DSC 140A

Probabilistic Modeling & Machine Learning

Lecture 01 | Part 1

What is Machine Learning??

What is Machine Learning?

- Computers can do things very quickly.

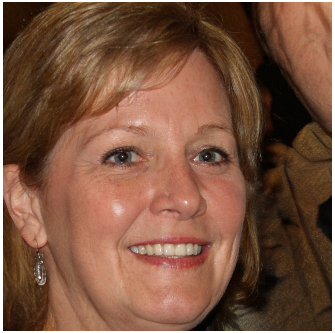
What is Machine Learning?

- 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`

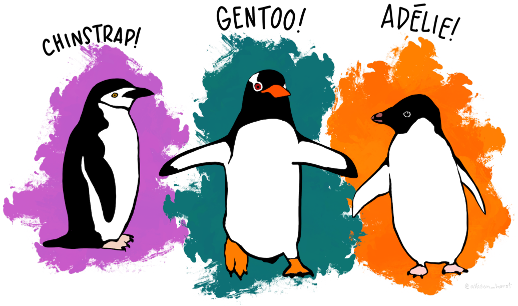
DSC 140A

Probabilistic Modeling & Machine Learning

Lecture 01 | Part 2

A Simple Prediction Algorithm

Penguin Prediction



[1]

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

[1] 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.

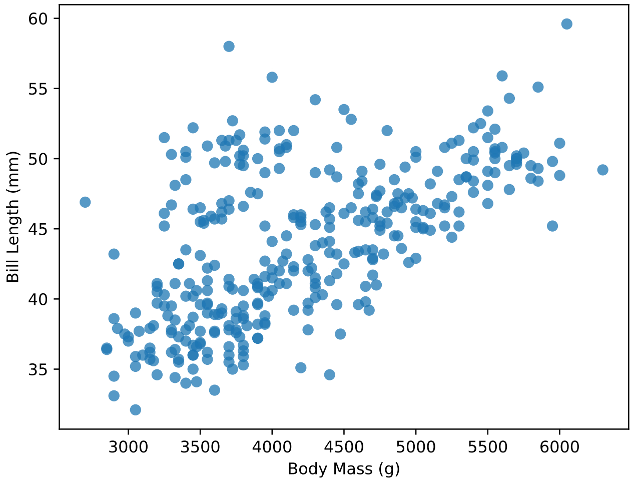
First Step: Featurization

- Chosen features should contain enough info to distinguish between species.

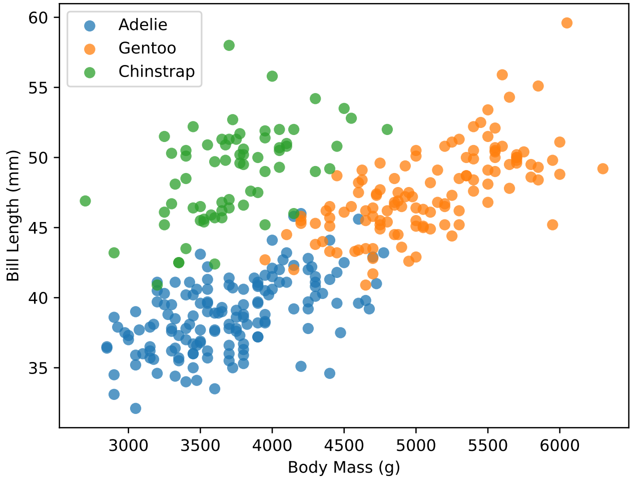
- We will represent each penguin by two numbers:
  1. Body Mass (in grams)
  2. Bill Length (in millimeters)

- Allows us to **embed** penguins as **point cloud** in $\mathbb{R}^2$.

Penguin Embedding

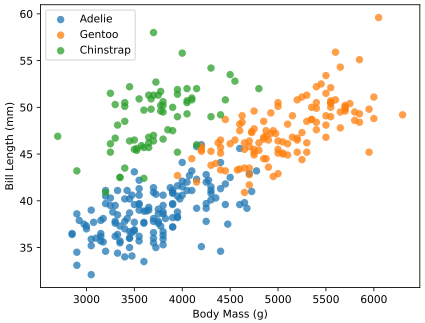


Penguin Embedding



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

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

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

A Note About Distances

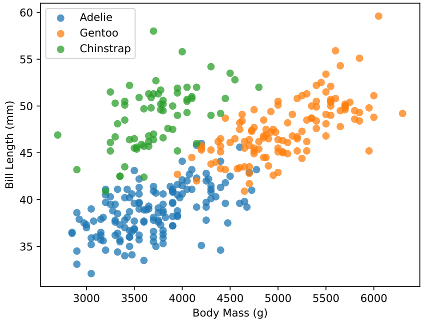
- We found the nearest neighbor with **Euclidean distance**:

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

- 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:

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

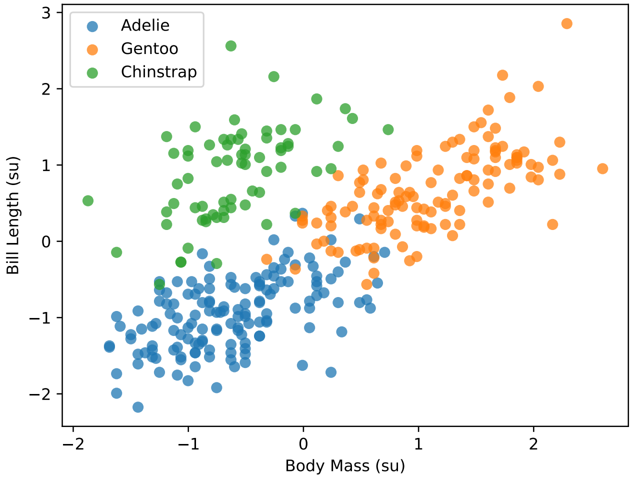
A Common Fix

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

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

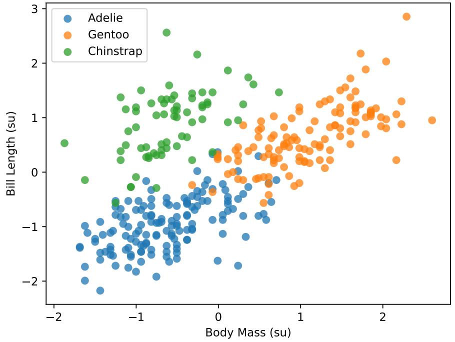
- Not always the right approach, though!

Standardized Penguins



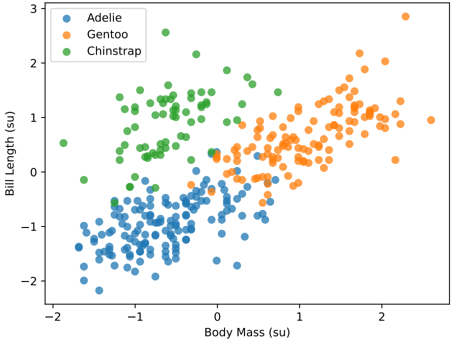
The Decision Boundary

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

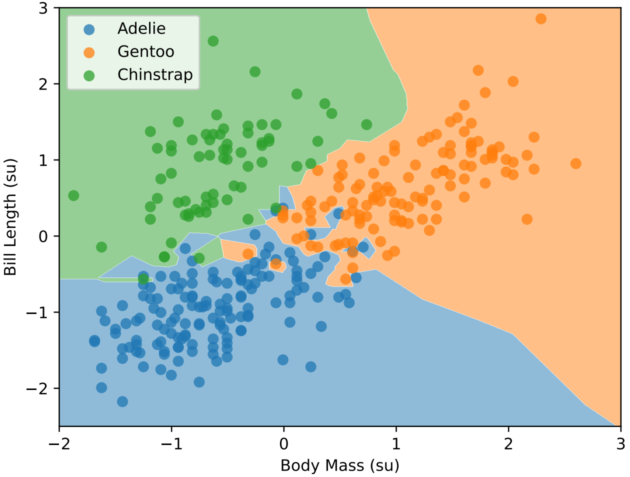


Exercise

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



The Decision Boundary

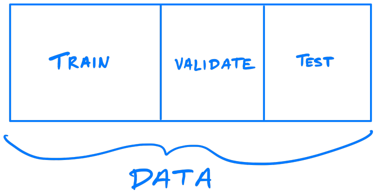


k-Nearest Neighbors

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

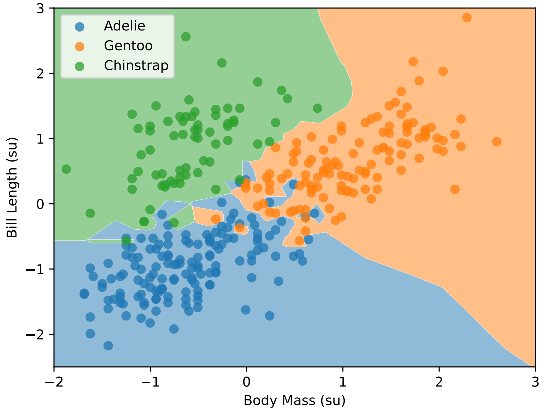
Choosing *k*

- 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

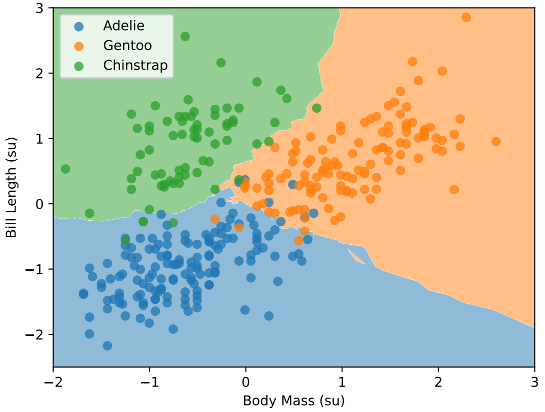
- How might the decision boundary change as we increase *k*?



$k = 1$

k and the Decision Boundary

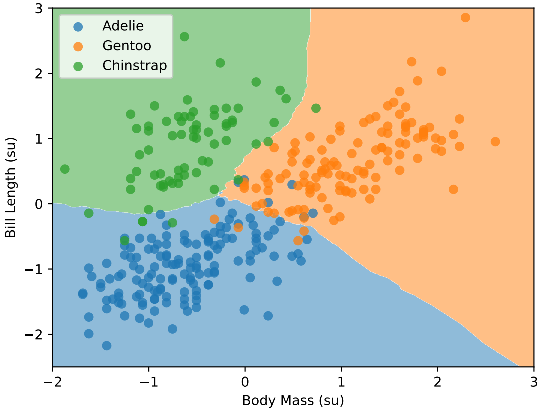
- How might the decision boundary change as we increase k?



$k = 10$

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.

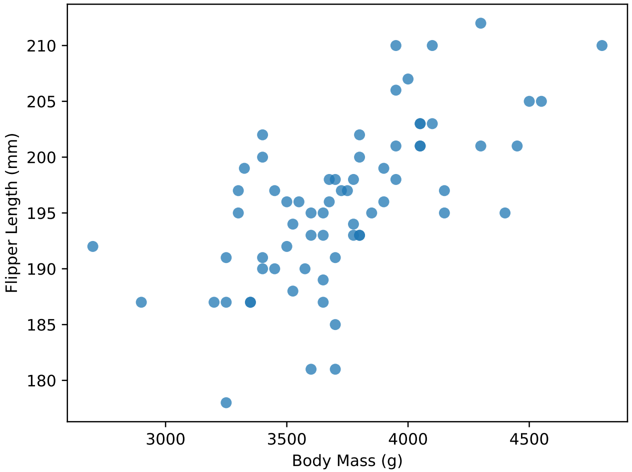
Exercise

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

Nearest Neighbor Regression

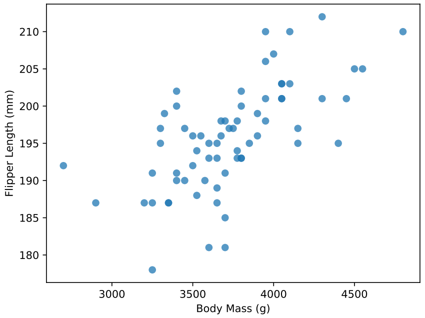
- The nearest neighbor rule can be used for **regression**, too.

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: $\{(\vec{x}^{(i)}, y_i)\} = \{(\vec{x}^{(1)}, y_1), \ldots, (\vec{x}^{(n)}, y_n)\}$

- **Given:** a new point, $\vec{z}$ with unknown target.

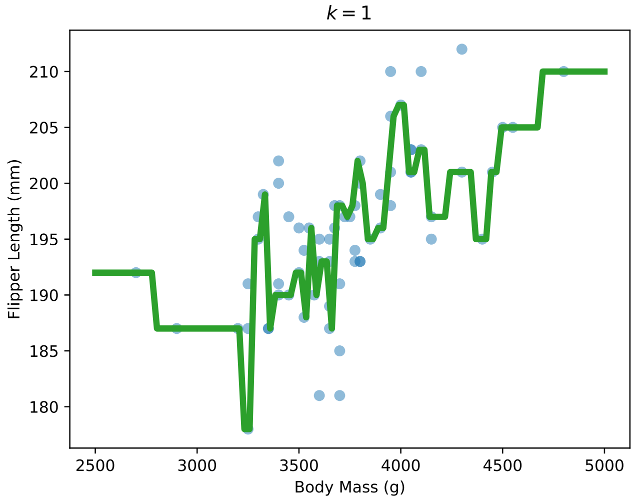
- **Predict:**
  1. Find the closest point to $\vec{z}$ in $\mathcal{D}$:
  $$i^* = \arg\min_{i \in \{1,\ldots,n\}} \|\vec{x}^{(i)} - \vec{z}\|$$
  2. Use y_{i^*} as the predicted target.

*k*NN Regression

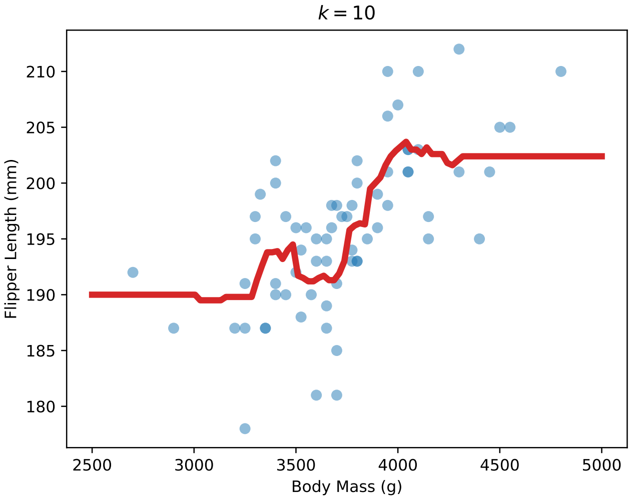
- As with classification, can generalize to *k* nearest neighbors.

- Natural prediction: the **mean** of the targets of the *k* closest neighbors.

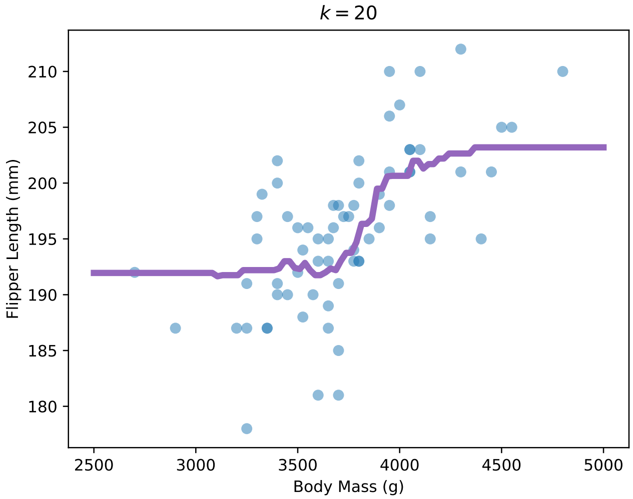
kNN Penguin Regression



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

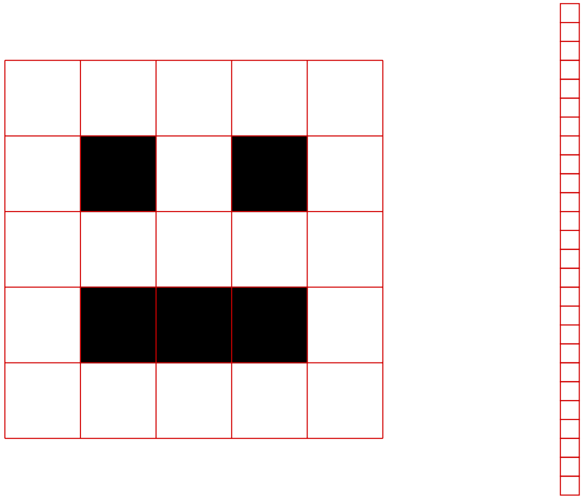
- 60,000 training images and associated labels.
- Each image is 28×28 pixels.
- Each pixel is 8 bit grayscale (0 - 255)

kNN on MNIST

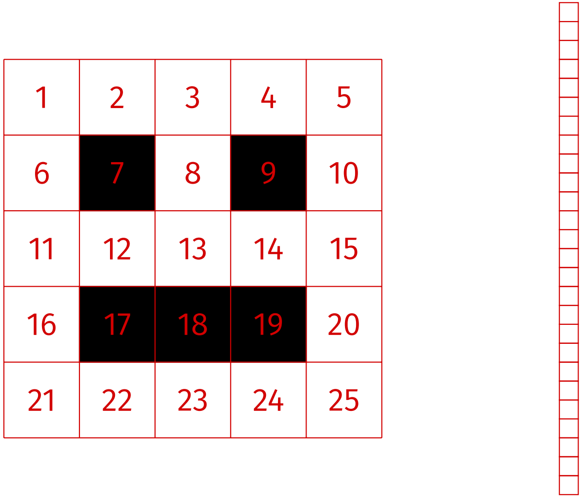
- We'll use kNN to do **character recognition**.
  - $\mathcal{X}$ = set of 28 × 28, 8-bit grayscale images
  - $\mathcal{Y} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.

- How do we represent an image as a feature vector?

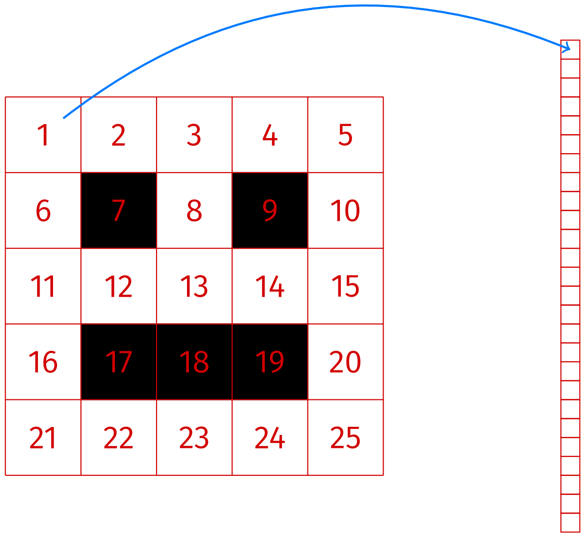
Images as Feature Vectors



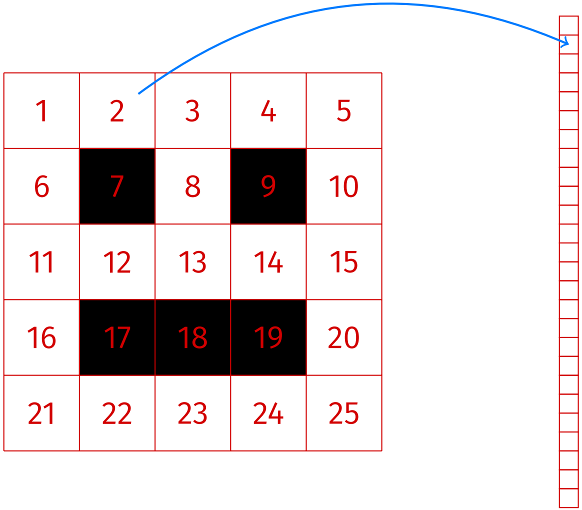
Images as Feature Vectors



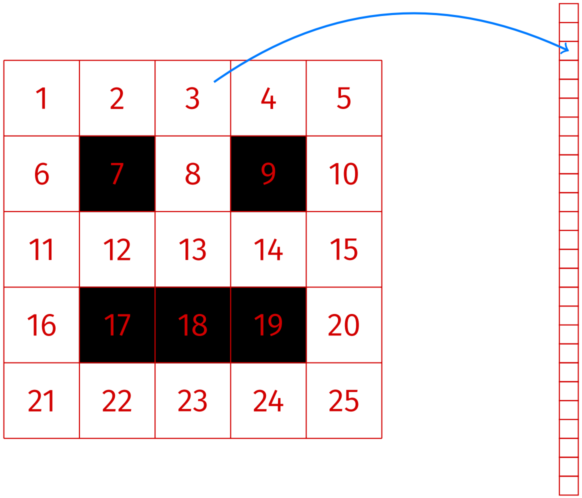
Images as Feature Vectors



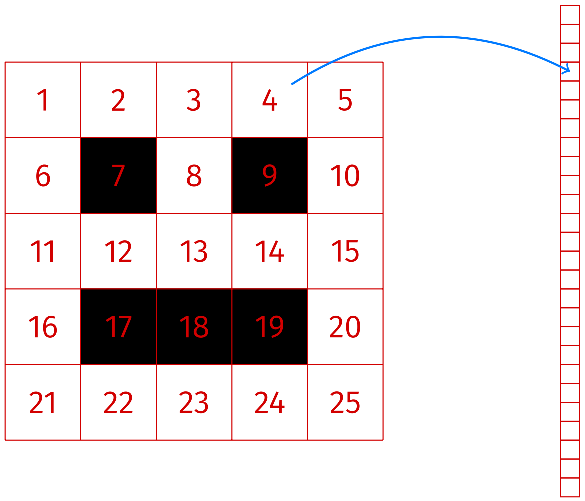
Images as Feature Vectors



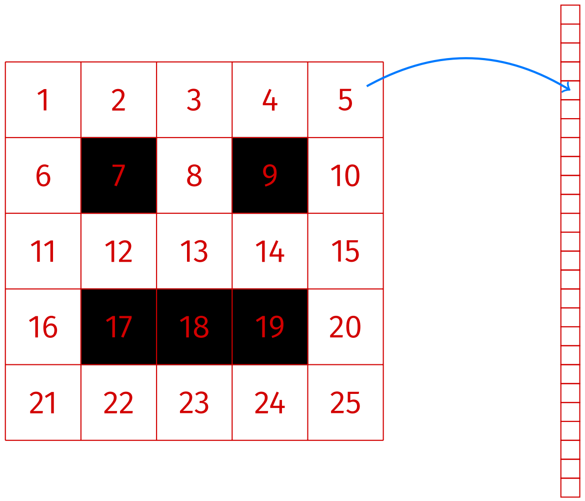
Images as Feature Vectors



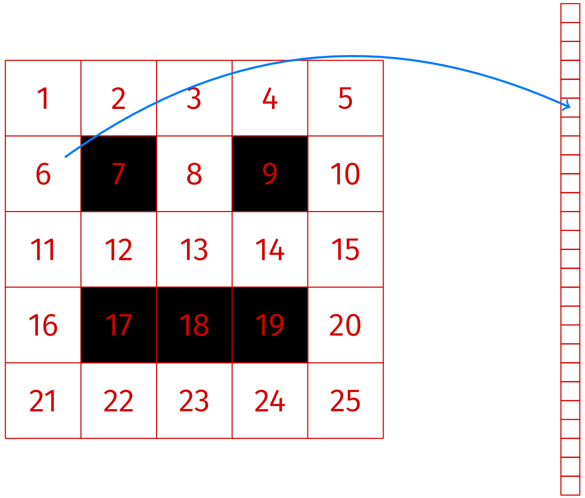
Images as Feature Vectors



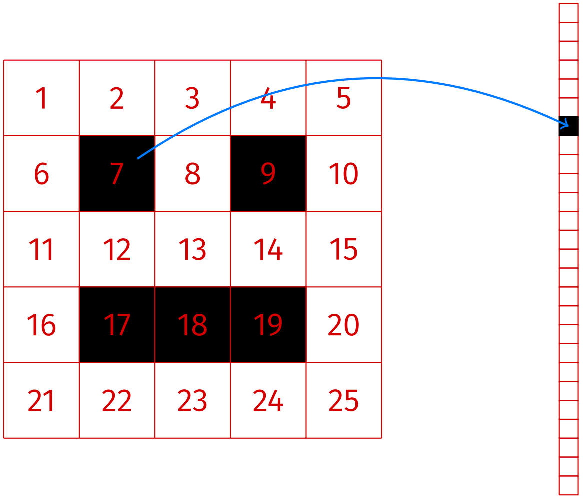
Images as Feature Vectors



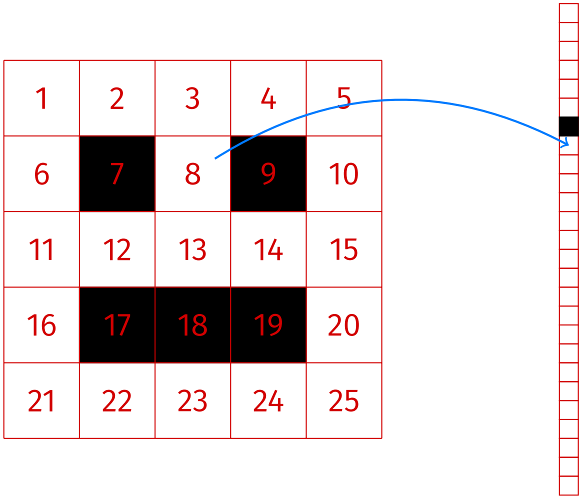
Images as Feature Vectors



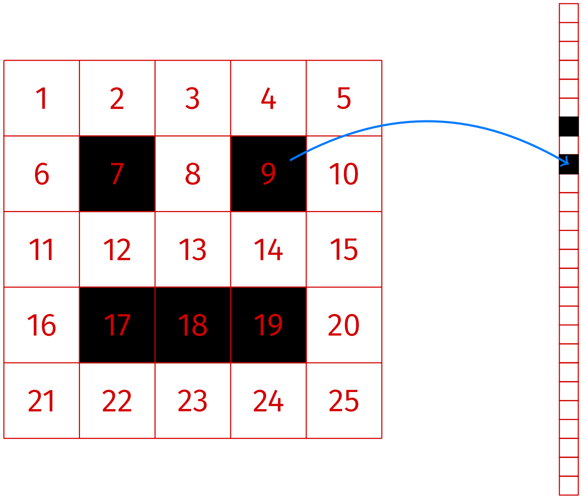
Images as Feature Vectors



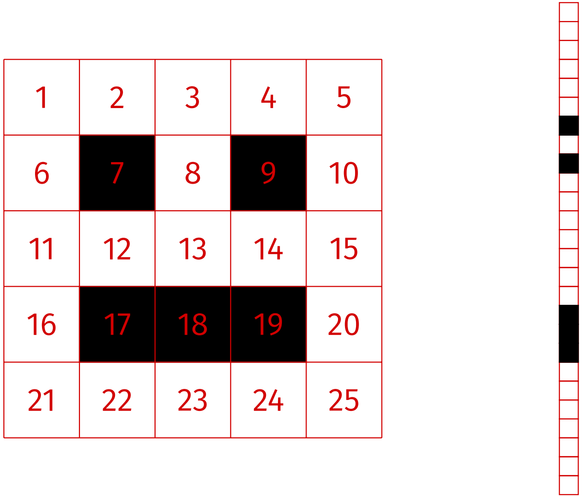
Images as Feature Vectors



Images as Feature Vectors



Images as Feature Vectors



Euclidean Distance Between Images

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

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

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?

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

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

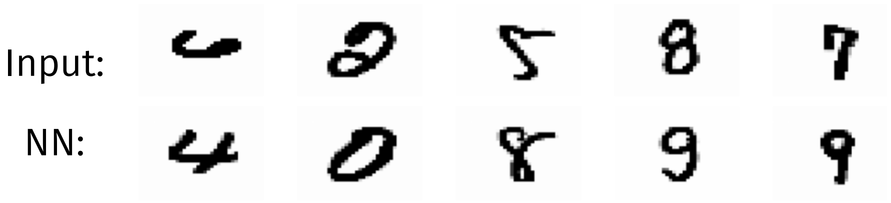
- Test set contains 1,000 images from each digit.
- Suppose our classifier guesses at random.
  Question: what is the expected test error?

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:



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 140A

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

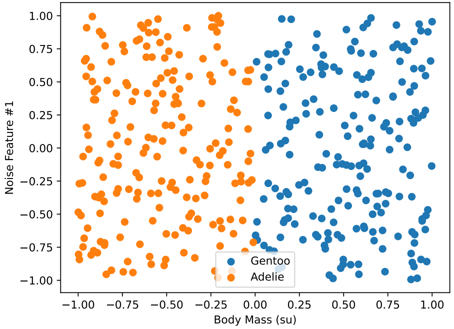
- Nearest neighbor approaches **do not learn** which features are **useful** and which **are not**.

Example

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

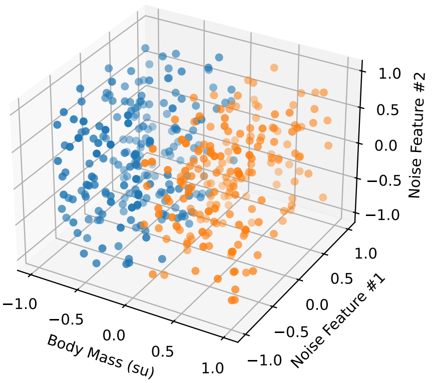
Example: One Noisy Feature

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



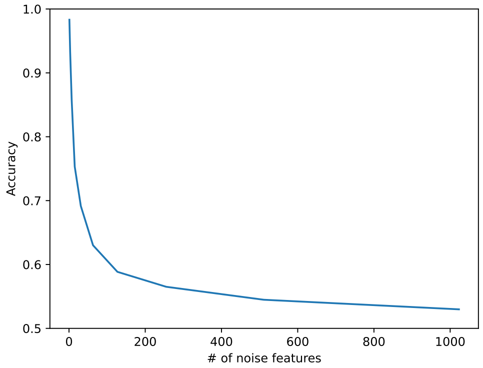
Example: Two Noisy Features

- 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.[2]
- Distance becomes less meaningful as *noisy* features are added.

[2]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

- kNN 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.