k-Nearest Neighbour

Outline

- Hour 1
  - Student reviews, discussion and measures to incorporate them
  - Review, announcements
- Hour 2: k-Nearest Neighbour (k-NN)

Student reviews and how I accounted for them

1) Hand-writing/pace of course is fast, audio quality is not good
   - I will write slower and improve recording audio

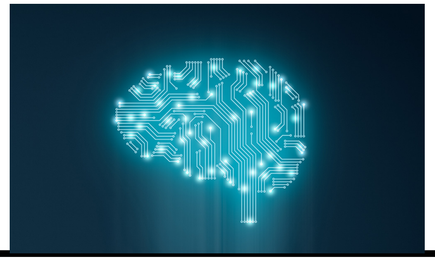
2) Python programming/exercise motivation
   - See AI in real-world datasets/mechanical engineering applications so less abstract
   - Iterate some of the main theory concepts: python complements lectures
   - Coding is not the main point: I will post the python with solutions from now on

3) More problem sets or some projects?
   - See all past problem sets and quizzes are provided in Moodle
   - I will do more worked out examples in class

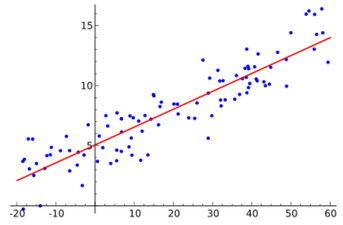
4) Complexity of the course material?
   - It's a mathematical course: lecture, python and problem sets all work together to help
   - Goal: see the math behind AI tools, connect with engineering applications

5) Level of noise in class

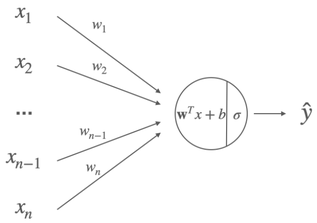
Introduction



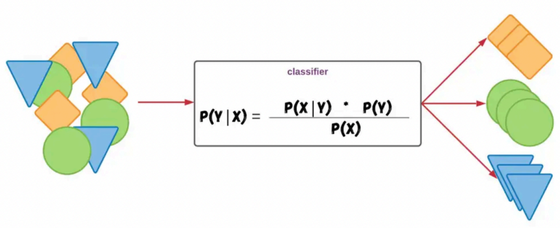
Linear regression



Logistic regression



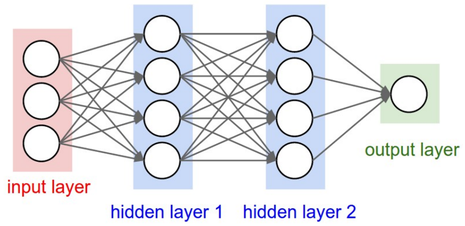
Naive Bayes



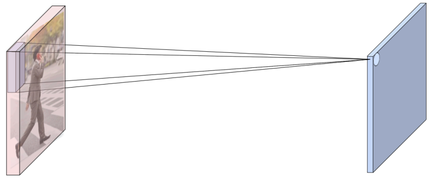
KNN

AI ethics

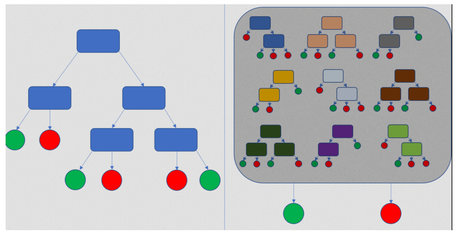
Neural networks



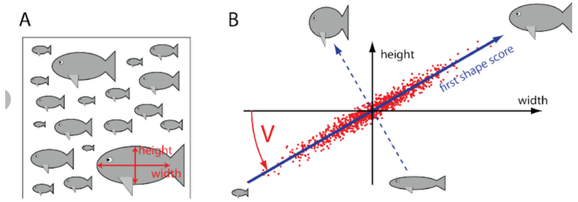
Convolutional neural networks



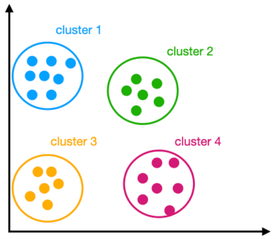
Decision-trees



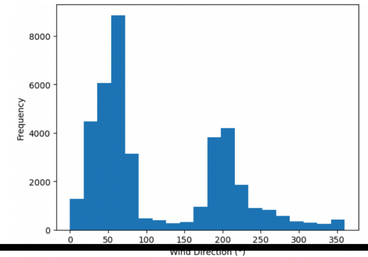
Dimensionality reduction



Clustering



Reinforcement learning



Announcements

1) Special lectures:
    1) Dec 9: AI in Industry, representative from Swiss Data Science Center
    2) Dec 16: AI ethics, representative from EPFL ethics instructor

2) Exam: Friday 31.01.2025 from 15h15 to 18h15 (CE11, CE12, CE1515)

   See past exams with solutions posted on Moodle

3) EPFL AI Day: https://memento.epfl.ch/event/artificial-intelligence-day/

Artificial Intelligence Day

Event details

Date	23.11.2024
Hour	10:00 › 17:00
Speaker	Programme complet
Location	Rolex Learning Center Online
Category	Conferences - Seminars
Event Language	French

Review - Naive Bayes classifier and additional exercises

$$P(y = c \mid x) = \frac{\prod_{j=1}^{d} P(x_j \mid y = c) P(y = c)}{P(x)}$$

Naive $\approx P(x \mid y=c)$, $x \in \mathbb{R}^d$

$\{x^i, y^i\}_{i=1}^N$

$x^i \in \mathbb{R}^d$ ✓

$y^i \hookrightarrow$ label $\in \{1, 2, \dots, K\}$

Naive:

x_j, $j = 1, \dots, d$

are independent

given label

y

- **Exercises and examples of Naive Bayes classifier**

  1) Python exercise: you did this last week and saw how to code this: dataset for predicting energy efficiency of home devices

  2) Online video: additional sources of help and examples: link to StatQuest videos provided

  3) Exam 2023, Problem 3, parts 1-7: this week you can go through this and ask questions during exercise hours, note that solutions are provided online

  4) Example today: on conditional independence

Exercise - conditional independence

Background of dataset and problem

Credit: Michael Fleshman via flickr.com

Data from the New York Civil Liberties Union (NCLU)

Group	Population (NYC)	Arrests/Summons	Stops
White	2,717,796	1,938	10,228
Latino	2,346,883	6,540	26,181
Black	1,875,108	9,840	49,362
Others	1,245,527	1,010	6,612
Total	8,185,314	19,328	92,383

Table 1: Stops and arrests/summons by NYC police force, 2014–2017.

Source: Manuel Foerster and Dominik Karos Article

Step by step example - conditional independence

- **1) Show the probability of being arrested is not independent of being** R1

Group	Population	Number arrested
R1	1,8 x 10^6	10 x 10^3
R2	2,7 x 10^6	2 x 10^3

two events A, B are independent if

$P(A|B) = P(A)$

(equivalently, $P(A, B) = P(A)P(B)$)

Event A : being from R1

Event B : being arrested

$$P(A|B) = \frac{\#\ R1 \text{ is arrested}}{\#\ \text{arrested}} = \frac{10 \times 10^3}{(10+2) \times 10^3} = \frac{5}{6}$$

$$P(A) = \frac{\#\ R1}{\#\ \text{population}} = \frac{1.8 \times 10^6}{(1.8+2.7) \times 10^6} = \frac{2}{5}$$

as $\frac{5}{6} \neq \frac{2}{5}$ then events are not independent

Step by step example - conditional independence

- **2) Show, conditioned on being stopped, probability of being arrested is independent of being black**

$$P(A, B \mid C) = P(A \mid C)\, P(B \mid C) \quad ?$$

(we used this assumption in Naive Bayes classifier)

Group	Population	Number arrested
R1	1,8 x 10^6	10 x 10^3
R2	2,7 x 10^6	2 x 10^3

Group	Population	Number arrested	Number stopped
R1	1,8 x 10^6	10 x 10^3	5 x 10^5
R2	2,7 x 10^6	2 x 10^3	1 x 10^5

Event C : being stopped

1) $P(A, B \mid C) = \frac{10 \times 10^3}{(5+1) \times 10^5} = \frac{10 \times 10^3}{6 \times 10^5}$

2) $P(A \mid C) = \frac{5 \times 10^5}{6 \times 10^5} =$

3) $P(B \mid C) = \frac{(10+2) \times 10^3}{6 \times 10^5} = \frac{12 \times 10^3}{6 \times 10^5}$

$\frac{10 \times 10^3}{6 \times 10^5} \overset{?}{=} \frac{5 \times 10^5}{6 \times 10^5} \times \frac{12 \times 10^3}{6 \times 10^5}$ ✓

In Naive Bayes classifier, we assume given label $y = c$, each feature is independent

assumption

$$P(x \mid y=c) \overset{\text{assumption}}{=} P(x_1 \mid y=c) P(x_2 \mid y=c) \cdots P(x_d \mid y=c)$$

↳ we saw example where $x \in \mathbb{R}^d$ was presence / absence of d words in emails

After this assumption, Naive Bayes classifier ...

$$P(y=c \mid x) = \frac{P(x_1 \mid y=c) P(x_2 \mid y=c) \cdots P(x_d \mid y=c) P(y=c)}{P(x)}$$

k-Nearest Neighbour

k-NN Problem setup
Supervised machine learning

Recall machine learning goal: Use a dataset to produce useful predictions on never-before-seen data.

$\{x^i, y^i\}_{i=1}^N$ $\quad x^i \in \mathbb{R}^d$, $y^i \in \{1, 2, \dots, K\}$

Recall terminology:

- **Features:** input variables $x^i \in \mathbb{R}^d$
- **Label:** what we are predicting y^i

k-NN Abstraction for classification

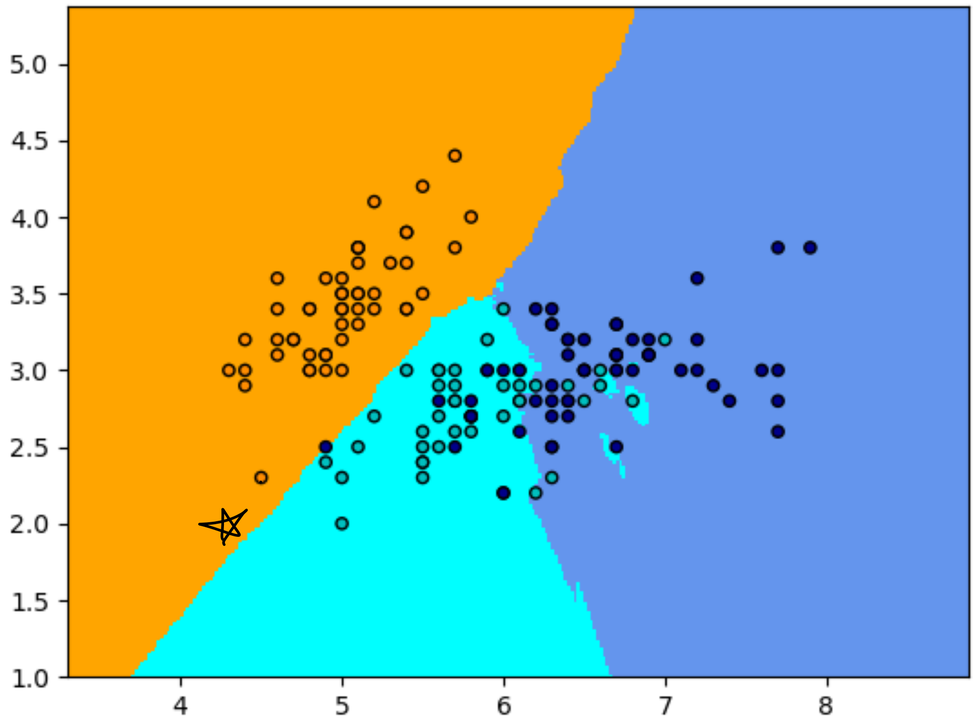
Problem: Classifying data points among different categories.

k-NN (***k-Nearest Neighbors***) algorithm assumes that data points of similar classes exist in close proximity

(Similar Inputs have similar outputs)

It classifies an unknown data point according to the category of its k nearest neighbors

(k = 1, 3, 5, …0) for binary classification



k-NN Distance Metric

How to measure the proximity between data points? → Measure distance

Examples of distance metrics:

$x^1, x^2 \in \mathbb{R}^d$

Euclidean (L2) distance

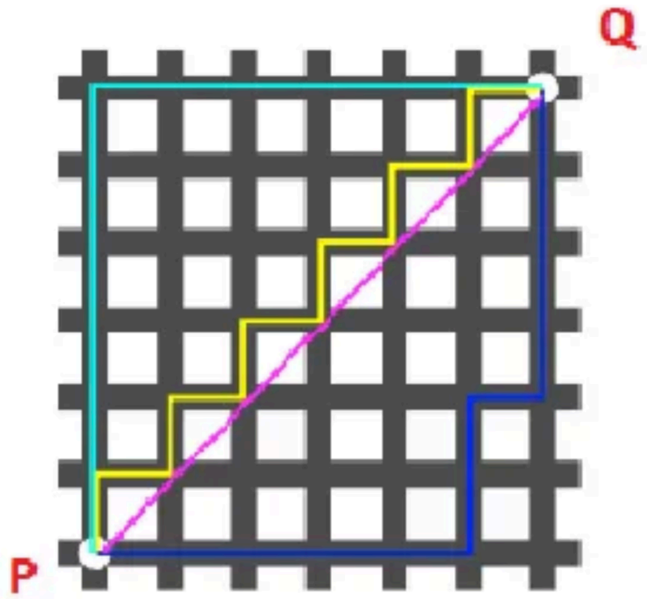
$$D(x^1, x^2) = \left((x_1^1 - x_1^2)^2 + (x_2^1 - x_2^2)^2 + \cdots + (x_d^1 - x_d^2)^2 \right)^{\frac{1}{2}}$$

Euclidean

Manhattan (L1) distance

$$D(x^1, x^2) = |x_1^1 - x_1^2| + |x_2^1 - x_2^2| + \cdots + |x_d^1 - x_d^2|$$

Manhattan



k-NN Distance Metric

How to measure the proximity between data points? → Measure distance

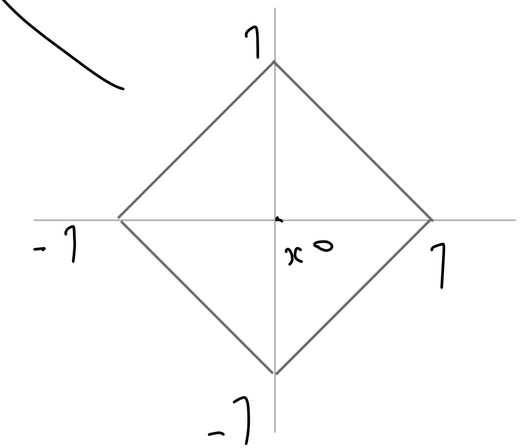
$$|x_1 - x_1^0| + |x_2 - x_2^0| = 1$$

$$\|x - x^0\|_2 = 1 \Rightarrow$$

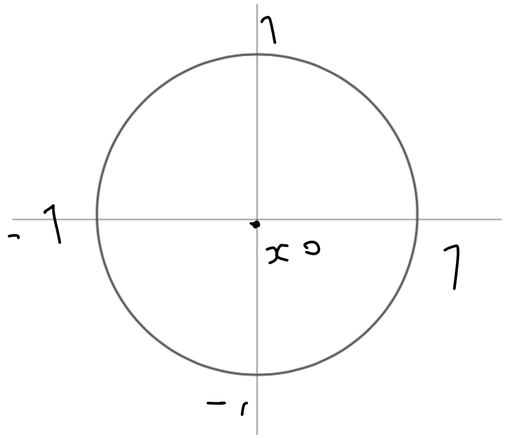
$$\left((x_1 - x_1^0)^2 + (x_2 - x_2^0)^2 \right)^{1/2} = 1$$

Level sets of the two most common distances

L1 (Manhattan) distance



L2 (Euclidean) distance



$$(x - x_1^0)^2 + (x_2 - x_2^0)^2 = 1$$

{ x satisfys above } is a circle of radius 1

k-NN Feature scaling

Features might have different scales

Distance metric gives more importance to the feature with largest scale

Normalizing data allows equal exploitation of information from features

Z-Score Standardisation: Set mean (**μ**) to 0, standard deviation (**σ**) to 1

Lets consider $x \in \mathbb{R}^d$

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} \begin{matrix} \longrightarrow \text{feature 1} \\ \longrightarrow \text{feature 2} \end{matrix}$$

to scale each feature x_j

$$\begin{cases} \mu_j = \dfrac{\sum_{i=1}^{N} x_j^i}{N} \\ \sigma_j = \dfrac{\sum_{i=1}^{N} (x_j^i - \mu_j)^2}{N-1} \end{cases}$$

$\{x^i\}_{i=1}^N$ are the data point.

$$\tilde{x}_j^i = \frac{x_j^i - \mu_j}{\sigma_j}, \quad \begin{matrix} i = 1, 2, \dots, N \\ j = 1, \dots, d. \end{matrix}$$

EPFL

k-NN How would you implement it?

fail pass

$x \in \mathbb{R}^2$, x_1 = # of hours studied, $y \in \{0, 1\}$

x_2 = # lectures attended

Problem: you are given a dataset, say students study hours and lecture attendance and whether they pass a course or not

$\{x^i, y^i\}_{i=1}^{N}$ N : number of students we have data for.

You want to predict for a given student how she'll do.

$x^{test} = (x_1^{test}, x_2^{test})$

k-NN (***k-Nearest Neighbors***) algorithm assumes that data points of similar classes exist in close proximity (Similar Inputs have similar outputs)

For simplicity take k=1, and use Manhattan distance

$x^i \in \mathbb{R}^2$

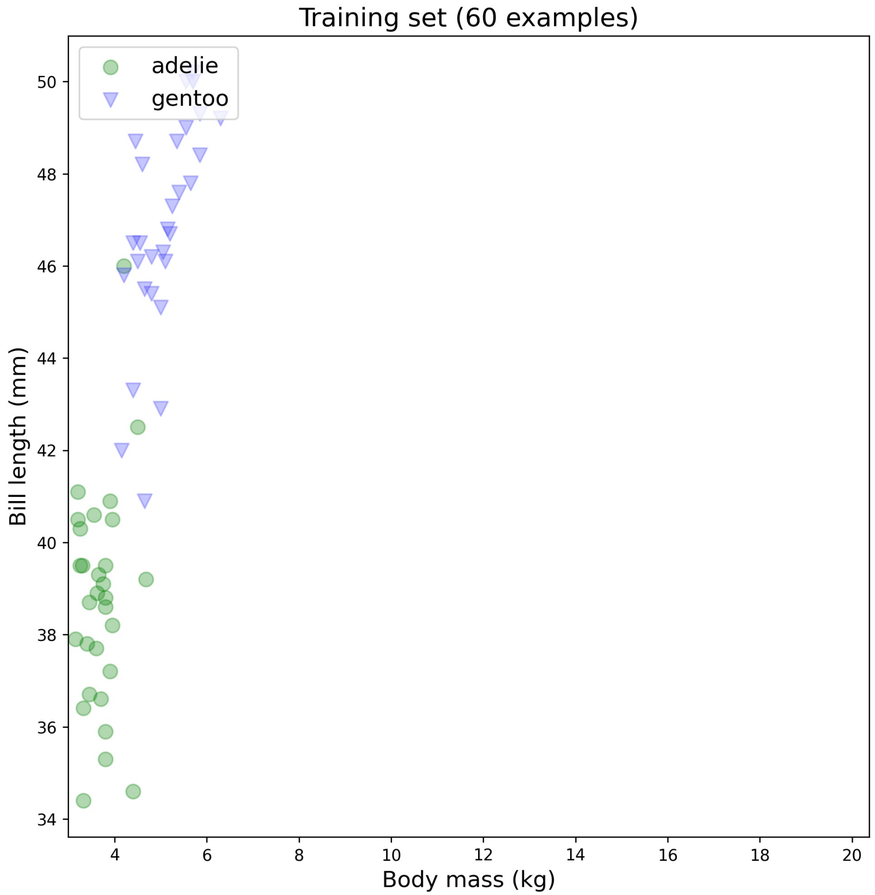
we go through each point in dataset & compute

Distance of x^{test} from x^i $\therefore$ $|x_1^{test} - x_1^i| + |x_2^{test} - x_2^i| = d^{test,i}$

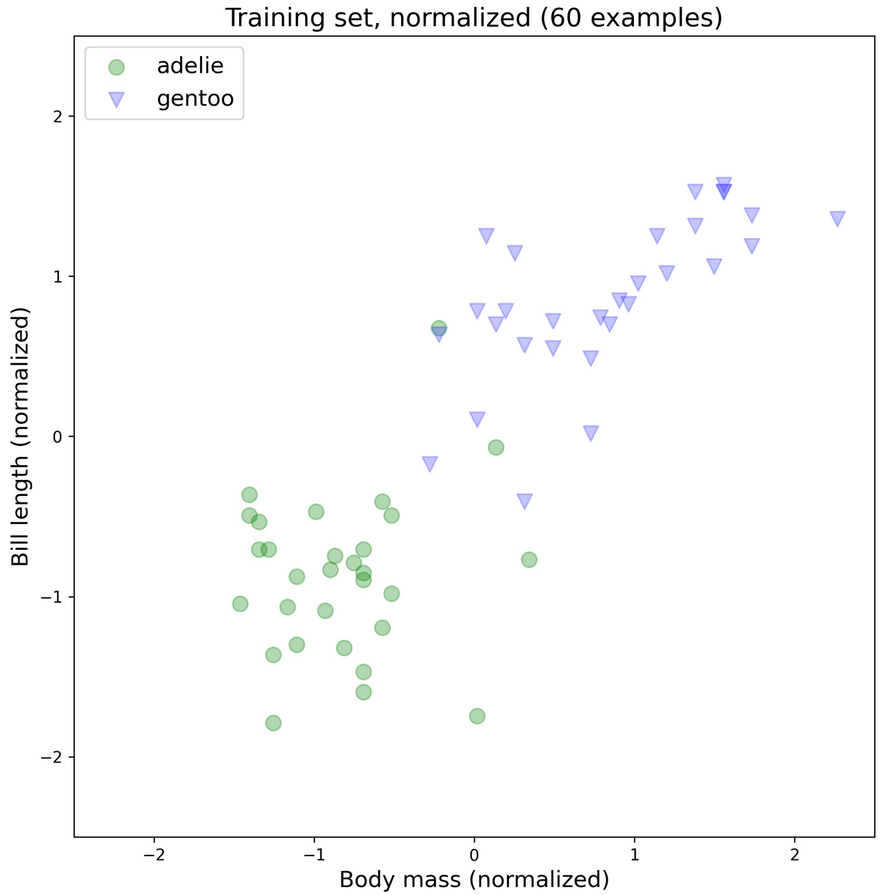
$\min_i (d^{test,i}) = d^{test,i^*}$, we give label of x^{i^*} to x^{test}.

k-NN

Feature scaling - Visualisation



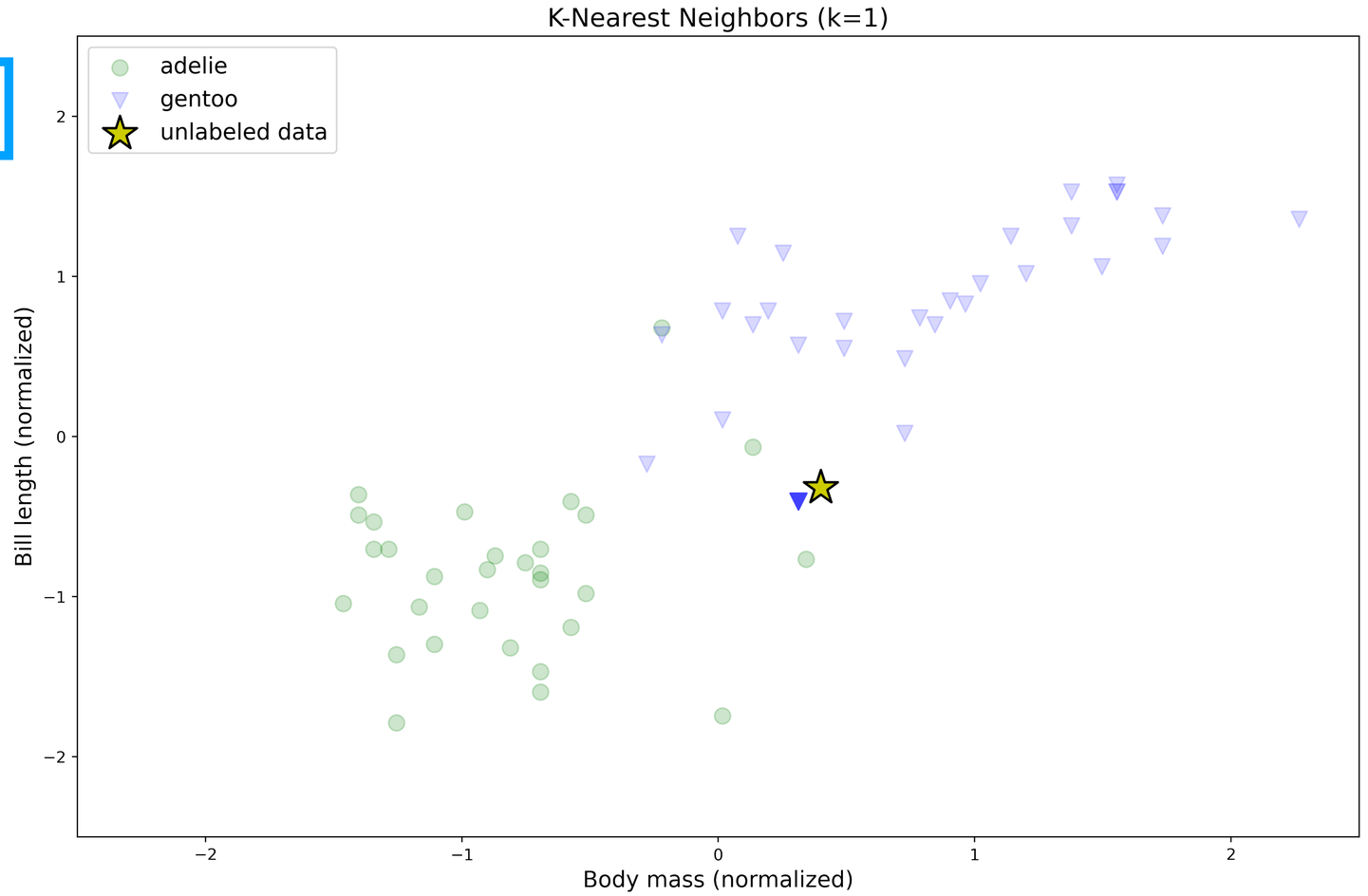
Normalize



k-NN
Visualisation

When **k=1**:
take label of nearest neighbor

k=1



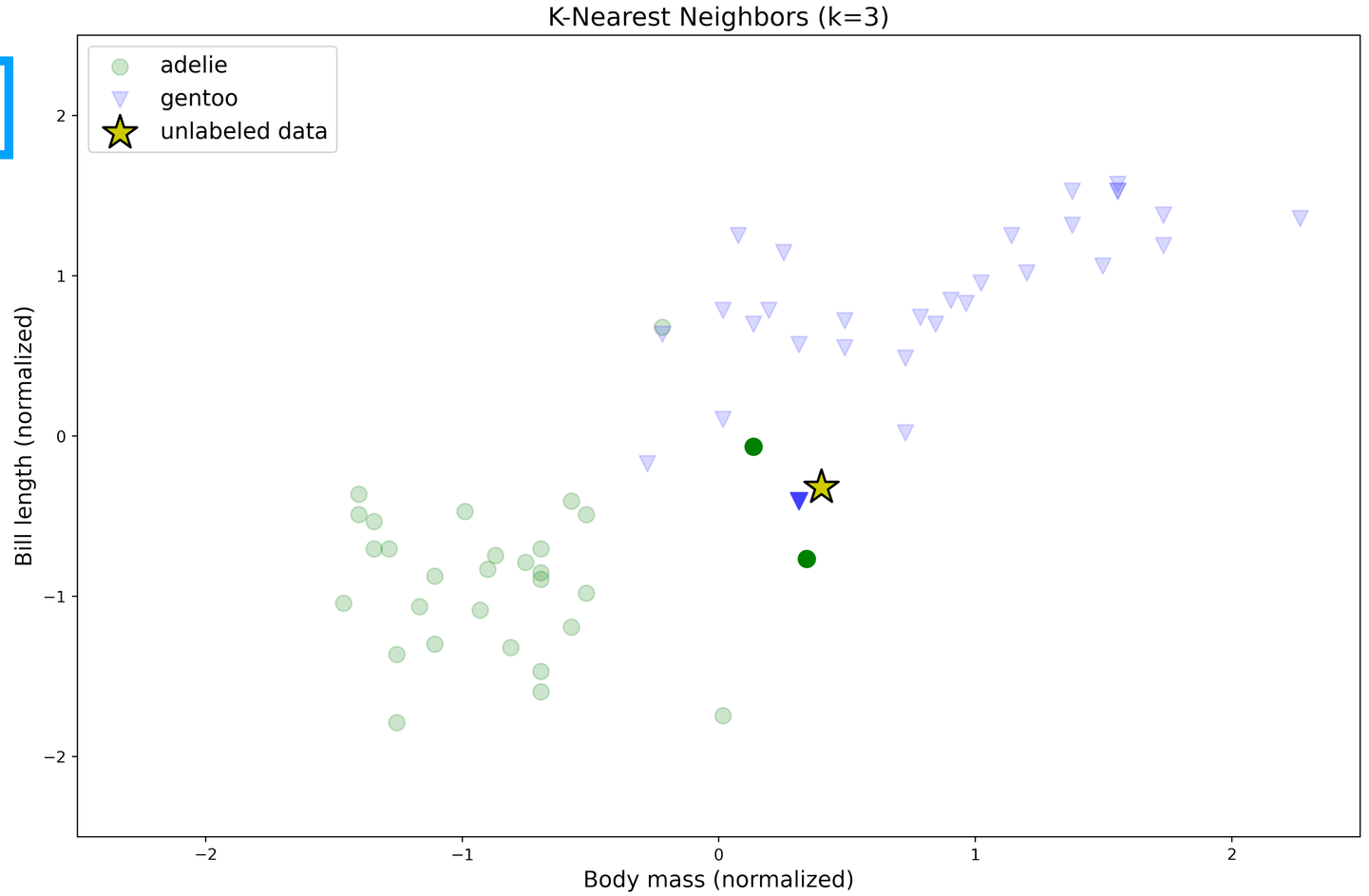
In which category belongs the ★ point? → **Gentoo**

k-NN

$k > 1$

Instead of copying label from nearest neighbor,
take **majority vote** from k closest points

k=3



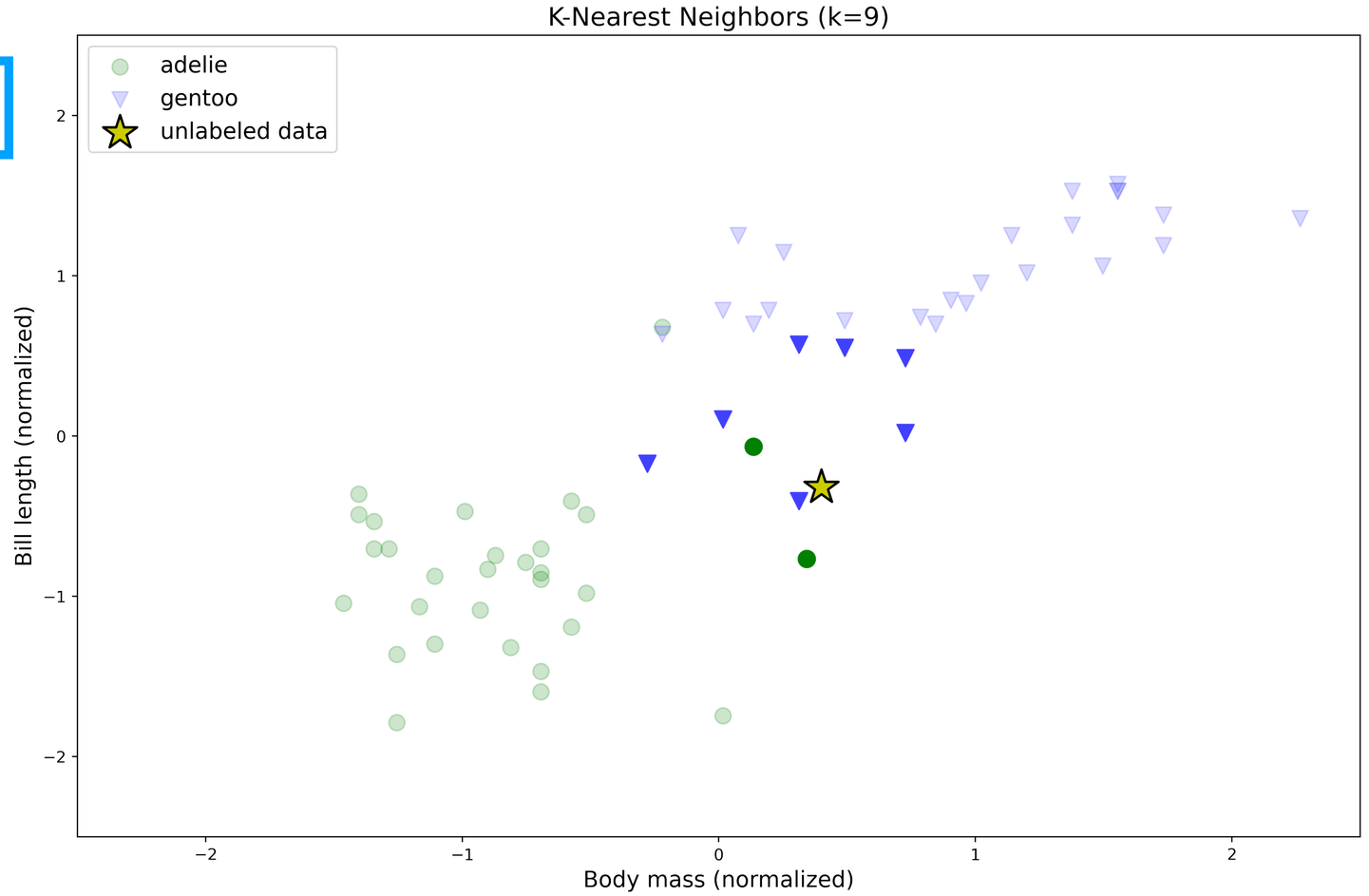
In which category belongs the ★ point? → **Adélie**

k-NN

k > 1

Instead of copying label from nearest neighbor,
take **majority vote** from k closest points

k=9



In which category belongs the ★ point? → **Gentoo**

Programming - What is a pseudo-code?

- Description of an algorithm in a language independent way
- It's what we think the algorithm should do before we encode it in python, matlab, C, etc.
- It's the most important step. If you can think in the pseudo-code, then you understand the problem and can code it in language of your choice
- In this course, I don't require a formal syntax of python (otherwise, it becomes a programming again)
- But you are required to know how a pseudo-code would work

EPFL

k-NN
Pseudo-code

1. Initialize k , inihlize distance choice (Euclidean / Manhattan / ...)
2. For every data point in the training set x^i for $i = 1, \dots, N$

   Calculate the **distance** with the unknown data point $Dist(x^{test}, x^i)$
3. Pick the **k nearest known** data points from the unknown data point ,
4. Get the **labels** of the selected **k** known data points
5. Return the mode (also known as majority vote) of the **k** labels

y^{test} = mode label of the label of k closest points.

How would you use k-NN for regression? $y \in \mathbb{R}$

we can still find k closest points,

and for label : $\dfrac{y^{i_1} + y^{i_2} + \dots + y^{i_k}}{k}$,

$i_1, \dots, i_k$ indices of the k closest data points.

k-NN

Implementation (in your exercises this week)

```python
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        self.X_train = X
        self.y_train = y

    def euclidean_dist(self, x):
        return np.sqrt(((self.X_train - x) ** 2).sum(axis=1))

    def predict(self, X, k):
        # Get the number of rows to predict
        num_test = X.shape[0]

        y_pred = np.zeros(num_test, dtype=self.y_train.dtype)

        # Find nearest neighbor for each row
        for i in range(num_test):
            # Calculate distances between data point and all points in the train set
            distances = self.euclidean_dist(X[i, :])
            # Find k-closest neighbors in the train set, then find labels of closest neighbors
            neighbor_indices = np.argsort(distances)[:k]
            neighbor_labels = self.y_train[neighbor_indices]

            # Find most frequent label among k-cloest neighbors
            best_label = np.argmax(np.bincount(neighbor_labels))
            y_pred[i] = best_label

        return y_pred
```

k-NN
Implementation

```python
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        self.X_train = X
        self.y_train = y

    def euclidean_dist(self, x):
        return np.sqrt(((self.X_train - x) ** 2).sum(axis=1))

    def predict(self, X, k):
        # Get the number of rows to predict
        num_test = X.shape[0]

        y_pred = np.zeros(num_test, dtype=self.y_train.dtype)

        # Find nearest neighbor for each row
        for i in range(num_test):
            # Calculate distances between data point and all points in the train set
            distances = self.euclidean_dist(X[i, :])
            # Find k-closest neighbors in the train set, then find labels of closest neighbors
            neighbor_indices = np.argsort(distances)[:k]
            neighbor_labels = self.y_train[neighbor_indices]

            # Find most frequent label among k-cloest neighbors
            best_label = np.argmax(np.bincount(neighbor_labels))
            y_pred[i] = best_label

        return y_pred
```

Save training data

k-NN
Implementation

```python
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        self.X_train = X
        self.y_train = y

    def euclidean_dist(self, x):
        return np.sqrt(((self.X_train - x) ** 2).sum(axis=1))

    def predict(self, X, k):
        # Get the number of rows to predict
        num_test = X.shape[0]

        y_pred = np.zeros(num_test, dtype=self.y_train.dtype)

        # Find nearest neighbor for each row
        for i in range(num_test):
            # Calculate distances between data point and all points in the train set
            distances = self.euclidean_dist(X[i, :])
            # Find k-closest neighbors in the train set, then find labels of closest neighbors
            neighbor_indices = np.argsort(distances)[:k]
            neighbor_labels = self.y_train[neighbor_indices]

            # Find most frequent label among k-cloest neighbors
            best_label = np.argmax(np.bincount(neighbor_labels))
            y_pred[i] = best_label

        return y_pred
```

For each test sample:

- Find k-closest training data
- Predict mode of k-closest training data

k-NN
Implementation

```python
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        self.X_train = X
        self.y_train = y

    def euclidean_dist(self, x):
        return np.sqrt(((self.X_train - x) ** 2).sum(axis=1))

    def predict(self, X, k):
        # Get the number of rows to predict
        num_test = X.shape[0]

        y_pred = np.zeros(num_test, dtype=self.y_train.dtype)

        # Find nearest neighbor for each row
        for i in range(num_test):
            # Calculate distances between data point and all points in the train set
            distances = self.euclidean_dist(X[i, :])
            # Find k-closest neighbors in the train set, then find labels of closest neighbors
            neighbor_indices = np.argsort(distances)[:k]
            neighbor_labels = self.y_train[neighbor_indices]

            # Find most frequent label among k-cloest neighbors
            best_label = np.argmax(np.bincount(neighbor_labels))
            y_pred[i] = best_label

        return y_pred
```

Q: With N examples, how fast are training and prediction?

k-NN
Implementation

```python
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        self.X_train = X
        self.y_train = y

    def euclidean_dist(self, x):
        return np.sqrt(((self.X_train - x) ** 2).sum(axis=1))

    def predict(self, X, k):
        # Get the number of rows to predict
        num_test = X.shape[0]

        y_pred = np.zeros(num_test, dtype=self.y_train.dtype)

        # Find nearest neighbor for each row
        for i in range(num_test):
            # Calculate distances between data point and all points in the train set
            distances = self.euclidean_dist(X[i, :])
            # Find k-closest neighbors in the train set, then find labels of closest neighbors
            neighbor_indices = np.argsort(distances)[:k]
            neighbor_labels = self.y_train[neighbor_indices]

            # Find most frequent label among k-cloest neighbors
            best_label = np.argmax(np.bincount(neighbor_labels))
            y_pred[i] = best_label

        return y_pred
```

Q: With N examples, how fast are training and prediction?

A: Train O(1), predict O(N)

k-NN
Implementation

```python
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        self.X_train = X
        self.y_train = y

    def euclidean_dist(self, x):
        return np.sqrt(((self.X_train - x) ** 2).sum(axis=1))

    def predict(self, X, k):
        # Get the number of rows to predict
        num_test = X.shape[0]

        y_pred = np.zeros(num_test, dtype=self.y_train.dtype)

        # Find nearest neighbor for each row
        for i in range(num_test):
            # Calculate distances between data point and all points in the train set
            distances = self.euclidean_dist(X[i, :])
            # Find k-closest neighbors in the train set, then find labels of closest neighbors
            neighbor_indices = np.argsort(distances)[:k]
            neighbor_labels = self.y_train[neighbor_indices]

            # Find most frequent label among k-cloest neighbors
            best_label = np.argmax(np.bincount(neighbor_labels))
            y_pred[i] = best_label

        return y_pred
```

Q: With N examples, how fast are training and prediction?

A: Train O(1), predict O(N)

If we are using for real-time decision-making, this could be bad: we want classifiers that are **fast** at prediction; **slow** for training can be ok

EPFL

k-NN
Hyperparameters

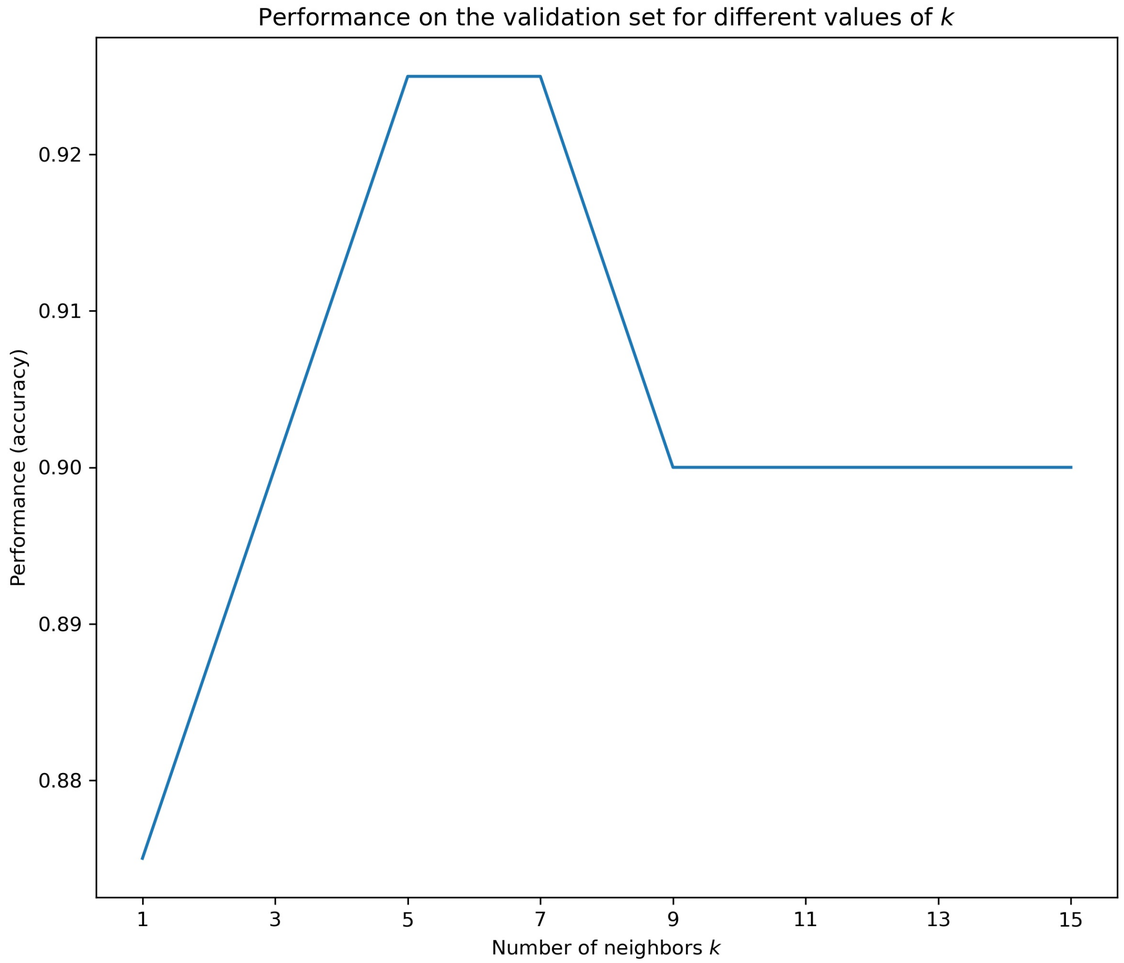
What is the best value of **k** to use ?

What is the best **distance** to use ?

These are **hyperparameters**: choices about the model/algorithm that we set rather than learn

Very problem-dependent.
Must try them all out and see what works best.

k-NN
Setting hyperparameters

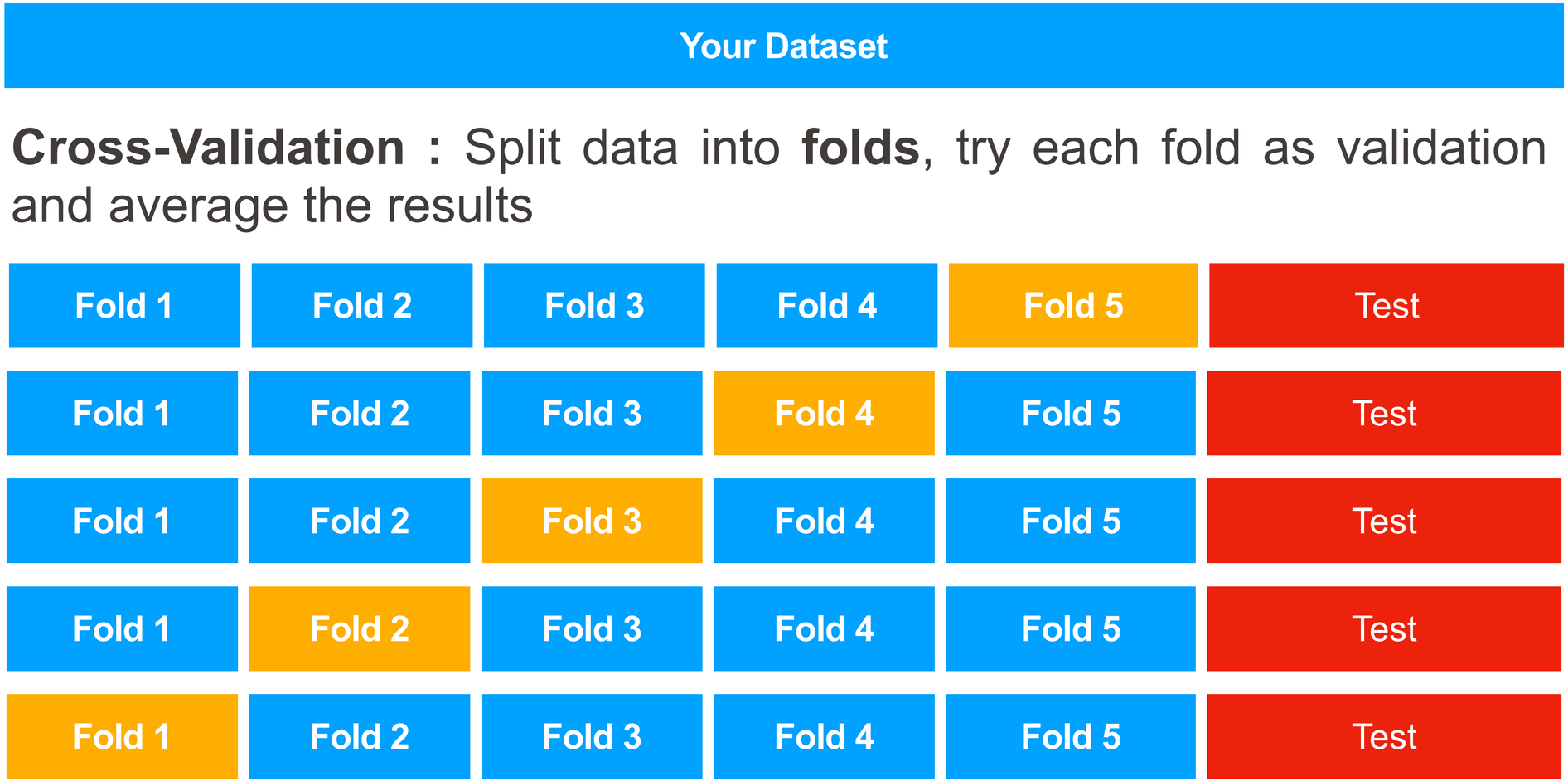


Validation set accuracy for different values of **k** for our penguin classification task

(Seems that k = 5 or 7 works best for this example)

Train, validate, test in ML

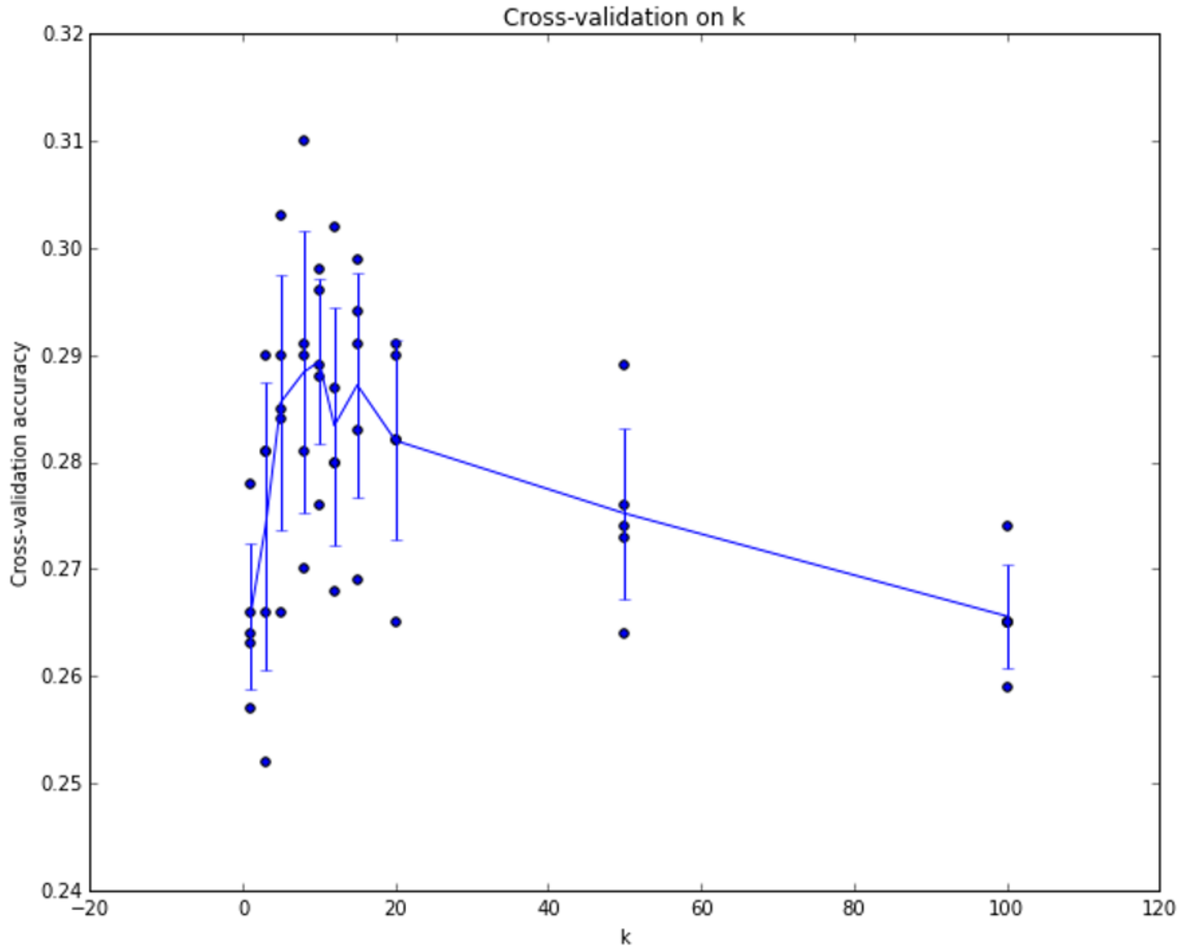
Setting hyperparameters



Cross-Validation : Split data into **folds**, try each fold as validation and average the results

Useful for small datasets

k-NN
Setting hyperparameters



Different dataset:
5-fold cross-validation
for the value of **k**.

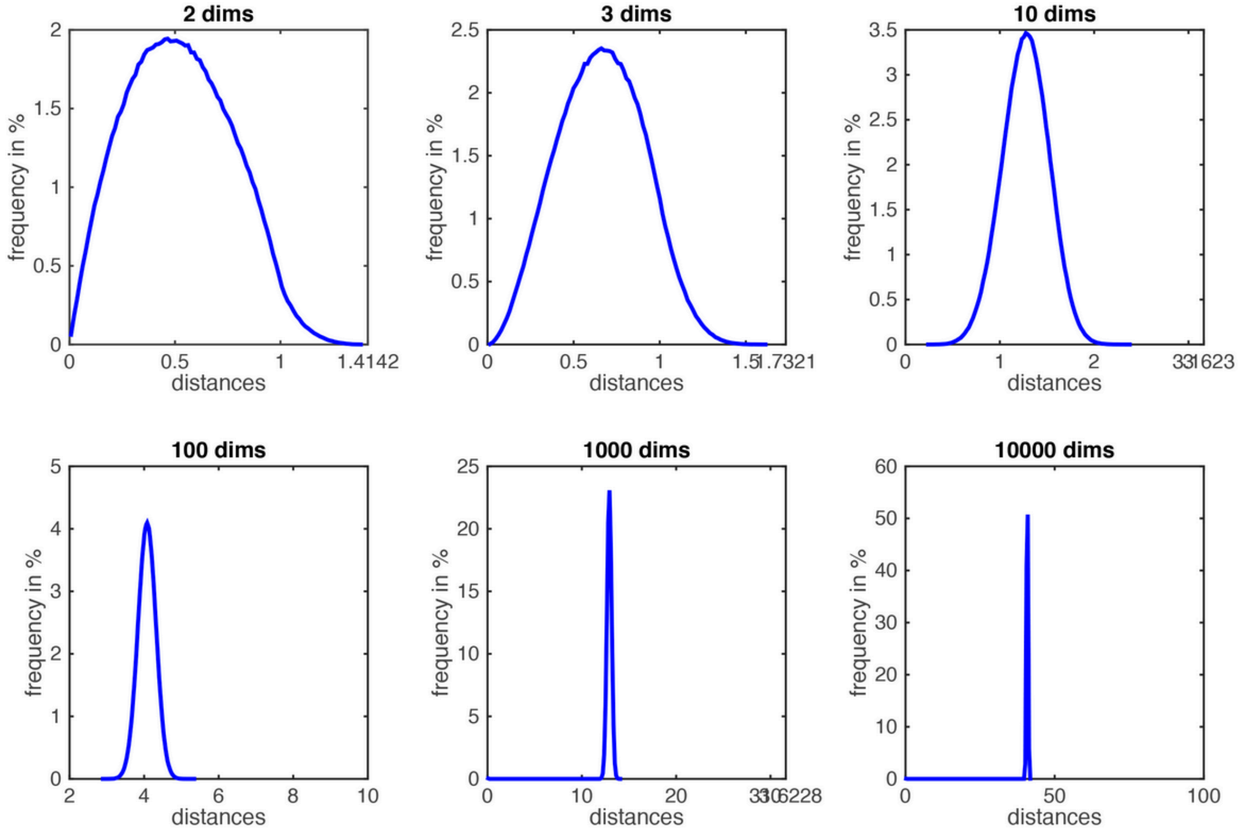
Each point: single
outcome.

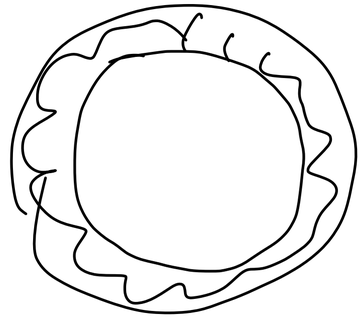
The line goes
through the mean, bars
indicate standard
deviation

(Seems that k ~= 7 works best
for this data)

k-NN

For randomly distributed points in high dimensions, distances concentrate within a very small range





Distribution of all pairwise distances between randomly distributed points within **d**-dimensional unit squares, Reference

Theory: Aggarwal et al. 2001, On the Surprising Behavior of Distance Metrics in High Dimensional Space

More intuition: StackExchange article

k-NN
Summary

Advantages

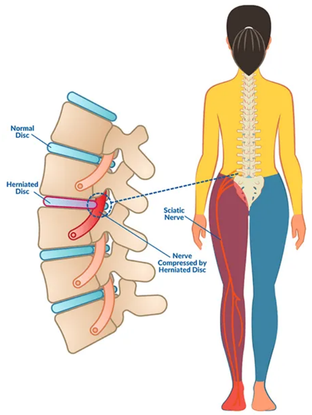
- Easy to implement
- No training required
- New data can be added seamlessly
- Versatile - useful for regression and classification

Disadvantages

- Does not work as well in high dimensions
- Sensitive to noisy data and skewed class distribution
- Requires high memory
- Prediction stage is slow with large data, requires comparison with all samples in dataset

k-Nearest Neighbours in python

- **Dataset:** Biomechanic features of orthopaedic patients, and the type of injury
- **Goal:** Classify the condition of patients based on biomechanic features



	pelvic_tilt	degree_spondylolisthesis	class
0	22.552586	-0.254400	Hernia
1	10.060991	4.564259	Hernia
2	22.218482	-3.530317	Hernia
3	24.652878	11.211523	Hernia
4	9.652075	7.918501	Hernia
...	...	...	...
85	9.906072	20.315315	Spondylolisthesis
86	17.879323	22.123869	Spondylolisthesis
87	10.218996	37.364540	Spondylolisthesis
88	16.800200	24.018575	Spondylolisthesis
89	23.896201	27.283985	Spondylolisthesis

- **Dataset:** Pinguin body features and their specie type
- **Goal:** Classify the specie of a new observed pinguin

	species	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g
0	Chinstrap	49.0	19.5	210.0	3950.0
1	Chinstrap	50.9	19.1	196.0	3550.0
2	Gentoo	42.7	13.7	208.0	3950.0
3	Chinstrap	43.5	18.1	202.0	3400.0
4	Chinstrap	49.8	17.3	198.0	3675.0