

# DSC 140A

*Probabilistic Modeling & Machine Learning*

Lecture 01 | Part 1

**Welcome**

# This Class

- ▶ This course focuses on machine learning **theory**.
  - ▶ 80% theory, 20% practice
- ▶ Other classes (DSC 80, DSC 148) focus on machine learning **practice**.

# Math Background

- ▶ The most important prereqs for this class are:
  - ▶ DSC 40A (mathematical foundations of ML)
  - ▶ DSC 80 (ML practice + pandas)
  - ▶ MATH 20C (multivariable calculus)
  - ▶ MATH 18 (linear algebra)
  - ▶ MATH 183 (probability/statistics)
  
- ▶ We'll review some of the math (Discussion today, 3-3:50), but you might want to fill in some gaps on your own.

# If you're a DSC major/minor...

- ▶ There are several ways to satisfy the DSC major's ML requirement
  - ▶ CSE 150A, CSE 151A, DSC 140A, DSC 140B
- ▶ **Recommendation:** take DSC 140A and DSC 140B
- ▶ Avoid “mixing and matching”
  - ▶ e.g., avoid DSC 140A + CSE 151A

# If you're not a DSC major/minor..

- ▶ Welcome!
- ▶ This class assumes that you've seen some machine learning before.
  - ▶ least squares regression, gradient descent, empirical risk minimization
- ▶ If not, DSC 40A might be a good place to start.
- ▶ You'll also want to be comfortable with Python.

# Class structure

- ▶ Course materials (for now): canvas, later: [dsc140a.com](http://dsc140a.com)
- ▶ Weekly HW 10%
- ▶ 4 quizzes 30%
- ▶ midterm 25%
- ▶ final 35%

Check the course schedule weekly for reading material.

# Homeworks

- ▶ credit/no credit (Based on effort/showing work)
- ▶ 2 “slip days”
- ▶ no limit to working in groups or use of AI
- ▶ Learning is your responsibility
- ▶ The purpose of HW is to provide you with materials to prepare you for the Quizzes

# Quizzes

- ▶ Held during discussion section
- ▶ Paper & Pen/pencil (bring appropriate materials to discussion)
- ▶ Based on HW material
- ▶ 1st quiz will be next week

## **Final: 2 parts**

- ▶ Part 1 – Midterm material (weeks 1-5)
- ▶ Part 2 – New material (weeks 6-10)
- ▶ If Part 1 score is better than midterm score then Part 1 score will replace midterm.

## **Course Instructor: Mikio Aoi, PhD**

- ▶ Jointly appointed in Data Science & Neurobiology
- ▶ BS in Kinesiology from CSULB
- ▶ Research: Data science for computational neuroscience
- ▶ Fun Fact: Used to get C's in math

# Course staff: Pengcen Jiang



- ▶ 4th year DSC PhD Student
- ▶ Thesis topic:  
Computational neuroscience with RNNs and dynamical systems
- ▶ Enjoys reading
- ▶ Fun Fact: “It’s my 4th year in US and I’ve been to 13 states!”

# Course staff: Jake Ryan



- ▶ 2nd year DSC PhD Student
- ▶ Thesis topic: Feature learning
- ▶ Enjoys snowboarding
- ▶ Fun Fact: “I’ve been at UCSD for 7 years”

# Course staff: Jude Mariadas



- ▶ 2nd year DSC MS student
- ▶ Thesis topic: Using data from wearables
- ▶ Likes soccer, sports science, and playing with his 4 mo
- ▶ Fun Fact: “I help run a high ropes course (through UCSD recreation) where I guide teams through obstacle challenges”

# DSC 140A

*Probabilistic Modeling & Machine Learning*

Lecture 01 | Part 2

**Prediction**

# Prediction

- ▶ **Prediction** is the most common task in machine learning.
  - ▶ **Given** some input information...
  - ▶ **Predict** some related output.

# Examples

- ▶ **Given** a data scientist's age, college GPA, and state of residence, **predict** their salary.
- ▶ **Given** a penguin's bill length and body mass, **predict** its species.
- ▶ **Given** a digital image, **predict** the gesture being made.

# Features and Labels

- ▶ Each piece of input information is called a **feature**.
- ▶ The output we're trying to predict is called the **label** (or **target**).
- ▶ **Example:**
  - ▶ Features: age, college GPA, state of residence
  - ▶ Label: salary

# Regression

- ▶ When the label is a continuous number, we call it a **regression** problem.
- ▶ **Examples:** predicting salary

# Classification

- ▶ When the label is one of a finite number of choices, we call it a **classification** problem.
- ▶ **Examples:** predicting species, predicting gestures

# Binary vs. Multiclass Classification

- ▶ In **binary classification**, there are only two possible labels.
- ▶ In **multiclass classification**, there are more than two possible labels.
- ▶ For simplicity, we'll focus on binary classification.

# Features

- ▶ Features are most often numerical.
- ▶ Why?
  1. Computers process numbers (not penguins).
  2. Allows us to use mathematical machinery.

# Feature Vectors

- ▶ We often package features into a **feature vector**.
- ▶ **Example:**

$$\vec{x} = (\text{bill length}, \text{body mass})^T$$

- ▶ The **dimensionality** of a feature vector is the number of features it contains.

# Choosing Features

- ▶ Features should contain information relevant to predicting the label.
- ▶ There should be a relationship between them.
  - ▶ It might be quite complex!
- ▶ Choosing good features is crucial.
  - ▶ “Garbage in, garbage out.”

# Learning from Data

- ▶ To teach the computer, we provide it with many **training examples**.
- ▶ Each example consists of an input feature vector  $\vec{x}$  and the correct output label  $y$ .
- ▶ The set of examples is called the **training set**:

$$\mathcal{X} = \{(\vec{x}^{(1)}, y_1), (\vec{x}^{(2)}, y_2), \dots, (\vec{x}^{(n)}, y_n)\}$$

# Learning from Data

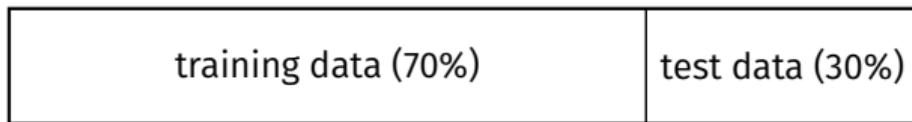
- ▶ Hope: given enough examples, the computer will detect a pattern between the features and labels.
- ▶ This process is called **learning**.
- ▶ The more complex the relationship, the more examples we'll need.

# Train Error

- ▶ To see how well the computer has learned, we can compute the **train error**.
  - ▶ Make predictions on the training set.
  - ▶ E.g., for classification, the fraction of training examples misclassified.
- ▶ But this is not always a good indicator of how well the computer will do on **new** examples.

# Test Error

- ▶ Instead, we reserve some examples for a **test set**.
- ▶ Randomly choose, say, 30% of the examples to be in the test set.



- ▶ Randomizing is important!

# Test Error

- ▶ Train **only** on the training set.
- ▶ Make predictions on the test set.
- ▶ The error on the test set is the **test error**.
- ▶ The test error is a better indicator of how well the computer will do on new examples.

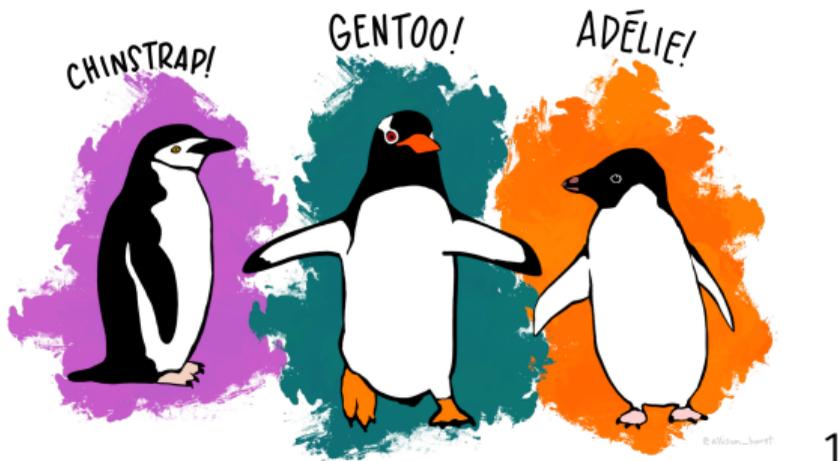
# Generalization

- ▶ The ability of the model to perform well on new, unseen examples is called **generalization**.
- ▶ In prediction, it's what we're after.
- ▶ Training error can be useful, but we care mostly about test error.

# Overfitting and Underfitting

- ▶ **Overfitting:** model does not generalize.
  - ▶ Train error is much lower than test error.
- ▶ **Underfitting:** model is not learning the pattern.
  - ▶ Both train and test error are high.
  - ▶ Need more features, more complex model, etc.

# Example: Penguin Prediction



- **Task:** given bill length and body mass, predict species.

---

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

# Training Set

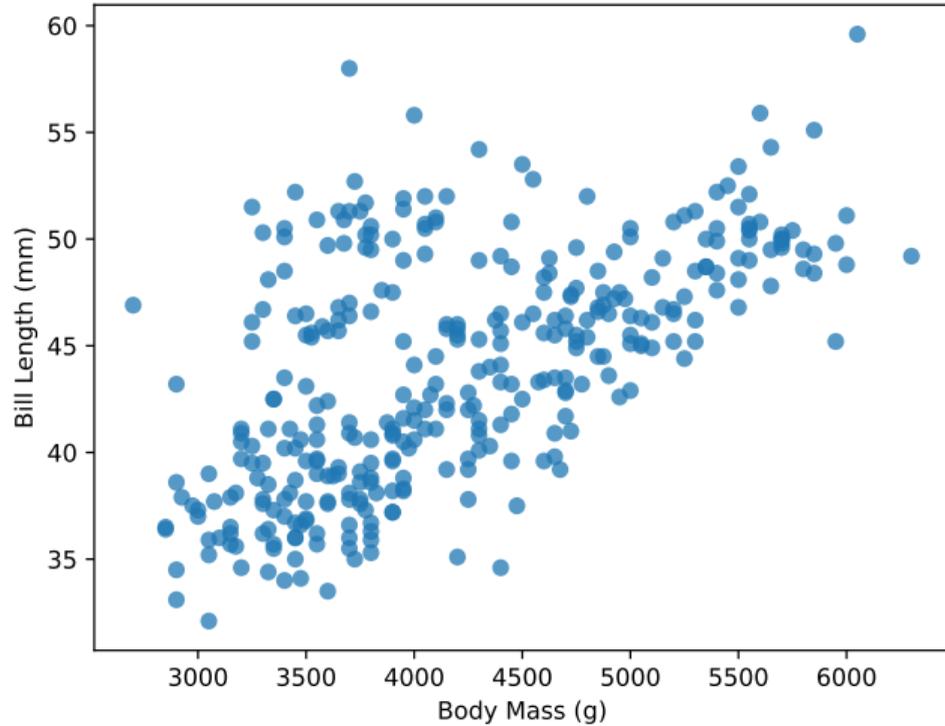
- ▶ We collect a training set of 344 penguins. For each penguin, we record:
  - ▶ The features: bill length, body mass
  - ▶ The label: species

- ▶ Each penguin becomes a feature vector in  $\mathbb{R}^2$ .

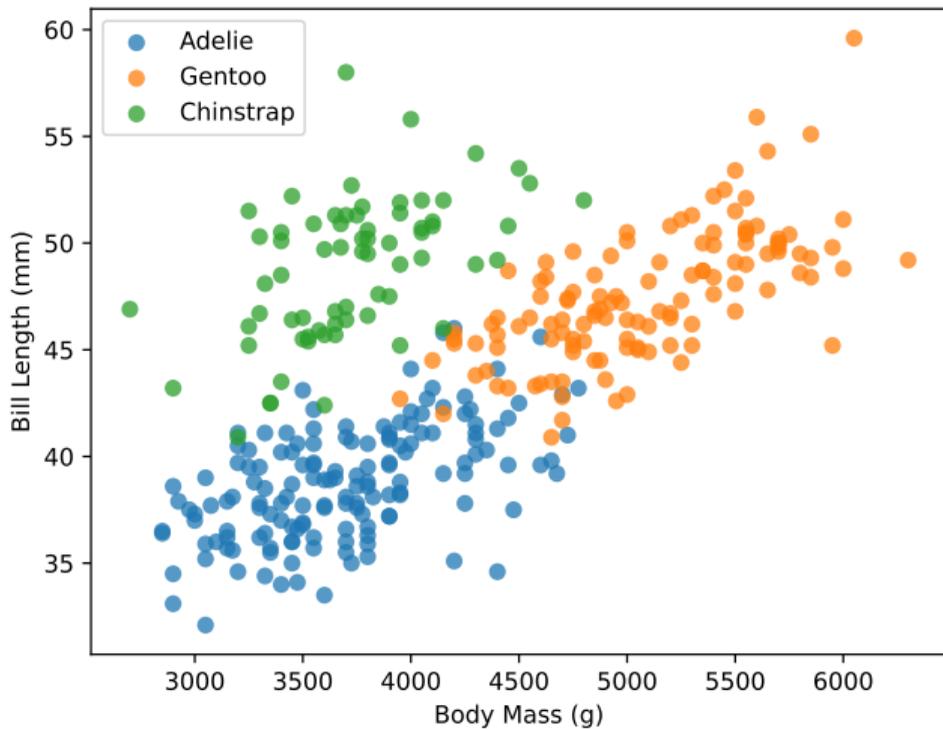
$$\vec{x}^{(i)} = (\text{body mass of penguin } i, \text{ bill length of penguin } i)^T$$

- ▶ We can **embed** penguins as **point cloud** in  $\mathbb{R}^2$ .

# Penguin Embedding

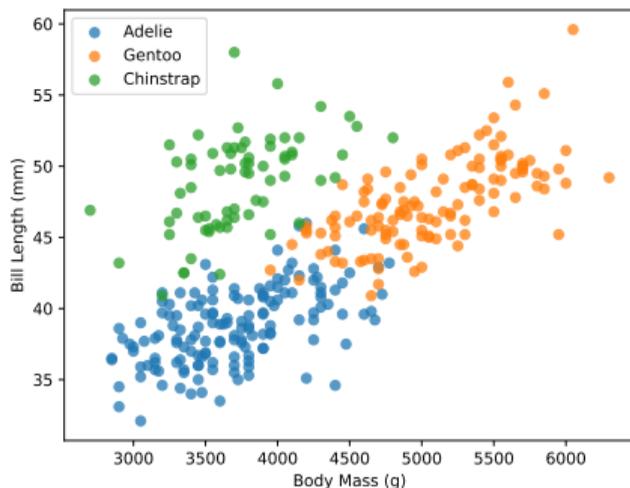


# Penguin Embedding



## 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*. Nearby (similar) feature vectors have similar labels.

# DSC 140A

*Probabilistic Modeling & Machine Learning*

Lecture 01 | Part 3

**Nearest Neighbors Predictors**

# Nearest Neighbors Predictors

- ▶ **Idea:** to predict the label of a new example:
  1. find the most similar example in the training set
  2. predict the same label
- ▶ This is called a **nearest neighbor predictor**.
- ▶ Useful for both regression and classification.

# Nearest Neighbor Classifier

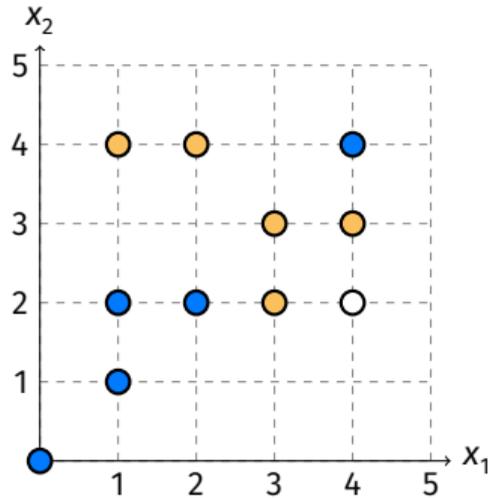
- ▶ **Data:** a training set  $\mathcal{X}$  of  $n$  feature vectors with labels:  $\{(\vec{x}^{(i)}, y_i)\} = \{(\vec{x}^{(1)}, y_1), \dots, (\vec{x}^{(n)}, y_n)\}$
- ▶ **Given:** a new point,  $\vec{z}$  with unknown label.
- ▶ **Predict:**
  1. Find the closest point to  $\vec{z}$  in  $\mathcal{X}$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

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

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

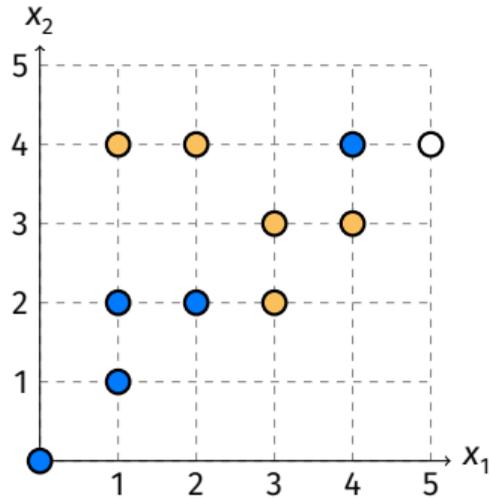
## Exercise

What is the predicted label for the new point?



## Exercise

What about for this new point?



# A Note About Distances

- ▶ We found the nearest neighbor using the **Euclidean distance**:

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

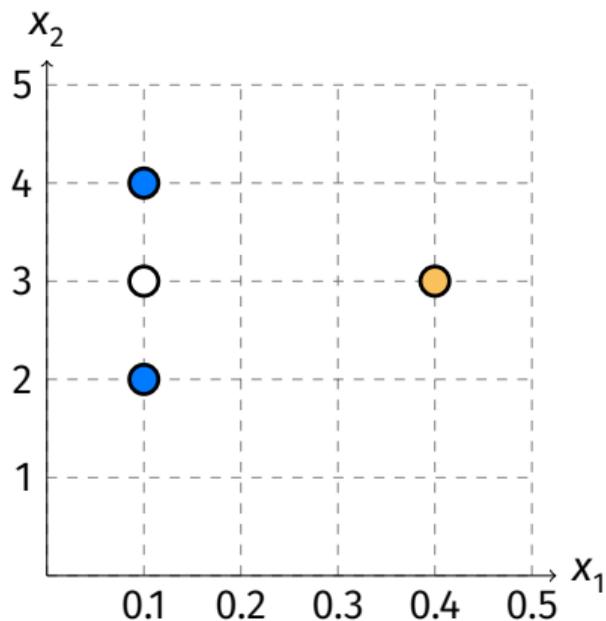
- ▶ Note that this is just one choice – there are other valid distances. E.g., cosine distance.

## A Note About Distances

- ▶ The Euclidean distance treats all features equally.
- ▶ In other words, all features contribute equally to the prediction.

## Exercise

What is the predicted label for the new point?



# Answer

- ▶ Predicted label: **yellow**.
- ▶ The features are measured on different scales.
- ▶ The blue points *look* closer, but the yellow point is closer in Euclidean distance.
- ▶ Not just a visual illusion; sometimes, features on different scales can cause problems.

# Example

- ▶ Person A is 6 ft tall, 180 lbs.
- ▶ Person B is 7 ft tall, 185 lbs.
- ▶ A new person is 7 ft tall, 180 lbs. Intuitively speaking, are they more similar to A or B?

# Standardizing Features

- ▶ When features are measured on different scales, it can help to **standardize**.
- ▶ **Idea:** shift and scale to make each feature have mean 0 and standard deviation 1.

# Standardizing Features

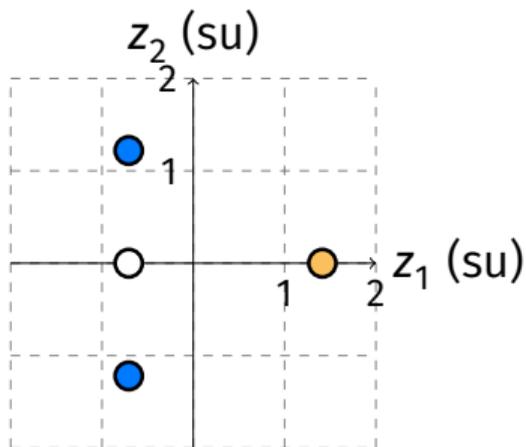
- ▶ Suppose we have two features,  $x_1$  and  $x_2$ , and let:
  - ▶  $\mu_1, \mu_2$  be the means of each feature in the training set,
  - ▶  $\sigma_1, \sigma_2$  be the standard deviations.
- ▶ When standardizing:

$$(x_1, x_2)^T \quad \text{becomes} \quad (z_1, z_2)^T = \left( \frac{x_1 - \mu_1}{\sigma_1}, \frac{x_2 - \mu_2}{\sigma_2} \right)^T$$

- ▶ Do this for all training data, **and** new test examples.

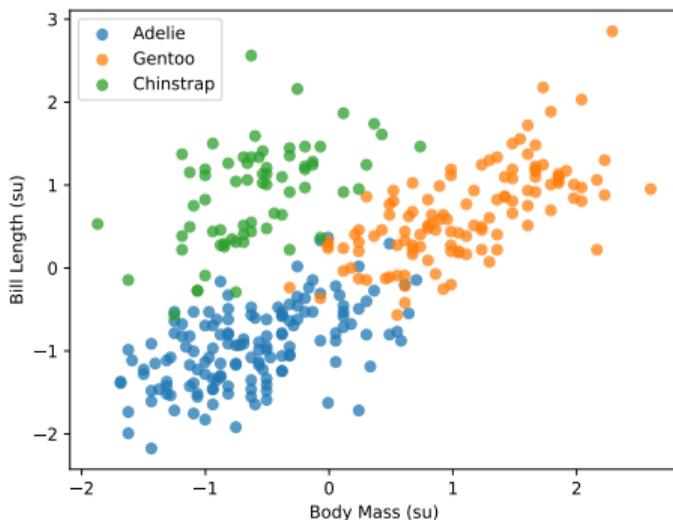
# Example

When plotted in standard units, the data now looks like this:



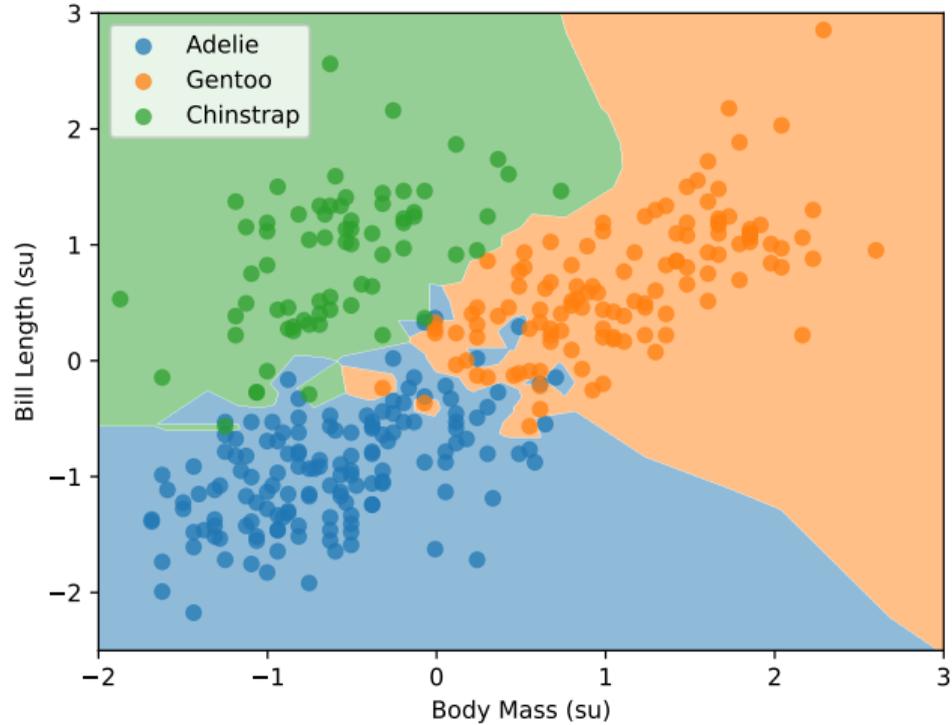
# The Decision Boundary

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





# The Decision Boundary



## Exercise

Suppose there are no duplicates in the training data.

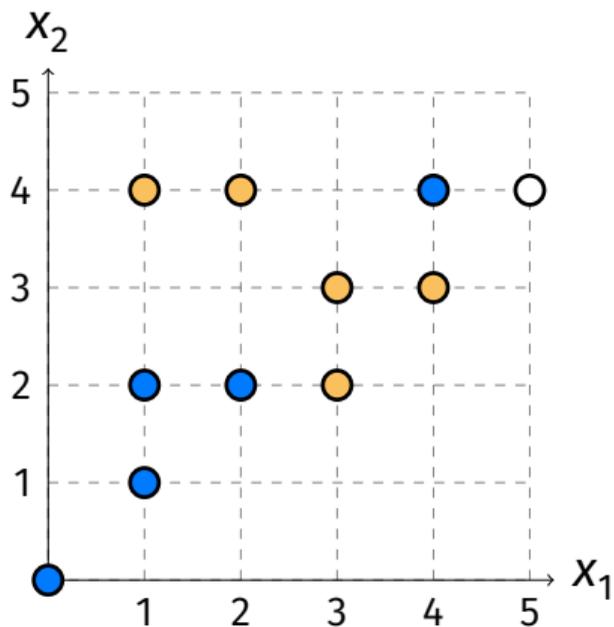
True or False: the nearest neighbor classifier will have 100% training accuracy.

# Answer

- ▶ **True.**
- ▶ If no duplicates, each training example is its own nearest neighbor.
- ▶ So for each training example, we predict the correct label.
- ▶ Takeaway: training accuracy can be **misleading**.

# Problem

- ▶ What if the nearest neighbor is an **outlier**?



# $k$ -Nearest Neighbors

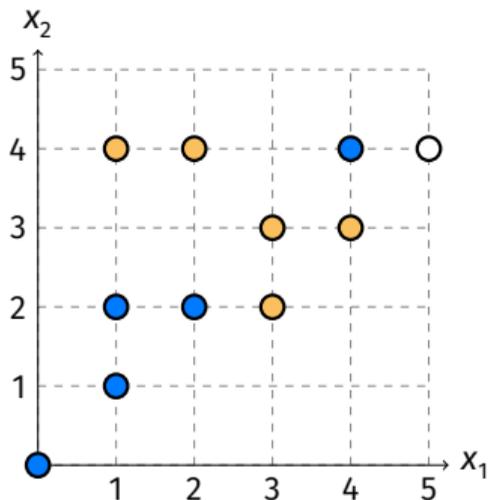
- ▶ Before: *single* closest neighbor determined prediction.
- ▶ Idea: have  $k$  closest neighbors “vote”.
- ▶ Can be useful to reduce noise.

# $k$ -Nearest Neighbors Classifier

- ▶ **Data:** a training set  $\mathcal{X}$  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, a choice for the parameter  $k$ .
- ▶ **Predict:**
  1. Find the  $k$  closest points to  $\vec{z}$  in  $\mathcal{X}$ :
  2. Use the most common label among those  $k$  points as the predicted label.

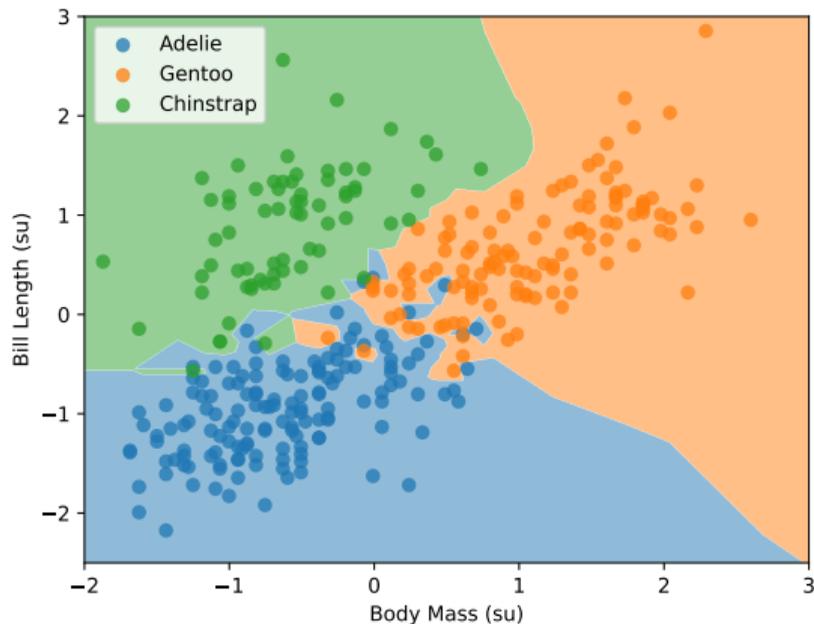
## Exercise

What is the predicted label for the new point using  $k$ NN with  $k = 3$ ?



# $k$ and the Decision Boundary

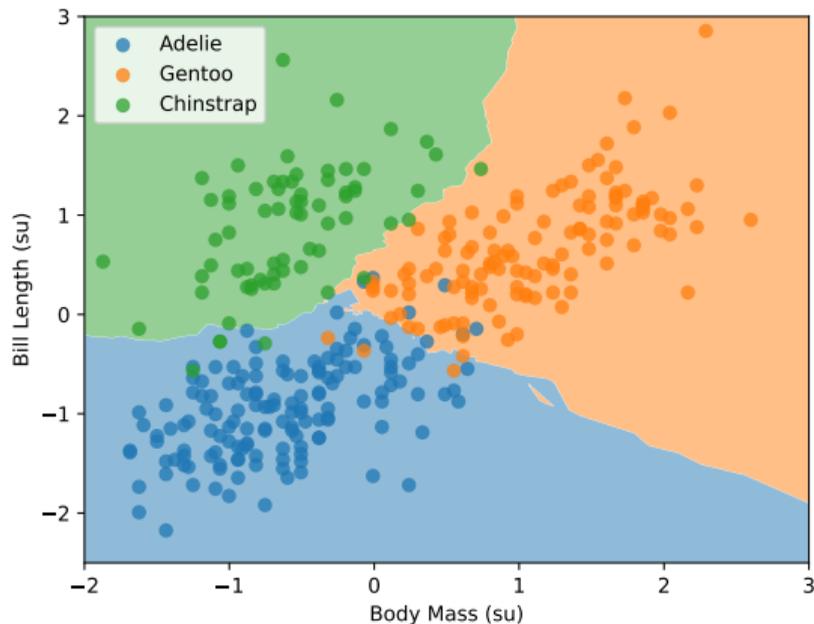
- ▶ How might the decision boundary change as we increase  $k$ ?



$k = 1$

# $k$ and the Decision Boundary

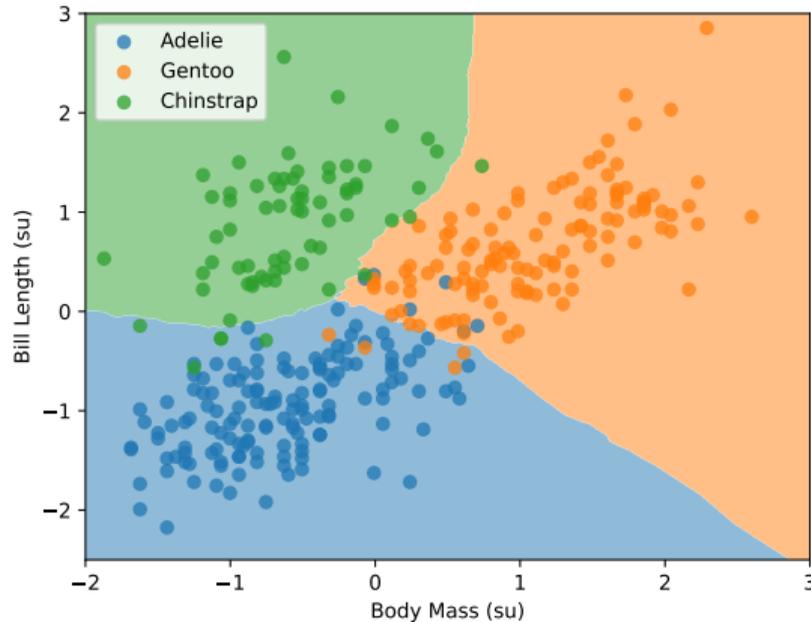
- ▶ How might the decision boundary change as we increase  $k$ ?



$k = 10$

# $k$ and the Decision Boundary

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



$k = 20$



## Exercise

What will the prediction be if we set  $k = n$ , where  $n$  is the number of training examples?

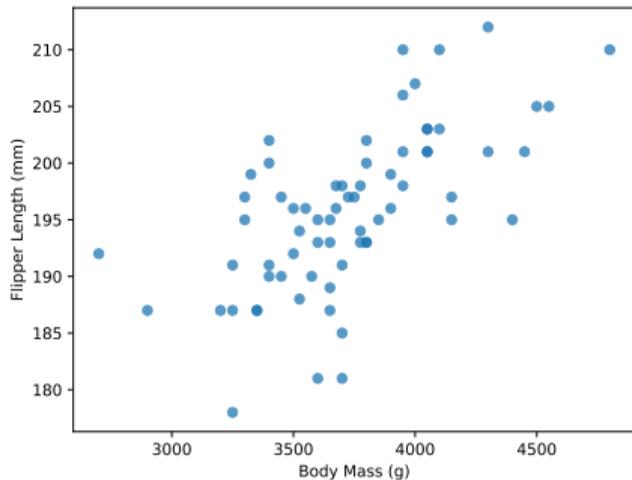
# Nearest Neighbor Regression

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



## 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 flipper length is known.
  2. Use that penguin's flipper length as our prediction.

# Nearest Neighbor Regression

▶ **Data:** a set  $\mathcal{X}$  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  $\mathcal{X}$ :

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

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

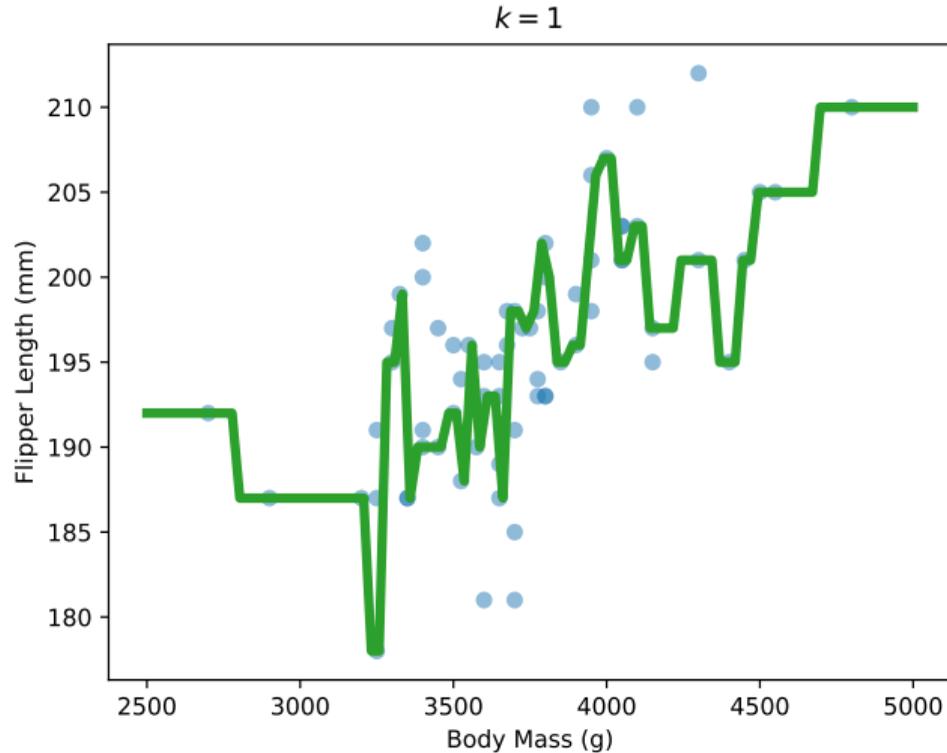
# *k*NN Regression

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

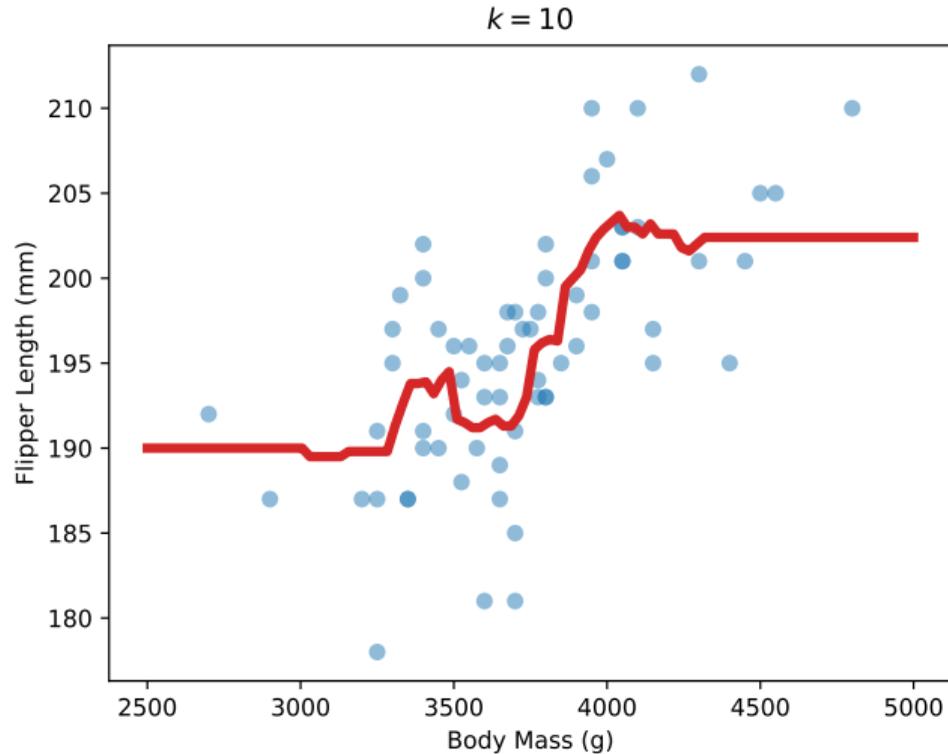
# $k$ -Nearest Neighbors Regression

- ▶ **Data:** a set  $\mathcal{X}$  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  $k$  closest points to  $\vec{z}$  in  $\mathcal{X}$
  2. Use the average of their labels as the predicted target

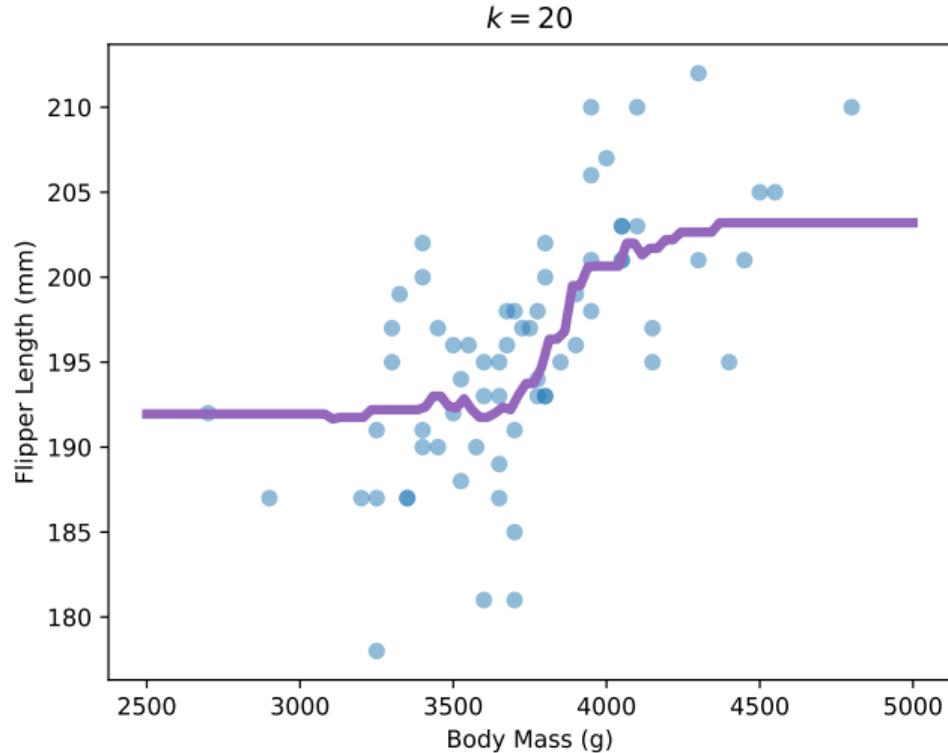
# Example: $kNN$ Penguin Regression



# Example: $kNN$ Penguin Regression



# Example: $kNN$ Penguin Regression



# DSC 140A

*Probabilistic Modeling & Machine Learning*

Lecture 01 | Part 4

**Gesture Recognition Demo, Revisited**

# Gesture Recognition

- ▶ The gesture recognition demo we saw earlier is a *k*NN classifier.

# Features

- ▶ Each video frame is made into a  $d$ -dimensional feature vector.
  1. Converted to grayscale.
  2. Divided into  $d$  horizontal strips.
  3. Feature  $i$  is the average brightness of strip  $i$ .

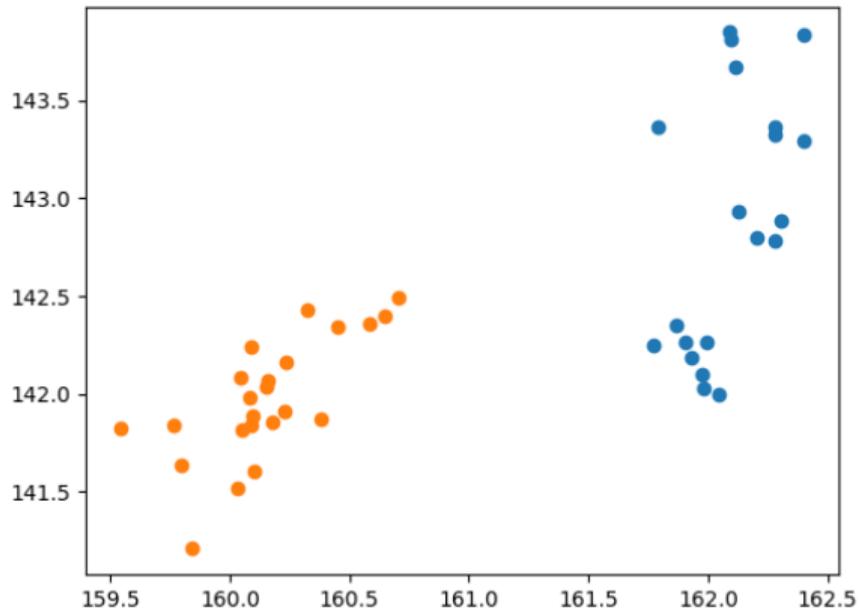
# Example: $d = 2$



# Example: $d = 2$



# Example: $d = 2$



# Observation

- ▶ The feature vectors for the same gesture are close together.
- ▶ The data has “organized” itself.

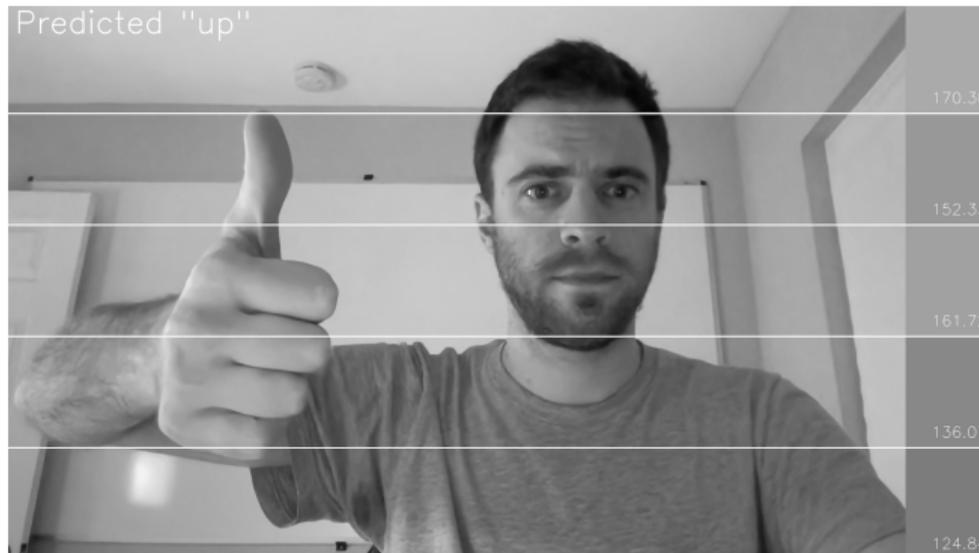
# Prediction

- ▶ Given a new frame, convert it to a feature vector and find the  $k$  closest training frames.

## Beyond $d = 2$

- ▶ We can use more features.
  - ▶ We can still apply  $k$ NN in high dimensions.
  - ▶ (But we can't visualize the feature vectors with a scatter plot)
  
- ▶ The original demo used  $d = 5$  features.

$$d = 5$$



# Takeaway

- ▶ Even seemingly-intelligent behavior can be achieved with simple algorithms + data.

# DSC 140A

*Probabilistic Modeling & Machine Learning*

Lecture 01 | Part 5

**From Theory to Practice**

## Tip #0: Implementation

- ▶ sklearn has a *k*NN implementation.
- ▶ `sklearn.neighbors.KNeighborsClassifier` for classification.
- ▶ But this a theory class; we'll implement it ourselves.

## Tip #0: Implementation

- ▶  $k$ NN can be implemented in a few lines of code with numpy.
- ▶ useful functions:
  - ▶ `np.linalg.norm`: computes distances,
  - ▶ `np.argmin`: finds index of the minimum value
  - ▶ `np.argpartition`: finds indices of  $k$  smallest values
  - ▶ `np.bincount`: counts occurrences of each value

```
import numpy as np

def knn_predict(X_train, y_train, x, k=1):
    # compute distances between test and training examples
    distances = np.linalg.norm(X_train - x, axis=1)

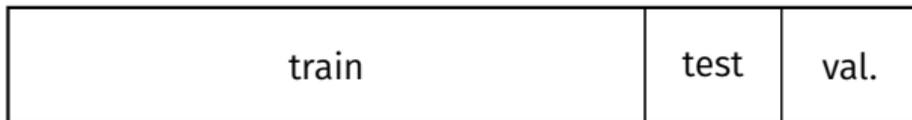
    # find the indices of the k smallest distances
    nearest = np.argpartition(distances, k, axis=0)[:k]

    # get the labels of the k nearest neighbors
    nearest_labels = y_train[nearest]

    # return the most common label
    return np.bincount(nearest_labels).argmax()
```

## Tip #1: Choosing $k$

- ▶ To choose  $k$ , further divide your data into training, test, and **validation** sets.



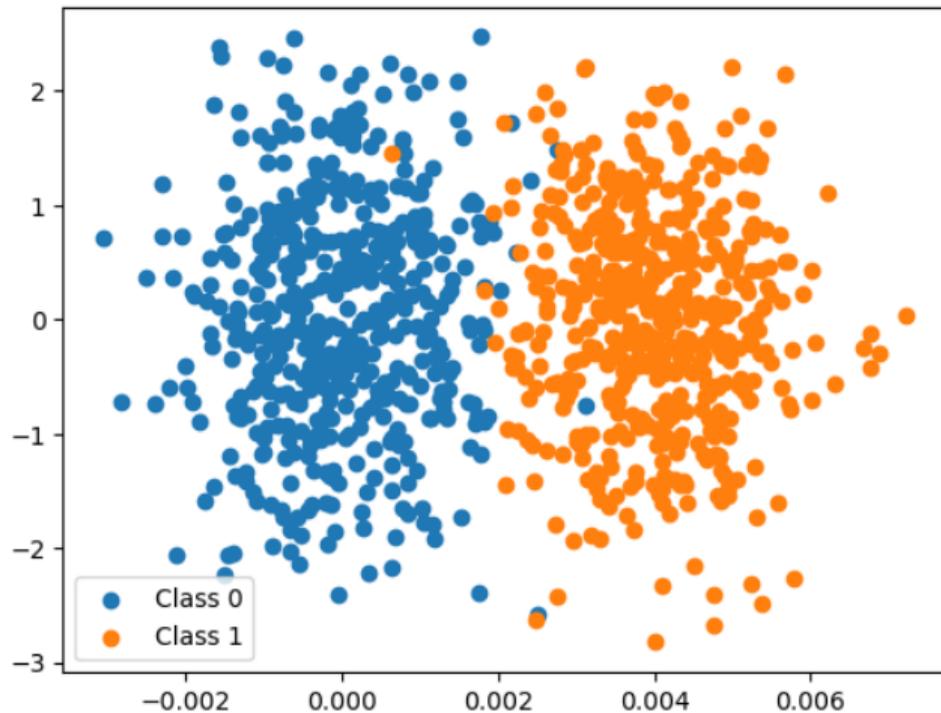
## Tip #1: Choosing $k$

- ▶ Pick a few different values of  $k$ , train model on each, and compute error on the validation set.
- ▶ Keep the  $k$  that gives the lowest error; this is your choice.
- ▶ Compute test error using this  $k$ .

## Tip #1: Choosing $k$

- ▶ When you're done and ready to use the model "in production":
  1. combine all of your data into one large training set,
  2. use the  $k$  you chose in validation,
  3. train the final model.

# Tip #2: Standardizing Features



```
»> knn = sklearn.neighbors.KNeighborsClassifier(n_neighbors=1)
»> knn.fit(X_train, y_train)
»> knn.score(X_test, y_test)
0.672
```

```
»» mu, sigma = X_train.mean(axis=0), X_train.std(axis=0)
»» Z_train = (X_train - mu) / sigma
»» Z_test = (X_test - mu) / sigma
»» knn.fit(Z_train, y_train)
»» knn.score(Z_test, y_test)
0.972
```

## Trivia: Speeding it Up

- ▶ Making a prediction requires computing the distance to *every* training example.
- ▶ There are ways of speeding this up:
  - ▶ Approximate nearest neighbors,
  - ▶  $k$ -d trees, ball trees, etc.
  - ▶ Subsampling the data.

# DSC 140A

*Probabilistic Modeling & Machine Learning*

Lecture 01 | Part 6

**The End?**

# The End?

- ▶ We have developed a simple prediction algorithm:  $k$ -nearest neighbors.
- ▶ Can used for both **classification** and **regression**.
- ▶ Often works well!
- ▶ Have we “solved” machine learning?

# No

- ▶ Nearest neighbor predictors have significant limitations in two areas:
  1. Computational efficiency
    - ▶ I.e., they are slow, or require a lot of memory.
  2. Predictive performance
    - ▶ I.e., they aren't always as accurate as other methods.

# Something Unsatisfying

- ▶ Do nearest neighbor models **learn** anything?
- ▶ They seem to just “memorize” the training data.

# The Main Problem

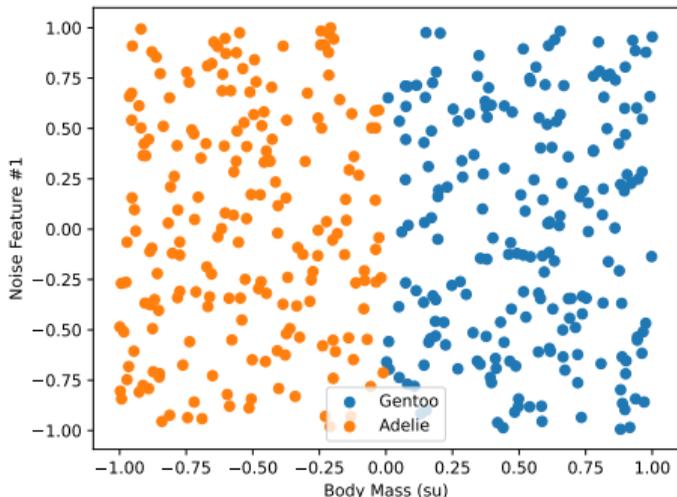
- ▶ Nearest neighbor approaches **do not learn** which features are **useful** and which **are not**.

# Example

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

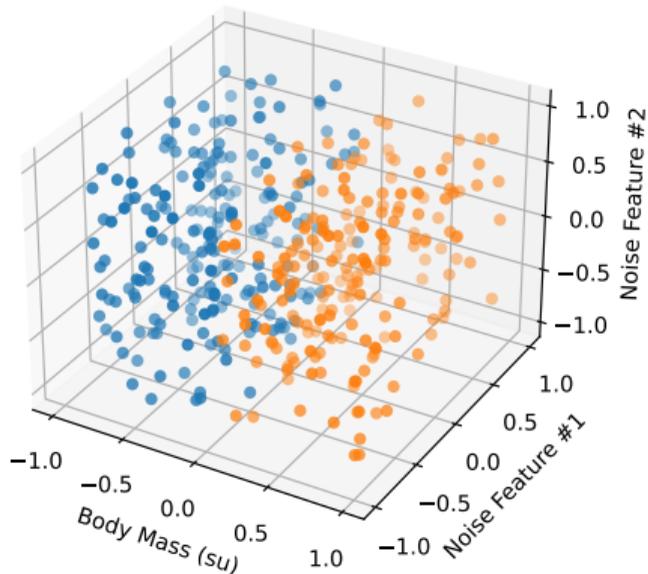
# Example: One Noisy Feature

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



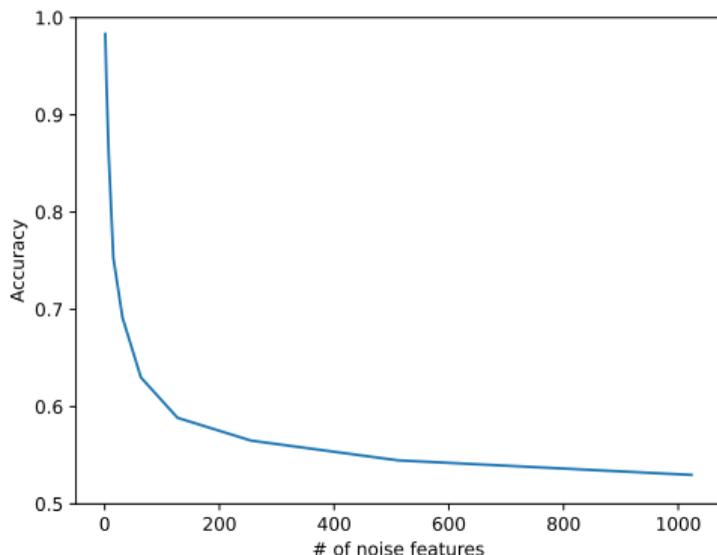
# Example: Two Noisy Features

- ▶ Suppose we add another 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.