Lecture 10

24.11.2025

Today's plan and announcements

- Hour 1:
  - Review of Naive Bases, reinforcement learning, recurrent neural networks
- Hour 2: k-nearest neighbor (k-NN) approach to supervised learning
  - Review of concepts in supervised learning: train/validation/test, hyper parameters
- Announcements
  - Quiz 2: Wednesday 26.11.2025 in exercise hour, see Moodle announcements
  - ~~19.12~~: Lecture on AI in Industry from Swiss Data Science Center representative

last lecture

Review - Naive Bayes classifier

x is categorical

$$P(y = c \mid x = (v_1, \ldots, v_d)) = \frac{\Pi_{j=1}^{d} P(x_j = v_j \mid y = c) P(y = c)}{P(x)}$$

Bayes rule

x is continuous-valued

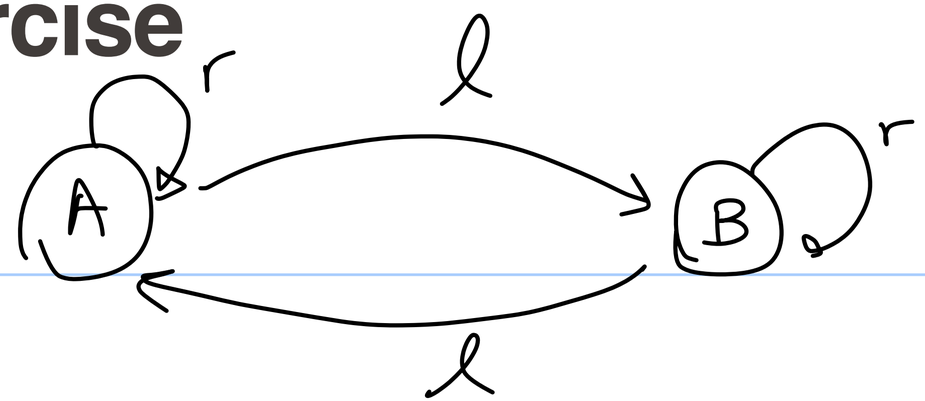
$$P(y = c \mid x) = \frac{f_{x_1 \mid y=c}(x_1) f_{x_2 \mid y=c}(x_2) \ldots f_{x_d \mid y=c}(x_d) P(y = c)}{f_x(x)}$$

- Training is easy, need to compute the right-hand-side probabilities from data
- To determine performance of the classifier, you do the same approach as usual: divide the dataset into training and test set. Use training set to compute the right-hand-side probabilities. Evaluate the performance (confusion matrix, accuracy, ….) on the test set

- Notes
  - no hyper parameters (unless we tune which features to use, …), so we avoid validation set
  - if a feature value v never appears in the training data set $P(x_j = v_j \mid c)$ =0. To address this, we apply

$$P(x_j = v \mid y = c) = \frac{\text{count}(x_j = v, y = c) + 1}{\text{count}(y = c) + |V_j|}$$, where $|V_j|$ is number of possible values feature j takes

Reinforcement learning exercise



- Recall the MDP from lecture 8, with 2 states $\{A, B\}$, and 2 actions $\{r, l\}$. In each state, taking the action r leads to remaining in the state and the action l leads to transitioning to the other state.
- The reward: $r(A, r) = 1, r(A, l) = 0, r(B, r) = 0, r(B, l) = 2,$ and the discount factor: $\gamma = 0.9$.
- The probability of taking action r in state A and B is given by p_A and p_B, respectively. Consider an episode with two time steps ($H = 1$). With an initialization of $p_A = 0.6$ and $p_B = 0.4$, you obtained the 3 trajectories shown in the following table. Estimate the gradient of the objective function with respect to the policy parameters p_A, p_B.

trajectory	Discounted sum of rewards of the trajectory	Gradient at t=0, t=1 with respect to p_A	Gradient at t=0, t=1 with respect to p_B
(A, r, A, r, A)	1 + 0.9x1 = 1.9	$\left(\frac{1}{p_A}, \frac{1}{p_A}\right)$	(0, 0)
(A, r, A, l, B)	$1 + 0.9 \times 0 = 1$	$\left(\frac{1}{p_A}, \frac{-1}{1-p_A}\right)$	(0, 0)
(A, l, B, l, A)	$0 + 0.9 \times 2 = 1.8$	$\left(\frac{-1}{1-p_A}, 0\right)$	$\left(0, \frac{-1}{1-p_B}\right)$

Reinforcement learning exercise solution

Recall: $\nabla_\theta J(\pi_\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=0}^{H}\gamma^t r(s_t^i, a_t^i)\right)\left(\sum_{t=0}^{H}\nabla_\theta \log \pi_\theta(a_t^i \mid s_t^i)\right)$. Since the policy parameters are p_A, p_B, we need to compute

$\sum_{t=0}^{H}\nabla_{p_A}\log\pi_\theta(a_t^i \mid s_t^i)$, and $\sum_{t=0}^{H}\nabla_{p_B}\log\pi_\theta(a_t^i \mid s_t^i)$ and the trajectory rewards. First, compute $\sum_{t=0}^{H}\nabla_{p_s}\log\pi_\theta(a \mid s)$ for actions $a \in \{r, l\}$

and state $s \in \{A, B\}$. Then, evaluate these gradients for a_t, s_t corresponding to the trajectories we have.

parameters $\theta = \{p_A, p_B\}$

1/

$\nabla_{p_A} \log \pi_\theta(a=r \mid s=A) = \nabla_{p_A} \log p_A = \frac{1}{p_A}$

$\nabla_{p_A} \log \pi_\theta(a=l \mid s=A) = \nabla_{p_A} \log(1-p_A) = \frac{-1}{1-p_A}$

$\nabla_{p_B} \log \pi_\theta(a \mid s=A) = 0$, for $a \in \{l, r\}$

$\nabla_{p_B} \log \pi_\theta(a=r \mid s=B) = \nabla_{p_B} \log p_B = \frac{1}{p_B}$

$\nabla_{p_B} \log \pi_\theta(a=l \mid s=B) = \nabla_{p_B} \log(1-p_B) = \frac{-1}{1-p_B}$

$\nabla_{p_A} \log \pi_\theta(a \mid s=B) = 0$, for $a \in \{l, r\}$

2/

plug in ★

$\nabla_{p_A} J(\pi_\theta) = \frac{1}{3}\left(1.9 \times \left(\frac{1}{p_A} + \frac{1}{p_A}\right) + 1 \times \left(\frac{1}{p_A} - \frac{1}{1-p_A}\right) + 1.8\left(\frac{-1}{1-p_A}\right) = 1\right.$

$\nabla_{p_B} J(\pi_\theta) = \frac{1}{3}\left(1.9 \times 0 + 1 \times 0 + 1.8 \times \frac{-1}{1-p_B}\right) \approx -2.6$

Reinforcement learning exercise solution

- Recall: $\nabla_\theta J(\pi_\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=0}^{H}\gamma^t r(s_t^i, a_t^i)\right)\left(\sum_{t=0}^{H}\nabla_\theta \log \pi_\theta(a_t^i \mid s_t^i)\right)$. Since the policy parameters are p_A, p_B, we need to compute

$\sum_{t=0}^{H}\nabla_{p_A}\log\pi_\theta(a_t^i \mid s_t^i)$, and $\sum_{t=0}^{H}\nabla_{p_B}\log\pi_\theta(a_t^i \mid s_t^i)$ and the trajectory rewards. First, compute $\sum_{t=0}^{H}\nabla_{p_s}\log\pi_\theta(a \mid s)$ for actions $a \in \{r, l\}$ and state $s \in \{A, B\}$. Then, evaluate these gradients for a_t, s_t corresponding to the trajectories we have.

What's the eshmated expected discounted reward ?

1) Expected discounted reward: $E\sum_{t=0}^{1}\gamma^t r(s_t, a_t)$

2) eshmaton from trajectories: $\{(s_t^i, a_t^i)\}_{i=1}^{N}$

$$\frac{1}{N}\sum_{t=0}^{1}\gamma^t(s_t^i, a_t^i) = \frac{1}{3}(1.9 + 1.0 + 1.8) = \frac{1.7}{3}$$

Recurrent neural network exercise

We aim to predict whether renewable energy supply will be greater or smaller than power demand for a given time period, given past data on demand d_t, supply r_t and temperature w_t and energy prices p_t. To this end, we will build a recurrent neural network with 5 hidden states and $\tanh$ activation function for the hidden states, and a single output indicating probability of the energy supply being greater than the demand.

- Write the resulting recurrent neural network
- Using historical data $\{d_t, r_t, w_t, p_t\}_{t=0}^{T-1}$, write the binary cross entropy loss to train the network.

Recurrent neural network exercise solution

- Write the resulting recurrent neural network

$s_t \in \mathbb{R}^S$, $a_t = \begin{bmatrix} d_t \\ r_t \\ w_t \\ p_t \end{bmatrix} \in \mathbb{R}^4$

$$s_{t+1} = \tanh(W_{ss} s_t + W_{sa} a_t + b_s),$$

$$\hat{y}_t = \sigma(W_{os} s_t + b_o)$$

↳ sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$

$$y_t = \begin{cases} 1 & d_t > r_t \\ 0 & \text{else} \end{cases}$$

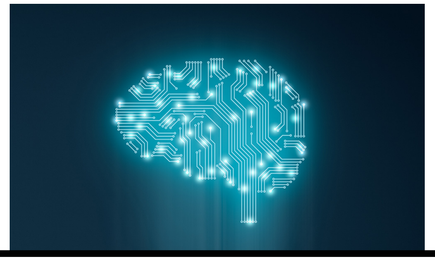
$$\theta = \{ W_{ss} \in \mathbb{R}^{S \times S},\ W_{sa} \in \mathbb{R}^{S \times 4},\ b_s \in \mathbb{R}^S,\ W_{os} \in \mathbb{R}^{1 \times S},\ b_o \in \mathbb{R} \}$$

- Using historical data $\{d_t, r_t, w_t, p_t\}_{t=0}^{T-1}$, write the binary cross entropy loss to train the network.

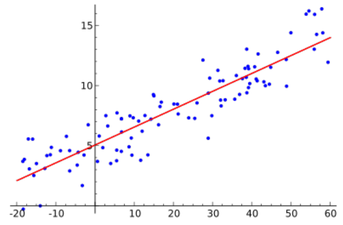
$$L(\theta) = \frac{-1}{T} \sum_{t=0}^{T-1} y_t \log \hat{y}_t + (1-y_t) \log(1-\hat{y}_t)$$

Note: hidden state in RNN is not the same as hidden layer in a deep neural network

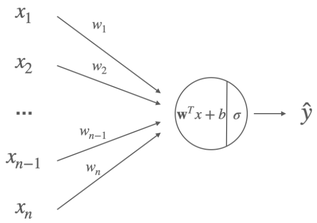
Introduction



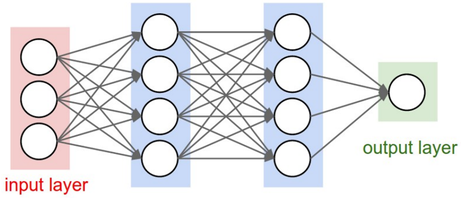
Linear regression



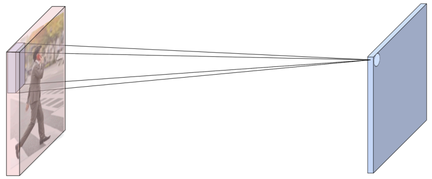
Logistic regression



Neural networks

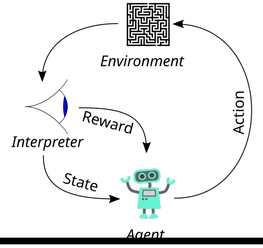


Convolutional neural networks

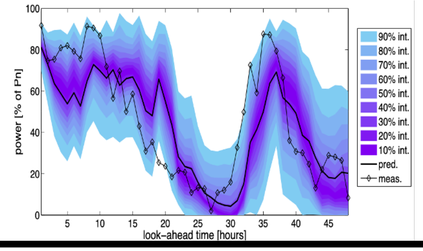


AI & sustainability

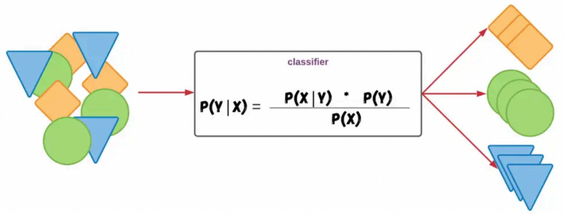
Reinforcement learning



Recurrent neural networks

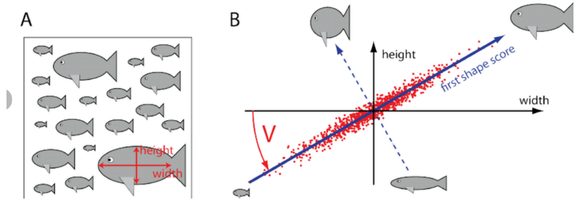


Naive Bayes

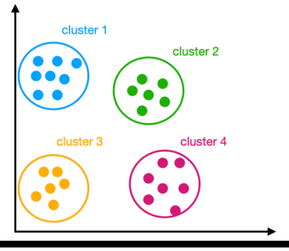


KNN

Dimensionality reduction



Clustering



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.

Recall terminology:

- **Features:** input variables
- **Label:** what we are predicting

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

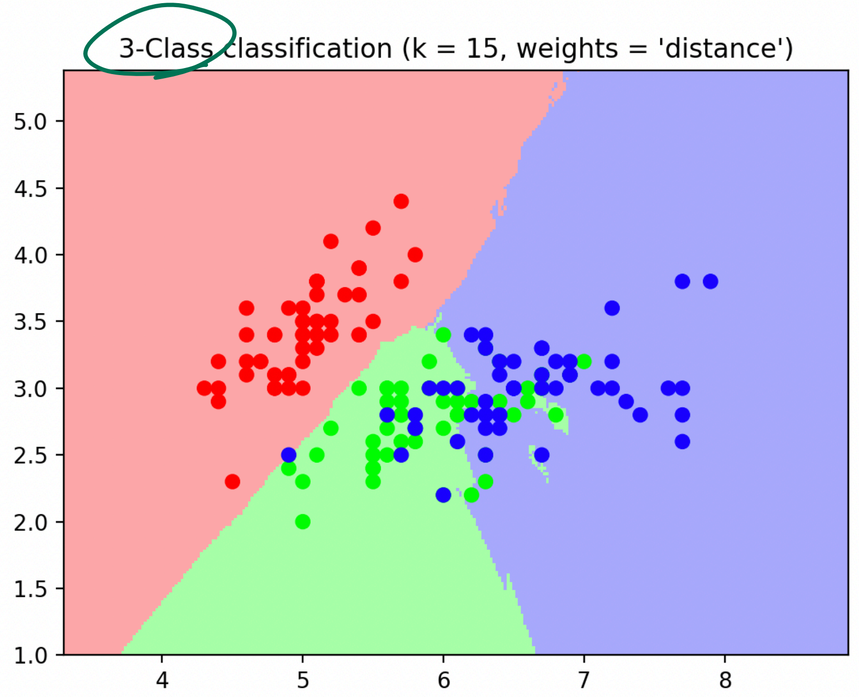
$y \in \mathbb{R}^m$ For regression.

$y \in \{1, 2, \cdots, C\}$ For classification within C possible classes

k-NN for classification

Problem: $\{x^i, y^i\}_{i=1}^N$, $y^i \in \{1, 2, \dots, C\}$

- k-Nearest Neighbors (k-NN) classifier 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 closest neighbors:
  - example: k = 1, 3, 5, … for binary classification
- Creates nonlinear decision boundaries
  - Contrast with linear boundaries (without feature engineering) in logistic regression



https://stackoverflow.com/questions/45075638/graph-k-nn-decision-boundaries-in-matplotlib

k-NN Distance Metric

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

Examples of distance metrics:

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

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

$x^i = (x^i_1, x^i_2, \ldots, x^i_d), \; i = 1, 2$

Euclidean (l_2) distance

$$\| x^1 - x^2 \|_2 = \left[(x^1_1 - x^2_1)^2 + (x^1_2 - x^2_2)^2 + \cdots + (x^1_d - x^2_d)^2 \right]^{1/2}$$

Manhattan (l_1) distance

$$\| x^1 - x^2 \|_1 = |x^1_1 - x^2_1| + |x^1_2 - x^2_1| + \cdots + |x^1_d - x^2_{d1}|$$

k-NN Distance Metric

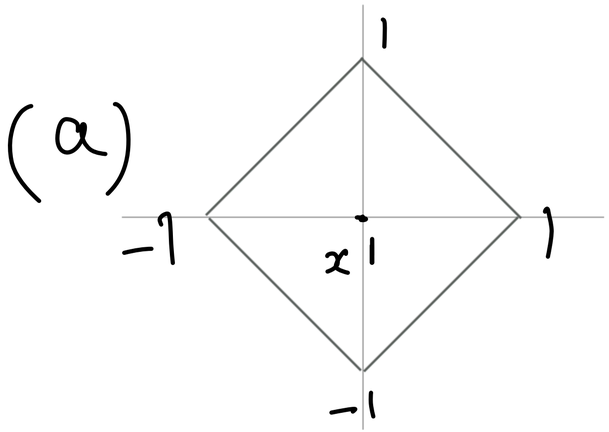
How to measure the proximity between data points? → measure distance

example: (a) l_1 distance $\{ x \in \mathbb{R}^2 \mid |x_1 - x_1'| + |x_2 - x_2'| = 1 \}$

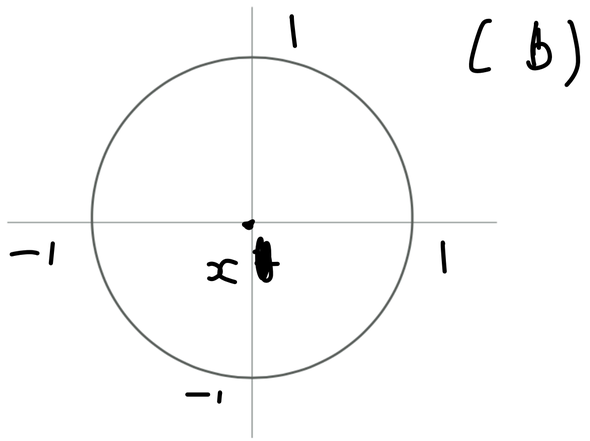
l_2 distance $\{ x \in \mathbb{R}^2 \mid (x_1 - x_1')^2 + (x_2 - x_2')^2 = 1 \}$

Level sets of the two most common distances $\{x \in \mathbb{R}^2 \mid \|x - x_0'\|_{l_i} = 1\}$, for $i = 1,2$

l_1 (Manhattan) distance

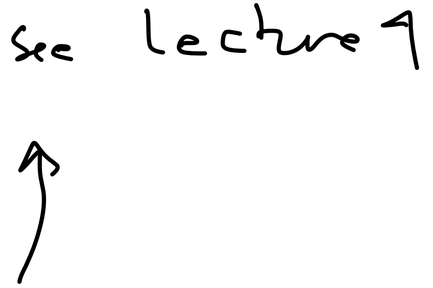


l_2 (Euclidean) distance



Data normalization/standardization

- k-NN is distance-based, so feature scale matters
- Larger-scale features dominate distances if data isn't scaled.
- Scaling ensures fair contribution of the features to the machine learning task



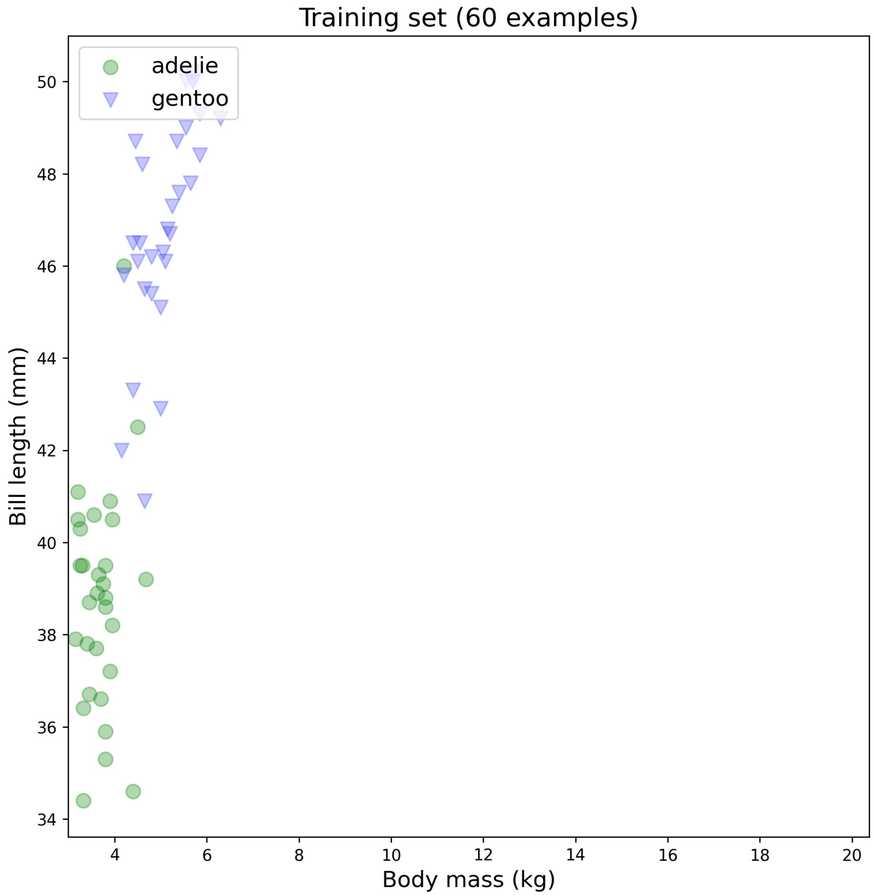
- **Z-Score Standardization:** for each feature, it sets mean (μ) to 0, standard deviation (σ) to 1

- Alternative approach: min-max scaling (see lecture 4) - might be more sensitive to outliers

- Feature scaling is also useful for models using gradient descent: linear regression, logistic regression, neural networks
  - Without scaling, gradients in large-scale feature directions become huge, forcing the use of very small learning rates to avoid overshooting, which slows convergence

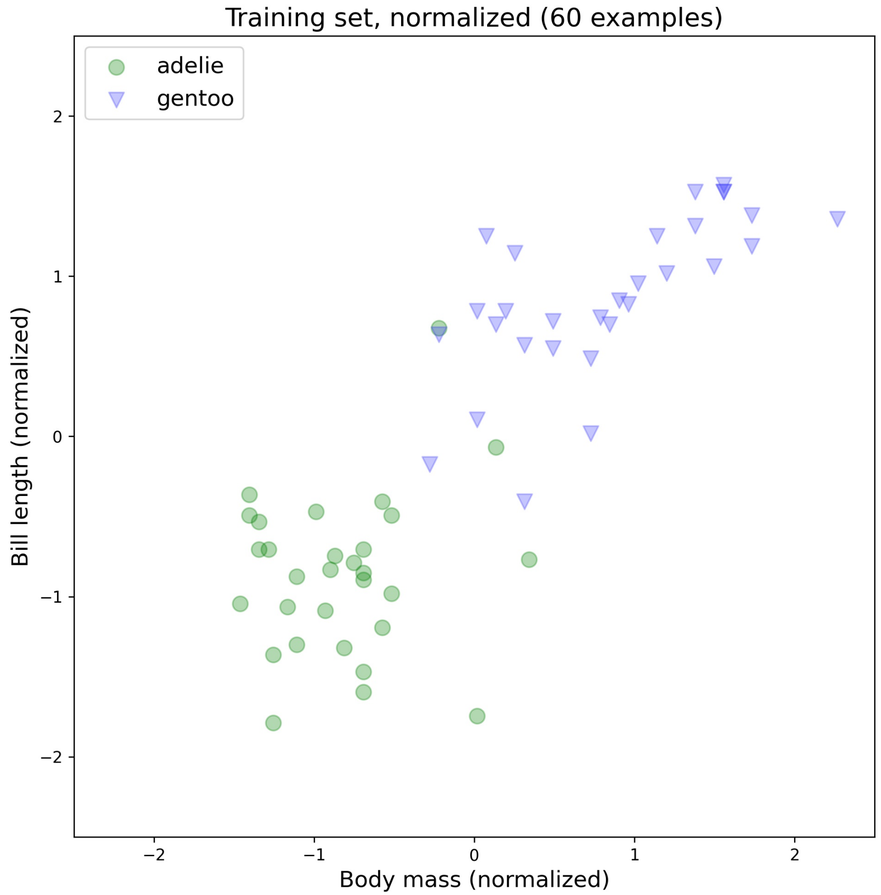
k-NN

Feature scaling - Visualisation



Normalize

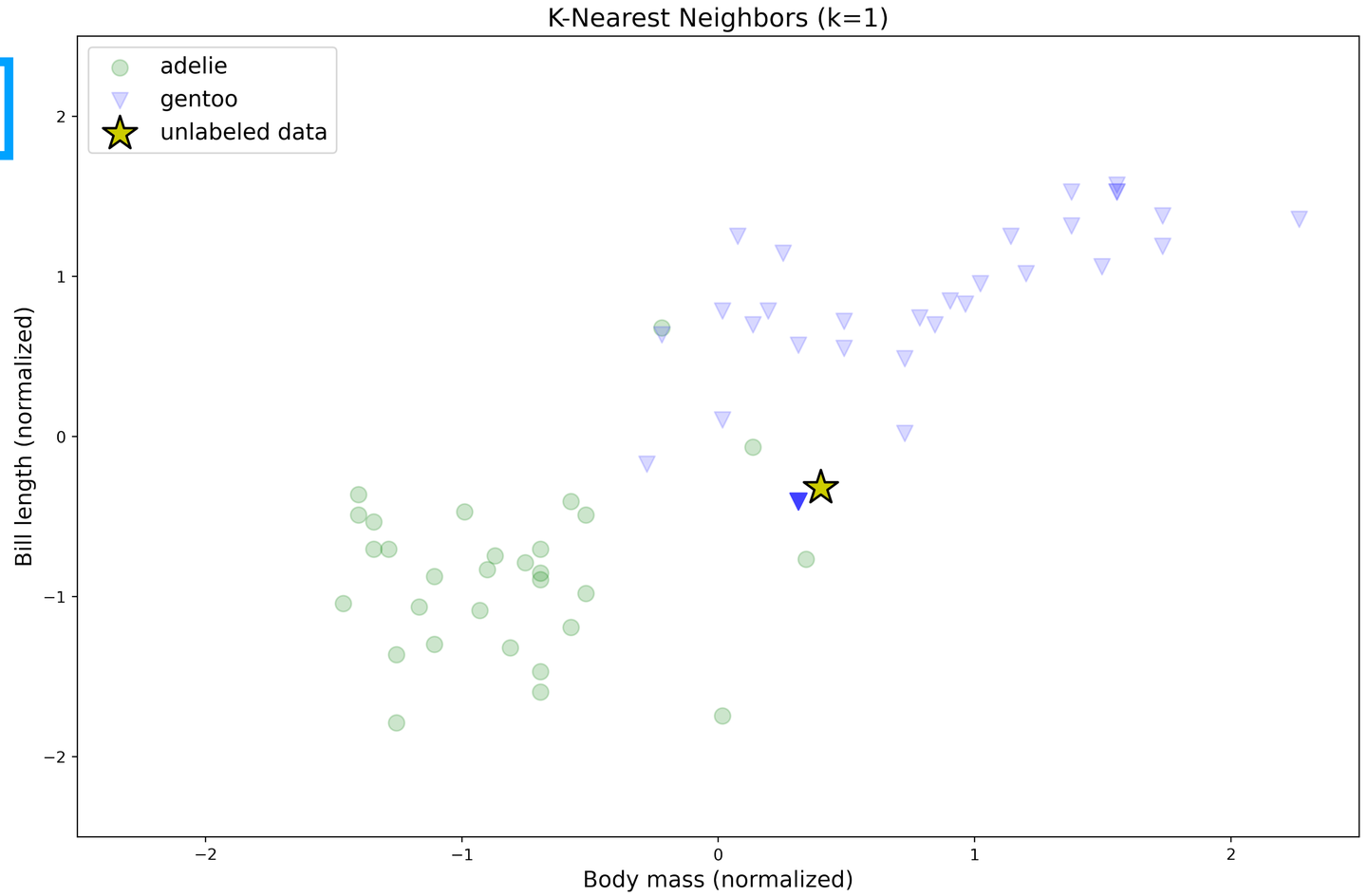
check approach in python code



k-NN
Visualisation

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

k=1



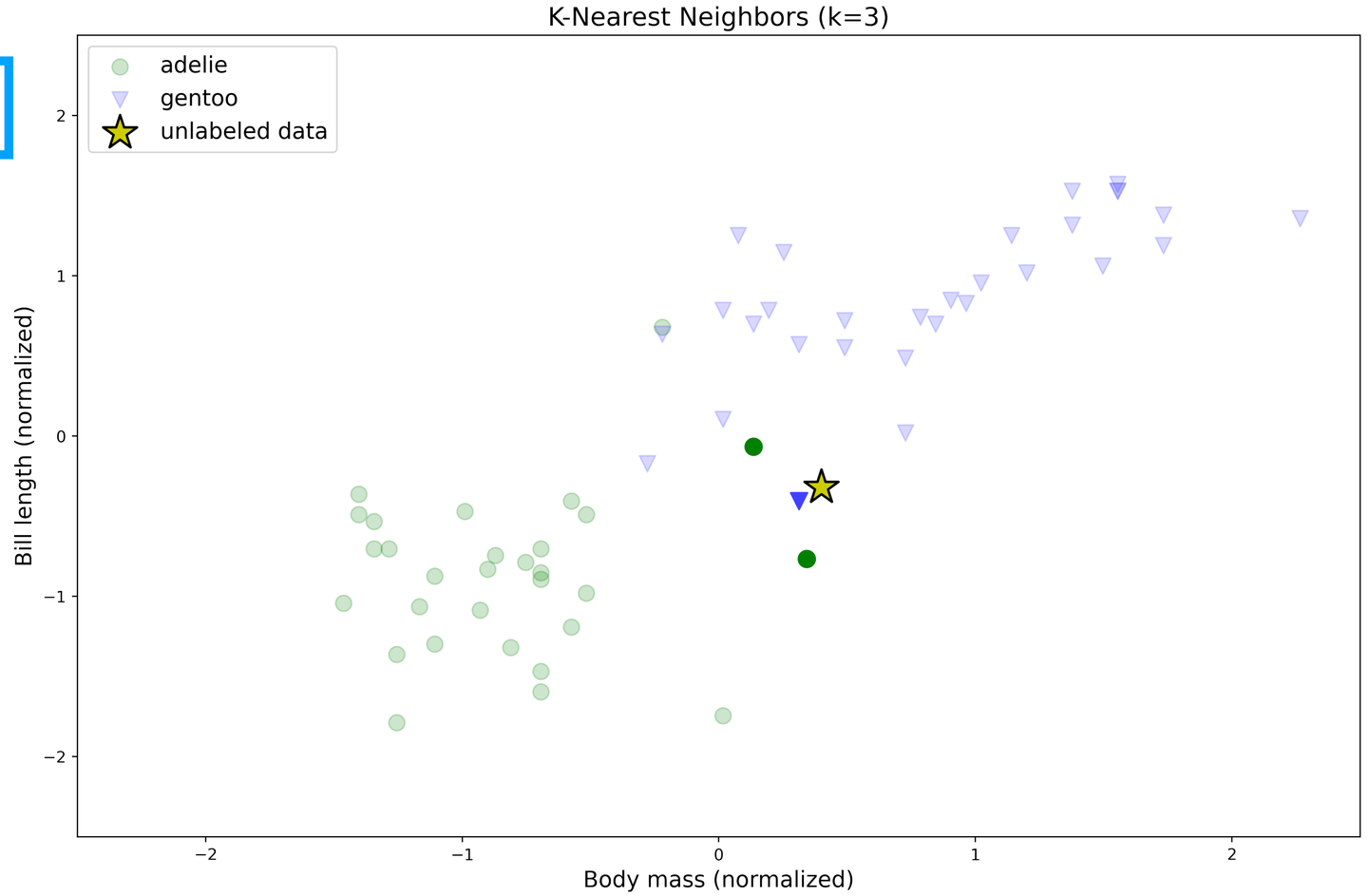
In which category belongs the ★ point? → **Gentoo**

k-NN

$k > 1$

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

k=3



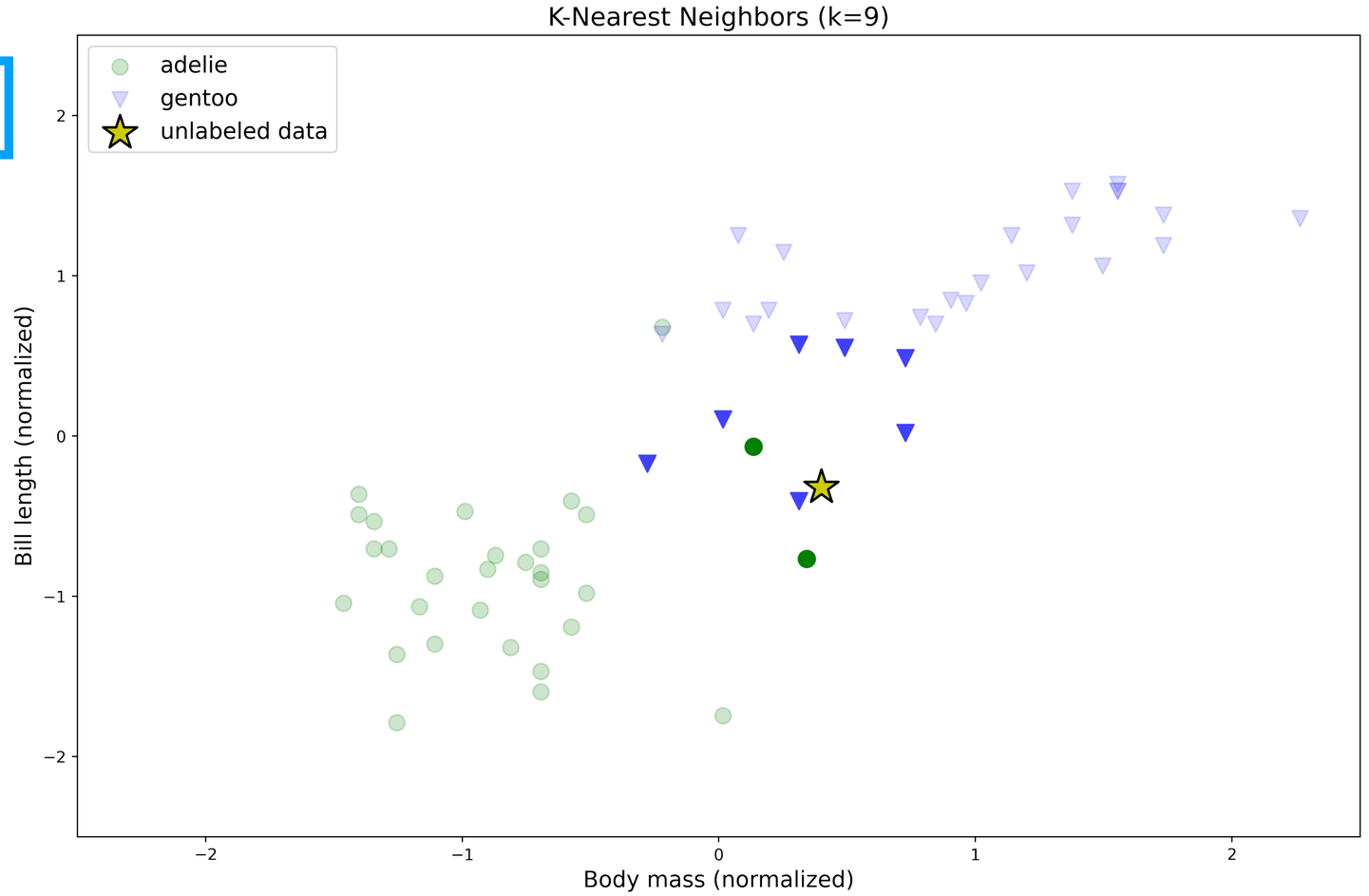
In which category belongs the ★ point? → **Adélie**

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?

- A language-independent description of an algorithm
- Focuses on what the algorithm does, not on programming syntax
- Written before implementation—in Python, MATLAB, C, etc.
- Helps clarify logic and structure; if you can express it in pseudocode, you understand the algorithmIn
- In this course, you don’t need exact Python syntax, only the ability to follow and write pseudocode

kNN pseudo-code

k (how many neighbors, which distance metric)

- kNN setting hyper-parameters given data set $\{x^i, y^i\}_{i=1}^N$
  - Split data into training, validation and test sets
  - For each candidate k and distance metric:
  - For every validation point:
    - compute distances to all training points
    - find the k nearest neighbors
    - predict using majority vote
  - compute validation error
  - Choose the k and distance metric with the lowest validation error

- kNN prediction of label for a new query point $x \in \mathbb{R}^d$
  - Compute the distance of x to all training points $\{x^i\}_{i=1}^N$
  - Select the k closest points
  - Look up their labels
  - Predict by majority vote

$\hat{y}$ (label of x) is the majority vote of labels of closest neighbors

kNN performance evaluation

- After you set the hyper parameters using the validation set
- Evaluate the final k-NN model on the test set to estimate its accuracy on unseen data (recall accuracy metrics for a classification problem from Lecture 3)
- k-NN does not learn explicit model parameters; it stores the training data and predicts using all training data
  - Contrast with other approaches we have looked at so far, where training data was just used for learning a model parameter. The model then was used for prediction

- Note: kNN can also be used for regression: predict label of a given point using the mean of the label of the k nearest neighbors

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
```

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?

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

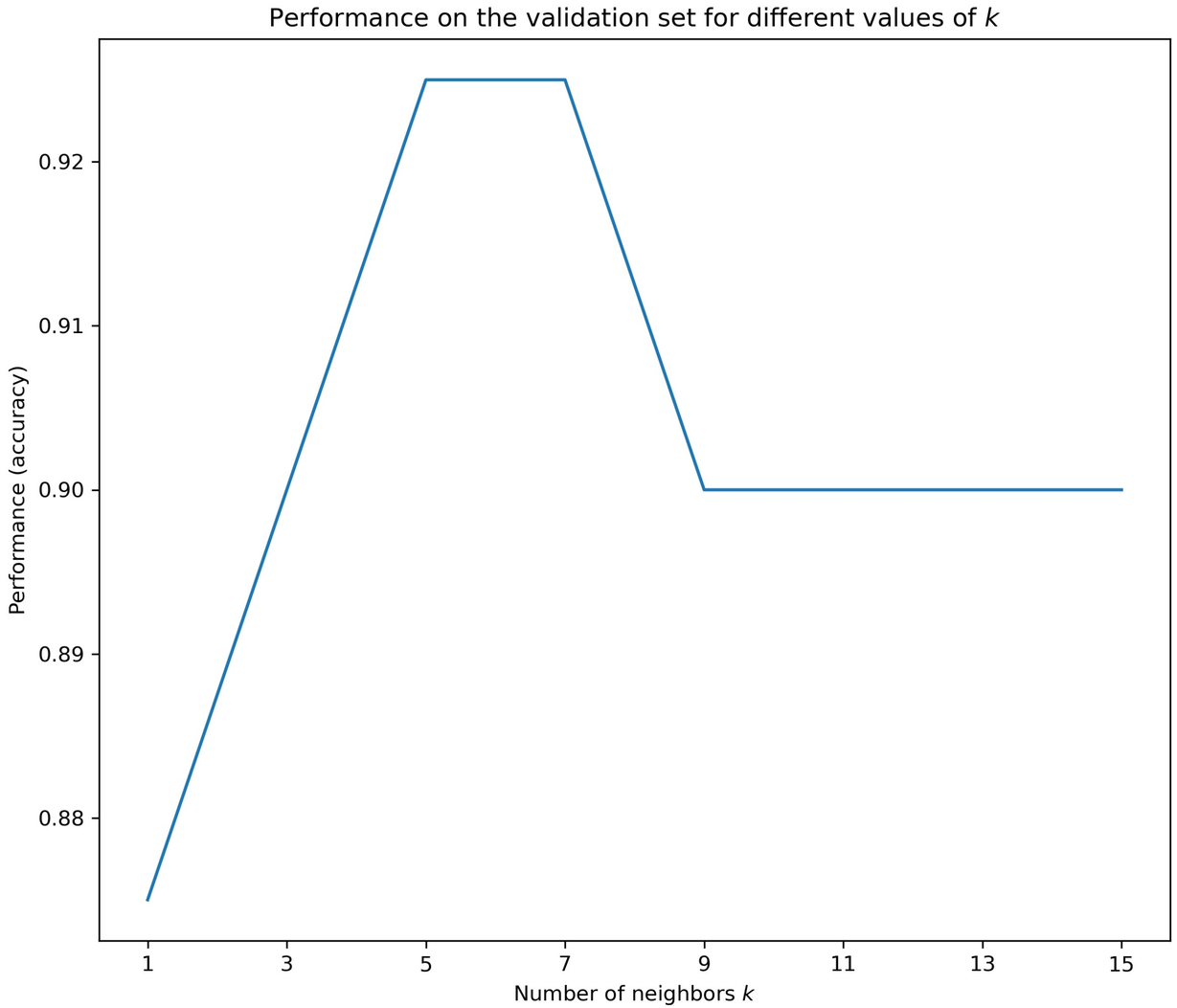
What is the best value of **k** to use ?

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

These are **hyper parameters**: 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



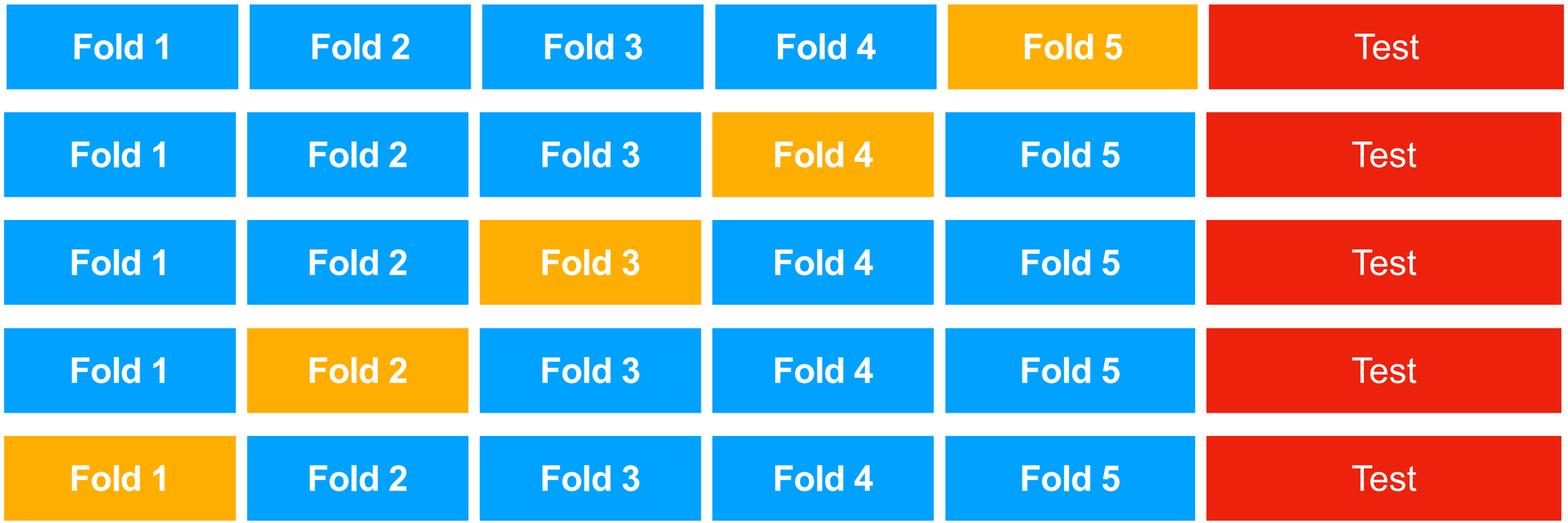
Validation set accuracy for different values of **k** for our penguin classification task

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

Train, validate, test in ML (Review)

Your Dataset

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



Useful for small datasets

Cross validation pseudo-code

How to choose k (similarly for distance metric)

- Split the training data into 5 folds
- Choose a candidate value of k
- Train on 4 folds and validate on the remaining fold
- Repeat so each fold serves once as validation
- Average the 5 validation accuracies for this k
- Repeat steps 2–5 for other candidate k values
- Select the k with the highest average validation accuracy
  - If averages are close, use variance across the folds to choose the k
  - Example:

- Retrain using all training data with the chosen k, then evaluate on the test set

k-Nearest Neighbours in python

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

label $y \in \{\text{Chinstrap}, \text{Gentoo}, \text{Adelie}\}$ features $x \in \mathbb{R}^4$

	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

Summary

- k-NN is a supervised learning approach
  - Can be used for classification or regression
  - No parameter to train, just hyper parameters to set
  - Large memory and time needed for prediction (inference) (store all data points and evaluate distance to the query point)
  - Not very effective in large dimensions (see discussion here)
- Next week, we move on to unsupervised learning

- Your tasks this week
  - Quiz 2
  - Python example (you just need to go through the code)