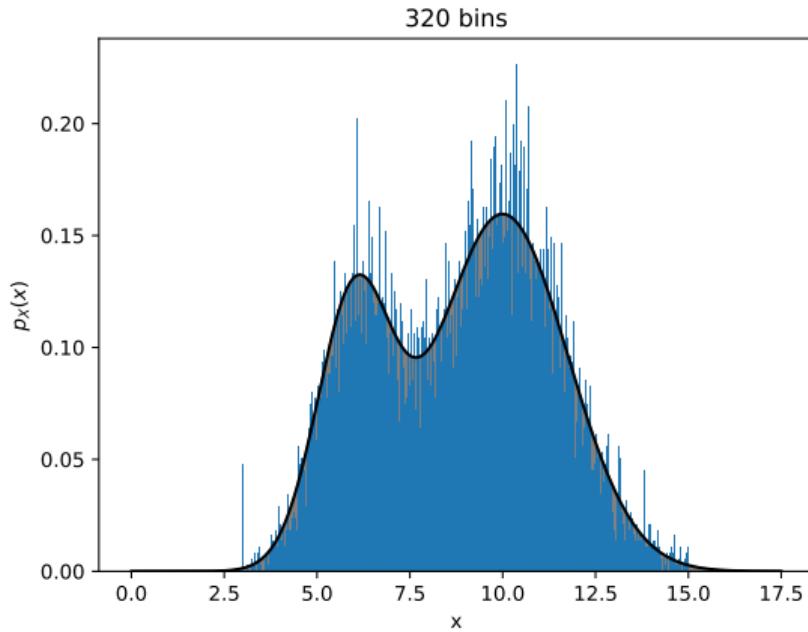
# Over- and Under-fitting

- ▶ The number of bins must be chosen appropriately to avoid over- or under-fitting.



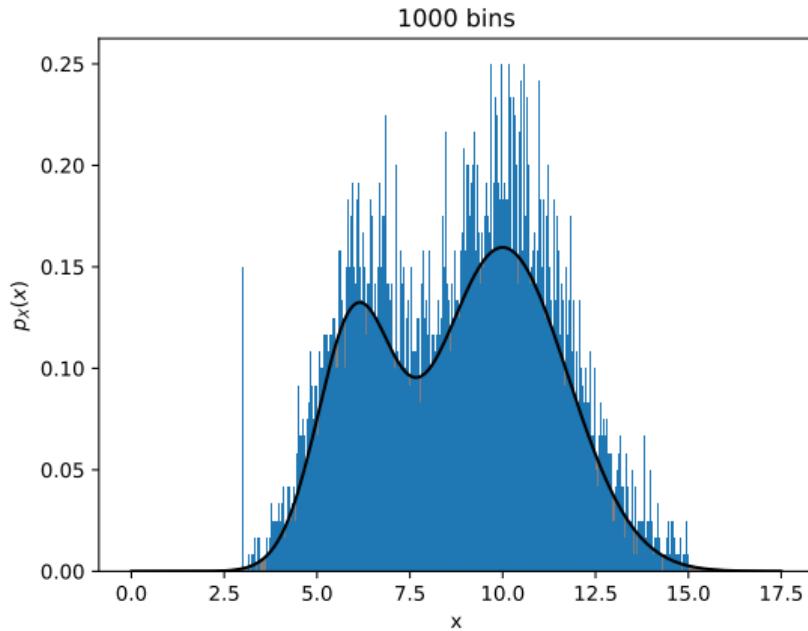
# Over- and Under-fitting

- ▶ The number of bins must be chosen appropriately to avoid over- or under-fitting.



# Over- and Under-fitting

- ▶ The number of bins must be chosen appropriately to avoid over- or under-fitting.



# DSC 140A

*Probabilistic Modeling & Machine Learning*

Lecture 8 | Part 3

## Multivariate Histogram Density Estimators

# Multivariate Estimation

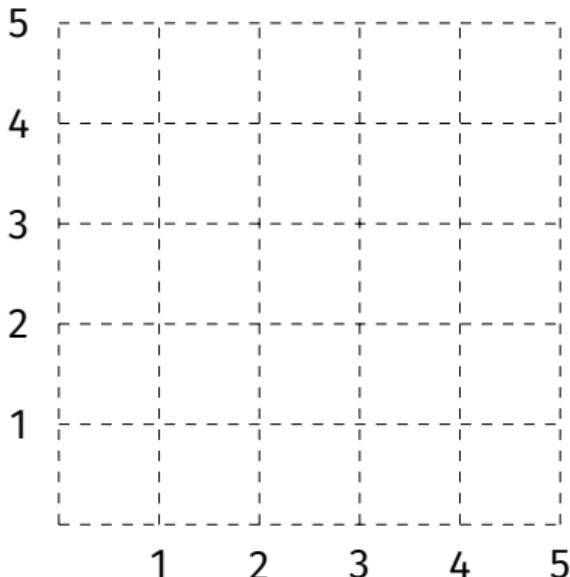
- ▶ In practice, we typically want to predict  $Y$  from many variables,  $X_1, X_2, \dots$
- ▶ How do we estimate densities  $p(\vec{x})$  of several variables?

# Histogram Estimators

- ▶ Histograms naturally generalize to  $d > 1$ :
- ▶ Suppose data  $\vec{x}^{(1)}, \dots, \vec{x}^{(n)}$  came from density  $f$
- ▶ Divide  $\mathbb{R}^d$  into rectangular bins **bins** with regular side-lengths  $\ell_1, \ell_2, \dots, \ell_d$
- ▶ Within a bin, estimate density:

$$f(\vec{x}) \text{ within bin} \approx \frac{\# \text{ data points } \in [a_i, b_i)}{n \times \underbrace{(\ell_1 \times \ell_2 \times \dots \times \ell_d)}_{\text{"bin volume"}}$$

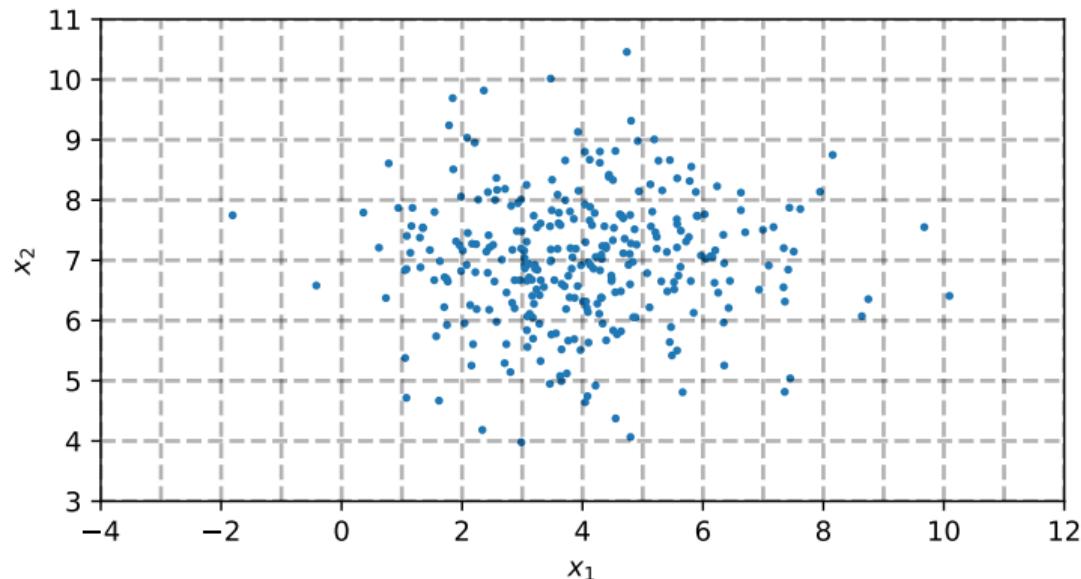
# Example: $d = 2$



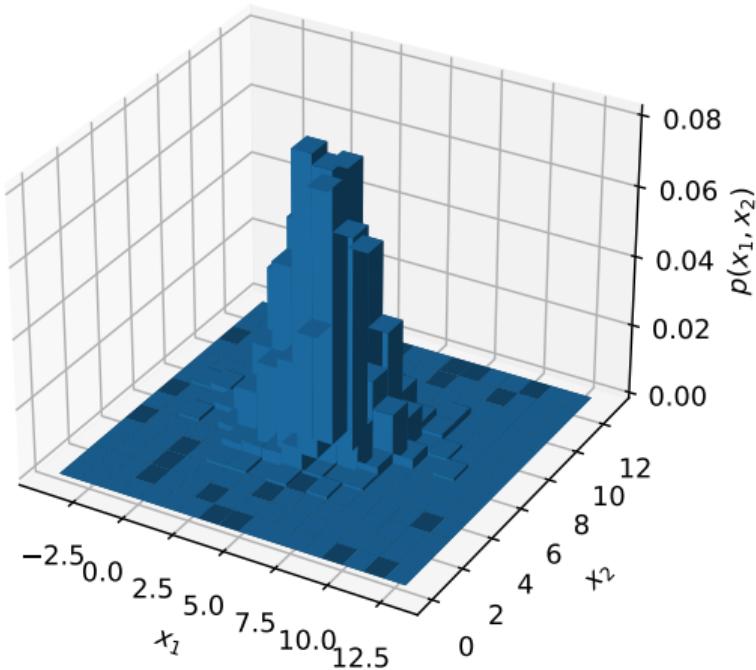
$X_1$	$X_2$	Y
4.1	1.8	0
3.6	3.0	0
4.2	2.2	1
4.2	2.4	1
2.3	3.2	0
4.9	2.4	1
2.1	0.8	1
3.2	1.1	1
4.7	2.3	0
3.8	4.9	0

E.g., estimate: 1)  $p_{x_1, x_2}(2.3, 2.5)$     2)  $p_{x_1, x_2}(3.3, 2.5)$

# Estimating 2-d Densities



# Estimating 2-d Densities



# Estimating in High Dimensions

- ▶ Histogram estimators can be used to estimate high-dimensional densities, *in principle.*
  - ▶ That is, densities of many continuous variables.
- ▶ But they typically do not work well due to the **curse of dimensionality.**

# Curse of Dimensionality

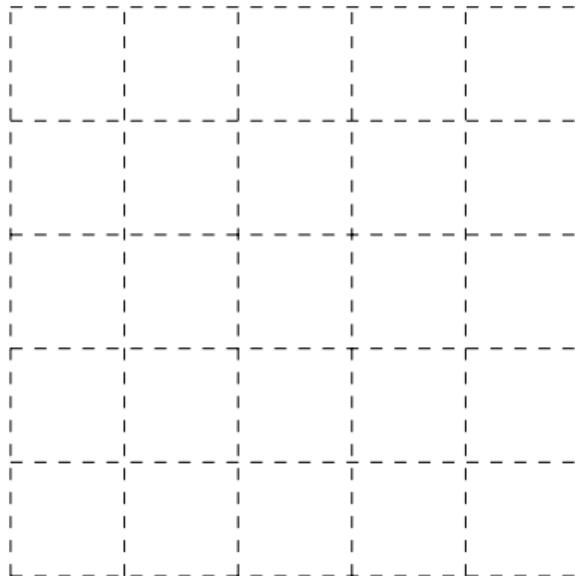
- ▶ Intuition: need sufficiently-many points in each bin to make good estimates.
  - ▶ Law of large numbers.
- ▶ Number of points needed is proportional to number of bins.
- ▶ **Many** bins in high dimensions.

# Curse of Dimensionality

- ▶ Suppose we have two continuous variables,  $X_1$  and  $X_2$ , each taking values between 0 and 1.
- ▶ Divide each feature into 5 equal bins:

0    0.2    0.4    0.6    0.8    1

# Curse of Dimensionality



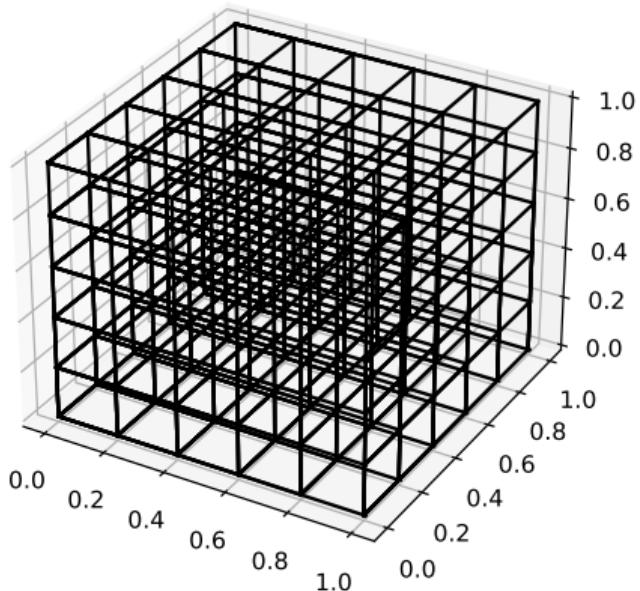
- ▶ Total number of bins:  $5 \times 5 = 25$

# Curse of Dimensionality

- ▶ Suppose we have two continuous variables,  $X_1$ ,  $X_2$ ,  $X_3$ , each taking values between 0 and 1.
- ▶ Divide each feature into 5 equal bins:

0    0.2    0.4    0.6    0.8    1

# Curse of Dimensionality



- ▶ Total number of bins:  $5 \times 5 \times 5 = 5^3 = 125$

# Curse of Dimensionality

- ▶ With  $d$  features, we'd have  $5^d$  bins.
- ▶ Example: with 20 features, we'd have

$$5^{20} \approx 10 \text{ trillion}$$



Put that in perspective using more familiar quantities.



10<sup>13</sup>, which is 10 trillion, is a very large quantity. Here are some ways to put it into perspective using more familiar quantities:



- Time: 10 trillion seconds is about 317,000 years. That's longer than human beings have been around!
- Distance: 10 trillion centimeters is about 106 light years, the distance light travels in 106 years. For comparison, the nearest star to our solar system, Proxima Centauri, is about 4.24 light years away.
- Money: 10 trillion dollars is more than the entire annual GDP of many countries, including Canada, Australia, and South Korea.
- Population: 10 trillion people is more than the total number of people who have ever lived on Earth. It's estimated that the total number of human beings who have ever lived is around 100 billion.

These examples show just how large 10 trillion is, and how it compares to other quantities we might be more familiar with.

# Curse of Dimensionality

- ▶ To accurately estimate densities in more than a few dimensions, we need **too much data**.
- ▶ Most bins will be empty.
- ▶ And so we take different approaches.

# A Different Approach

- ▶ Histogram estimators don't make assumptions about the **shape** of the density.
  - ▶ **Good:** very flexible.
  - ▶ **Bad:** requires a lot of data.
- ▶ **Next:** Assume a particular shape (e.g., a Gaussian) and try to learn it from data.

# DSC 140A

*Probabilistic Modeling & Machine Learning*

Lecture 12 | Part 1

**Parametric Density Estimation**

# Bayes Classifier

- ▶ Recall the **Bayes Classifier**: predict

$$\begin{cases} 1, & \text{if } \mathbb{P}(Y = 1 | \vec{X} = \vec{x}) > \mathbb{P}(Y = 0 | \vec{X} = \vec{x}), \\ 0, & \text{otherwise.} \end{cases}$$

- ▶ Equivalently, using **Bayes' rule**:

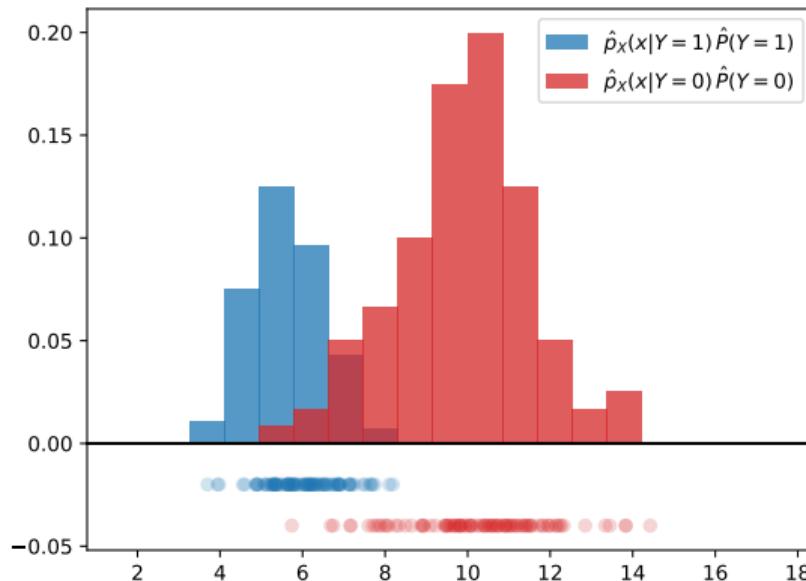
$$\begin{cases} 1, & \text{if } p_X(x | Y = 1)\mathbb{P}(Y = 1) > p_X(x | Y = 0)\mathbb{P}(Y = 0), \\ 0, & \text{otherwise.} \end{cases}$$

# Estimating Densities

- ▶ We rarely know the true distribution.
- ▶ We must **estimate** it from data.
- ▶ When  $\vec{X}$  is continuous, we estimate **density**.

# Last Time: Histogram Estimators

- ▶ **Histograms** provide one way of estimating densities.



# Histogram Drawbacks

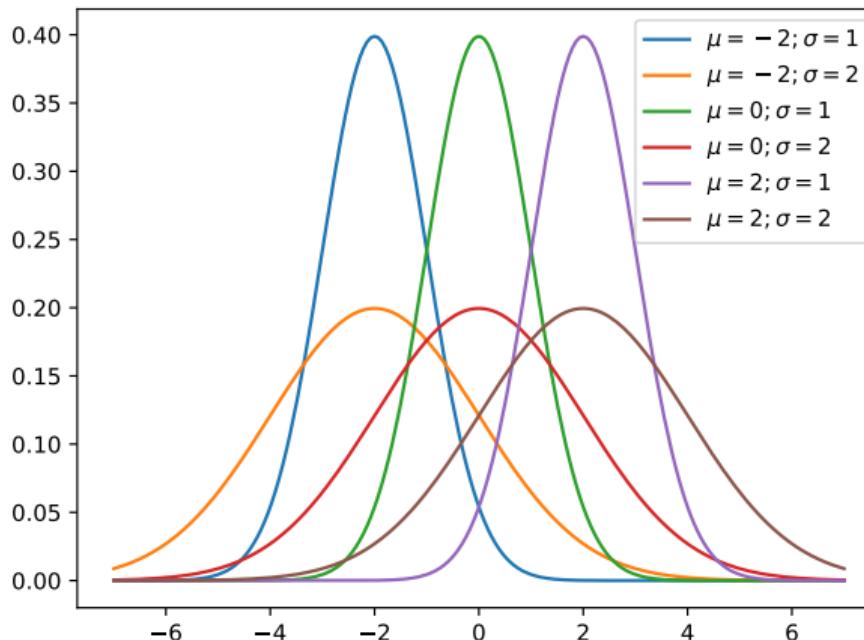
- ▶ We saw that histograms need massive amounts of data in high dimensions.
- ▶ The **Curse of Dimensionality**.

# Observation

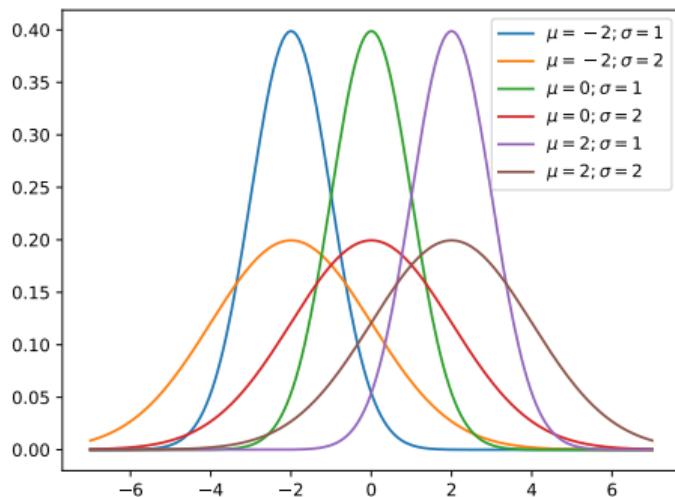
- ▶ Histogram estimators assume nothing about the **shape** of the true density.
- ▶ This makes them very flexible, but also data-hungry.
- ▶ **Idea:** Assume that the true, underlying density has a certain form.

# Example: Gaussians

- ▶ Often assume that the true distribution is **Gaussian** (aka, **Normal**).



# Example: Gaussians



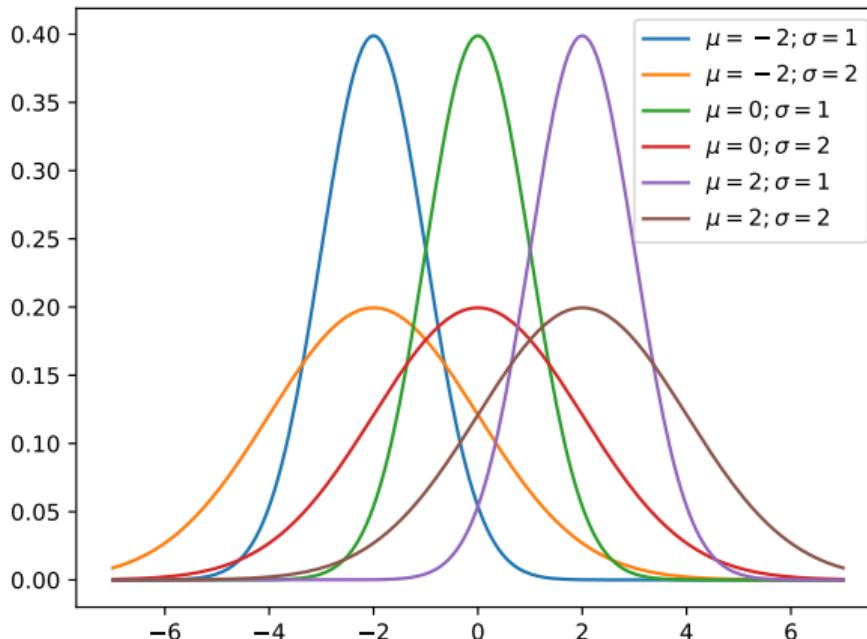
► **Recall:** the pdf of the Gaussian distribution:

$$p(x; \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

►  $\mu$  and  $\sigma$  are **parameters**  
►  $\mu$  controls center  
►  $\sigma$  controls width

# Gaussian

- ▶ **Central Limit Theorem:** sums of independent random variables are Gaussian
- ▶ **Examples:** test scores, heights, measurement errors, ...



# Parametric Distributions

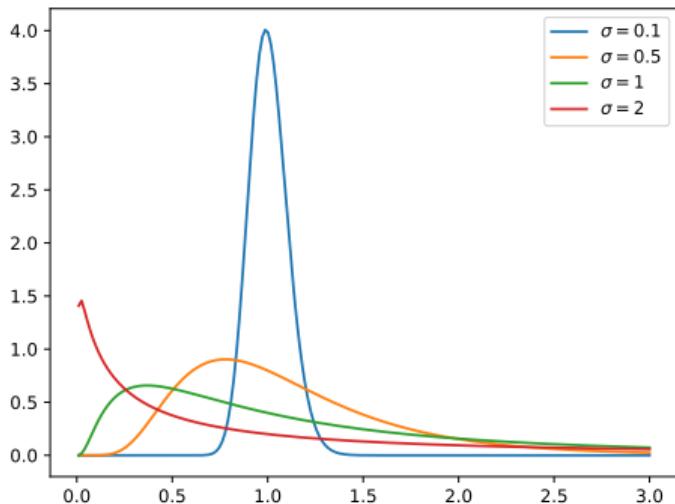
- ▶ A **parametric distribution** is **totally determined** by a finite number of **parameters**.
- ▶ **Example:** knowing  $\mu$  and  $\sigma$  tells you everything about a Gaussian distribution.

# Other Parametric Distributions

- ▶ There are many parametric distributions.
- ▶ **Discrete:** Bernoulli, Multinomial, Poisson, ...
- ▶ **Continuous:** Log-normal, Gamma, Pareto, ...

# Example: Lognormal

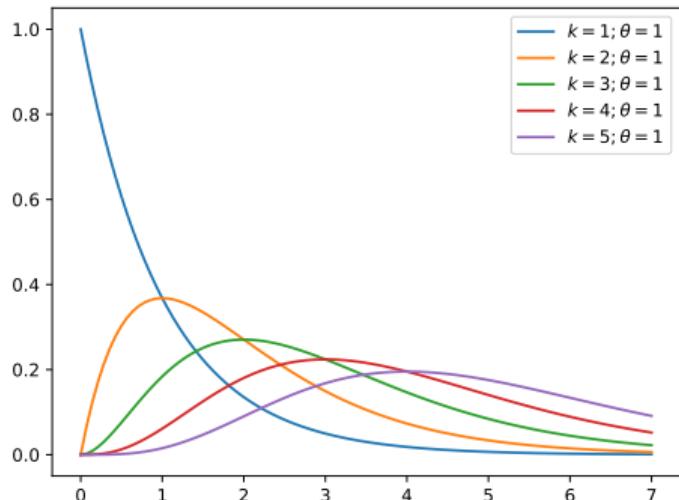
- ▶ Product of many independent positive random numbers.
- ▶ **Example:** length of comments in an internet forum



$$p(x; \mu, \sigma) = \frac{1}{x\sigma\sqrt{2\pi}} \exp\left(-\frac{(\ln x - \mu)^2}{2\sigma^2}\right)$$

# Example: Gamma

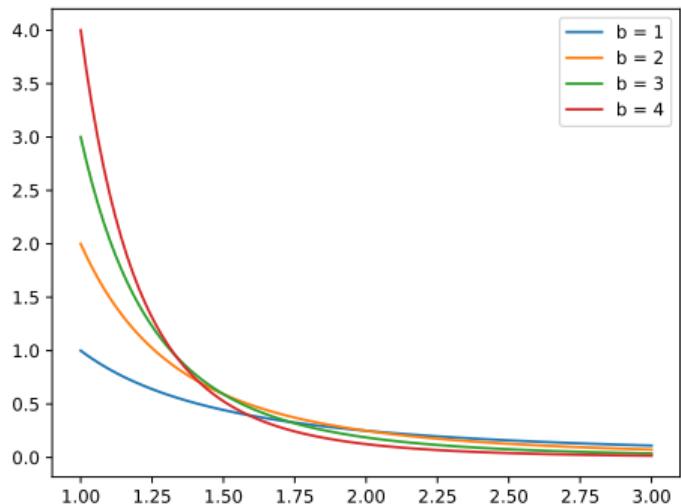
- **Examples:** wait times, size of rainfalls, insurance claims, ...



$$p(x; k, \theta) = \frac{1}{\Gamma(k)\theta^k} x^{k-1} e^{-x/\theta}$$

# Example: Pareto

- **Examples:** distribution of wealth, size of meteorites, ...



$$p(x; x_m, \alpha) = \frac{\alpha x_m^\alpha}{x^{\alpha+1}}$$

# Parametric Density Estimation

- ▶ In **parametric density estimation**, we assume data comes from some parametric density.
  - ▶ E.g., Gaussian, Log-Normal, Pareto, etc.
- ▶ But we don't know the parameters.
- ▶ Use data to **estimate** the parameters.

# Non-Parametric Density Estimation

- ▶ Contrast this with estimating density with histograms.
- ▶ There were no parameters controlling the shape of the density.
- ▶ Histograms are **non-parametric** density estimators.

# DSC 140A

*Probabilistic Modeling & Machine Learning*

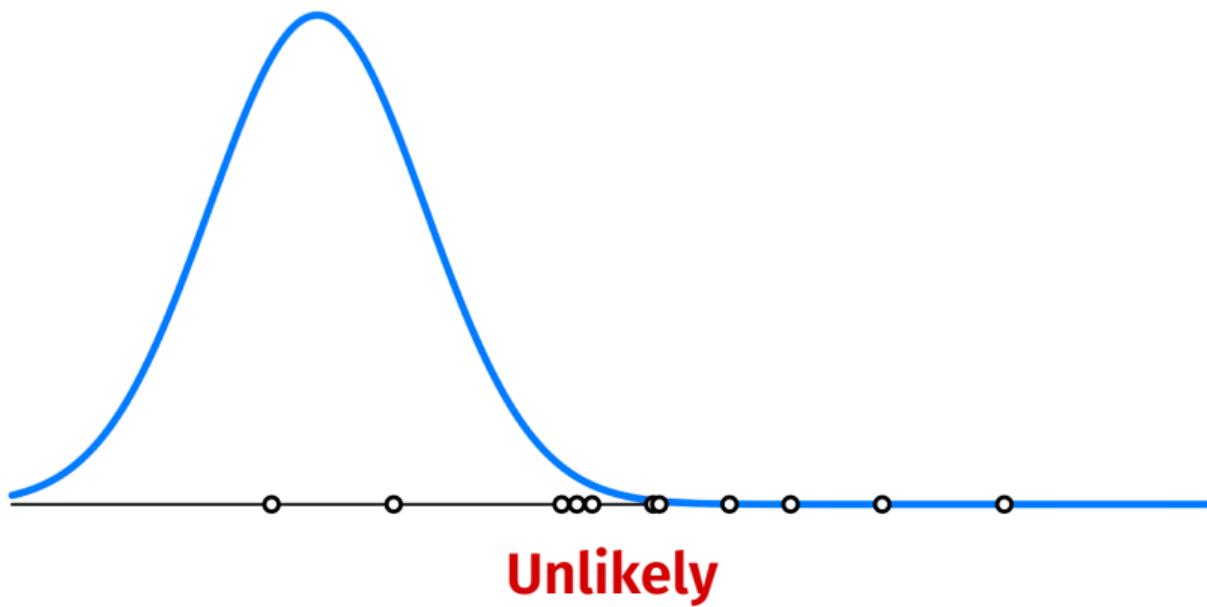
Lecture 12 | Part 2

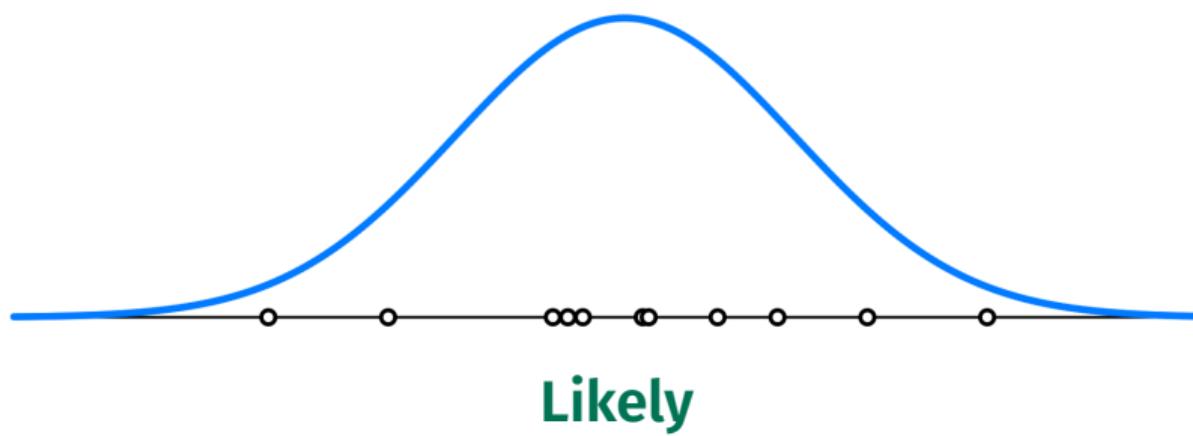
## Maximum Likelihood Estimation

# Parametric Density Estimation

- ▶ Suppose we have data  $x^{(1)}, \dots, x^{(n)} \in \mathbb{R}$ .
- ▶ Assume it came from a parametric distribution.
  - ▶ Say, a Gaussian.
- ▶ What were the parameter values used to generate the data?
- ▶ Using data to guess  $\mu$  and  $\sigma$  is called **estimating** the parameters.





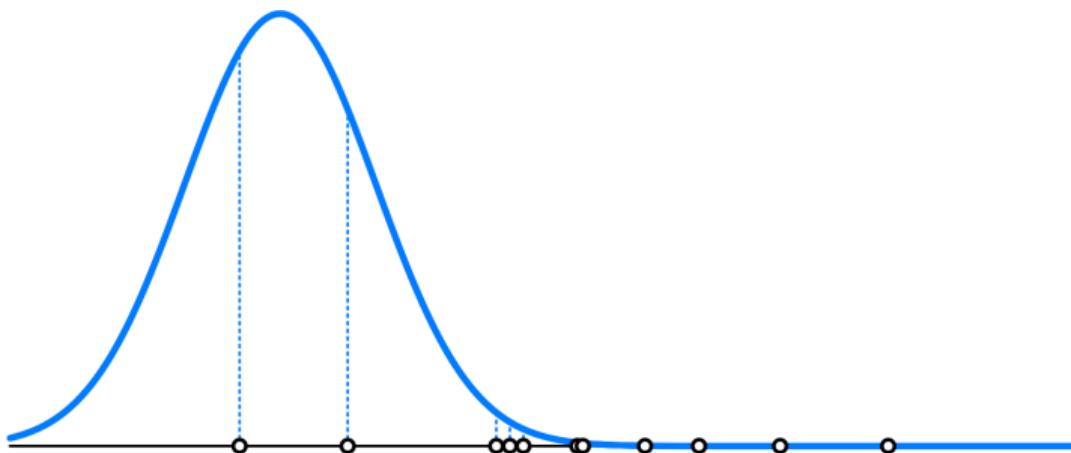


# Intuition

- ▶ Some parameter choices seem **more likely** than others.
- ▶ That is, there is a greater chance that the data could have been generated by them.
- ▶ How can we quantify this?

# Intuition

- ▶ Let  $p$  be the Gaussian probability density function.
- ▶  $p(x^{(i)}; \mu, \sigma)$  quantifies how likely it is to see  $x^{(i)}$  if parameters  $\mu$  and  $\sigma$  are used.



## Exercise

Assume that  $x^{(1)}, \dots, x^{(n)}$  are all sampled independently from a density with parameters  $\mu, \sigma$ .

Think of  $p(x^{(i)}; \mu, \sigma)$  as the “chance” of seeing  $x^{(i)}$  under parameters  $\mu$  and  $\sigma$ .

What is the chance of seeing  $x^{(1)}$  and  $x^{(2)}$  and  $x^{(3)}$  and ... and  $x^{(n)}$ ?

# Intuition

- ▶  $p(x^{(1)}; \mu, \sigma) \times p(x^{(2)}; \mu, \sigma) \times \dots \times p(x^{(n)}; \mu, \sigma)$  quantifies likelihood of seeing  $x^{(1)}, \dots, x^{(n)}$  simultaneously.
- ▶ In fact, it is the **joint density** of the data.
- ▶ But instead think of this as a function of  $\mu$  and  $\sigma$ .

# Likelihood

- ▶ The **likelihood** of  $\mu$  and  $\sigma$  with respect to data  $x^{(1)}, \dots, x^{(n)}$  is:

$$\begin{aligned}\mathcal{L}(\mu, \sigma; x^{(1)}, \dots, x^{(n)}) &= p(x^{(1)}; \mu, \sigma) \times p(x^{(2)}; \mu, \sigma) \times \cdots \times p(x^{(n)}; \mu, \sigma) \\ &= \prod_{i=1}^n p(x^{(i)}; \mu, \sigma)\end{aligned}$$

# Likelihood

- ▶ The likelihood function takes in parameters  $\mu$  and  $\sigma$  and returns a real number.
- ▶ **Interpretation:** likelihood that data was generated by this choice of  $\mu$  and  $\sigma$ .
- ▶ **Goal:** find  $\mu$  and  $\sigma$  that **maximize** the likelihood.

<http://dsc140a.com/static/vis/mle/>

# Maximizing Likelihood

- ▶ To maximize  $\mathcal{L}(\mu, \sigma)$ , we might take derivatives  $\frac{\partial \mathcal{L}}{\partial \mu}$  and  $\frac{\partial \mathcal{L}}{\partial \sigma}$ , set to 0, solve.
- ▶ But the likelihood is often difficult to work with.

# Example: Gaussian

- ▶ Assume that  $p$  is the Gaussian pdf.

$$p(x; \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

- ▶ Then the likelihood function is:

$$\mathcal{L}(\mu, \sigma) = \prod_{i=1}^n \left( \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x^{(i)}-\mu)^2}{2\sigma^2}} \right)$$

# Log Likelihood

- ▶ It is typically easier to work with the **log likelihood** instead.

$$\tilde{\mathcal{L}}(\mu, \sigma) = \ln \mathcal{L}(\mu, \sigma)$$

- ▶ **Fact:** Because  $\ln x$  is **monotonically increasing**, a maximizer of  $\ln \mathcal{L}$  also maximizes  $\mathcal{L}$

## Procedure: Gaussian

1. Write the log likelihood function  $\tilde{\mathcal{L}}$ .
2. Take derivatives  $\partial\tilde{\mathcal{L}}/\partial\mu$  and  $\partial\tilde{\mathcal{L}}/\partial\sigma$
3. Set to zero and solve for  $\mu$  and  $\sigma$ .

## Recall: Log Properties

- ▶ If  $a$  and  $b$  are positive:  $\ln(a \times b) = \ln a + \ln b$
- ▶ If  $a$  and  $b$  are positive:  $\ln(a/b) = \ln a - \ln b$
- ▶ If  $a$  is positive:  $\ln a^p = p \ln a$

# Step 1: Write Log Likelihood

- ▶ Write the log likelihood function for the Normal distribution.

## Step 2: Differentiate

- ▶ We have:  $\tilde{\mathcal{L}} = \sum_{i=1}^n \left[ -\ln \sigma - \ln \sqrt{2\pi} - \frac{(x^{(i)} - \mu)^2}{2\sigma^2} \right]$
- ▶ Compute  $\partial \tilde{\mathcal{L}} / \partial \mu$ :

## Step 2: Differentiate

- ▶ We have:  $\tilde{\mathcal{L}} = \sum_{i=1}^n \left[ -\ln \sigma - \ln \sqrt{2\pi} - \frac{(x^{(i)} - \mu)^2}{2\sigma^2} \right]$
- ▶ Compute  $\partial \tilde{\mathcal{L}} / \partial \sigma$ :

## Step 3: Solve

- ▶ We have  $\partial \tilde{L} / \partial \mu = \frac{1}{\sigma^2} \sum_{i=1}^n (x^{(i)} - \mu)$
- ▶ Solve  $\partial \tilde{L} / \partial \mu = 0$  for  $\mu$ .

## Step 3: Solve

- ▶ We have  $\partial \tilde{L} / \partial \sigma = \sum_{i=1}^n \left[ -\frac{1}{\sigma} + \frac{(x^{(i)} - \mu)^2}{\sigma^3} \right]$
- ▶ Solve  $\partial \tilde{L} / \partial \sigma = 0$  for  $\sigma$ .

# MLEs for Gaussian Distribution

- We have found the **maximum likelihood estimates** for the Gaussian distribution:

$$\mu_{\text{MLE}} = \frac{1}{n} \sum_{i=1}^n x^{(i)} \quad \sigma_{\text{MLE}} = \sqrt{\frac{1}{n} \sum_{i=1}^n (x^{(i)} - \mu_{\text{MLE}})^2}$$

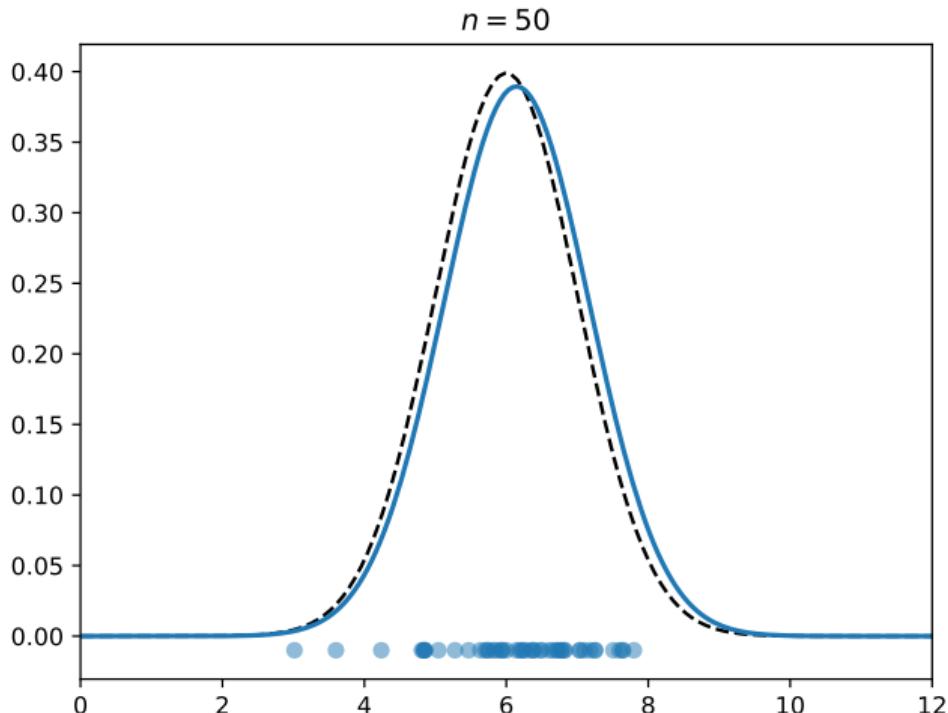
# “Fitting” a Gaussian

- ▶ Suppose we wish to “fit” a Gaussian to data  $x^{(1)}, \dots, x^{(n)}$ .
- ▶ The **maximum likelihood** approach:
  1. Compute:

$$\mu_{\text{MLE}} = \frac{1}{n} \sum_{i=1}^n x^{(i)} \quad \sigma_{\text{MLE}} = \sqrt{\frac{1}{n} \sum_{i=1}^n (x^{(i)} - \mu_{\text{MLE}})^2}$$

2. Use these as parameters of the Gaussian.

# Example



# In General

- ▶ Maximum Likelihood Estimation (MLE) can be used for a variety of densities.
- ▶ Suppose density  $p$  has parameters  $\theta_1, \dots, \theta_k$

1. Write log likelihood function:

$$\ln \mathcal{L}(\theta_1, \dots, \theta_k) = \sum_{i=1}^n \ln p(x^{(1)}, \dots, x^{(n)}; \theta_1, \dots, \theta_k)$$

2. Compute derivatives:  $\partial \tilde{\mathcal{L}} / \partial \theta_1, \partial \tilde{\mathcal{L}} / \partial \theta_2, \dots, \partial \tilde{\mathcal{L}} / \partial \theta_k$
3. Set derivatives to zero, solve for  $\theta_1, \dots, \theta_k$ .

## In Practice

- ▶ The MLE for a parameter only needs to be derived once.
- ▶ Many textbooks, statistics packages, and Wikipedia list the MLE parameter estimators.

# DSC 140A

*Probabilistic Modeling & Machine Learning*

Lecture 12 | Part 3

## Parametric vs. Non-Parametric Density Estimation

# Making Predictions

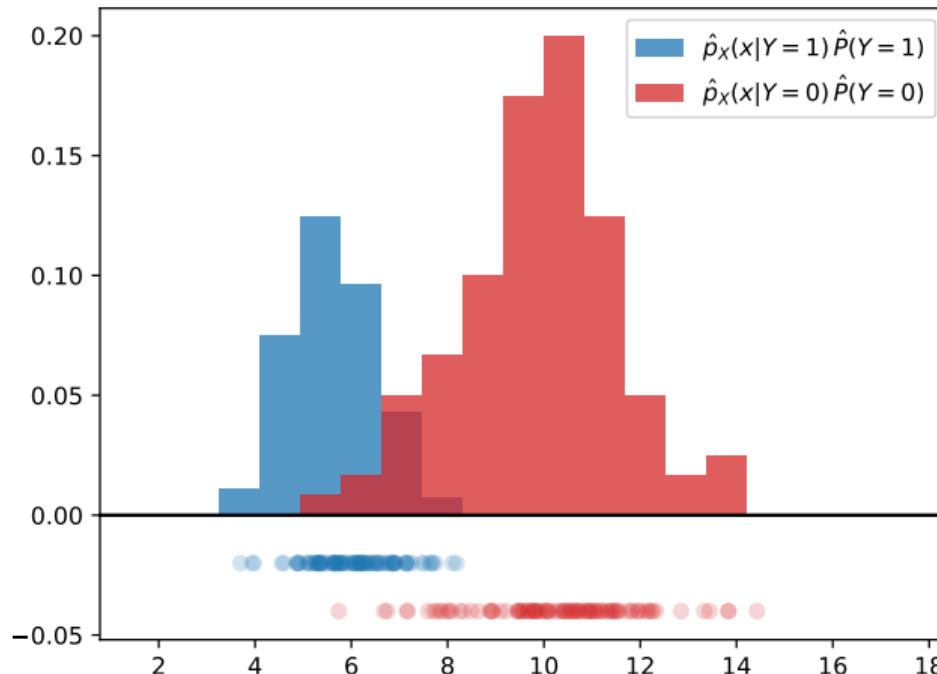
- ▶ We observe a data set  $\{(x^{(i)}, y_i)\}$  of flipper lengths and penguin species (0 or 1).
- ▶ **Task:** Given the flipper length of a new penguin, what is its species?
- ▶ Bayes' classifier: predict
$$\begin{cases} 1, & \text{if } p_X(x | Y = 1)\mathbb{P}(Y = 1) > p_X(x | Y = 0)\mathbb{P}(Y = 0), \\ 0, & \text{otherwise.} \end{cases}$$

# Estimating Densities

- ▶ We must estimate  $p_X(x | Y = 0)$  and  $p_X(x | Y = 1)$ .
- ▶ Approach 1: Non-parametric (histograms)
- ▶ Approach 2: Parametric

# Approach 1: Non-Parametric

- ▶ Estimate  $p_x(x | Y = 0)$  and  $p_x(x | Y = 1)$  with histograms.

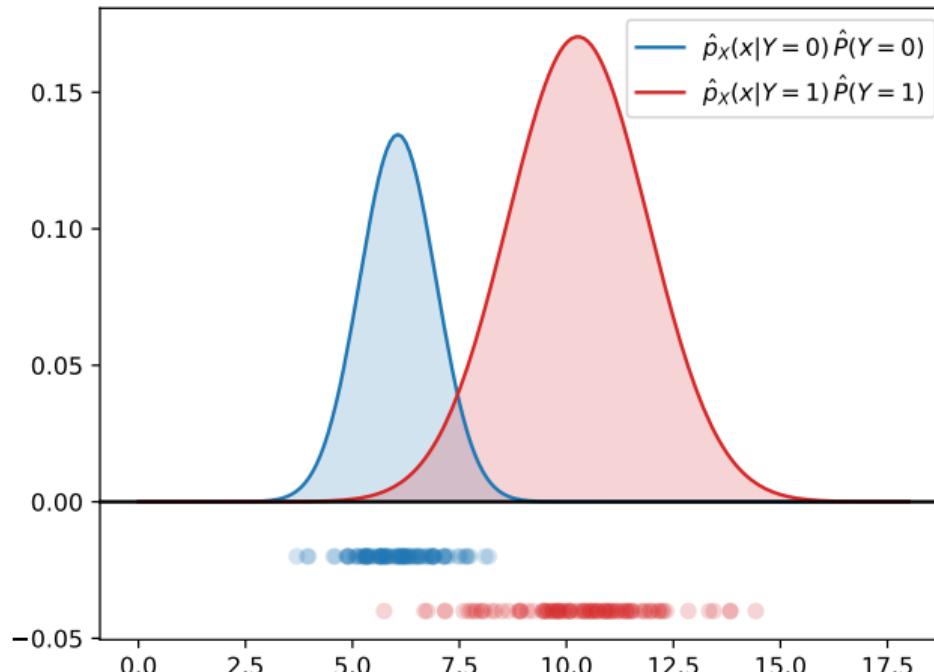


## **Approach 2: Parametric**

- ▶ Must choose a parametric distribution.
- ▶ Plotting a histogram, data looks roughly normal.
- ▶ We will fit Gaussians.

# Approach 2: Parametric

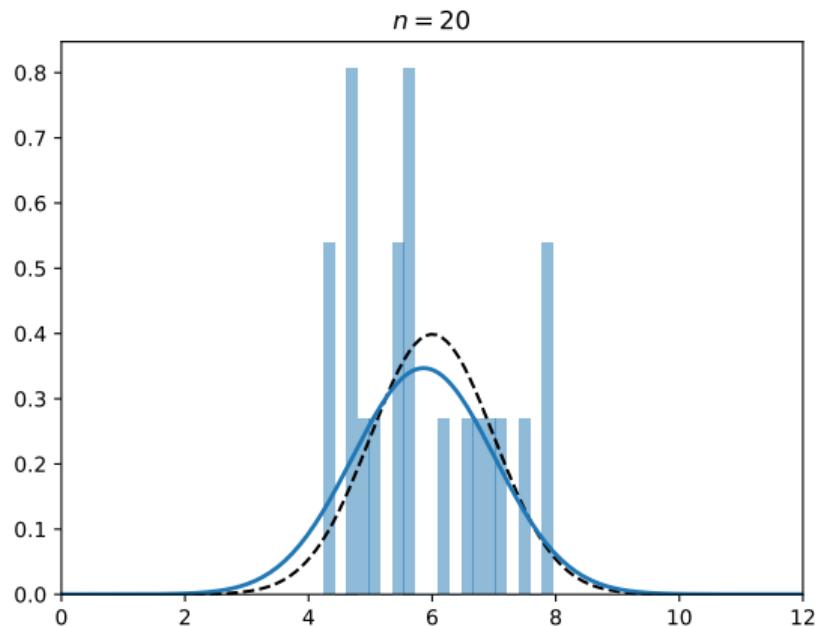
- ▶ Estimate  $p_X(x | Y = 0)$  and  $p_X(x | Y = 1)$  by fitting Gaussians with MLE.



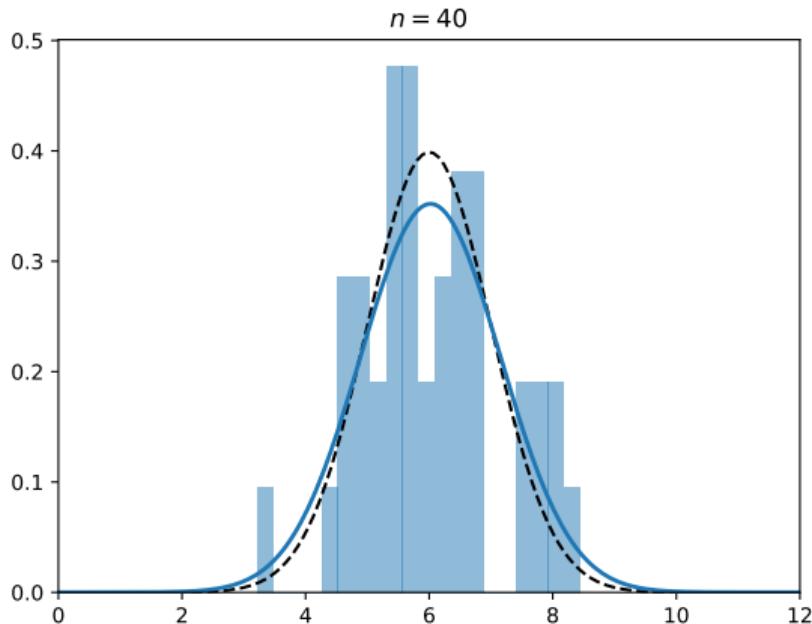
# Data Requirements

- ▶ Suppose the underlying distribution that produced the data actually was a Gaussian.
  - ▶ Or close to one.
- ▶ The parametric approach will require less data than the non-parametric.

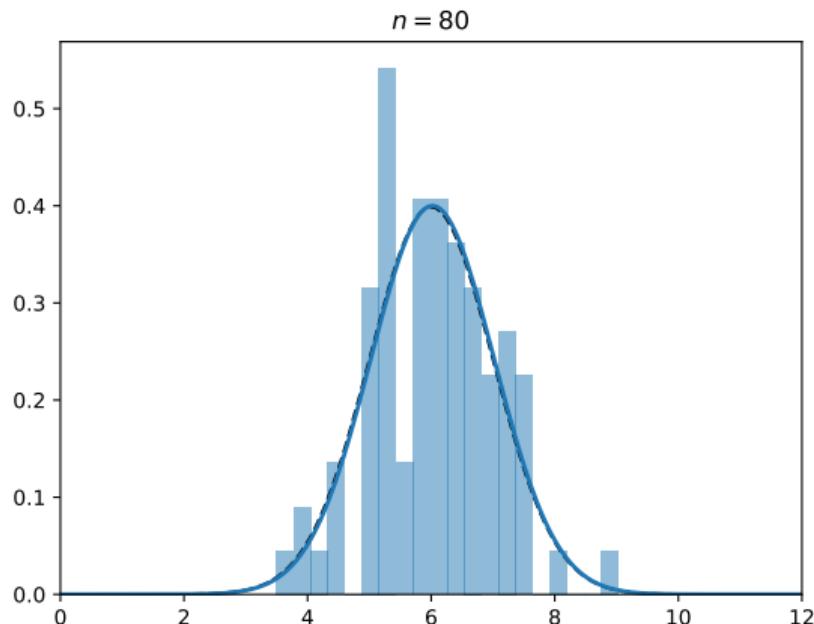
# Data Requirements



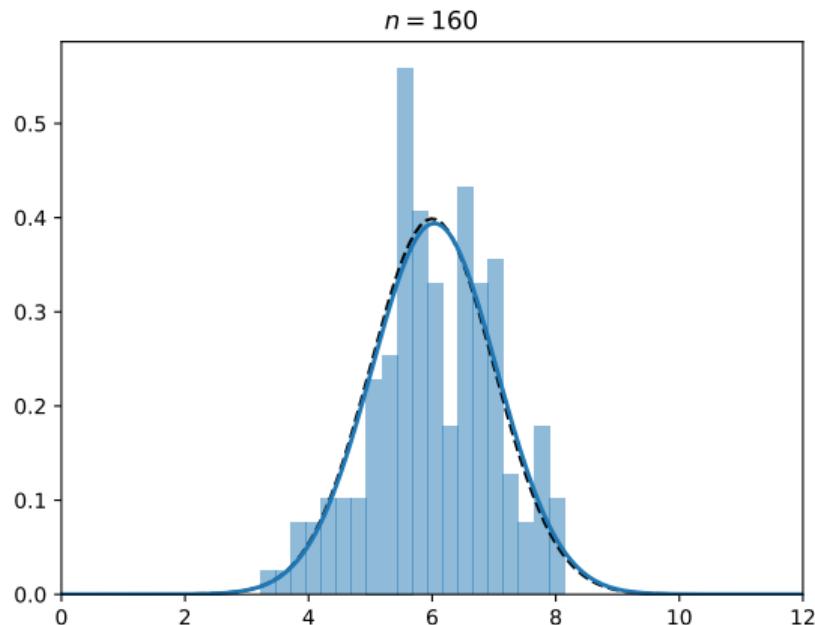
# Data Requirements



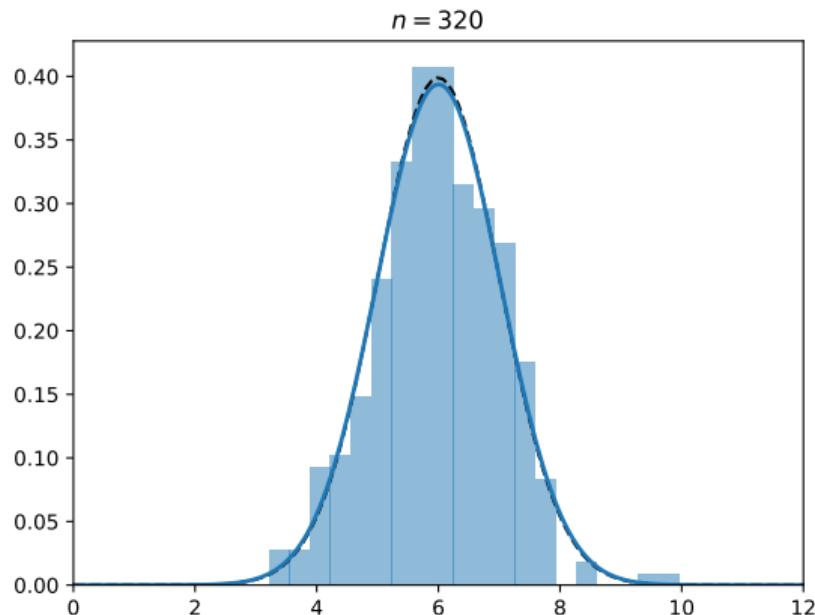
# Data Requirements



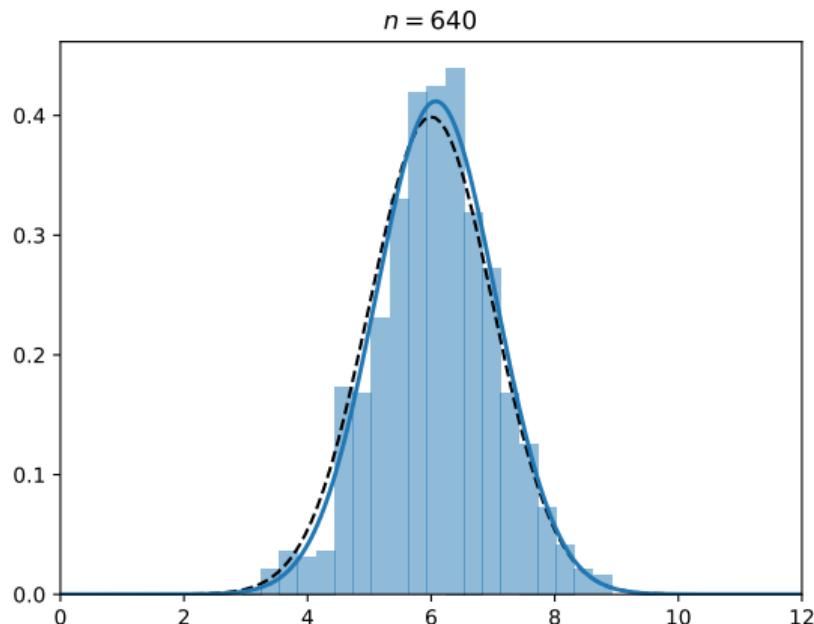
# Data Requirements



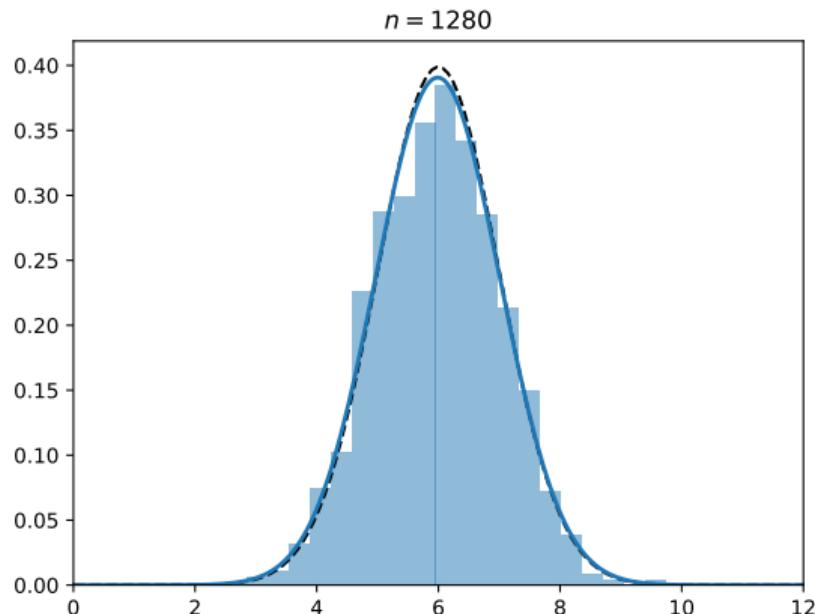
# Data Requirements



# Data Requirements



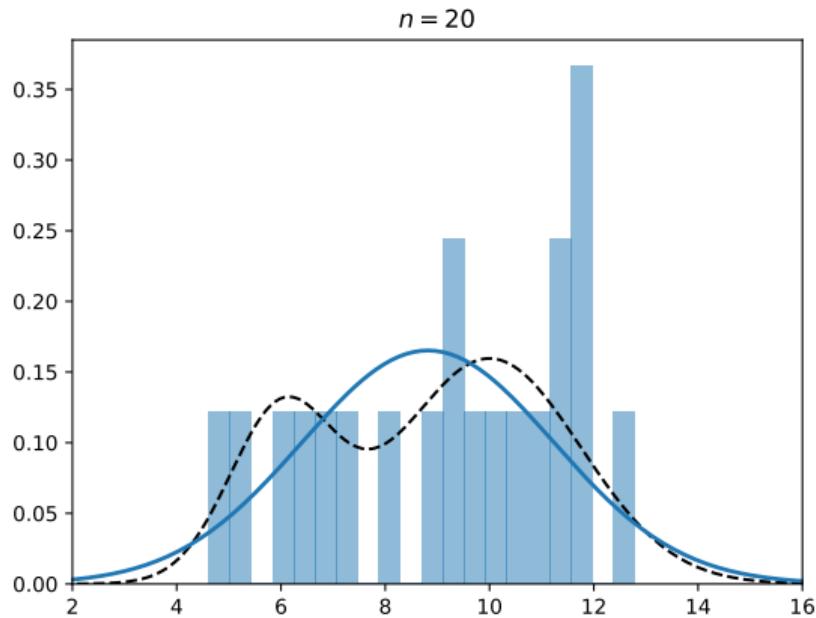
# Data Requirements



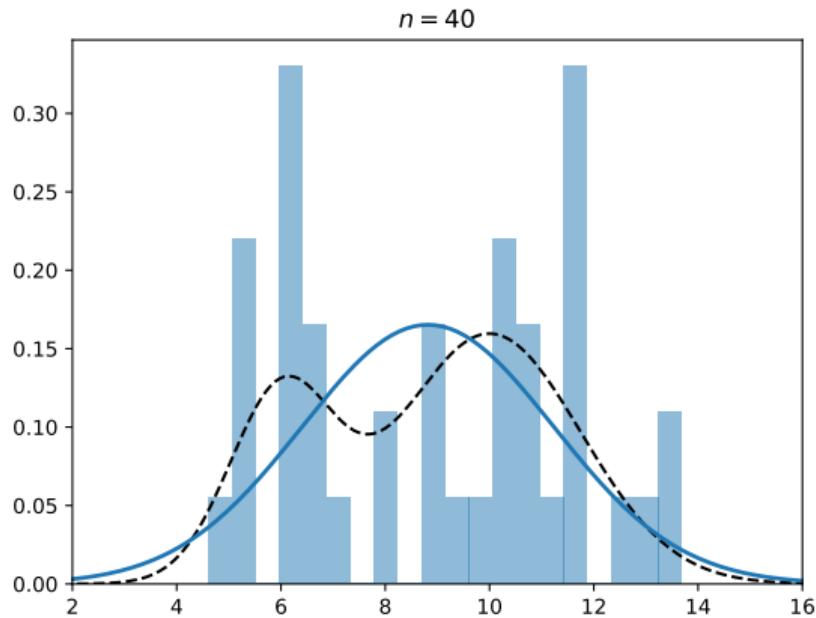
# Mis-specification

- ▶ However, suppose the underlying distribution is **not** Gaussian.
- ▶ No amount of data will allow the parametric approach to get close.
  - ▶ The model has been **mis-specified**.
- ▶ But the non-parametric approach will be close, eventually.

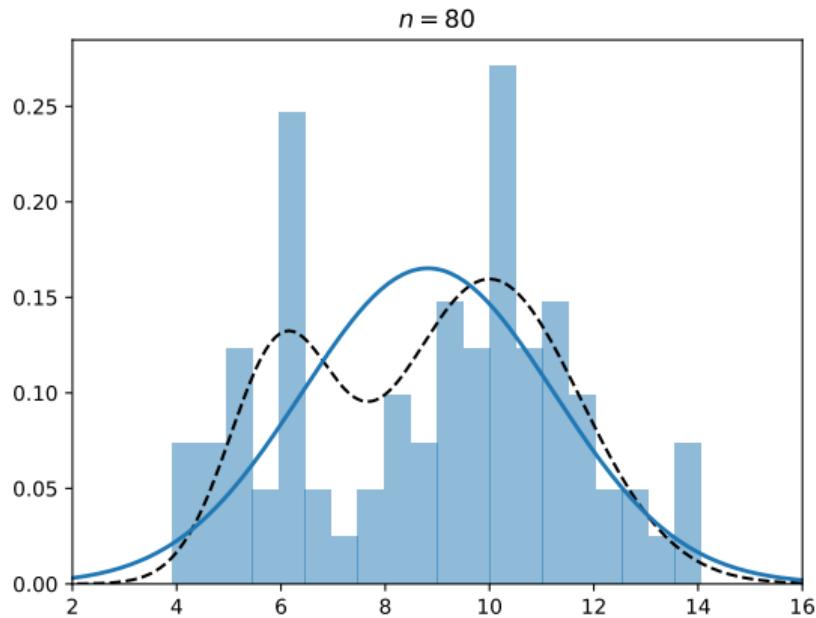
# Parametric vs. Non-Parametric



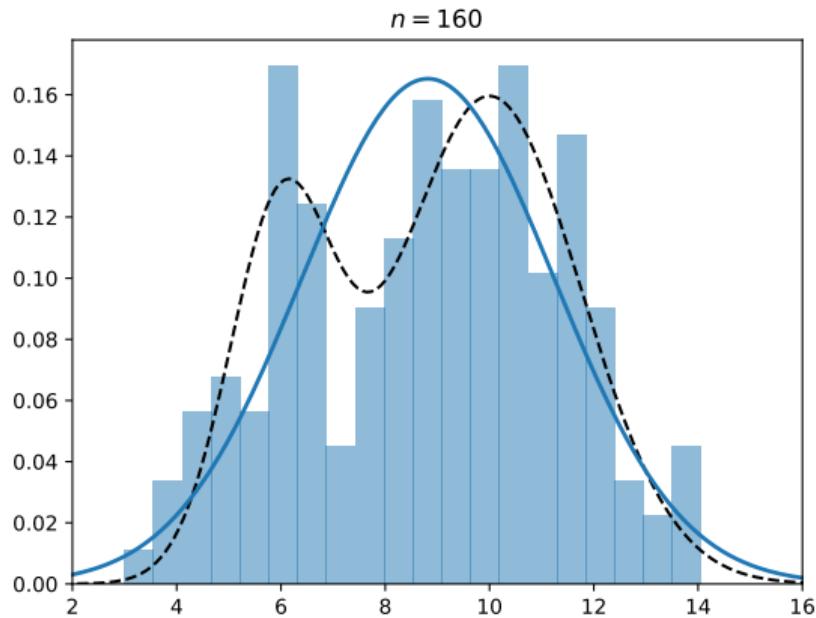
# Parametric vs. Non-Parametric



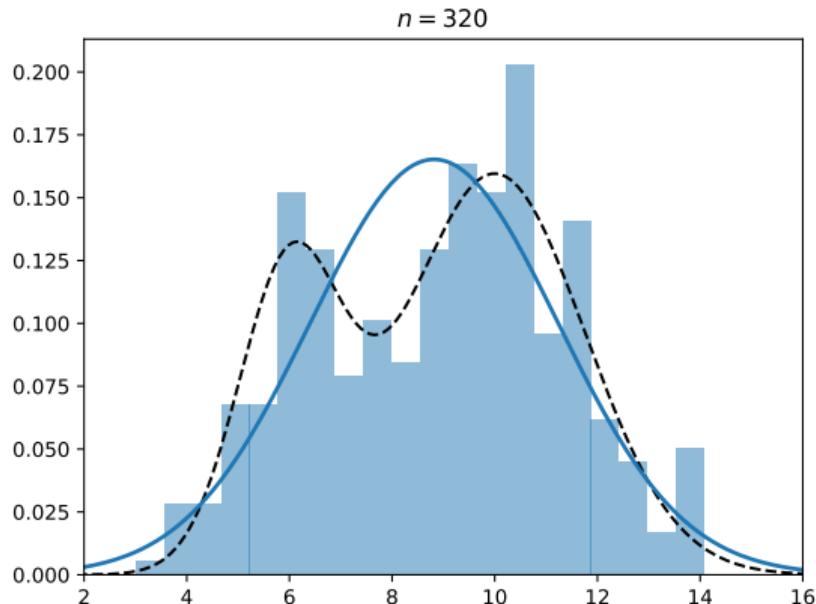
# Parametric vs. Non-Parametric



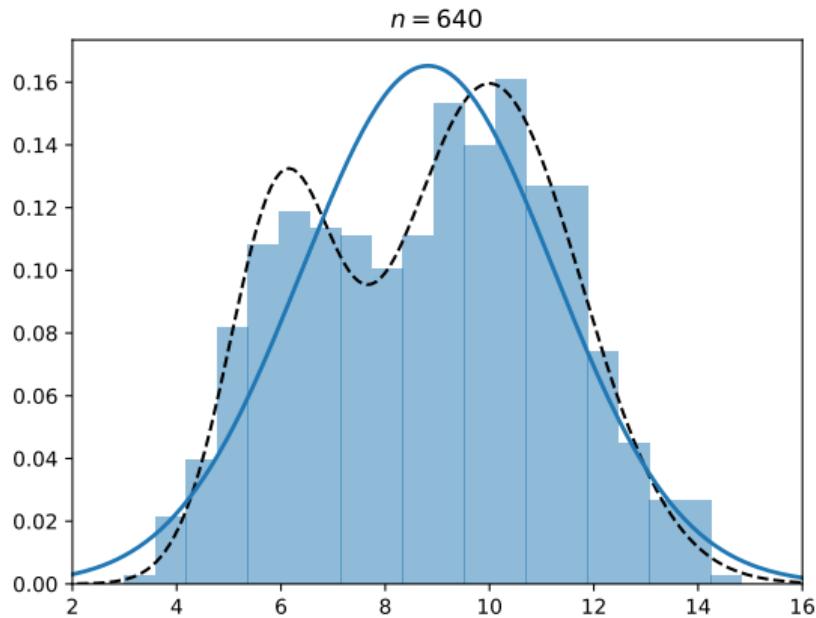
# Parametric vs. Non-Parametric



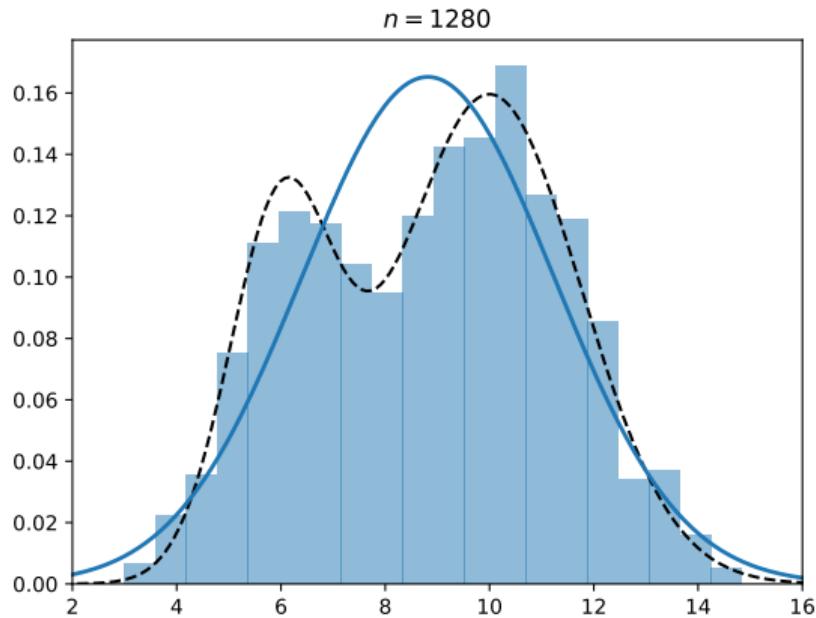
# Parametric vs. Non-Parametric



# Parametric vs. Non-Parametric



# Parametric vs. Non-Parametric



# High Dimensions

- ▶ Non-parametric approaches can fit arbitrary densities, but they require lots of data.
  - ▶ Especially in high dimensions!
- ▶ Parametric approaches require less data, provided that they are correctly specified.
- ▶ **Next time:** parametric density estimation in high dimensions.

# DSC 140A

*Probabilistic Modeling & Machine Learning*

Lecture 13 | Part 1

**Bayes with Multiple Features**

# Recap

- ▶ **Bayes Classifier:** predict  $y$  that maximizes  $\mathbb{P}(Y = y | X = x)$
- ▶ **Alternatively:** predict  $y$  that maximizes  $p_X(x | Y = y)\mathbb{P}(Y = y)$
- ▶ We must estimate these probabilities/densities.

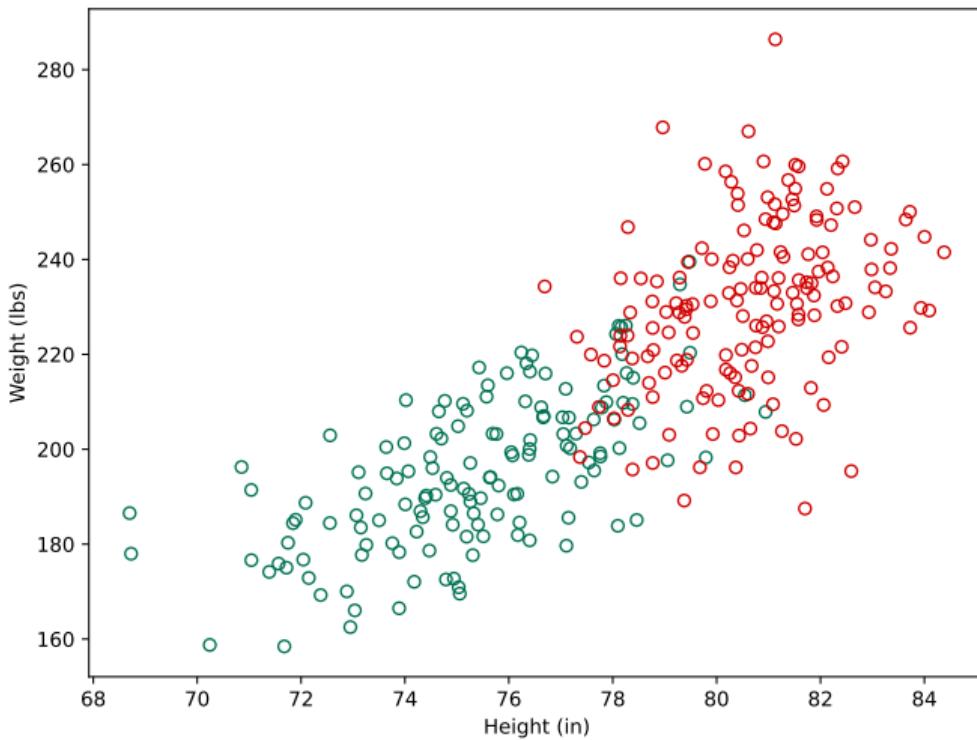
# Example: NBA Players

- ▶ **Guard** and **Forward** are two positions in basketball.
- ▶ Forwards tend to be larger than guards.



# Example: NBA Players

- ▶ Suppose we have a data set of  $n$  NBA players:
  - ▶  $X_1$ : the player's height
  - ▶  $X_2$ : the player's weight
  - ▶  $Y$ : the player's position (1 = guard, 0 = forward)
- ▶ **Given:** a new player's height and weight, predict their position.



# Bayes in $\geq 2$ Dimensions

- ▶ With one feature, Bayes said to pick  $y$  maximizing:

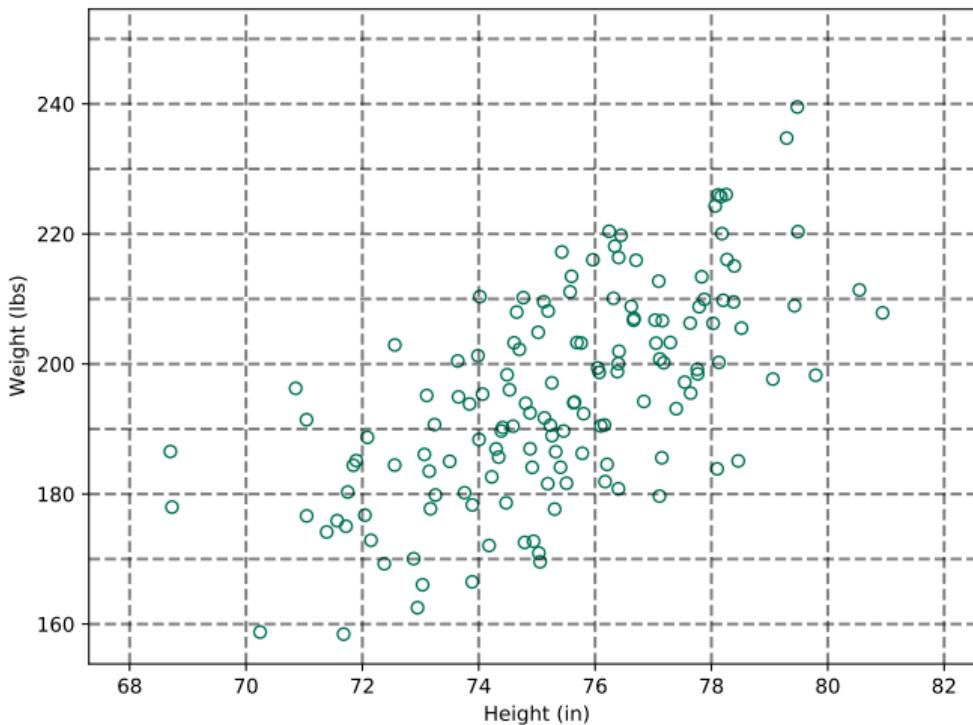
$$p_X(x | Y = y) \mathbb{P}(Y = y)$$

- ▶ With  $k$  features, pick  $y$  maximizing:

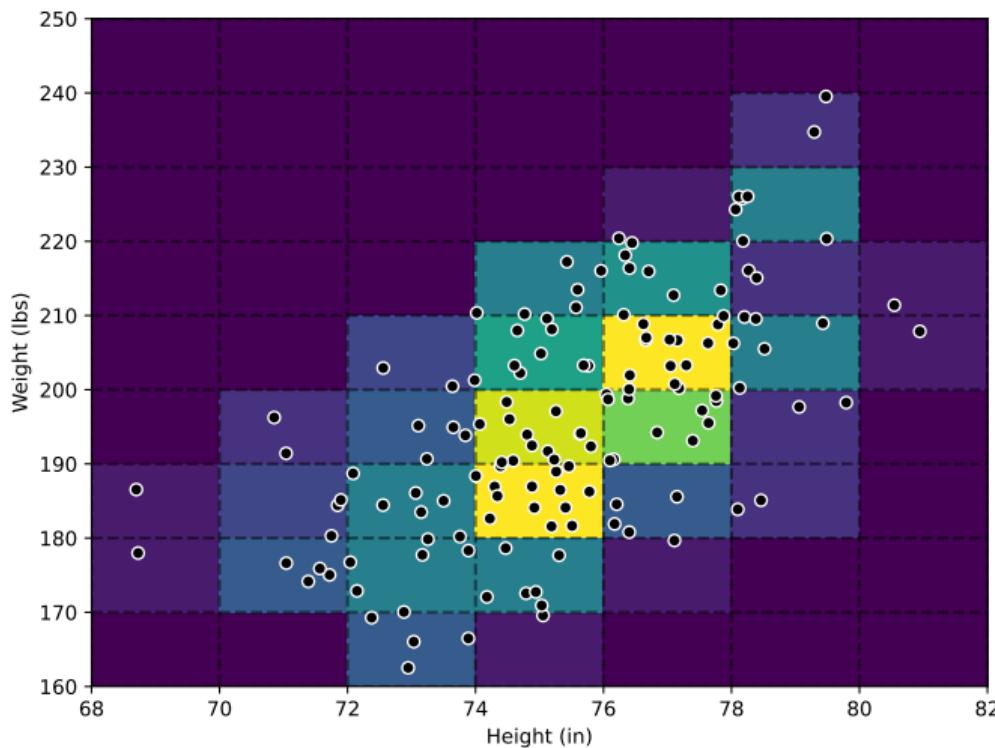
$$p_{\vec{X}}(\vec{x} | Y = y) \mathbb{P}(Y = y)$$

- ▶  $\vec{x}$  is the **feature vector**. Here: (height, weight) $^T$
- ▶ We need to estimate density  $p(\vec{x} | Y = y)$  for each class.

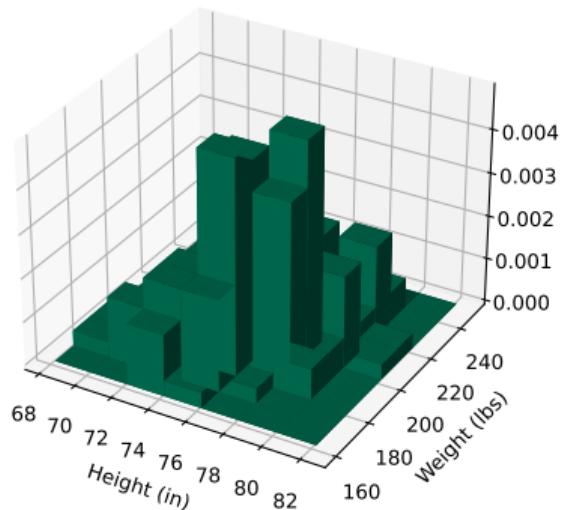
# Estimating with Histograms



# Estimating with Histograms



# Estimating with Histograms



# Predicting with Histograms

To predict the class of an input  $\vec{x}$ :

1. Use histograms to estimate  $p_{\vec{X}}(\vec{x} | Y = y)$  for each class separately.
2. Predict the class  $y$  maximizing

$$p_{\vec{X}}(\vec{x} | Y = y) \mathbb{P}(Y = y)$$

# Histogram Estimators

- ▶ Histogram density estimators are very flexible.
- ▶ But suffer heavily from **curse of dimensionality**.
- ▶ Not feasible for estimating density in more than a few dimensions.

# Today

- ▶ **Last time:** we saw the **parametric** approach to density estimation.
  - ▶ Pick a parametric distribution (e.g., Gaussian)
  - ▶ Find parameters by maximizing likelihood
- ▶ We saw how to do this for one-dimensional data.
- ▶ **Today:** multidimensional data.

## In particular...

- ▶ **Today:** multivariate Gaussian density estimation.
- ▶ That is: fitting multivariate Gaussians to data with maximum likelihood.

# DSC 140A

*Probabilistic Modeling & Machine Learning*

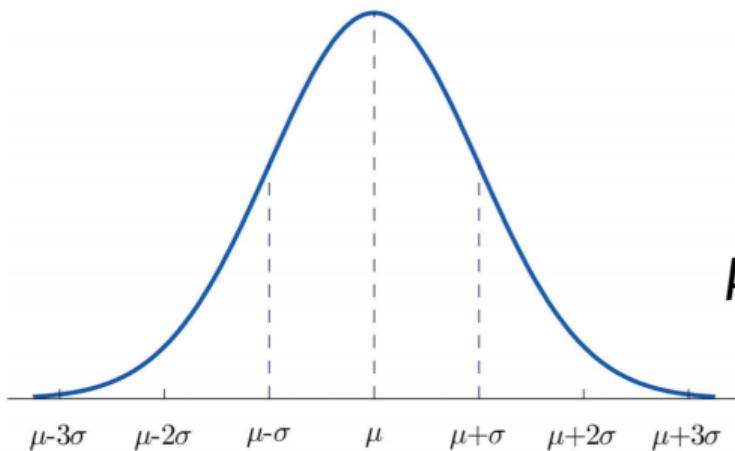
Lecture 13 | Part 2

Multivariate Gaussians

# Multivariate Gaussians

- ▶ In 1 dimension, a Gaussian seemed to describe distribution of heights.
- ▶ Does a **multivariate** Gaussian describe distribution of heights and weights?

# “Deriving” Multivariate Gaussians



$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}(x-\mu)^2/\sigma^2}$$

$$X \sim \mathcal{N}(\mu, \sigma^2)$$

# Setting #1

- ▶ Suppose we have  $d$  independent random variables  $X_1, \dots, X_d$ .
- ▶ Assume that each is Gaussian; different mean, but **same** variance:
$$X_1 \sim \mathcal{N}(\mu_1, \sigma^2), \quad X_2 \sim \mathcal{N}(\mu_2, \sigma^2), \dots, \quad X_d \sim \mathcal{N}(\mu_d, \sigma^2).$$

# Setting #1

- ▶ What is the **joint density**  $p(x_1, x_2, \dots, x_d)$ ?
- ▶ Since we assumed  $X_1, \dots, X_d$  are independent:

$$\begin{aligned} p(x_1, x_2, \dots, x_d) &= p(x_1)p(x_2) \cdots p(x_d) \\ &= \left( \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}(x-\mu_1)^2/\sigma^2} \right) \cdot \left( \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}(x-\mu_2)^2/\sigma^2} \right) \cdots \left( \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}(x-\mu_d)^2/\sigma^2} \right) \end{aligned}$$

# Setting #1

- ▶ What is the **joint density**  $p(x_1, x_2, \dots, x_d)$ ?
- ▶ Since we assumed  $X_1, \dots, X_d$  are independent:

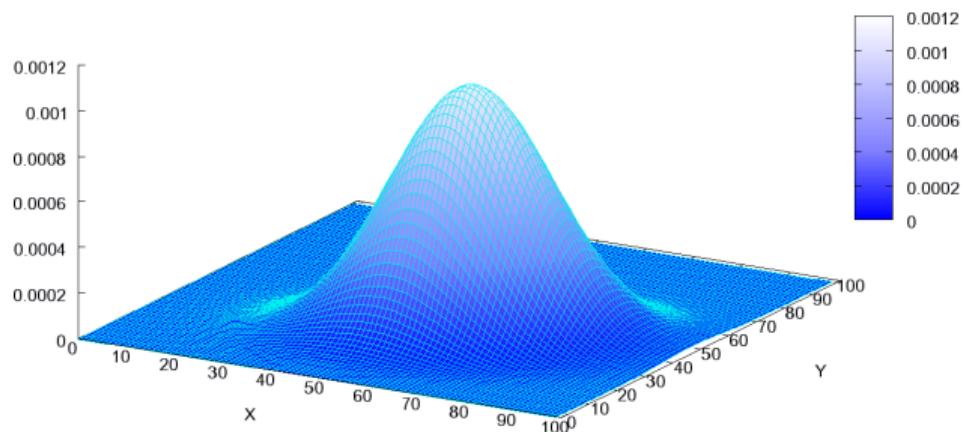
$$\begin{aligned} p(x_1, x_2, \dots, x_d) &= p(x_1)p(x_2) \cdots p(x_d) \\ &= \left( \frac{1}{\sqrt{2\pi}\sigma^2} e^{-\frac{1}{2}(x-\mu_1)^2/\sigma^2} \right) \cdot \left( \frac{1}{\sqrt{2\pi}\sigma^2} e^{-\frac{1}{2}(x-\mu_2)^2/\sigma^2} \right) \cdots \left( \frac{1}{\sqrt{2\pi}\sigma^2} e^{-\frac{1}{2}(x-\mu_d)^2/\sigma^2} \right) \\ &= \frac{1}{(2\pi\sigma^2)^{d/2}} \exp\left(-\frac{(x_1 - \mu_1)^2 + (x_2 - \mu_2)^2 + \cdots + (x_d - \mu_d)^2}{2\sigma^2}\right) \end{aligned}$$

# Setting #1

- ▶ What is the **joint density**  $p(x_1, x_2, \dots, x_d)$ ?
- ▶ Since we assumed  $X_1, \dots, X_d$  are independent:

$$\begin{aligned} p(x_1, x_2, \dots, x_d) &= p(x_1)p(x_2) \cdots p(x_d) \\ &= \left( \frac{1}{\sqrt{2\pi}\sigma^2} e^{-\frac{1}{2}(x-\mu_1)^2/\sigma^2} \right) \cdot \left( \frac{1}{\sqrt{2\pi}\sigma^2} e^{-\frac{1}{2}(x-\mu_2)^2/\sigma^2} \right) \cdots \left( \frac{1}{\sqrt{2\pi}\sigma^2} e^{-\frac{1}{2}(x-\mu_d)^2/\sigma^2} \right) \\ &= \frac{1}{(2\pi\sigma^2)^{d/2}} \exp\left(-\frac{(x_1 - \mu_1)^2 + (x_2 - \mu_2)^2 + \dots + (x_d - \mu_d)^2}{2\sigma^2}\right) \\ &= \frac{1}{(2\pi\sigma^2)^{d/2}} \exp\left(-\frac{\|\vec{x} - \vec{\mu}\|^2}{2\sigma^2}\right) \end{aligned}$$

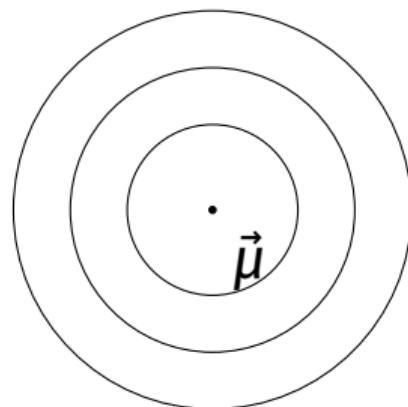
# Setting #1



## Setting #1: Spherical Gaussians

$$p(\vec{x}) = \frac{1}{(2\pi\sigma^2)^{d/2}} \exp\left(-\frac{1}{2} \frac{\|\vec{x} - \vec{\mu}\|^2}{\sigma^2}\right)$$

- ▶ Contours are (hyper)spheres.
- ▶ Every slice through middle gives same Gaussian.



## Setting #2

- ▶ Still assume  $X_1, \dots, X_d$  are independent, Gaussian.
- ▶ But they now have different variances:

$$X_1 \sim \mathcal{N}(\mu_1, \sigma_1^2), \quad X_2 \sim \mathcal{N}(\mu_2, \sigma_2^2), \dots, \quad X_d \sim \mathcal{N}(\mu_d, \sigma_d^2).$$

## Setting #2

$$\begin{aligned} p(x_1, x_2, \dots, x_d) &= p(x_1)p(x_2) \cdots p(x_d) \\ &= \left( \frac{1}{\sqrt{2\pi\sigma_1^2}} e^{-\frac{1}{2}(x-\mu_1)^2/\sigma_1^2} \right) \cdot \left( \frac{1}{\sqrt{2\pi\sigma_2^2}} e^{-\frac{1}{2}(x-\mu_2)^2/\sigma_2^2} \right) \cdots \left( \frac{1}{\sqrt{2\pi\sigma_d^2}} e^{-\frac{1}{2}(x-\mu_d)^2/\sigma_d^2} \right) \end{aligned}$$

# Setting #2

$$\begin{aligned} p(x_1, x_2, \dots, x_d) &= p(x_1)p(x_2) \cdots p(x_d) \\ &= \left( \frac{1}{\sqrt{2\pi\sigma_1^2}} e^{-\frac{1}{2}(x-\mu_1)^2/\sigma_1^2} \right) \cdot \left( \frac{1}{\sqrt{2\pi\sigma_2^2}} e^{-\frac{1}{2}(x-\mu_2)^2/\sigma_2^2} \right) \cdots \left( \frac{1}{\sqrt{2\pi\sigma_d^2}} e^{-\frac{1}{2}(x-\mu_d)^2/\sigma_d^2} \right) \\ &= \frac{1}{(2\pi)^{d/2}\sigma_1 \cdot \sigma_2 \cdots \sigma_d} \exp\left(-\frac{1}{2} \left[ \frac{(x_1 - \mu_1)^2}{\sigma_1^2} + \frac{(x_2 - \mu_2)^2}{\sigma_2^2} + \cdots + \frac{(x_d - \mu_d)^2}{\sigma_d^2} \right]\right) \end{aligned}$$

# Setting #2

- ▶ Define

$$C = \begin{pmatrix} \sigma_1^2 & 0 & \dots & 0 \\ 0 & \sigma_2^2 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \sigma_d^2 \end{pmatrix}$$

- ▶ Then:

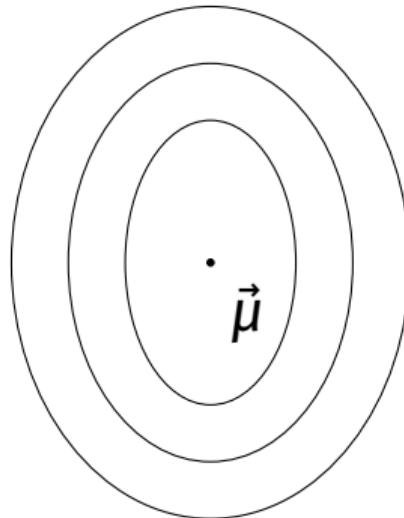
$$p(\vec{x}) = \frac{1}{(2\pi)^{d/2} |C|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\vec{x} - \vec{\mu})^T C^{-1}(\vec{x} - \vec{\mu})\right)$$

where  $|C|$  is the **determinant** of  $C$ .

## Setting #2: Axis-Aligned Gaussians

$$p(\vec{x}) = \frac{1}{(2\pi)^{d/2} |C|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\vec{x} - \vec{\mu})^T C^{-1} (\vec{x} - \vec{\mu})\right)$$

- ▶ Contours are axis-aligned (hyper)ellipses.
- ▶  $C$  is the **covariance matrix**.
  - ▶ Diagonal.
  - ▶ Entries are variances.



## Setting #3: General Gaussians

- ▶ We have assumed that  $X_1, \dots, X_d$  are independent.
- ▶ Now assume that they're not. Define **covariance**:

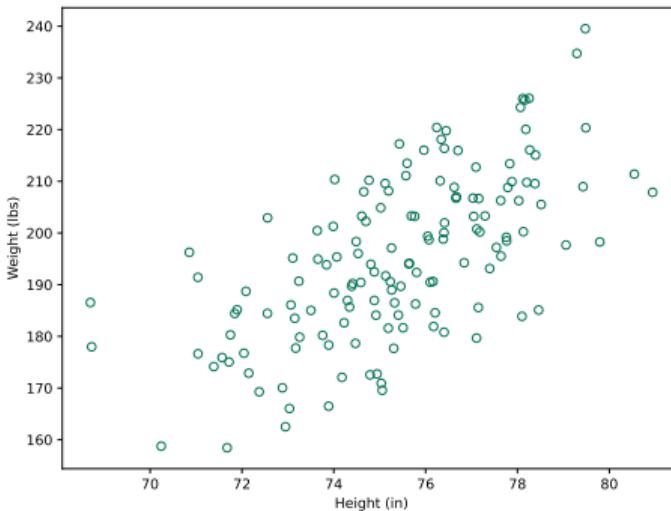
$$\text{Cov}(X_i, X_j) = \mathbb{E}[(X_i - \mu_i)(X_j - \mu_j)]$$

- ▶ **Note:**

$$\text{Var}(X_i) = \text{Cov}(X_i, X_i)$$

# Covariance

- ▶ Covariance measures how much two quantities **vary together**.



$$\text{Cov}(X_i, X_j) = \mathbb{E}[(X_i - \mu_i)(X_j - \mu_j)]$$

## Setting #3: General Gaussians

- Now the **covariance matrix** has off-diagonal elements:

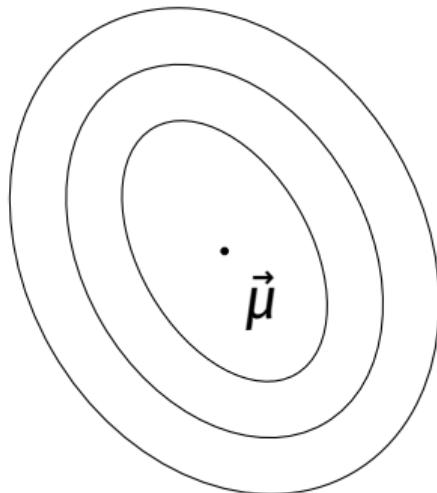
$$C = \begin{pmatrix} \text{Var}(X_1) & \text{Cov}(X_1, X_2) & \cdots & \text{Cov}(X_1, X_d) \\ \text{Cov}(X_2, X_1) & \text{Var}(X_2) & \cdots & \text{Cov}(X_2, X_d) \\ \cdots & \cdots & \cdots & \cdots \\ \text{Cov}(X_d, X_1) & \text{Cov}(X_d, X_2) & \cdots & \text{Var}(X_d) \end{pmatrix}$$

- Since  $\text{Cov}(X_i, X_j) = \text{Cov}(X_j, X_i)$ ,  $C$  is symmetric.

## Setting #3: General Gaussians

$$p(\vec{x}) = \frac{1}{(2\pi)^{d/2} |C|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\vec{x} - \vec{\mu})^T C^{-1} (\vec{x} - \vec{\mu})\right)$$

Contours are general (hyper)ellipses.  
 $C$  need not be diagonal.



# Overview

- ▶ The probability density function for a multivariate Gaussian distribution is:

$$p(\vec{x}) = \frac{1}{(2\pi)^{d/2} |C|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\vec{x} - \vec{\mu})^T C^{-1} (\vec{x} - \vec{\mu})\right)$$

- ▶ Here,  $C$  is the **covariance matrix**.

# Overview

- ▶ There are three cases:
  1.  $C$  is diagonal, with all the same entries.
  2.  $C$  is diagonal, with different entries.
  3.  $C$  is not diagonal.

# DSC 140A

*Probabilistic Modeling & Machine Learning*

Lecture 13 | Part 3

**Fitting Multivariate Gaussians**

# Fitting Multivariate Gaussians

- ▶ Suppose  $\vec{x}^{(1)}, \dots, \vec{x}^{(n)}$  came from a multivariate Gaussian.
- ▶ What were the parameters of that Gaussian?
- ▶ We can use the principle of **maximum likelihood**.

# What are the parameters?

$$p(\vec{x}) = \frac{1}{(2\pi)^{d/2} |C|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\vec{x} - \vec{\mu})^T C^{-1}(\vec{x} - \vec{\mu})\right)$$

- ▶  $\vec{\mu}$ : controls Gaussian's location
- ▶  $C$ : controls Gaussian's shape

# Estimating $\vec{\mu}$

- The maximum likelihood estimator for  $\mu$  is:

$$\vec{\mu}_{\text{MLE}} = \frac{1}{n} \sum_{i=1}^n \vec{x}^{(i)}$$

# Estimating $C$

- ▶ First: make assumptions on covariance matrix.
- ▶ In order from strict to weak:
  - ▶ Spherical:  $C$  is diagonal, with all the same entries.
  - ▶ Axis-Aligned:  $C$  is diagonal, with different entries.
  - ▶ General:  $C$  is not diagonal.
- ▶ The weaker the assumptions, the more parameters to estimate.

# Fitting Spherical Gaussians

- ▶ Only one variance parameter:  $\sigma^2$ .
- ▶ The density function becomes:

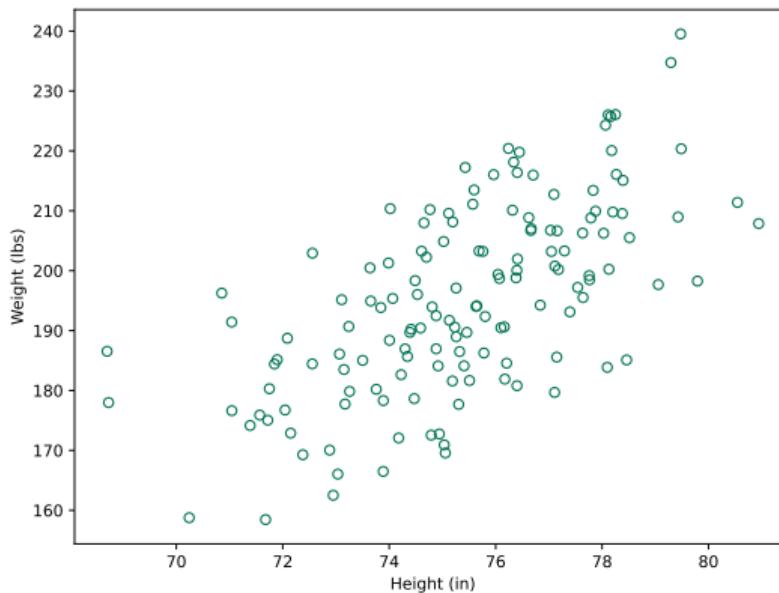
$$p(\vec{x}) = \frac{1}{(2\pi\sigma^2)^{d/2}} \exp\left(-\frac{(\vec{x} - \vec{\mu})^T(\vec{x} - \vec{\mu})}{2\sigma^2}\right)$$

- ▶ The maximum likelihood estimator:

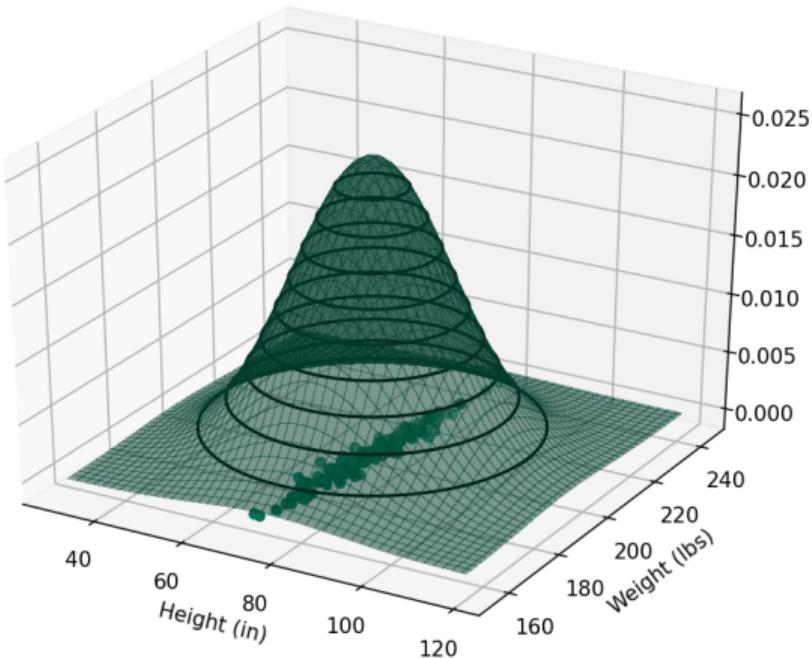
$$\sigma_{MLE}^2 = \frac{1}{n} \sum_{i=1}^n \|\vec{x}^{(i)} - \vec{\mu}_{MLE}\|^2$$

# Example: NBA Data

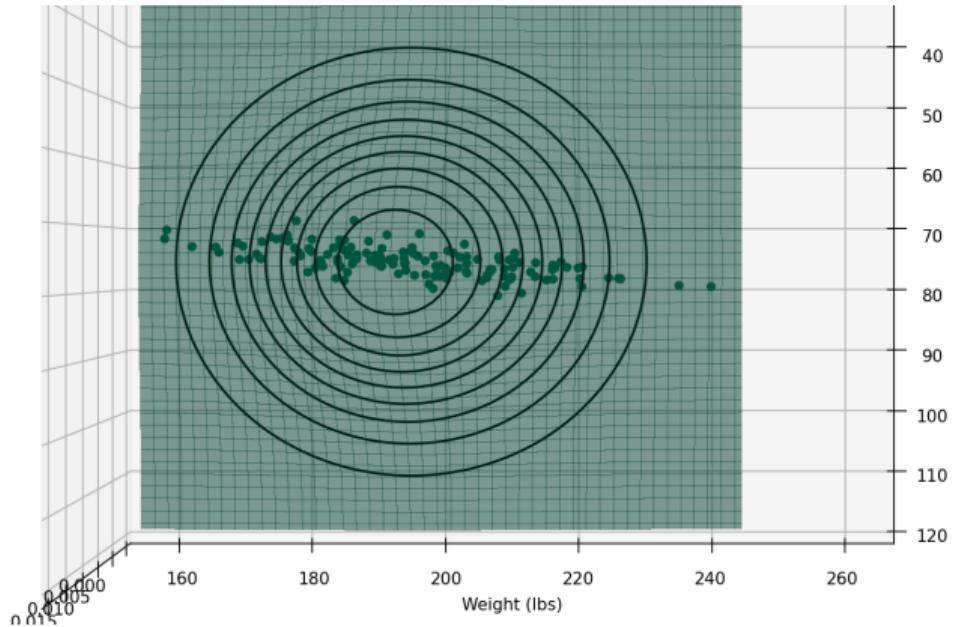
- ▶ What if we fit a spherical Gaussian to the NBA data?



# Fitting Spherical Gaussians



# Fitting Spherical Gaussians



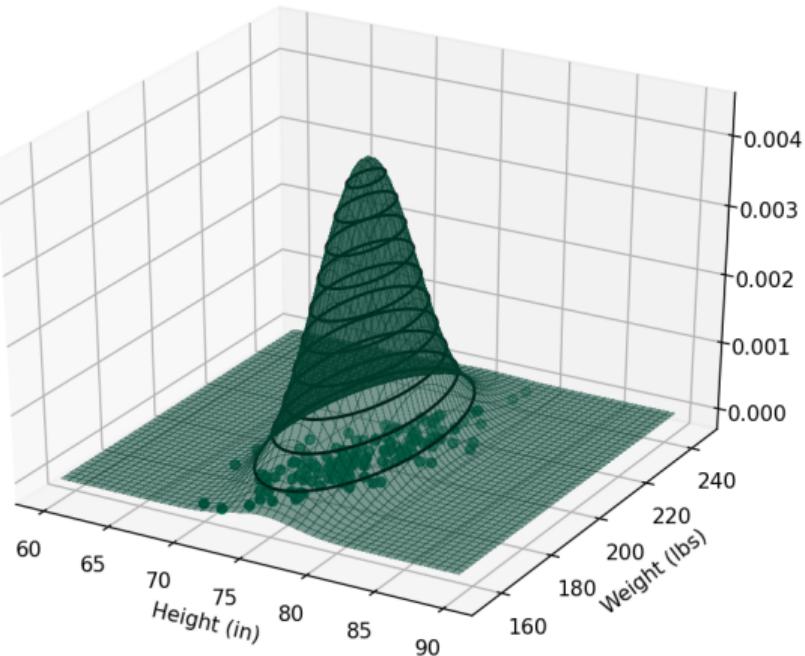
## Example: NBA Data

- ▶ Spherical Gaussians are not well-suited to this data.
- ▶ Perhaps if the data were **standardized...**
- ▶ Instead, try axis-aligned Gaussians.

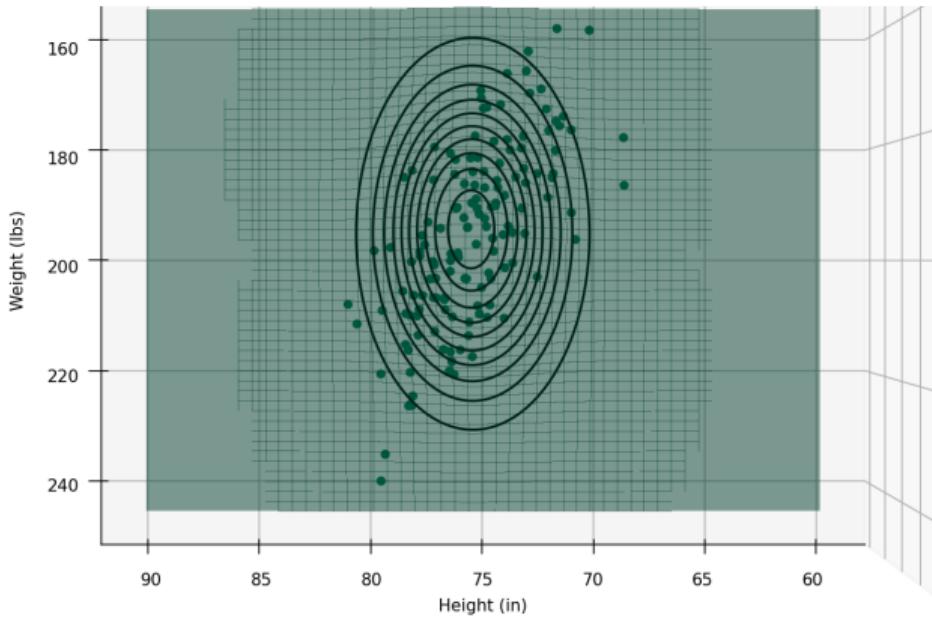
# Fitting Axis-Aligned Gaussians

- ▶ Variance for each axis:  $\sigma_1^2$  and  $\sigma_2^2$ .
- ▶ Maximum likelihood estimates:
  - $\sigma_1^2$  = sample variance of heights
  - $\sigma_2^2$  = sample variance of weights

# Fitting Axis-Aligned Gaussians



# Fitting Axis-Aligned Gaussians



## Example: NBA Data

- ▶ Axis-aligned Gaussian does not capture correlation between height and weight.
- ▶ Try general Gaussian with full covariance.

# Fitting General Gaussians

- ▶ Must compute covariance for each pair of dimensions.
- ▶ Maximum likelihood estimate for covariance of feature  $i$  and  $j$ :

$$C_{ij} = \left( \frac{1}{n} \sum_{k=1}^n \vec{x}_i^{(k)} \vec{x}_j^{(k)} \right) - \mu_i \mu_j$$

# Computing the Covariance Matrix

Step 1. Make matrix with heights in first column,  
weights in second:

$$\begin{pmatrix} \text{height 1} & \text{weight 1} \\ \text{height 2} & \text{weight 2} \\ \dots & \dots \\ \text{height } n & \text{weight } n \end{pmatrix}$$

# Computing the Covariance Matrix

Step 2. Subtract sample mean height, mean weight from each column. Call this matrix  $X$ :

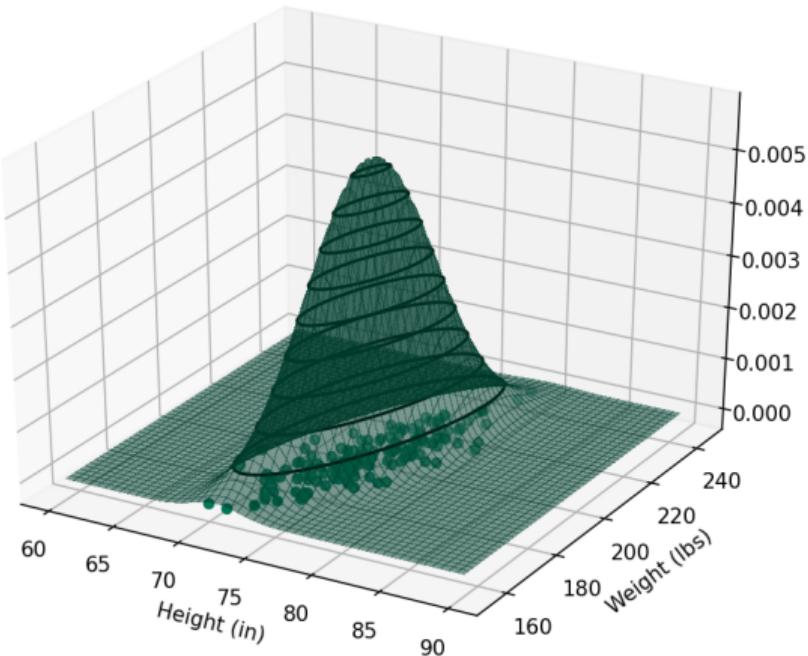
$$X = \begin{pmatrix} \text{height 1} - \text{mean height} & \text{weight 1} - \text{mean weight} \\ \text{height 2} - \text{mean height} & \text{weight 2} - \text{mean weight} \\ \dots & \dots \\ \text{height } n - \text{mean height} & \text{weight } n - \text{mean weight} \end{pmatrix}$$

# Computing the Covariance Matrix

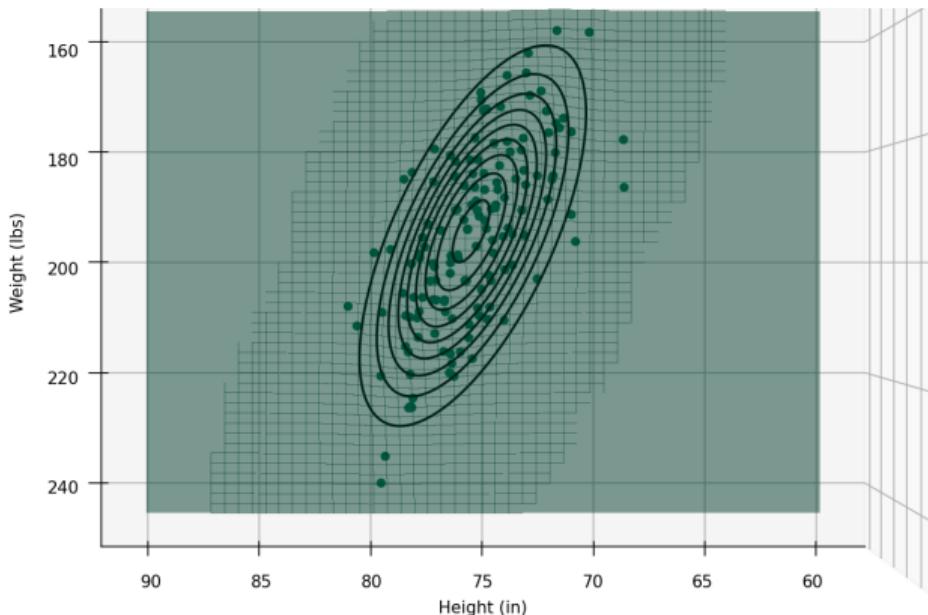
The empirical covariance matrix is then:

$$C = \frac{1}{n} X^T X$$

# Fitting General Gaussians



# Fitting General Gaussians



# **Up next...**

Making predictions using these fitted Gaussians.

# DSC 140A

*Probabilistic Modeling & Machine Learning*

Lecture 13 | Part 4

**Discriminant Analysis**

# Bayes Classifier with MV Gaussians

1. Fit Gaussian for  $p(\vec{X} | Y = y)$  for each class,  $y$ .
2. For new point, predict  $y$  maximizing:

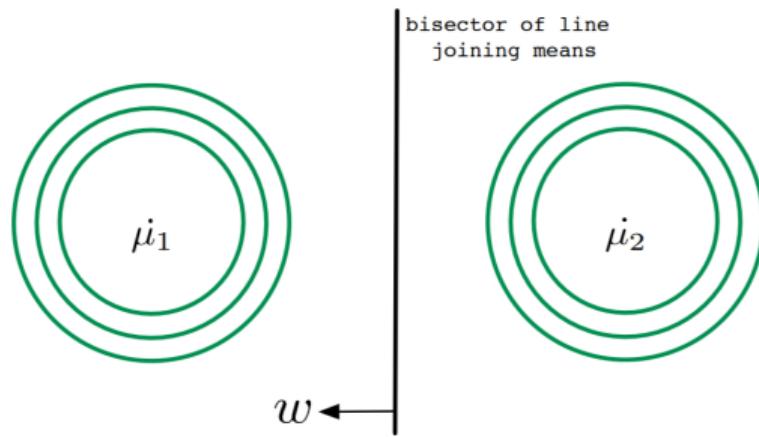
$$p(\vec{X} = \vec{x} | Y = y) \mathbb{P}(Y = y)$$

# Decision Boundary

- ▶ For every point in space, we have a classification.
- ▶ The **decision boundary**: surface between different classifications.
  - ▶ On one side, prediction is  $y_1$ ;
  - ▶ on the other, prediction is  $y_2$ .

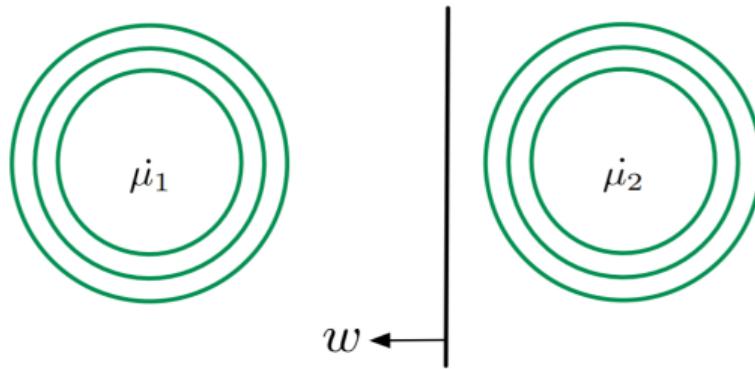
# Setting #1

- ▶ Assume:
  - ▶ classes equally likely:  $\mathbb{P}(Y = 1) = \mathbb{P}(Y = 0)$
  - ▶ identical covariance matrices



# Setting #1

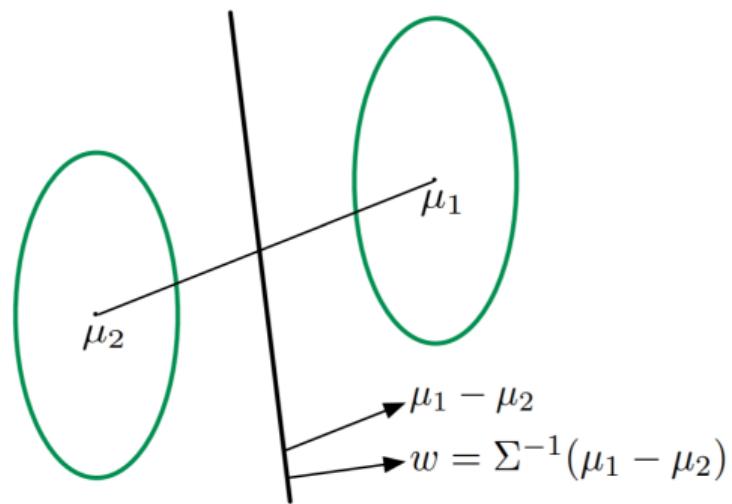
- If  $\mathbb{P}(Y = y_1) > \mathbb{P}(Y = y_2)$ :



Choose class 1 if  $\vec{w} \cdot \frac{(\vec{\mu}_1 - \vec{\mu}_2)}{\sigma^2} \geq \theta$ .

# Setting #2

- ▶ Assume:
  - ▶ covariance matrices identical, diagonal
  - ▶ that is: axis-aligned Gaussians



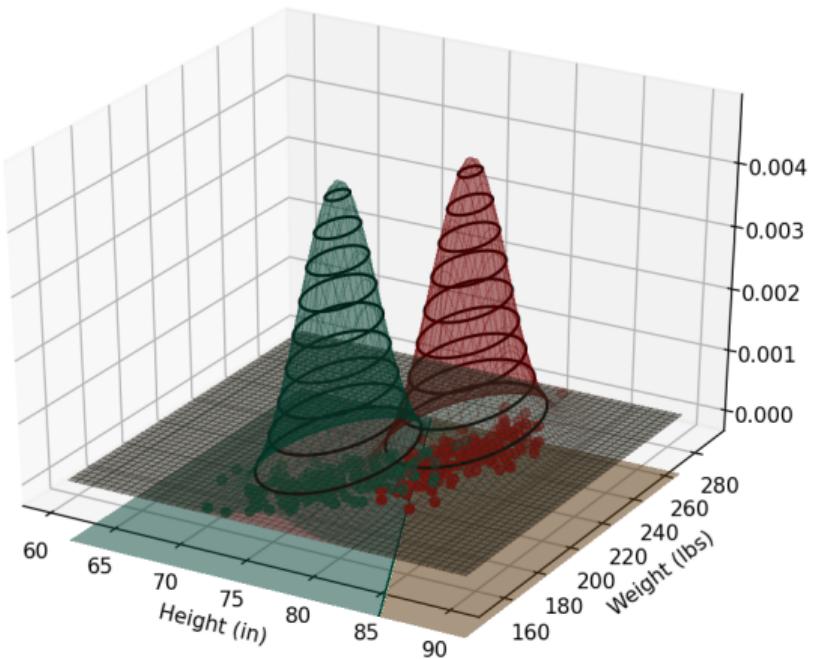
Predict class 1 if  
 $\vec{x} \cdot \vec{w} \geq \theta$ .

# Example

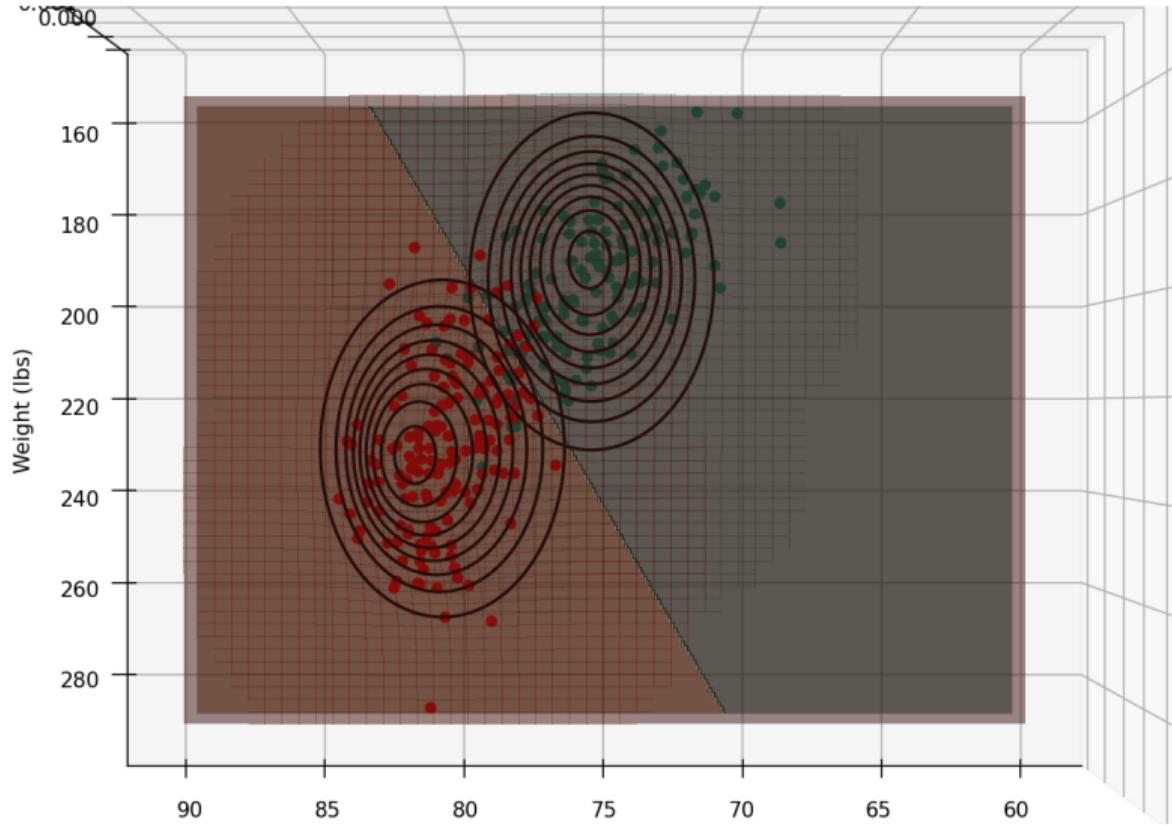
- ▶ Use to predict position given height and weight.
- ▶ How do we get one covariance matrix?
- ▶ Don't lump data together...
- ▶ Instead, compute covariance matrix for each class, perform weighted average:

$$C = \frac{n_1 C_1 + n_2 C_2}{n_1 + n_2}$$

# Example



# Example

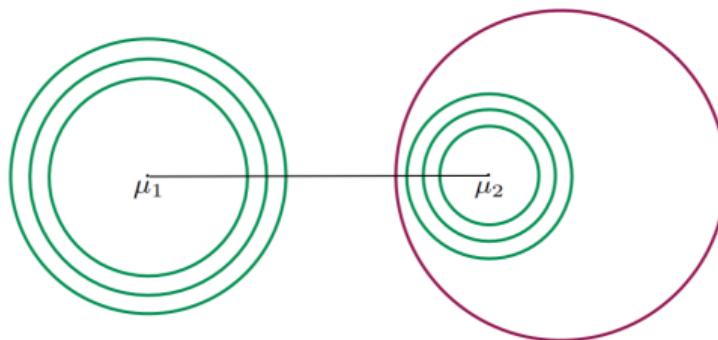


# Linear Discriminant Analysis

- ▶ When covariance matrices are **equal**, decision boundary is linear.
- ▶ This procedure is called **linear discriminant analysis** (LDA).
- ▶ True even if the Gaussians have full covariance.

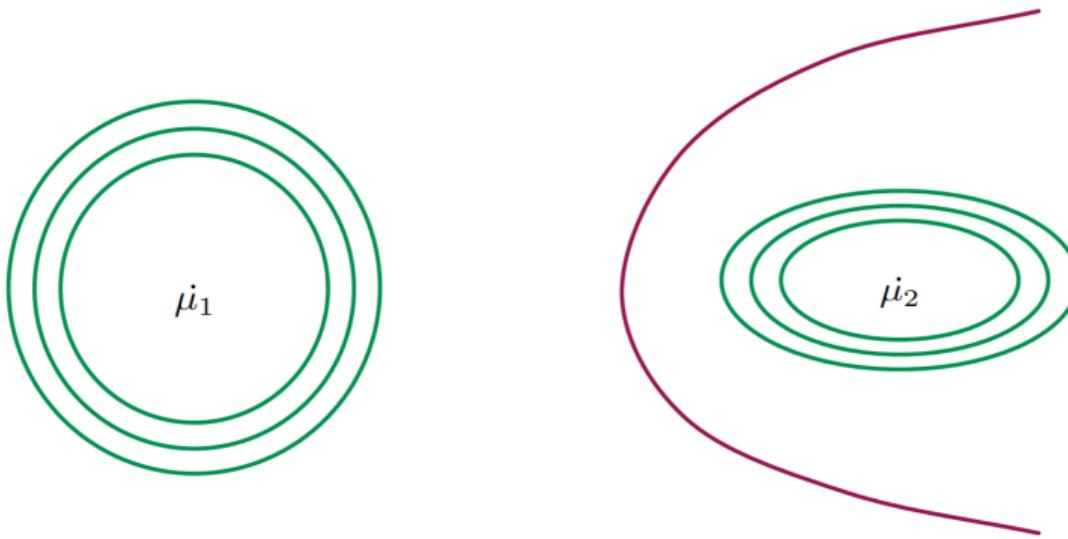
# Setting #3

- ▶ Assume:
  - ▶ covariance matrices  $C_1, C_2$  different, non-diagonal

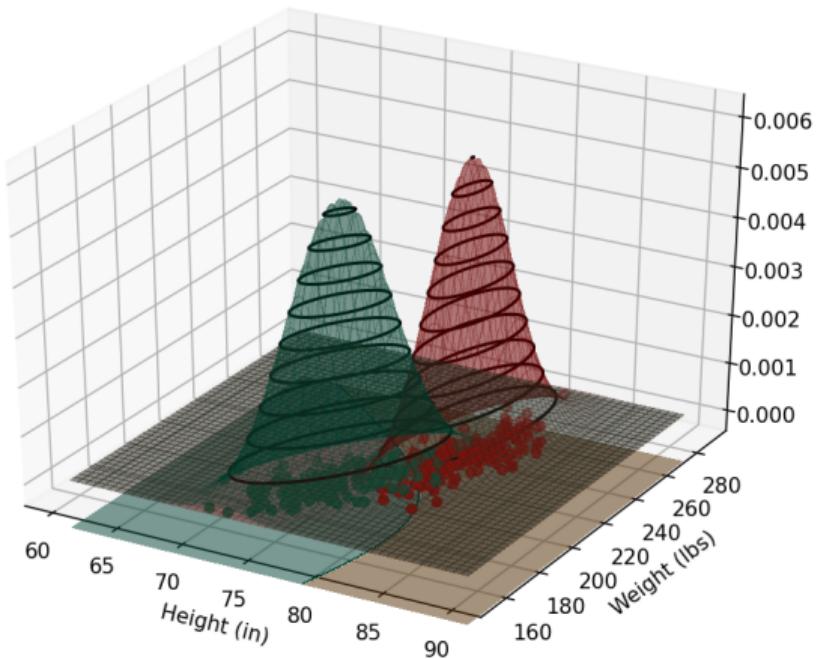


# Setting #3

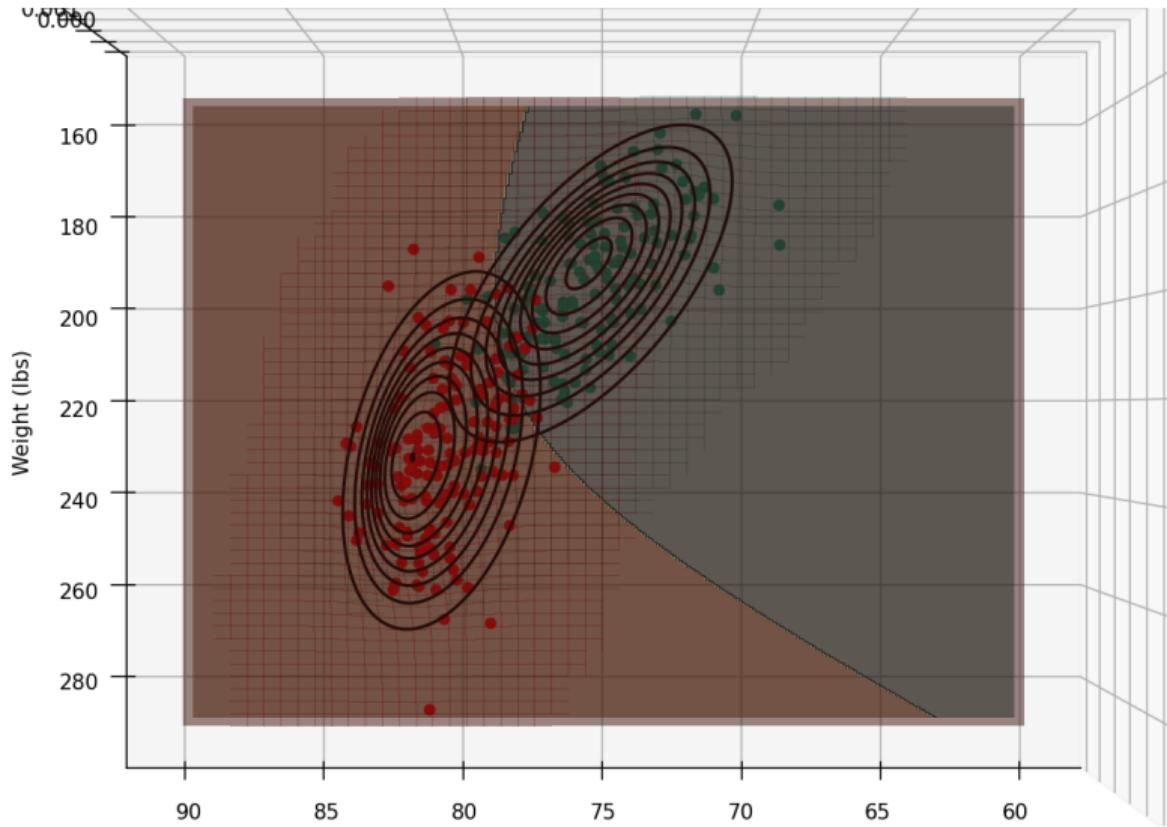
- ▶ Assume:
  - ▶ covariance matrices  $C_1, C_2$  different, non-diagonal



# Example



# Example



# Quadratic Discriminant Analysis

- ▶ When covariance matrices are equal, decision boundary is quadratic (ellipsoidal, paraboloidal, hyperboloidal).
- ▶ This procedure is called **quadratic discriminant analysis** (QDA).

## In practice...

- ▶ A full covariance requires estimating  $\Theta(d^2)$  parameters; needs more data.
- ▶ Gaussian assumption may be a poor match for data.

# DSC 140A

*Probabilistic Modeling & Machine Learning*

Lecture 01 | Part 1

**Recap**

# Applying the Bayes Classifier

- ▶ Predict the class  $y$  which maximizes:

$$p_X(\vec{X} = \vec{x} \mid Y = y) \mathbb{P}(Y = y)$$

- ▶ We must **estimate** the density,  $p_X$ .
- ▶ Two approaches:
  1. Non-parametric (e.g., histograms)
  2. Parametric (e.g., fit Gaussian with MLE)

# Curse of Dimensionality

- ▶ In practice, we have many features.
- ▶ This means  $p_X(\vec{X} = \vec{x} | Y = y)$  is **high dimensional**.
- ▶ Non-parametric estimators do not do well in high dimensions due to the **curse of dimensionality**:
  - ▶ Data required grows exponentially with number of features.

# Responses

- ▶ Parametric density estimation can fare better.
- ▶ However, it too can suffer from the curse.
- ▶ **Today**, a different approach: assume **conditional independence**.

# DSC 140A

*Probabilistic Modeling & Machine Learning*

Lecture 01 | Part 2

**What is Conditional Independence?**

# Remember: Independence

- ▶ Events  $A$  and  $B$  are **independent** if

$$\mathbb{P}(A, B) = \mathbb{P}(A) \cdot \mathbb{P}(B).$$

- ▶ Equivalently,  $A$  and  $B$  are independent if<sup>1</sup>

$$\mathbb{P}(A | B) = \mathbb{P}(A)$$

---

<sup>1</sup>or  $\mathbb{P}(B) = 0$

## Informally

- ▶  $A$  and  $B$  are **independent** if learning  $B$  does not influence your belief that  $A$  happens.

# Example

You draw one card from a deck of 52 cards.  $A$  is the event that the card is a heart,  $B$  is the event that the card is a face card (J,Q,K,A). Are these independent?

♥: 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, A

♦: 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, A

♣: 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, A

♠: 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, A

# Example

We've lost the King of Clubs! You draw one card from this deck of 51 cards.  $A$  is the event that the card is a heart,  $B$  is the event that the card is a face card (J,Q,K,A). Are these independent?

♥: 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, A

♦: 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, A

♣: 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, A

♠: 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, A

# In the Real World...

- ▶ ...true independence is rare.
  - ▶ Example, survivors of the titanic:

## In the Real World...

- ▶  $P(\text{Survived} = 1) = .408$
- ▶  $P(\text{Survived} = 1 | \text{FavColor} = \text{purple}) = .4$
- ▶ **Not independent...**

## In the Real World...

- ▶  $P(\text{Survived} = 1) = .408$
- ▶  $P(\text{Survived} = 1 | \text{FavColor} = \text{purple}) = .4$
- ▶ **Not independent... ...but “close”!**

## In the Real World...

- ▶  $P(\text{Survived} = 1) = .408$
- ▶  $P(\text{Survived} = 1 | \text{Pclass} = 1) =$

## In the Real World...

- ▶  $P(\text{Survived} = 1) = .408$
- ▶  $P(\text{Survived} = 1 | \text{Pclass} = 1) = .657$

## In the Real World...

- ▶  $P(\text{Survived} = 1) = .408$
- ▶  $P(\text{Survived} = 1 | \text{Pclass} = 1) = .657$
- ▶ **Strong dependence.**

# Remember: Conditional Independence

- ▶ Events  $A$  and  $B$  are **conditionally independent** given  $C$  if

$$\mathbb{P}(A, B | C) = \mathbb{P}(A | C) \cdot \mathbb{P}(B | C)$$

- ▶ Equivalently<sup>2</sup>:

$$\mathbb{P}(A | B, C) = \mathbb{P}(A | C)$$

---

<sup>2</sup>Or  $\mathbb{P}(B) = 0$

# Informally

- ▶ Suppose you know that  $C$  has happened.
- ▶ You have some belief that  $A$  happens, given  $C$ .
- ▶  $A$  and  $B$  are **conditionally independent** given  $C$  if learning that  $B$  happens in addition to  $C$  does not influence your belief that  $A$  happens given  $C$ .

## **Very informally**

- ▶  $A$  and  $B$  are **conditionally independent** given  $C$  if learning that  $B$  happens in addition to  $C$  gives you no more information about  $A$ .

# Example

We've lost the King of Clubs! You draw one card from this deck of 51 cards.  $A$  is the event that the card is a heart,  $B$  is the event that the card is a face card (J,Q,K,A). Now suppose you know that the card is **red**. Are  $A$  and  $B$  independent **given** this information?

♥: 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, A

♦: 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, A

♣: 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, A

♠: 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, A

# Titanic Example

- ▶ Survival and class are **not** independent.
  - ▶  $\mathbb{P}(\text{Survived} = 1) = .408$
  - ▶  $\mathbb{P}(\text{Survived} = 1 | \text{Pclass} = 1) = .657$
- ▶ But they're (close) to **conditionally independent** given ticket price:
  - ▶  $\mathbb{P}(\text{Survived} = 1 | \text{PClass} = 1, \text{Fare} > 50) = .708$
  - ▶  $\mathbb{P}(\text{Survived} = 1 | \text{Fare} > 50) = .696$

# More Variables

- ▶  $X_1, X_2, \dots, X_d$  are **mutually conditionally independent** given  $Y$  if

$$\mathbb{P}(X_1, X_2, \dots, X_d | Y) = \mathbb{P}(X_1 | Y) \cdot \mathbb{P}(X_2 | Y) \cdots \mathbb{P}(X_d | Y)$$

# Densities

- ▶ If  $A$  and  $B$  are **continuous** random variables, their joint density can be factored:

$$p(a, b) = p_A(a) \cdot p_B(b)$$

- ▶ If  $A$  and  $B$  are **conditionally independent** given  $C$ , then:

$$p(a, b | C = c) = p_A(a | C = c) \cdot p_B(b | C = c)$$

# Densities

- ▶ Suppose  $X_1, \dots, X_d$  are  $d$  features,  $Y$  is class label.
- ▶ If the features are not independent given  $Y$ , then:

$$p(\vec{x} | Y = y) = p(x_1, x_2, \dots, x_d | Y = y)$$

- ▶ **Curse of dimensionality!**

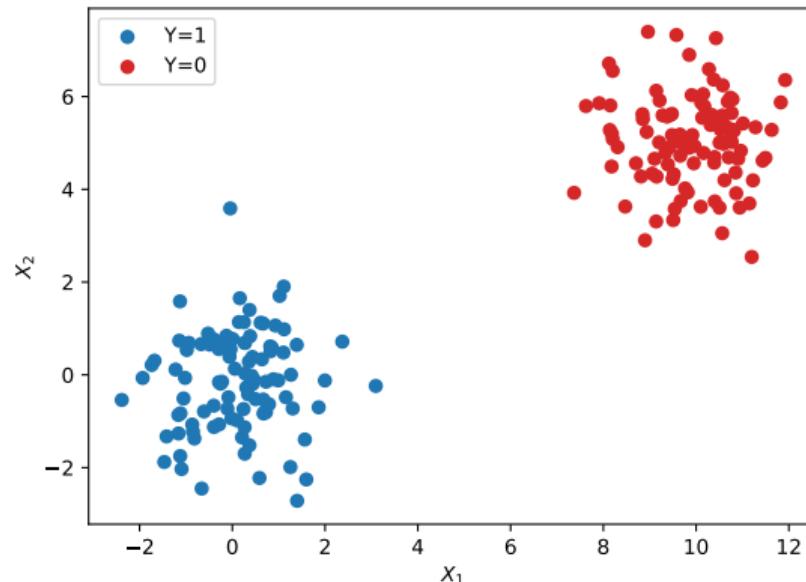
# Densities

- ▶ Suppose  $X_1, \dots, X_d$  are  $d$  features,  $Y$  is class label.
- ▶ However, if the features are **mutually conditionally independent** given  $Y$ , then:

$$\begin{aligned} p(\vec{x} | Y = y) &= p(x_1, x_2, \dots, x_d | Y = y) \\ &= p_1(x_1 | Y = y) \cdot p_2(x_2 | Y = y) \cdots p_d(x_d | Y = y) \end{aligned}$$

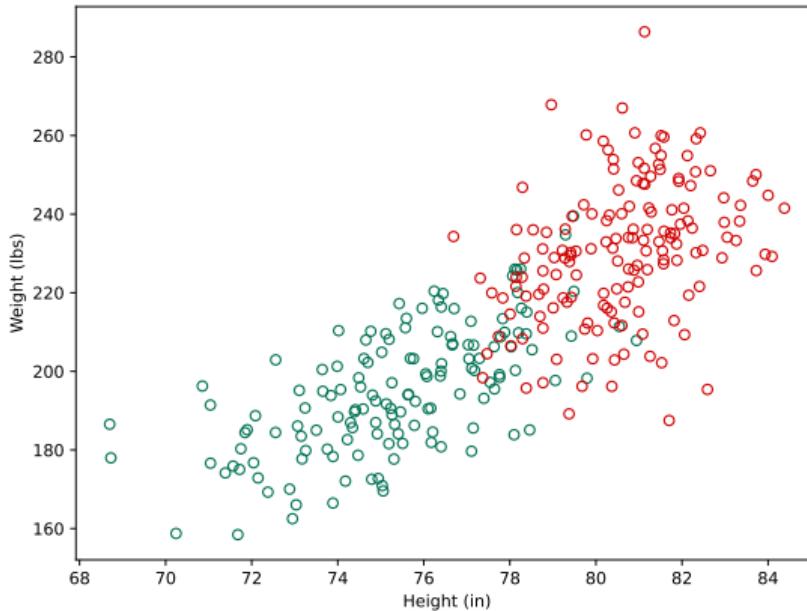
## Exercise

Are  $X_1$  and  $X_2$  (close to) conditionally independent given  $Y$ ?



## Exercise

Are height and weight (close to) conditionally independent given the player's position?



# DSC 140A

*Probabilistic Modeling & Machine Learning*

Lecture 01 | Part 3

**How Conditional Independence Helps**

# Recall: The Bayes Classifier

- ▶ To use the Bayes classifier, we must estimate

$$p(\vec{x} \mid Y = y_i)$$

for each class  $y_i$ , where  $\vec{x} = (x_1, x_2, \dots, x_d)$ .

- ▶ Written differently, we need to estimate:

$$p(x_1, \dots, x_d \mid Y = y_i)$$

# Recall: Histogram Estimators

- ▶ When  $X_1, \dots, X_d$  are continuous, we can use **histogram estimators**.
- ▶ **Curse of Dimensionality**: if we discretize each dimension into 10 bins, there are  $10^d$  bins.

# Conditional Independence to the Rescue

- ▶ Now suppose  $X_1, \dots, X_d$  are mutually conditionally independent given  $Y$ . Then:

$$p(x_1, \dots, x_d | Y = y_i) = p_1(x_1 | Y = y_i)p_2(x_2 | Y = y_i) \cdots p_d(x_d | Y = y_i)$$

- ▶ Instead of estimating  $p(x_1, \dots, x_d | Y)$ , estimate  $p_1(x_1 | Y), \dots, p_d(x_d | Y)$  separately.

# Breaking the Curse

- ▶ Suppose we use histogram estimators.
- ▶ If we discretize each dimension into 10 bins, we need:
  - ▶ 10 bins to estimate  $p_1(x_1|Y)$
  - ▶ 10 bins to estimate  $p_2(x_2|Y)$
  - ▶ ...
  - ▶ 10 bins to estimate  $p_d(x_d|Y)$
- ▶ We therefore need  $10d$  bins in total.

# Breaking the Curse

- ▶ Conditional independence **drastically reduced** the number of bins needed to cover the input space.
- ▶ From  $\Theta(10^d)$  to  $\Theta(d)$ .

# Idea

- ▶ Bayes Classifier needs a lot of data when  $d$  is big.
- ▶ But if the features are conditionally independent given the label, we don't need so much data.
- ▶ So let's just **assume** conditional independence.
- ▶ The result: the **Naïve Bayes Classifier**.

# Naïve Bayes: The Algorithm

- ▶ **Assume** that  $X_1, \dots, X_d$  are mutually independent given the class label.
- ▶ Estimate **one-dimensional** densities  $p_1(x_1 | Y = y_i), \dots, p_d(x_d | Y = y_i)$  however you'd like.
  - ▶ histograms, fitting univariate Gaussians, etc.
- ▶ Pick the  $y_i$  which maximizes

$$p_1(x_1 | Y = y_i) \cdots p_d(x_d | Y = y_i) \mathbb{P}(Y = y_i)$$

## But wait...

- ▶ ...are we allowed to just **assume** conditional independence?
- ▶ Sure!
- ▶ The independence assumption is usually **wrong**, but it can work surprisingly well in practice.

# Estimating Probabilities

- ▶ You can estimate  $p(X_i|Y)$  however makes sense.
- ▶ Popular: **Gaussian Naïve Bayes.**

## Example: NBA

- ▶ **Given:** player with height = 75 in, weight = 210 lbs.
- ▶ **Predict:** whether they are a forward or a guard.
- ▶ Let's use Gaussian Naïve Bayes.

# Example: NBA

- ▶ Compute:

$$p(75 \text{ in}, 210 \text{ lbs} | Y = \text{forward})\mathbb{P}(Y = \text{forward})$$

$$p(75 \text{ in}, 210 \text{ lbs} | Y = \text{guard})\mathbb{P}(Y = \text{guard})$$

- ▶ Using conditional independence assumption:

$$p_1(75 \text{ in} | Y = \text{forward}) \cdot p_2(210 \text{ lbs} | Y = \text{forward})\mathbb{P}(Y = \text{forward})$$

$$p_1(75 \text{ in} | Y = \text{guard}) \cdot p_2(210 \text{ lbs} | Y = \text{guard})\mathbb{P}(Y = \text{guard})$$

# Example: NBA

- We need to estimate:

$$p_1(75 \text{ in} \mid Y = \text{forward})$$

$$p_1(75 \text{ in} \mid Y = \text{guard})$$

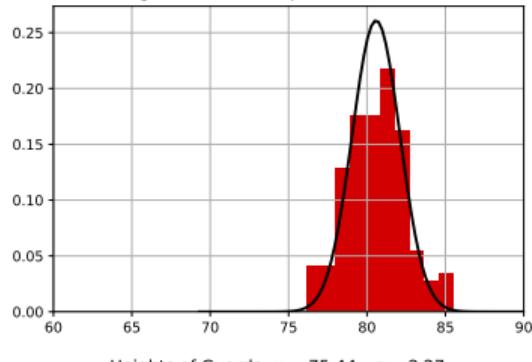
$$p_2(210 \text{ lbs} \mid Y = \text{forward})$$

$$p_2(210 \text{ lbs} \mid Y = \text{guard})$$

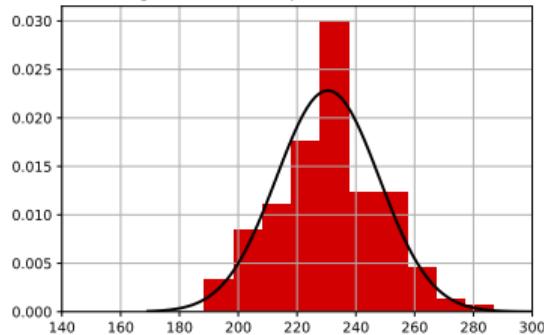
# Example: NBA

- ▶ We'll fit 1-d Gaussians to:
  - ▶ heights of forwards.
  - ▶ heights of guards.
  - ▶ weights of forwards.
  - ▶ weights of guards.

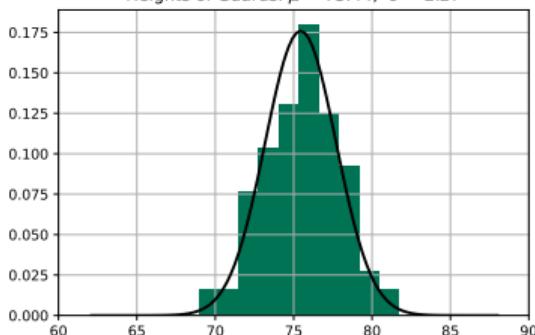
Heights of Forwards:  $\mu = 80.58$ ,  $\sigma = 1.53$



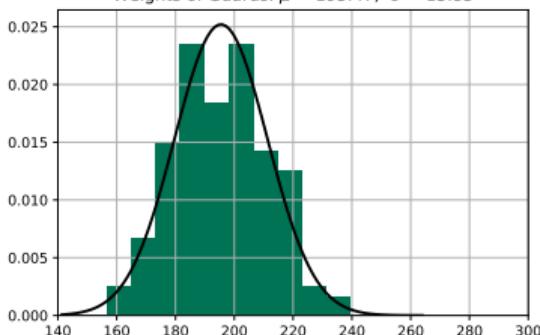
Weights of Forwards:  $\mu = 230.46$ ,  $\sigma = 17.48$



Heights of Guards:  $\mu = 75.44$ ,  $\sigma = 2.27$



Weights of Guards:  $\mu = 195.47$ ,  $\sigma = 15.83$



# Example: NBA

$$\begin{aligned} & p_1(75 \mid Y = \text{forward}) \cdot p_2(210 \mid Y = \text{forward}) \cdot \mathbb{P}(Y = \text{forward}) \\ &= \mathcal{N}(75; 80.58, 1.53^2) \cdot \mathcal{N}(210; 230.46, 17.48^2) \cdot \frac{156}{300} \\ &\approx 6.73 \times 10^{-6} \end{aligned}$$

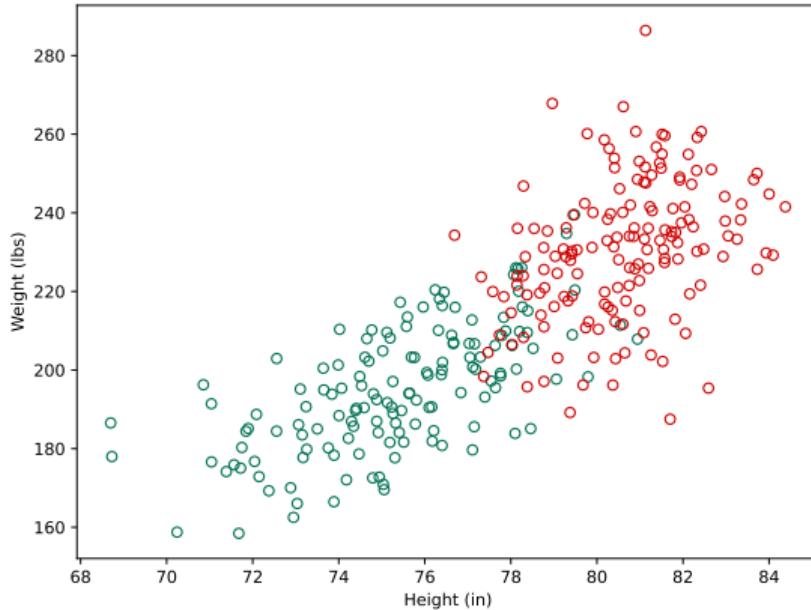
$$\begin{aligned} & p_1(75 \mid Y = \text{guard}) \cdot p_2(210 \mid Y = \text{guard}) \cdot \mathbb{P}(Y = \text{guard}) \\ &= \mathcal{N}(75; 75.44, 2.27^2) \cdot \mathcal{N}(210; 195.47, 15.83^2) \cdot \frac{144}{300} \\ &\approx 5.88 \times 10^{-5} \end{aligned}$$

## **Example: NBA**

- ▶ About 85% accurate on test set.

## Exercise

Are height and weight conditionally independent given the player's position?



## Example: NBA

- ▶ No!
- ▶ Gaussian Naïve Bayes worked well even though the conditional independence assumption is not accurate.

# Gaussian Naïve Bayes

- ▶  $p(X_1 | Y) \cdots p(X_d | Y)$  is a product of 1-d Gaussians with different means, variances.
- ▶ Remember: result is a  $d$ -dimensional Gaussian with diagonal covariance matrix:

$$C = \begin{pmatrix} \sigma_1^2 & 0 & \cdots & 0 \\ 0 & \sigma_2^2 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & \sigma_d^2 \end{pmatrix}$$

# Gaussian Naïve Bayes

- ▶ But in GNB, each class has own diagonal covariance matrix.
- ▶ Therefore: Gaussian Naïve Bayes is **equivalent** to QDA with diagonal covariances.

# Beyond Gaussian

- ▶ Naïve Bayes is very flexible.
- ▶ Can use different parametric distributions for different features.
  - ▶ E.g., normal for feature 1, log normal for feature 2, etc.
- ▶ Can use non-parametric density estimation (densities) for other features.
- ▶ Can also handle discrete features.

# **Up next...**

...predicting who survives on the Titanic.

# DSC 140A

*Probabilistic Modeling & Machine Learning*

Lecture 01 | Part 4

**The Titanic**

# The Titanic Dataset

PassengerID	Survived	Pclass	Sex	Age	Fare	Embarked	FavColor
0	0	3	female	23.0	7.9250	S	yellow
1	0	1	male	47.0	52.0000	S	purple
2	0	3	male	36.0	7.4958	S	green
3	0	3	male	31.0	7.7500	Q	purple
4	0	3	male	19.0	7.8958	S	purple
...	...	...	...	...	...	...	...

Goal: predict survival given Age, Sex, Pclass.

# Let's use Naïve Bayes

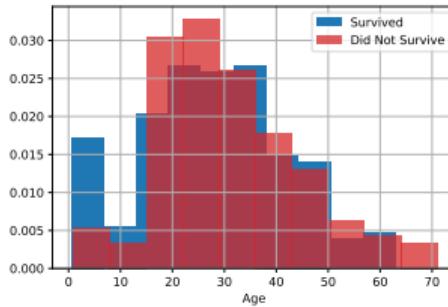
- ▶ We'll pick  $y_i$  so as to maximize

$$p(\text{Age} = x_1 \mid Y = y_i) \cdot \mathbb{P}(\text{Sex} = x_2 \mid Y = y_i) \cdot \mathbb{P}(\text{Pclass} = x_3 \mid Y = y_i) \cdot \mathbb{P}(Y = y_i)$$

- ▶ We must choose how to estimate probabilities.  
Gaussians?

# Estimating Probabilities

- ▶ How do we estimate  $p(\text{Age} = x_1 | Y = y_i)$ ?
- ▶ Age is a continuous variable.
- ▶ Looks kind of bell-shaped, we'll fit Gaussians.



# Estimating Probabilities

- ▶ How do we estimate  $\mathbb{P}(\text{Sex} = x_1 \mid Y = y_i)$ ?
- ▶ Sex is a **discrete** variable in this data set.
- ▶ Fitting Gaussian makes no sense.
- ▶ But estimating these probabilities is easy.

# Estimating Probabilities

$$\begin{aligned}\mathbb{P}(\text{Sex} = \text{male} \mid \text{Survived}) &\approx \frac{\# \text{ of survived and male}}{\# \text{ of survived}} \\ &= .4\end{aligned}$$

$$\begin{aligned}\mathbb{P}(\text{Sex} = \text{male} \mid \text{Did Not Survive}) &\approx \frac{\# \text{ of died and male}}{\# \text{ of died}} \\ &= .87\end{aligned}$$

# Estimating Probabilities

- ▶ Pclass, too, is categorical. Estimate in same way.
- ▶ You can estimate  $\mathbb{P}(X_i|Y)$  however makes sense.
- ▶ **Can use different ways for different features.**
- ▶ Gaussian for age, simple ratio of counts for class, sex.

## Example: The Titanic

- ▶ Using just age, sex, ticket class, Naïve Bayes is 70% accurate on test set.
- ▶ Not bad. Not great.
- ▶ To do better, add more features.

# In High Dimensions

- ▶ Naïve Bayes can work well in high dimensions.
- ▶ Example: document classification.
  - ▶ Document represented by a “bag of words”.
  - ▶ Pick a large number of words; say, 20,000.
  - ▶ Make a  $d$ -dimensional vector with  $i$ th entry counting number of occurrences of  $i$ th word.

# Practical Issues

- ▶ We are multiplying lots of small probabilities:

$$\mathbb{P}(X_1 | Y) \cdots \mathbb{P}(X_d | Y)$$

- ▶ Potential for **underflow**.

# Practical Issues

- ▶ “Trick”: work with log-probabilities instead.
- ▶ Pick the  $y_i$  which maximizes

$$\begin{aligned} & \log [\mathbb{P}(X_1 = x_1 | Y = y_i) \cdots \mathbb{P}(X_d = x_d | Y = y_i) \mathbb{P}(Y = y_i)] \\ &= \log \mathbb{P}(X_1 = x_1 | Y = y_i) + \dots + \log \mathbb{P}(X_d = x_d | Y = y_i) + \log \mathbb{P}(Y = y_i) \\ &= \left( \sum_{j=1}^d \log \mathbb{P}(X_j = x_j | Y = y_i) \right) + \log \mathbb{P}(Y = y_i) \end{aligned}$$

# DSC 140A

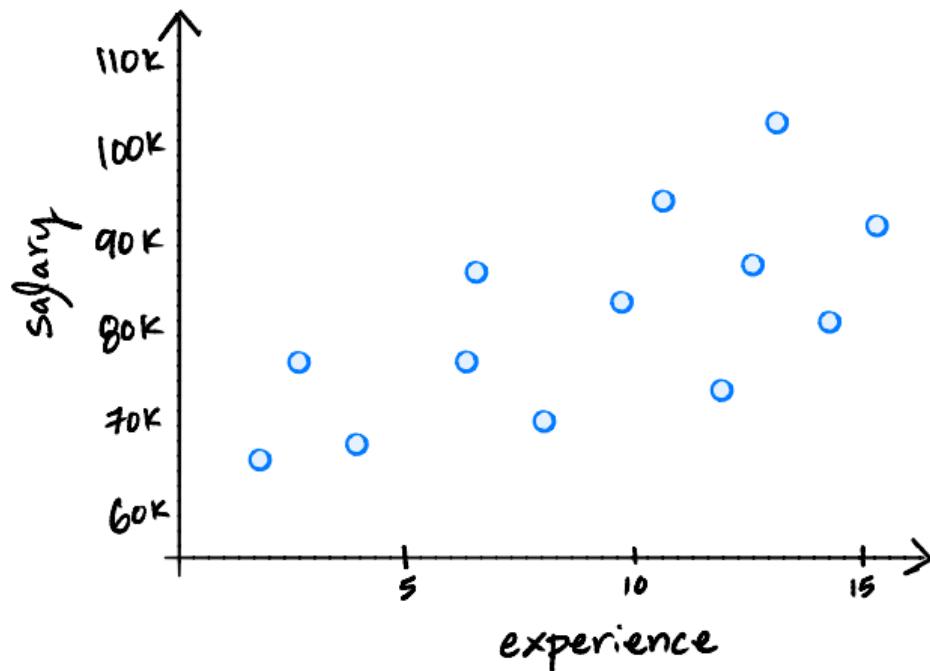
*Probabilistic Modeling & Machine Learning*

Lecture 15 | Part 1

**Recall: Regression**

# Recall

- We have seen the problem of regression.



# Recall

- ▶ Introduced **empirical risk minimization (ERM)**:
- ▶ Step 1: choose a **hypothesis class**
  - ▶ Let's assume we've chosen linear predictors
- ▶ Step 2: choose a **loss function**
  - ▶ Used square loss
- ▶ Step 3: minimize **expected loss (empirical risk)**
  - ▶ MSE (Mean Squared Error)

# Recall: Least Squares

- ▶ Goal: fit a function of the form  $H(\vec{x}; \vec{w}) = \text{Aug}(\vec{x}) \cdot \vec{w}$
- ▶ In (ordinary) least squares regression, we **minimized** the **mean squared error**:

$$\vec{w}^* = \arg \min_{\vec{w}} \frac{1}{n} \sum_{i=1}^n (H(\vec{x}^{(i)}; \vec{w}) - y_i)^2$$

- ▶ **Solution:**  $\vec{w}^* = (X^T X)^{-1} X^T \vec{y}$

# Observation

- ▶ This is the “curve fitting” approach to regression.
- ▶ I.e., find a “line of best fit”.
- ▶ There was no consideration of the (random) process that generated the data.

# **Today**

- ▶ Take a probabilistic approach to regression.

# DSC 140A

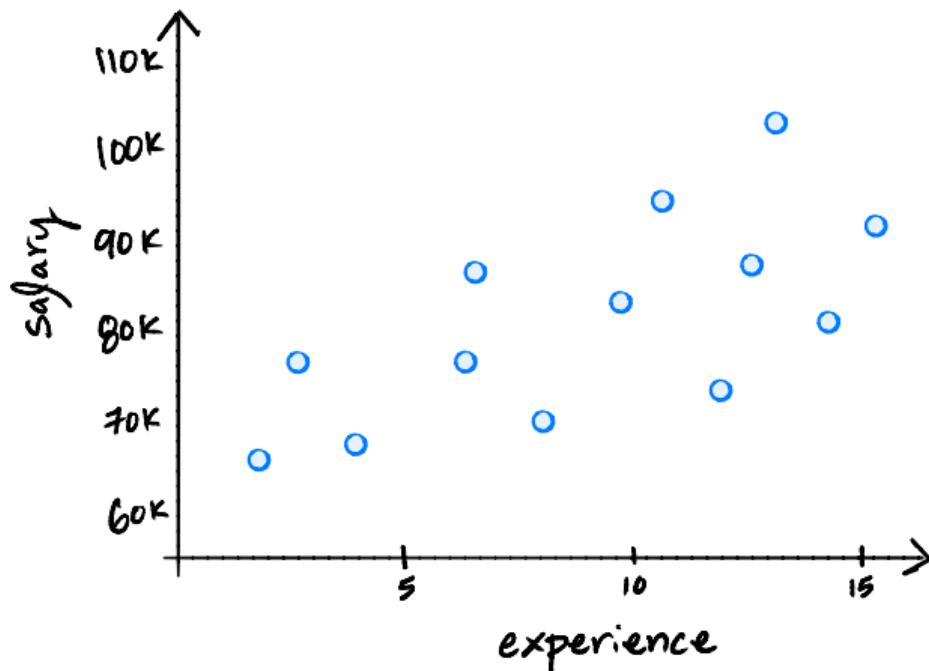
*Probabilistic Modeling & Machine Learning*

Lecture 15 | Part 2

## Probabilistic View of Regression

# Probabilistic View of Regression

- ▶ Note: There is **uncertainty** in the salary.

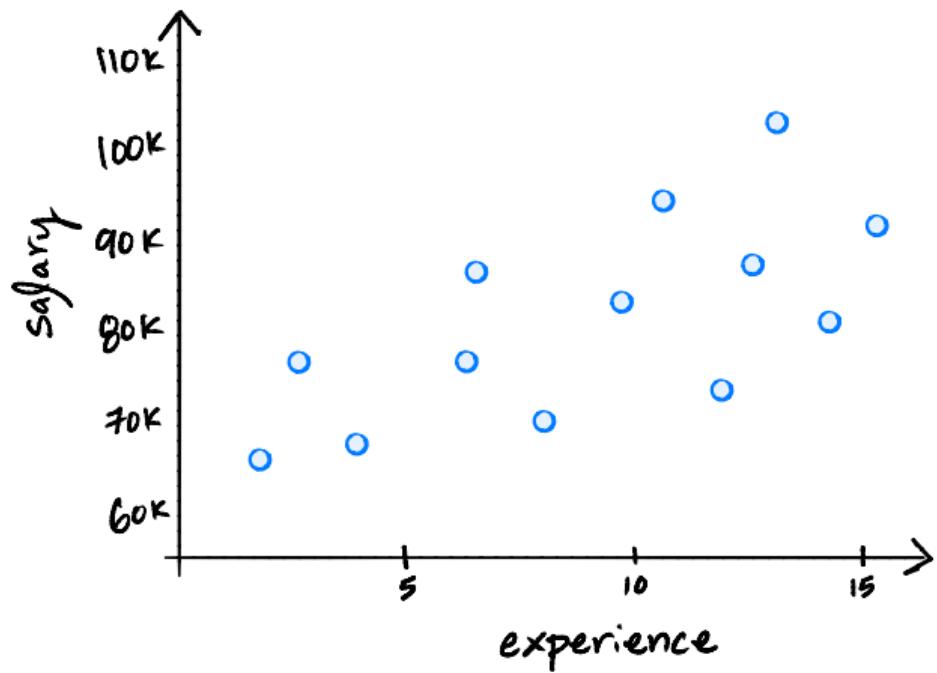


# Modeling Uncertainty

- ▶ We can model this uncertainty using probability.

$$\text{Salary} = w_0 + w_1 \times (\text{Experience}) + \varepsilon$$

- ▶ Here,  $\varepsilon$  is the (random) **error**.
- ▶ What is a reasonable choice of **distribution** for  $\varepsilon$ ?



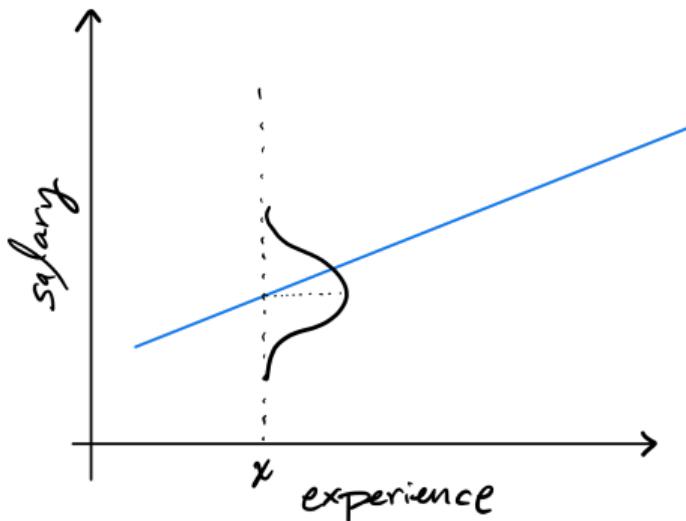
# Error Distribution

- ▶ It is reasonable to assume that the error distribution is:
  - ▶ **Symmetric:** equally as likely to predict high as to predict low
  - ▶ **Centered at zero:** mean error is zero
- ▶ The **Gaussian distribution** (with mean 0) satisfies this.

# Modeling Uncertainty

- ▶ Assuming a Gaussian (Normal) distribution:

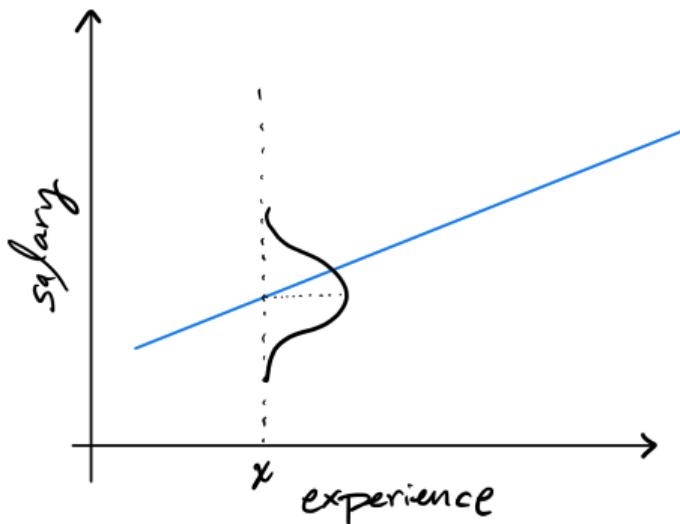
$$\text{Salary} = w_0 + w_1 \times (\text{Experience}) + \underbrace{\mathcal{N}(0, \sigma^2)}_{\varepsilon}$$



# Modeling Uncertainty

- ▶ Equivalently:

$$\text{Salary} \sim \mathcal{N}(w_0 + w_1 \times \text{Experience}, \sigma^2)$$



## In General

- ▶ In general:

$$Y \sim \mathcal{N}(\text{Aug}(\vec{x}) \cdot \vec{w}, \sigma^2)$$

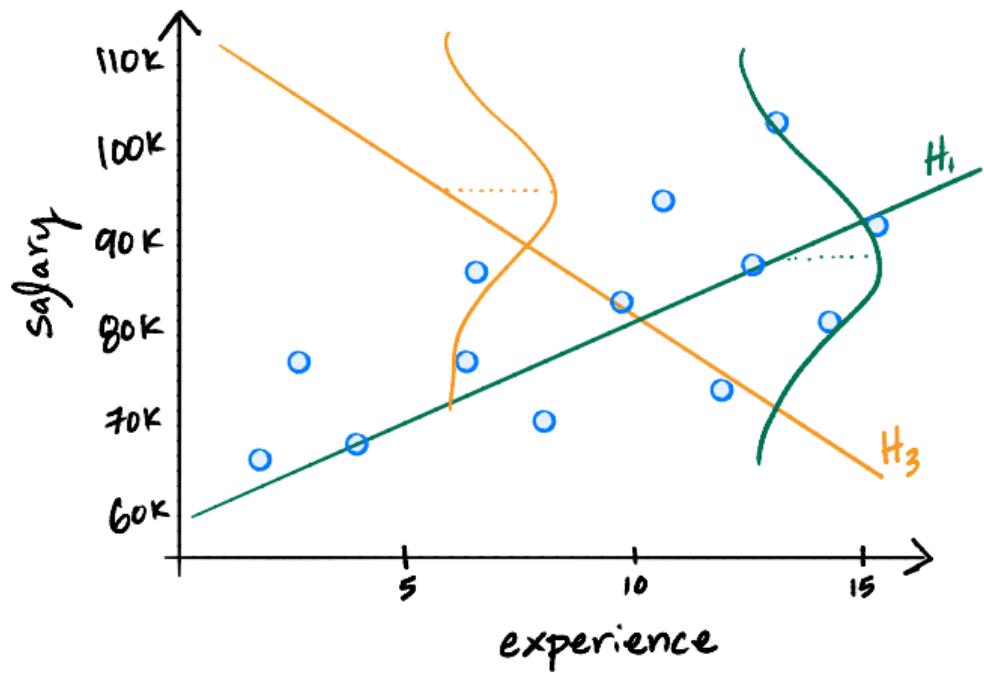
- ▶ That is: for any feature vector  $\vec{x}$ , the target  $Y$  is drawn from a Gaussian centered at  $\text{Aug}(\vec{x}) \cdot \vec{w}$ .

# Estimating Parameters

- ▶ We assume the model:

$$\text{Salary} \sim \mathcal{N}(w_0 + w_1 \times \text{Experience}, \sigma^2)$$

- ▶ Given some data, what parameters generated it?
  - ▶ What were  $w_0$ ,  $w_1$ ,  $\sigma$ ?
- ▶ **Estimate** them with maximum likelihood?



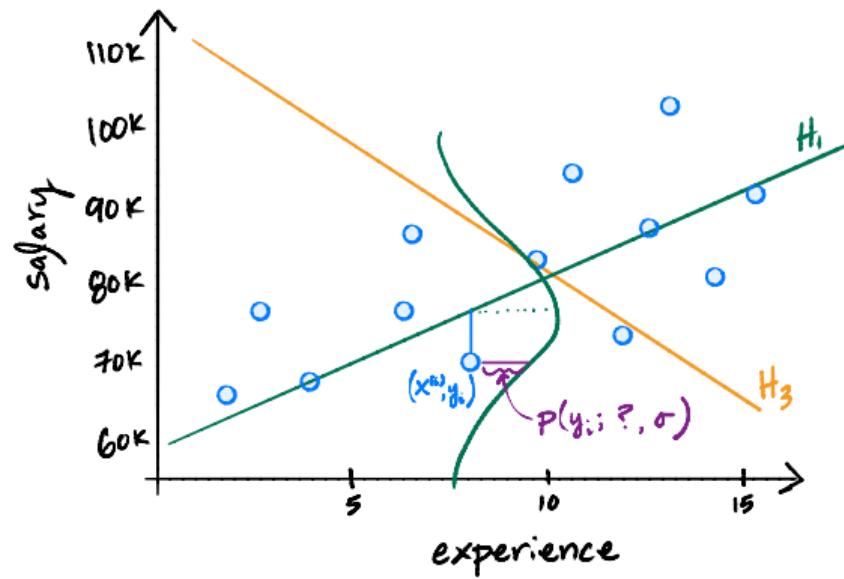
# Likelihood

- ▶ Let  $p(y; \mu, \sigma)$  be the Gaussian pdf:

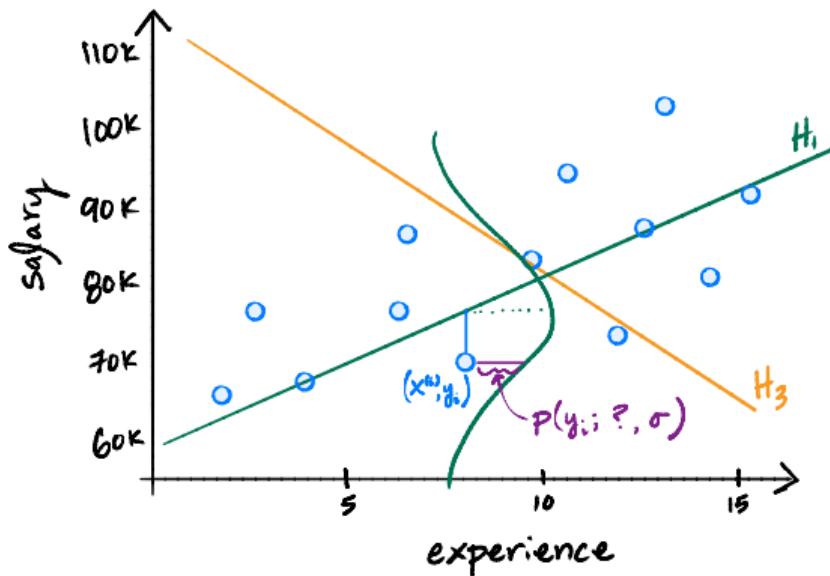
$$p(y; \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(y-\mu)^2/(2\sigma^2)}$$

- ▶ We observe a data set  $\{(\vec{x}^{(i)}, y_i)\}$ .
- ▶ What is the likelihood of a choice of parameters  $\vec{w}, \sigma$ , with respect to the data?

# Likelihood wrt a Point



# Likelihood wrt a Point



- ▶  $p(y_i; w_0 + w_1 x^{(i)}, \sigma)$  measures likelihood with respect to  $(x^{(i)}, y_i)$ .

# Likelihood

- ▶ In general,

$$p(y_i; \text{Aug}(\vec{x}^{(i)}) \cdot \vec{w}, \sigma)$$

measures likelihood with respect to single data point  $(\vec{x}^{(i)}, y_i)$ .

- ▶ Likelihood with respect to data set:

$$L(\vec{w}, \sigma) = \prod_{i=1}^n p(y_i; \text{Aug}(\vec{x}^{(i)}) \cdot \vec{w}, \sigma)$$

# Log-Likelihood

Compute the log-likelihood from  
 $\prod_{i=1}^n p(y_i; \text{Aug}(\vec{x}^{(i)}) \cdot \vec{w}, \sigma).$

# Log-Likelihood

- ▶ The log-likelihood is:

$$\tilde{L}(\vec{w}, \sigma) = -\frac{1}{2\sigma^2} \sum_{i=1}^n (\text{Aug}(\vec{x}^{(i)}) \cdot \vec{w} - y_i)^2 + \frac{n}{2} \ln \frac{1}{\sigma^2} - \frac{n}{2} \ln(2\pi)$$

- ▶ We want to **maximize** this quantity.

# Claim 1

$$\begin{aligned} & \arg \max_{\vec{w}} \left[ -\frac{1}{2\sigma^2} \sum_{i=1}^n (\text{Aug}(\vec{x}^{(i)}) \cdot \vec{w} - y_i)^2 + \frac{n}{2} \ln \frac{1}{\sigma^2} - \frac{n}{2} \ln(2\pi) \right] \\ & = \end{aligned}$$

$$\arg \max_{\vec{w}} \left[ -\frac{1}{2\sigma^2} \sum_{i=1}^n (\text{Aug}(\vec{x}^{(i)}) \cdot \vec{w} - y_i)^2 \right]$$

## Claim 2

$$\arg \max_{\vec{w}} \left[ -\frac{1}{2\sigma^2} \sum_{i=1}^n (\text{Aug}(\vec{x}^{(i)}) \cdot \vec{w} - y_i)^2 \right]$$

=

$$\arg \max_{\vec{w}} \left[ -\frac{1}{n} \sum_{i=1}^n (\text{Aug}(\vec{x}^{(i)}) \cdot \vec{w} - y_i)^2 \right]$$

# Claim 3

$$\arg \max_{\vec{w}} \left[ -\frac{1}{n} \sum_{i=1}^n (\text{Aug}(\vec{x}^{(i)}) \cdot \vec{w} - y_i)^2 \right]$$

=

$$\arg \min_{\vec{w}} \left[ \frac{1}{n} \sum_{i=1}^n (\text{Aug}(\vec{x}^{(i)}) \cdot \vec{w} - y_i)^2 \right]$$

- ▶ That is, minimize the **mean squared error**.

## Main Idea

Mazimizing the likelihood of  $\vec{w}$  with respect to the data (assuming Gaussian error term) is **equivalent** to minimizing mean squared error.

# Solution

- ▶ The maximum likelihood estimate for  $\vec{w}$  is therefore:

$$\vec{w}^* = (X^T X)^{-1} X^T \vec{y}$$

- ▶ That is, the exact same as we obtained by empirical risk minimization with the square loss.

# DSC 140A

*Probabilistic Modeling & Machine Learning*

Lecture 15 | Part 3

## A Probabilistic View of Regularization

# Recall: Ridge Regression

- ▶ In **ridge regression**, we added a regularization term:  $\|\vec{w}\|^2$ .

$$\vec{w}^* = \arg \min_{\vec{w}} \frac{1}{n} \sum_{i=1}^n (H(\vec{x}^{(i)}; \vec{w}) - y_i)^2 + \lambda \|\vec{w}\|^2$$

- ▶ **Solution:**  $\vec{w}^* = (X^T X + n\lambda I)^{-1} X^T \vec{y}$

- ▶ Helps control overfitting.

# Probabilistic View

- ▶ Regularization term  $\|\vec{w}\|^2$  was motivated by observing that  $\|\vec{w}\|$  tends to be large when overfitting.
- ▶ Now: motivate same term, probabilistically.
- ▶ Will adopt a **Bayesian** perspective.

# A Prior on Weights

- ▶ Imagine we have yet to see the data.
- ▶ There is no reason to believe that a given weight  $w_i$  is positive or negative.
- ▶ We believe it is more likely to be small (close to zero) than large.

# A Prior on Weights

- ▶ This **prior belief** is captured by assuming:

$$w_i \sim \mathcal{N}(0, s^2)$$

- ▶ Note that in truth,  $w_i$  is **not** random.
- ▶ We are adopting a **Bayesian** view of probability; it expresses level of belief.

# A Prior on Weights

- ▶ If each weight has distribution  $\mathcal{N}(0, s^2)$ , then:

$$\vec{w} \sim \mathcal{N}(\vec{0}, s^2 \cdot I)$$

- ▶ That is, the distribution of  $\vec{w}$  has density:

$$p_{\vec{w}}(\vec{w}) = \frac{1}{(2\pi s^2)^{d/2}} e^{-\frac{1}{2} \frac{\|\vec{w}-\vec{0}\|^2}{s^2}}$$

# Distribution of $\vec{w}$

- ▶ Using Bayes' Rule:

$$p_{\vec{w}}(\vec{w} | \vec{x}, y) \propto p_y(y | \vec{w}, \vec{x})p_{\vec{w}}(\vec{w})$$

- ▶ What is the most probable value of  $\vec{w}$ ?

$$\begin{aligned}\arg \max_{\vec{w}} [p_{\vec{w}}(\vec{w} | \vec{x}, y)] &= \arg \max_{\vec{w}} [p_y(y | \vec{w}, \vec{x}) p_{\vec{w}}(\vec{w})] \\&= \arg \max_{\vec{w}} \ln [p_y(y | \vec{w}, \vec{x}) p_{\vec{w}}(\vec{w})] \\&= \arg \max_{\vec{w}} [\ln p_y(y | \vec{w}, \vec{x}) + \ln p_{\vec{w}}(\vec{w})] \\&= \arg \min_{\vec{w}} [-\ln p_y(y | \vec{w}, \vec{x}) - \ln p_{\vec{w}}(\vec{w})] \\&= \arg \min_{\vec{w}} [\text{MSE}(\vec{w}) - \ln p_{\vec{w}}(\vec{w})]\end{aligned}$$

# Deriving the Regularizer

- ▶ Since

$$p_{\vec{w}}(\vec{w}) = \frac{1}{(2\pi s^2)^{d/2}} e^{-\frac{1}{2} \frac{\|\vec{w}-\vec{0}\|^2}{s^2}}$$

we have:

$$-\ln p_{\vec{w}}(\vec{w}) = c + \frac{1}{2s^2} \|\vec{w}\|^2$$

- ▶ So

$$\arg \min_{\vec{w}} [\text{MSE}(\vec{w}) - \ln p_{\vec{w}}(\vec{w})] = \arg \min_{\vec{w}} \left[ \text{MSE}(\vec{w}) + \underbrace{\frac{1}{2s^2}}_{\lambda} \|\vec{w}\|^2 \right]$$

## Main Idea

Placing a  $\mathcal{N}(0, s^2)$  prior on each weight and maximizing  $p_{\vec{w}}(\vec{w} | \vec{x}, y)$  is equivalent to minimizing the  $\|\vec{w}\|^2$ -regularized mean squared error (**ridge regression**).

# DSC 140A

Probabilistic Modeling & Machine Learning

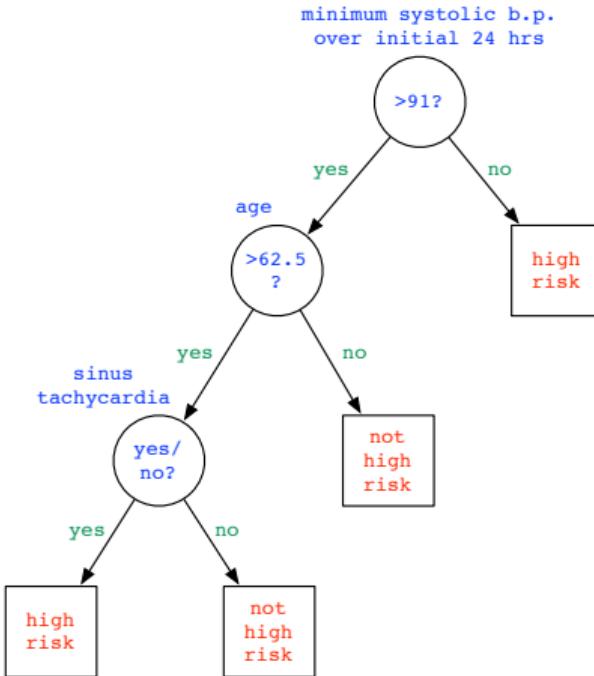
Lecture 16 | Part 1

**Decision Trees**

# The Problem

- ▶ UCSD Medical Center (1970s): identify patients at risk of dying within 30 days after heart attack.

# A Decision Tree



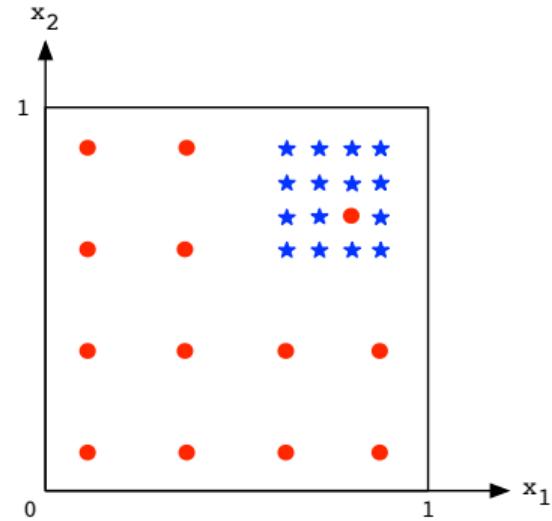
# Decision Trees

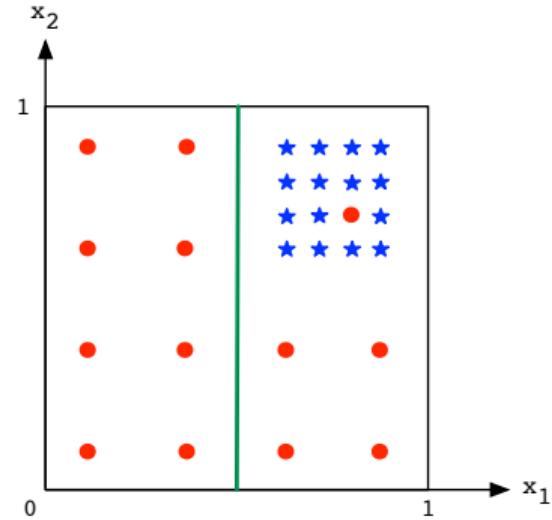
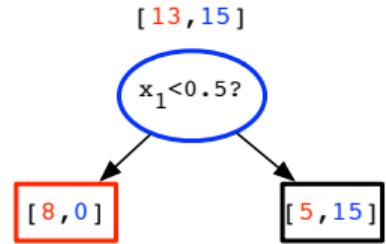
- ▶ A **decision tree** is a rooted tree.
- ▶ Internal nodes ask yes/no questions.
  - ▶ **Categorical:** Is patient a male?
  - ▶ **Numerical:** Is patient's age > 62.5 years?
- ▶ Leaf nodes are decisions (class labels).
- ▶ Path from root is a sequence of “and”s:
  - ▶ Is patient over 62.5 **and** male **and** BP > 100?  
Then high risk.

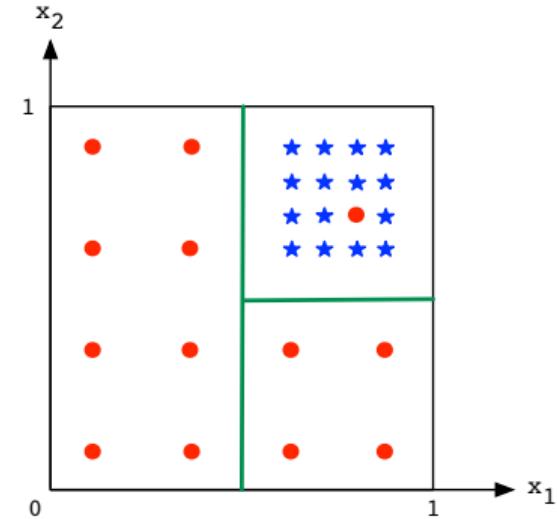
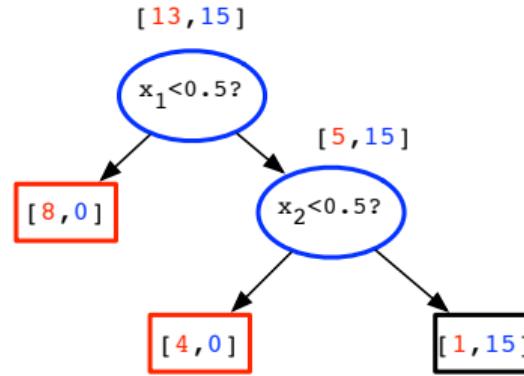
# Learning Decision Trees

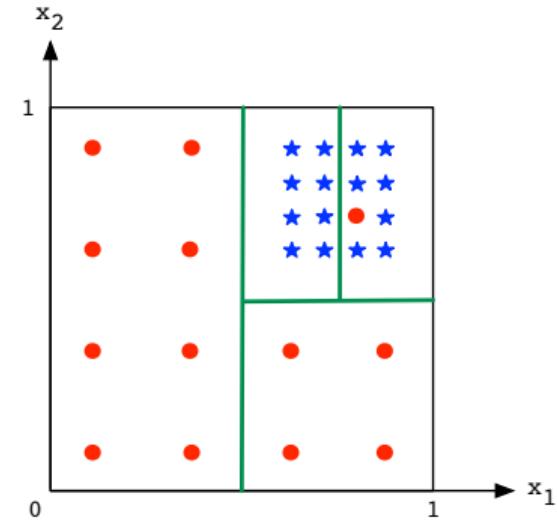
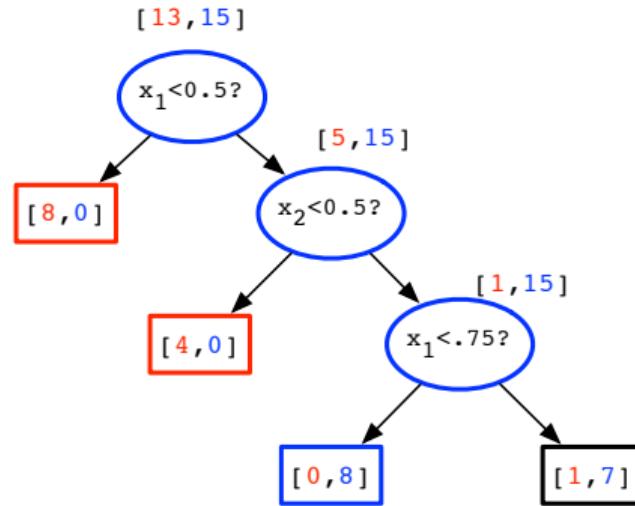
- ▶ How do we **learn** a tree from data?
  - ▶ Find right sequence of questions so that each training point is correctly classified.

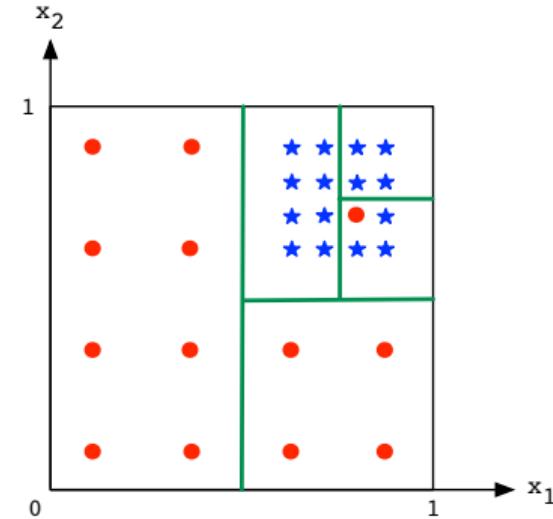
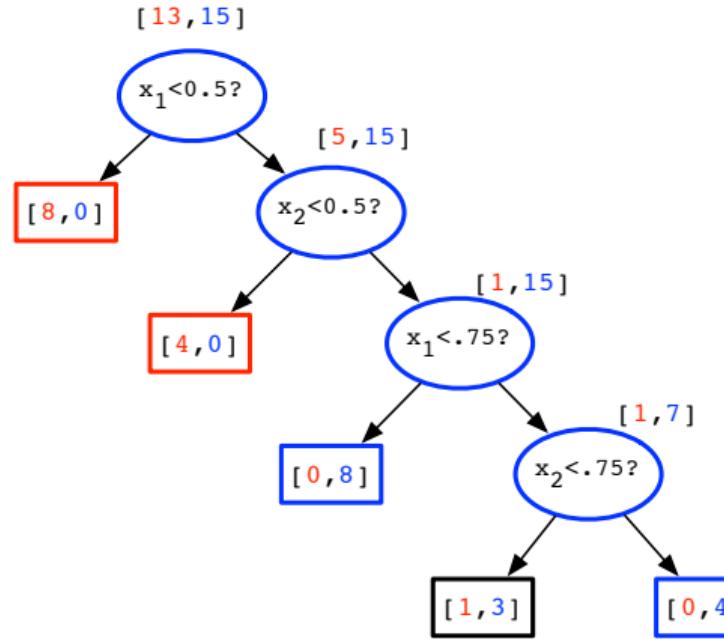
[ 13 , 15 ]

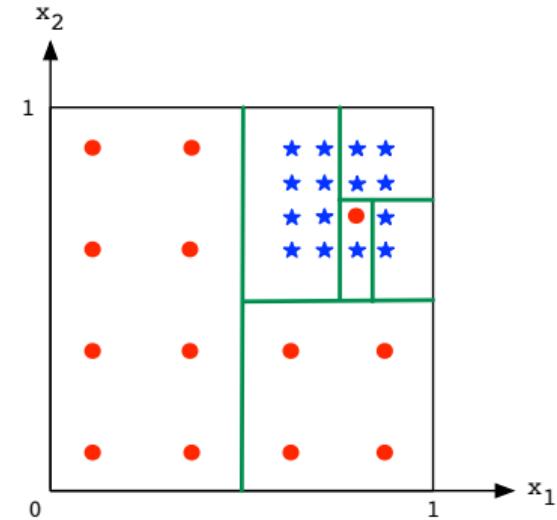
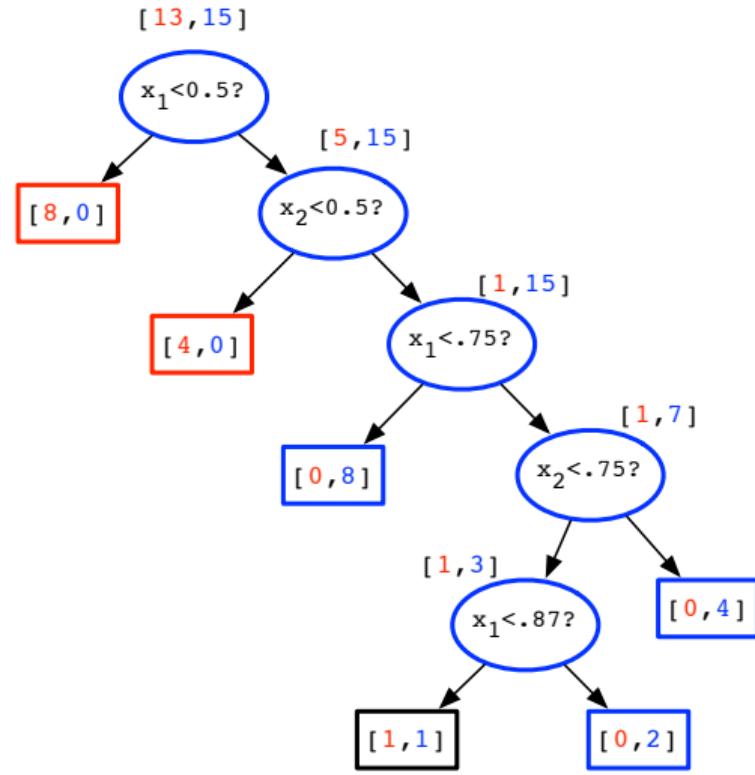


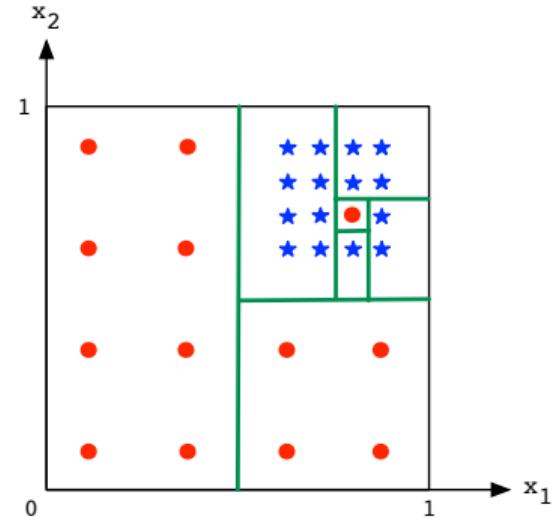
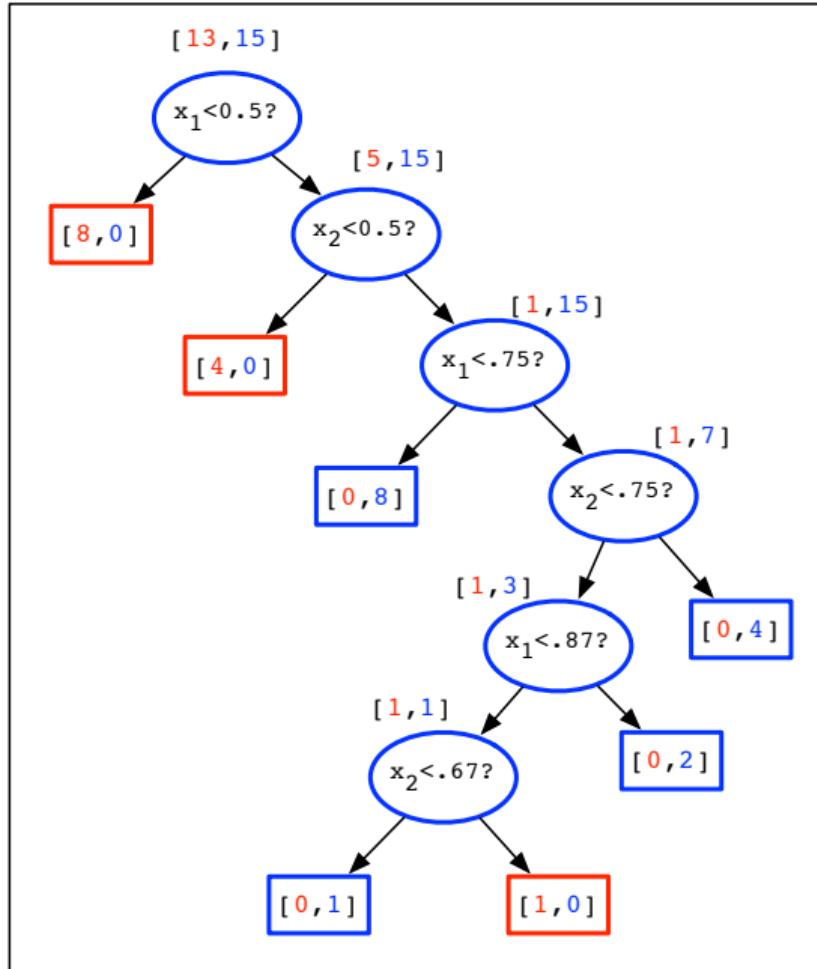












# Learning Decision Trees

- ▶ Start with single node containing all data points
- ▶ Repeat greedy procedure:
  - ▶ Look at all possible questions (splits)
  - ▶ Pick the one that most reduces **uncertainty**.
- ▶ Stop when each leaf node is **pure**.

# Aside: Generating Possible Questions

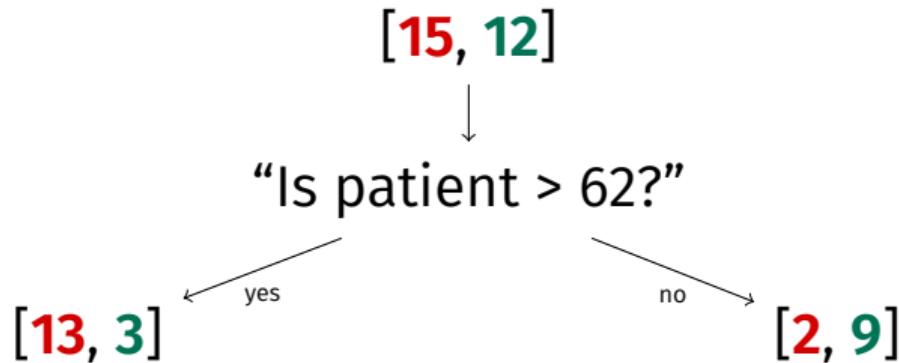
- ▶ **Categorical:** One question per value seen.
- ▶ E.g., county of residence.
  - ▶ Patient is from San Diego County?
  - ▶ Patient is from Riverside County?
  - ▶ Patient is from Orange County?

# Aside: Generating Possible Questions

- ▶ **Numerical:** one question between each pair of consecutive values.
- ▶ E.g., ages in data = {42, 43, 55, 57, 61, 75}
  - ▶ Patient is < 42.5?
  - ▶ Patient is < 49?
  - ▶ ...
  - ▶ Patient is < 68?

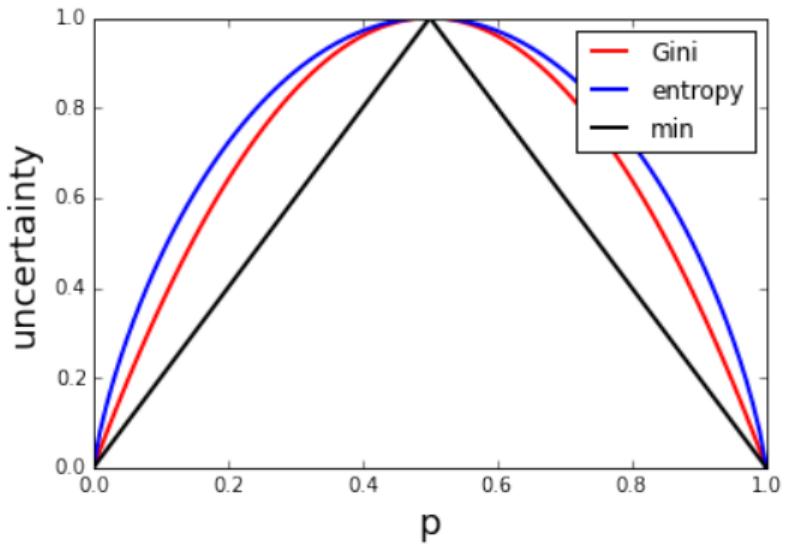
# Measuring Uncertainty

- ▶ A good question splits the data by class.



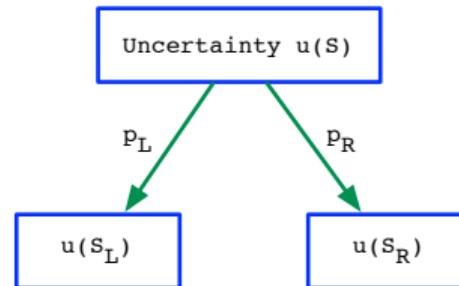
# Measuring Uncertainty

- ▶ Suppose our node contains proportions:
  - ▶  $p$  from class +
  - ▶  $(1 - p)$  from class -
- ▶ Common **uncertainty scores**:
  - ▶ **Misclassification rate**:  $\min\{p, 1 - p\}$
  - ▶ **Gini index**:  $2p(1 - p)$
  - ▶ **Entropy**:  $p \log \frac{1}{p} + (1 - p) \log \frac{1}{1-p}$

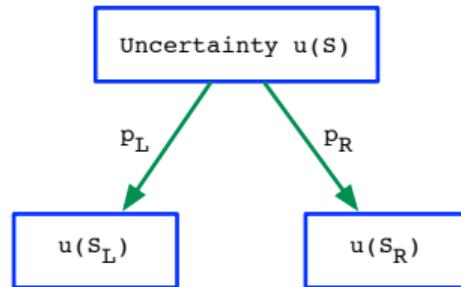


# Benefit of a Question

- ▶ Let  $u(S)$  be the uncertainty score for a set of labeled points,  $S$ .
- ▶ Consider a particular question (split):



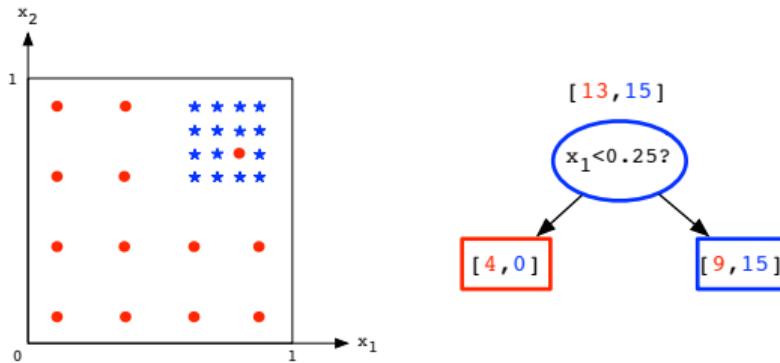
# Benefit of a Question



- ▶ Resulting uncertainty:

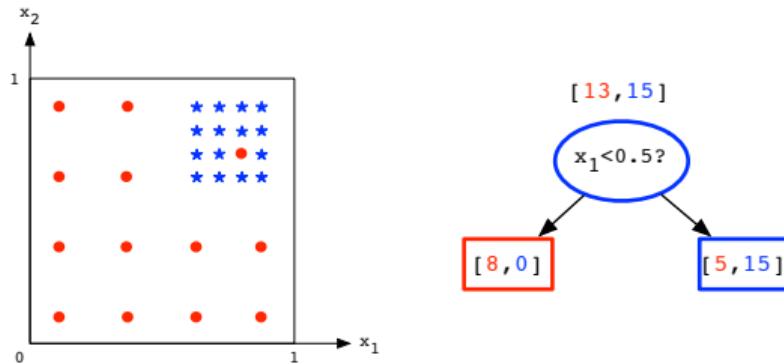
$$p_L u(S_L) + p_R u(S_R)$$

# Example



- ▶ Initial Gini uncertainty:  $2 \times \frac{13}{28} \times \frac{15}{28}$ .
- ▶  $p_L u(S_L) + p_R u(S_R) = \frac{4}{28} \cdot 0 + \frac{24}{28} \cdot 2 \cdot \frac{9}{24} \cdot \frac{15}{24} = \frac{45}{112}$

# Example



- ▶ Initial Gini uncertainty:  $2 \times \frac{13}{28} \times \frac{15}{28}$ .
- ▶  $p_L u(S_L) + p_R u(S_R) = \frac{8}{28} \cdot 0 + \frac{20}{28} \cdot 2 \cdot \frac{5}{20} \cdot \frac{15}{20} = \frac{30}{112}$

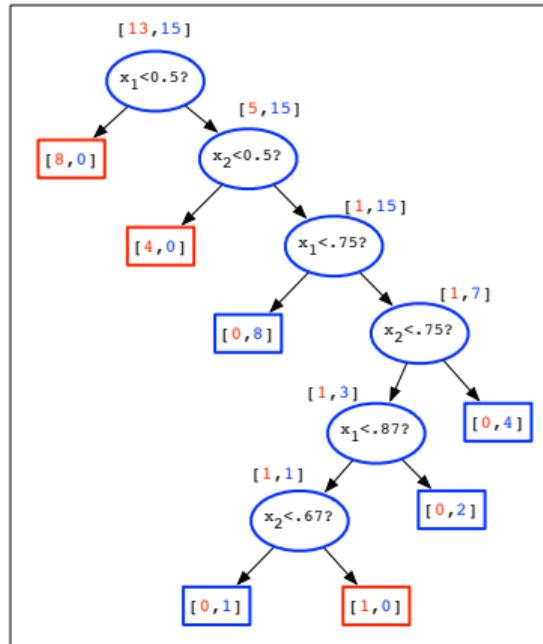
# Summary

To learn a decision tree:

- ▶ Pick a measure of uncertainty (Gini, Entropy, etc.)
- ▶ Recursively ask question minimizing uncertainty.

# Prediction

- ▶ To make prediction, traverse tree.
  - ▶ Example: (0.75, 0.6)



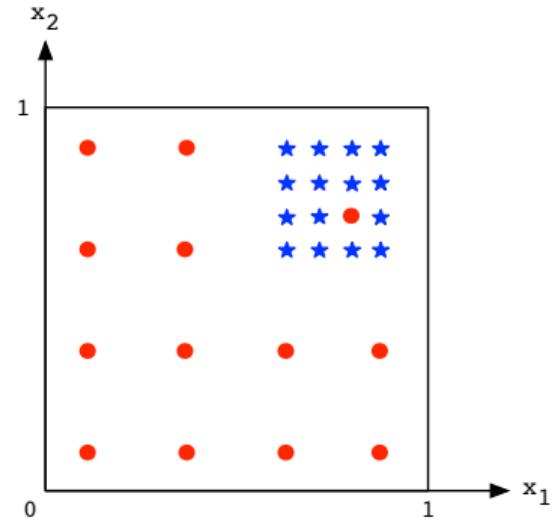
# DSC 140A

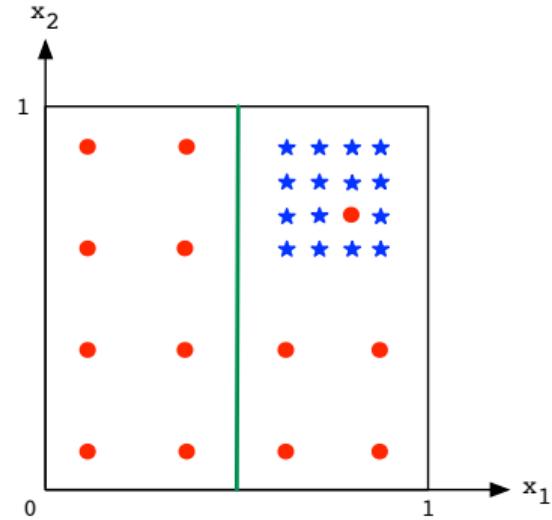
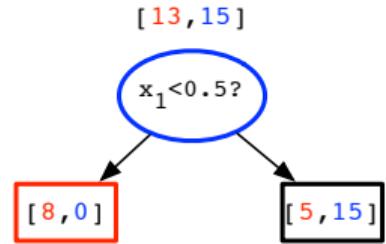
*Probabilistic Modeling & Machine Learning*

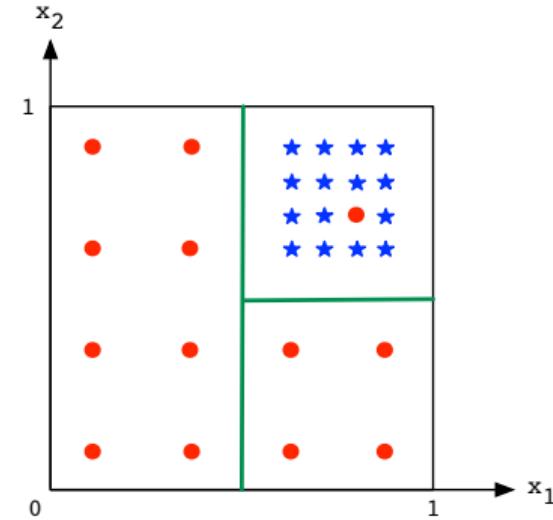
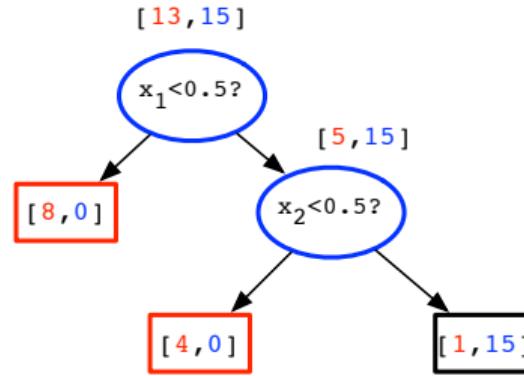
Lecture 16 | Part 2

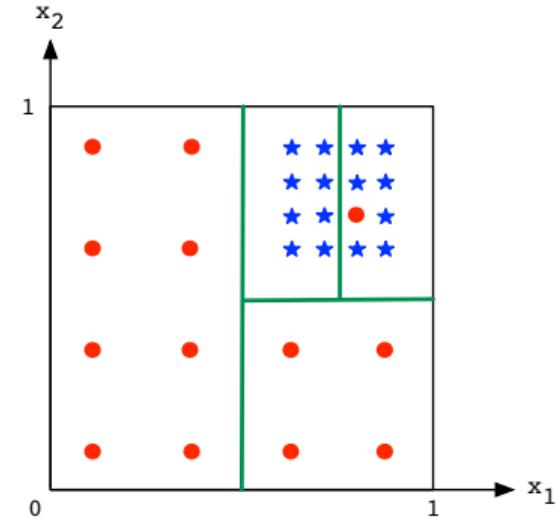
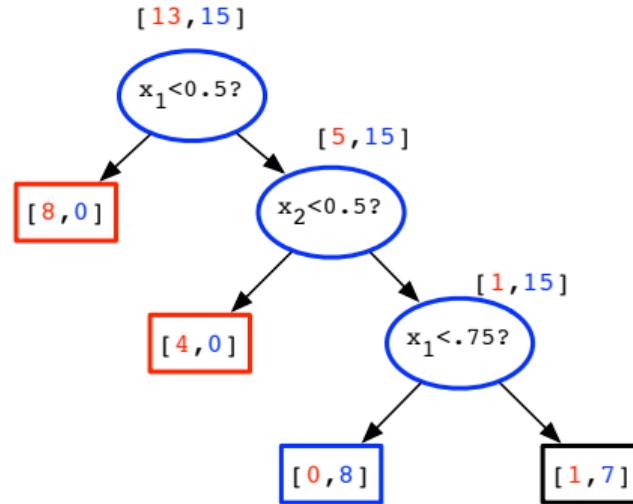
## Overfitting

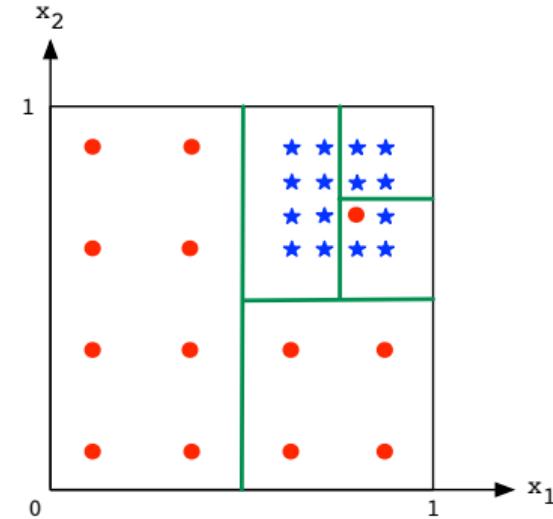
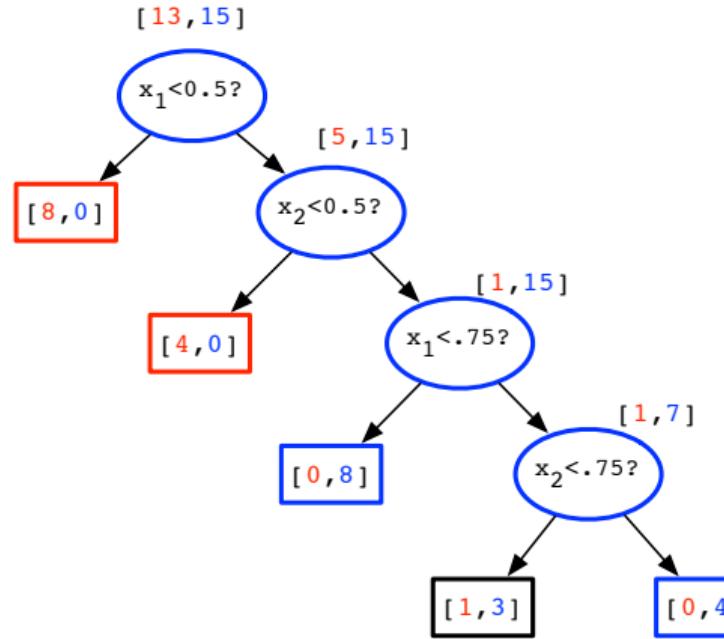
[ 13 , 15 ]

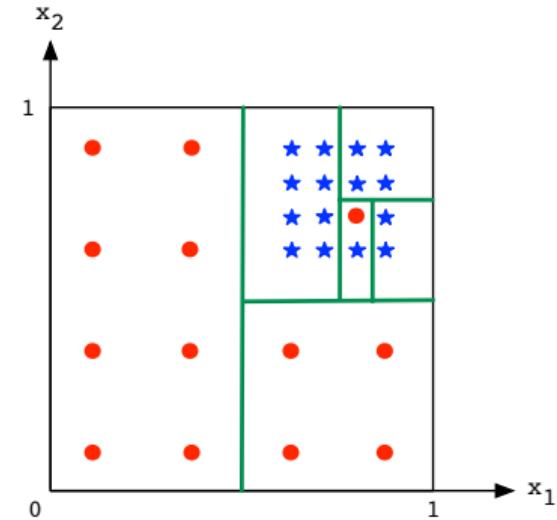
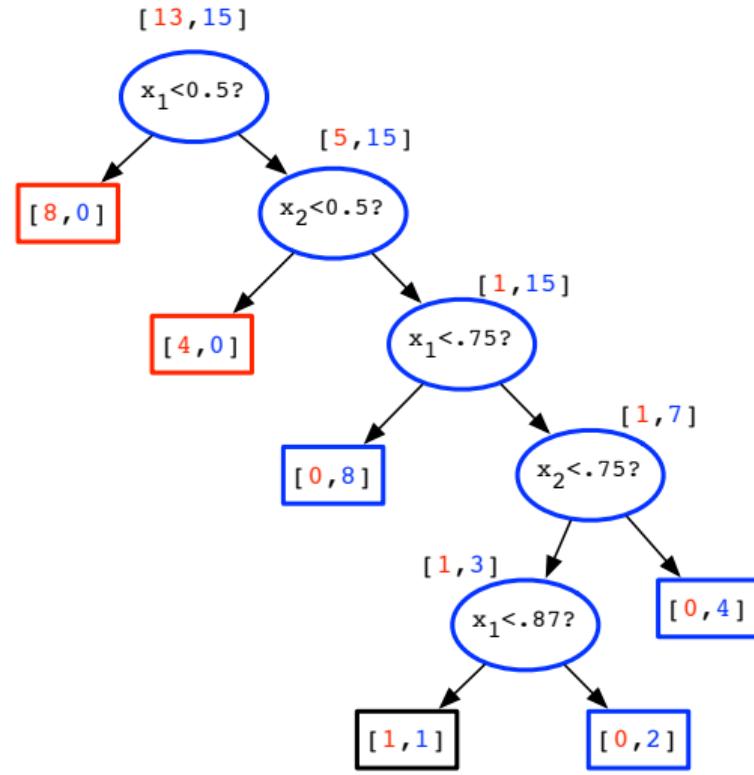


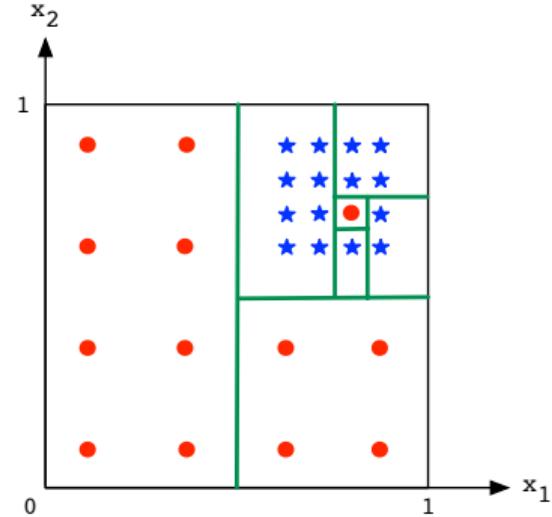
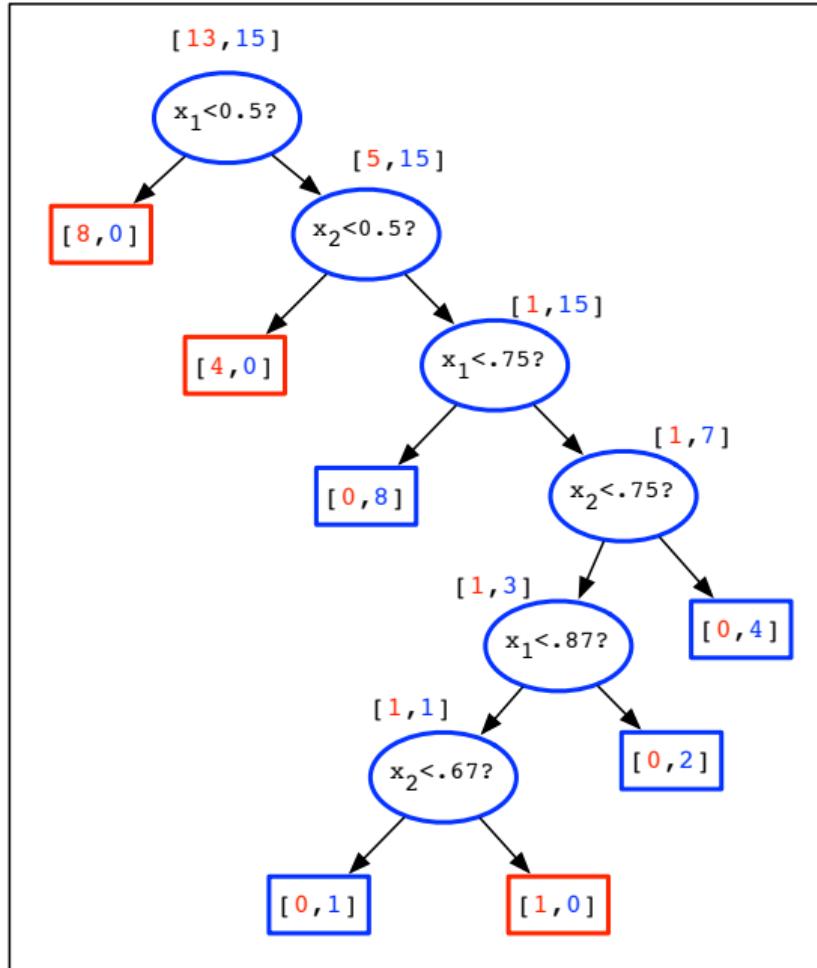






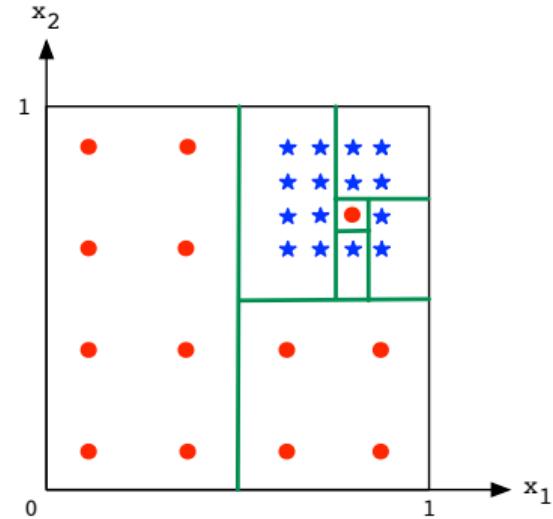
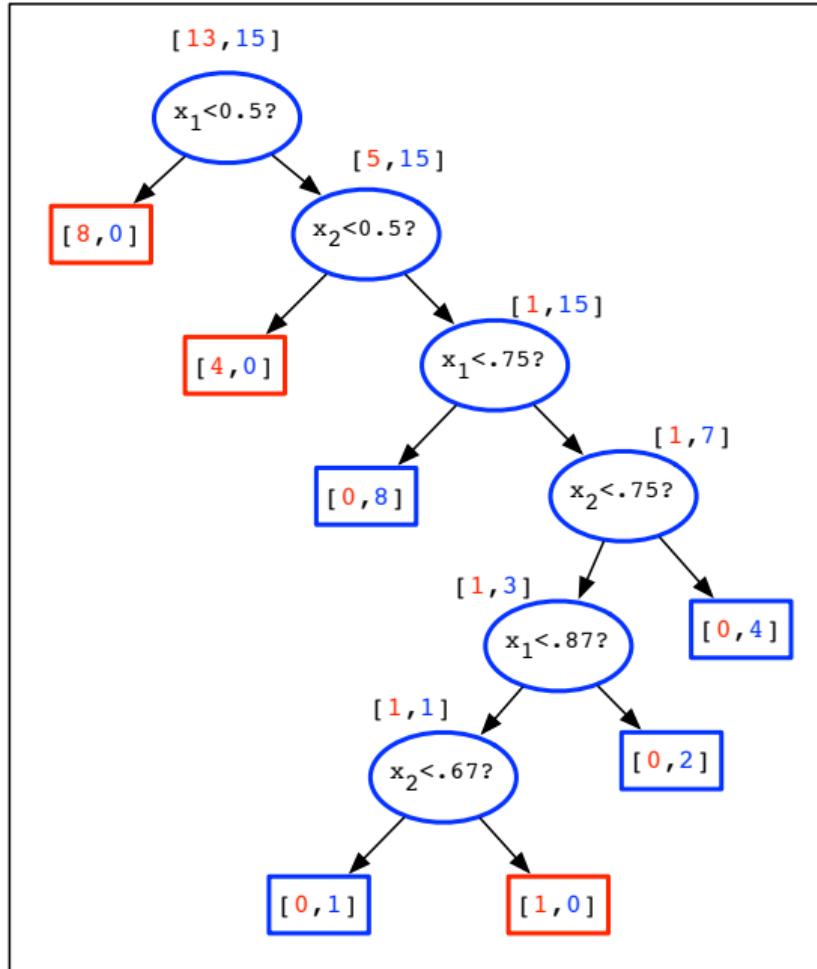


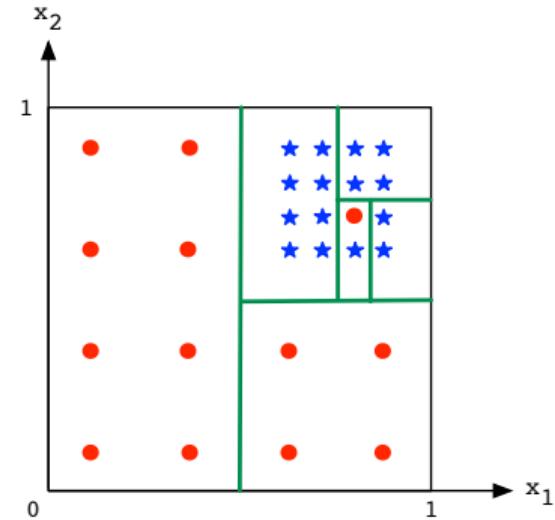
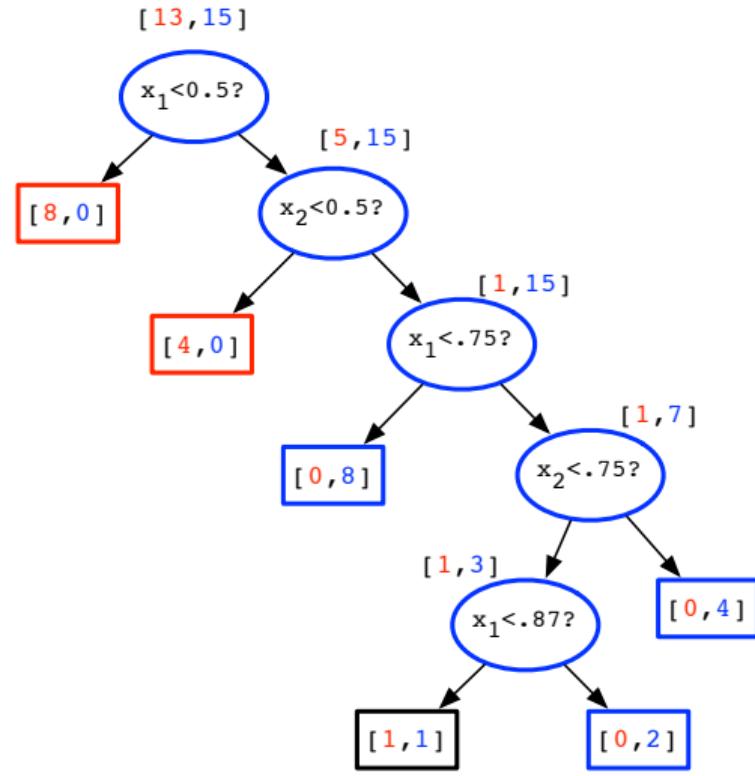


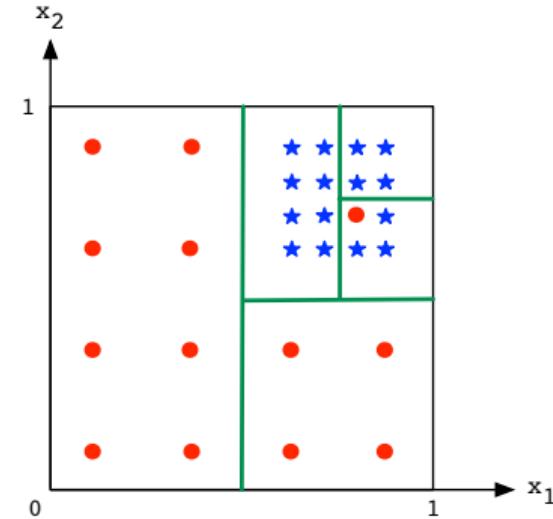
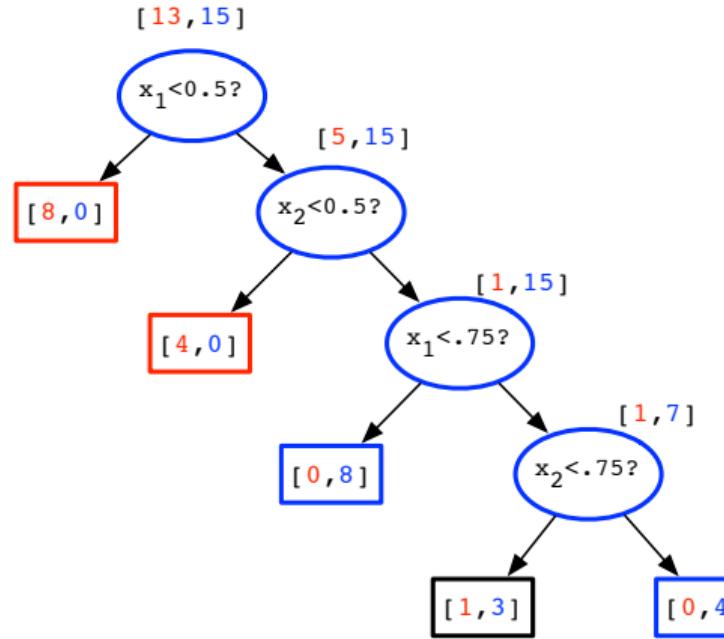


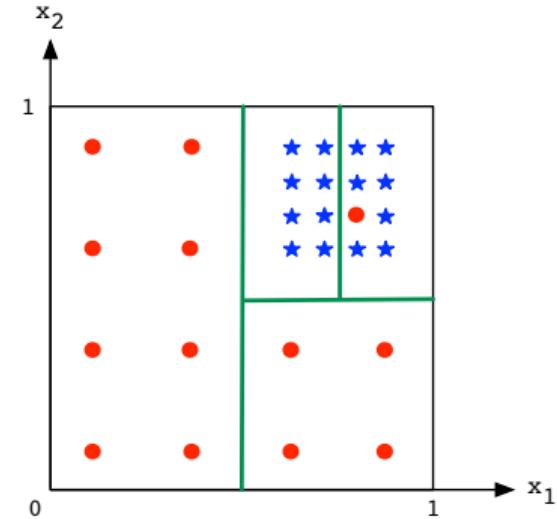
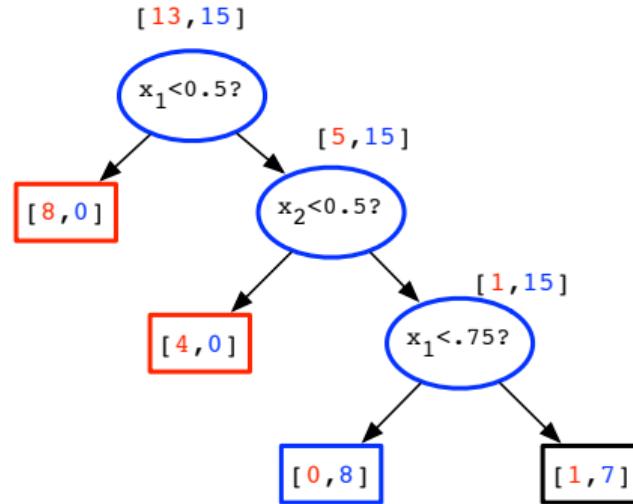
# Overfitting

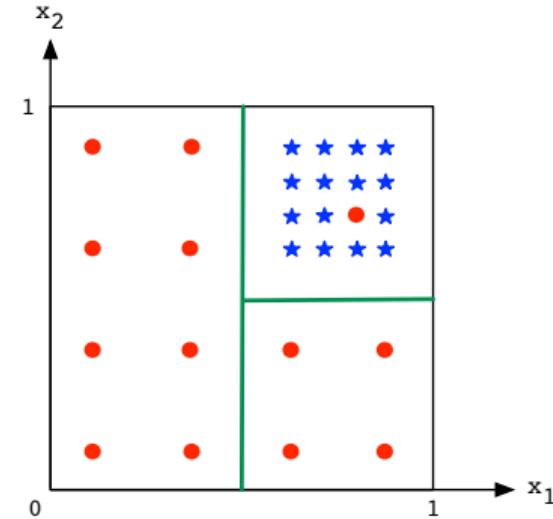
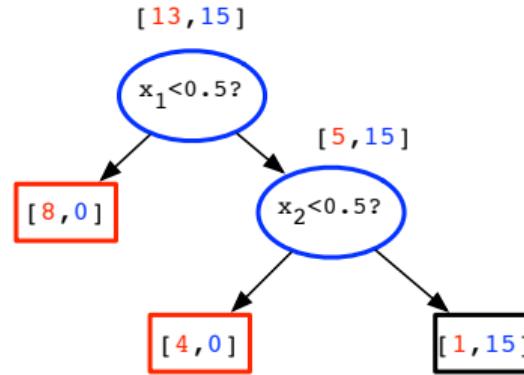
- ▶ The training error is **zero**.
  - ▶ We might be **overfitting**.
- ▶ (One) **solution**: rewind a few steps.



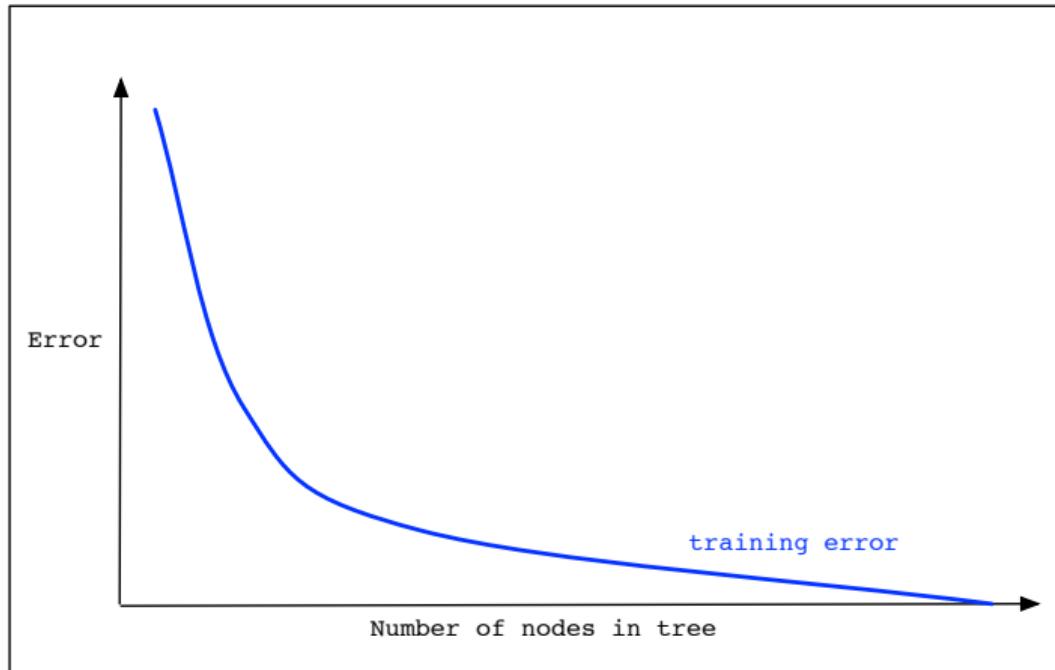




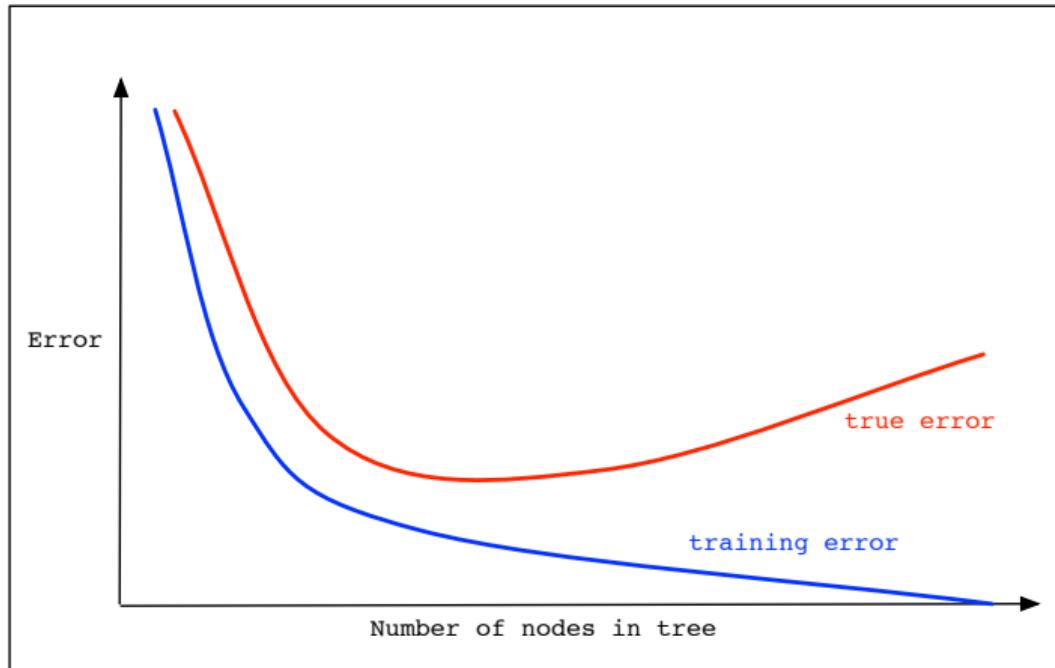




# Overfitting



# Overfitting



## Two Strategies

- ▶ **Pruning**: simplify already-constructed tree.
- ▶ **Early-stopping**: stop early.

# Pruning

- ▶ Given a full decision tree.
- ▶ Starting with predecessors of leaf nodes, replace node by most common class.
- ▶ If the change reduces validation error, keep it.  
Otherwise reverse it.

# Early-Stopping

- ▶ Stop recursion when:
  - ▶ node is “pure enough” (uncertainty is low).
  - ▶ tree is too deep.

# Decision Tree Properties

Very expressive:

- ▶ Can accommodate any type of data
  - ▶ numerical, Boolean, etc.
- ▶ Can accommodate any number of classes
- ▶ Can perfectly fit any data set
  - ▶ If data has no duplicates from different classes.
  - ▶ **Danger!** Overfitting!