

Lecture 1: Introduction & KNNs

Jikael Gagnon (Based on slides by Sheheryar Parvaz and Shidan Javaheri)

McGill Artificial Intelligence Society

Winter 2026



Who are we?

lecturers

lecturers

Ritchie

- From Calgary, Alberta
- Was once a MAIS student too
- Enjoys badminton

lecturers

Ritchie

- From Calgary, Alberta
- Was once a MAIS student too
- Enjoys badminton

Jikael

- Graduated Fall 2025
- Starting master's in AI for healthcare at UofT
- Once met Christian Bale (American Psycho) at a urinal

team

<https://mcgillai.com/team/2025-2026>

technical project managers (TPMs)

- Antoine Dulauroy - antoine.dulauroy@mail.mcgill.ca
- Emma Kondrup - emma.kondrup@mail.mcgill.ca
- Mikael D'Agostino - mikael.dagostino@mail.mcgill.ca
- Isaac Hu - isaac.hu@mail.mcgill.ca

goals

- Develop an intuition for fundamental concepts in machine learning
- Ease the transition to more advanced classes
- Learn to use standard tools
- Make friends

non-goals

We prioritize intuition over mathematical rigor. Consider the more advanced classes at McGill (such as COMP 451 or COMP551) for a more complete overview.

class logistics

class logistics

- **8 mandatory lectures** covering theory
 - ▶ 2 hours per week

class logistics

- **8 mandatory lectures** covering theory
 - ▶ 2 hours per week
- **3 assignments**, covering the practical components
 - ▶ 3-6 hours per week

class logistics

- **8 mandatory lectures** covering theory
 - ▶ 2 hours per week
- **3 assignments**, covering the practical components
 - ▶ 3-6 hours per week
- **a final project**, consisting of **4 deliverables** and a presentation

class logistics

- **8 mandatory lectures** covering theory
 - ▶ 2 hours per week
- **3 assignments**, covering the practical components
 - ▶ 3-6 hours per week
- **a final project**, consisting of **4 deliverables** and a presentation

Make sure you've joined **Slack** and **MyCourses**. You can find the syllabus and roadmap under “Content”.

announcements

- First deliverable for the final project will be uploaded tonight (due Feb 17th).
- Next lecture is in Leacock 109!

What is “machine learning”?

A model is said to learn from experience for some task, if its performance at the task improves with experience.

~ paraphrased from Tom Mitchell's "Machine Learning"

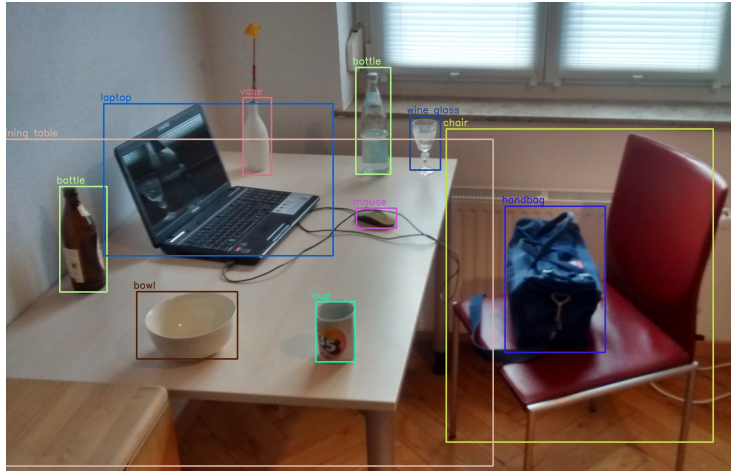
A **model** is said to learn from **experience** for some **task**, if its **performance** at the **task** improves with **experience**.

examples of **tasks**

image classification



object detection



language modelling

 How do I introduce myself in Mandarin?

 Introducing yourself in Mandarin Chinese typically involves providing your name and a few basic details about yourself. Here's a simple self-introduction in Mandarin:

1. Start with a greeting:
 - 你好 (Nǐ hǎo) - Hello
2. State your name:
 - 我叫 [Your Name] (Wǒ jiào [Your Name]) - My name is [Your Name]
3. Mention where you're from:
 - 我来自 [Your City or Country] (Wǒ láizì [Your City or Country]) - I am from [Your City or Country]
4. Optionally, you can include your profession or occupation:
 - 我是 [Your Occupation] (Wǒ shì [Your Occupation]) - I am a [Your Occupation]

Here's an example:
你好，我叫小明。我来自美国。我是一名学生。
(Nǐ hǎo, wǒ jiào Xiǎo Míng. Wǒ láizì Měiguó. Wǒ shì yī míng xuéshēng.)
Hello, my name is Xiao Ming. I am from the United States. I am a student.

Remember to adjust the introduction according to your own name, origin, and occupation. Additionally, it's a good idea to practice the pronunciation of these phrases to ensure you're understood correctly.

 Regenerate

image captioning

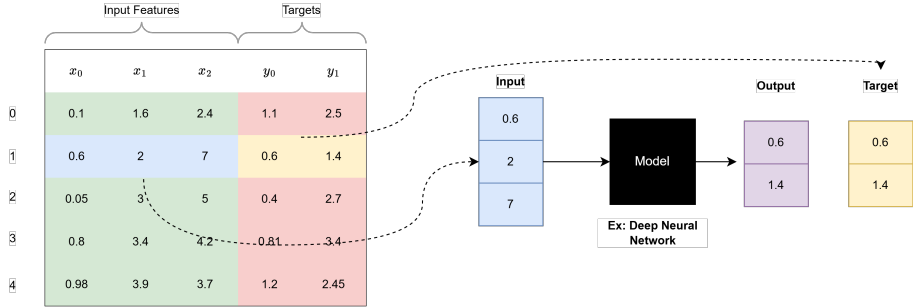
Oxford-102 Flowers	This flower has petals that are red with yellow tips.	The flower shown has white petals with yellow anther in the center.	Purple petals that are almost heart shaped with small green receptacles.	This flower has petals that are yellow with many layers.
				
	This bird is blue with white on the head. the feathers are blue and the belly is white.	This bird is completely black with a large blunt beak.	This is a bird with a green wing, a brown head and a red bill.	A bird with long black wings, tail and white breast, the bill is short and black.
				
CUB-200 Birds	Two people holding snowboards are standing in the snow.	Lunch of rice and beans with soup and juice.	A big rig truck in a parking lot without a trailer.	A group of men traveling on horses in the water.
				
COCO				

supervised learning (at a glance)

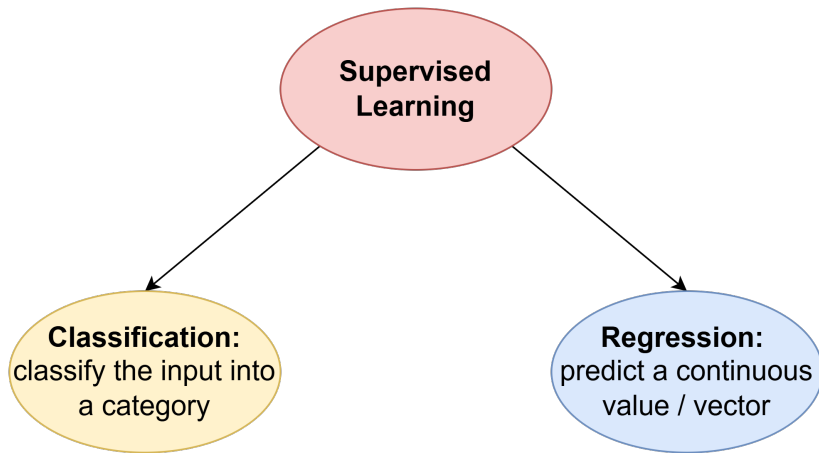
supervised learning (at a glance)
given a set of **features**, predict the **target**

Features and targets can be vectors or scalars:

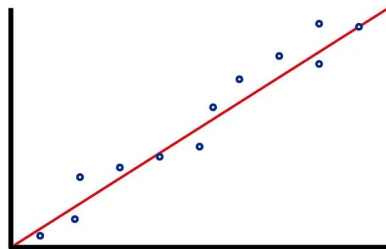
	Input Features			Targets	
	x_0	x_1	x_2	y_0	y_1
0	0.1	1.6	2.4	1.1	2.5
1	0.6	2	7	0.6	1.4
2	0.05	3	5	0.4	2.7
3	0.8	3.4	4.2	0.81	3.4
4	0.98	3.9	3.7	1.2	2.45



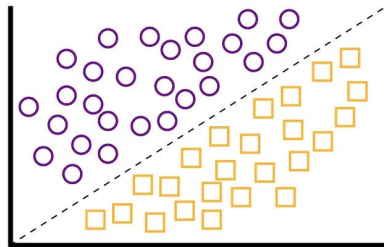
Two Types of Supervised Learning



Regression vs. Classification



Regression



Classification

classification (example #1)

General Task: Predict some discrete **label** from a set of **features**.

Example: Predict whether a patient has diabetes (label) based on blood pressure, BMI, and age (features).

classification (example #1)

General Task: Predict some discrete **label** from a set of **features**.

Example: Predict whether a patient has diabetes (label) based on blood pressure, BMI, and age (features).

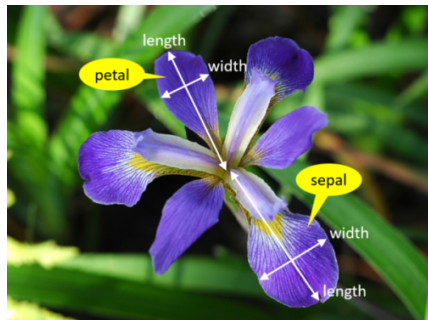
Dataset

#	Blood Pressure (x_0)	BMI (x_1)	Age (x_2)	label (y)
1	72	33.6	50	<i>Diabetes</i>
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
101	66	26.6	31	<i>No Diabetes</i>
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

classification (example # 2)

General Task: Predict some discrete **label** from a set of **features**.

Example: Predict species of Iris flower (label) based on a set of measurements (features).



classification (example # 2)

General Task: Predict some discrete **label** from a set of **features**.

Example: Predict species of Iris flower (label) based on a set of measurements (features).

#	sepal length (x_0)	sepal width (x_1)	petal length (x_2)	petal width (x_3)	label (y)
1	5.1	3.5	1.4	0.2	<i>I. setosa</i>
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
101	6.3	3.3	6.0	2.5	<i>I. virginica</i>
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$



(a) *Iris setosa*



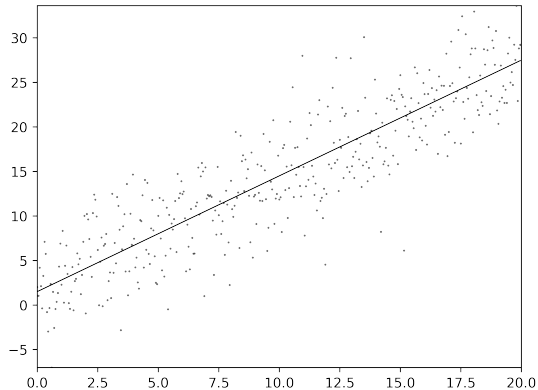
(b) *Iris versicolor*



(c) *Iris virginica*

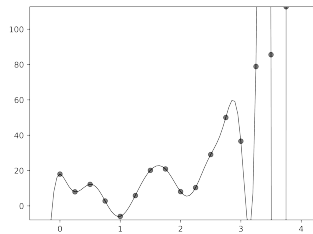
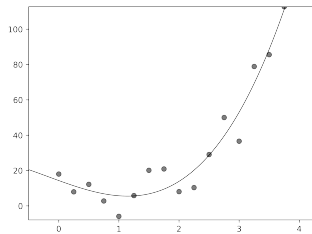
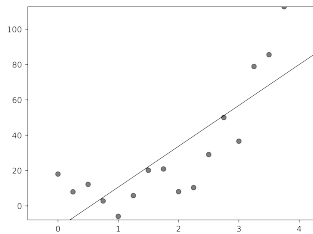
regression

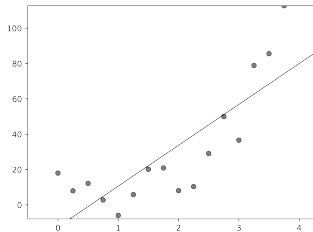
Predict a **continuous value** (target) from the **features**.



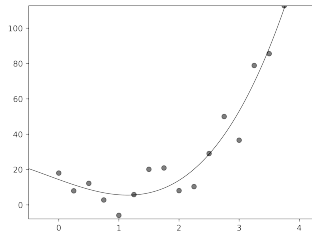
regression (example dataset)

#	med. income (x_0)	house age (x_1)	...	pop. (x_4)	median price (y)
1	8.3252	41.		322.	4.562
2	8.3014	21.	...	2401.	3.585
$\vdots$	$\vdots$	...	$\vdots$	$\vdots$	
100	7.2574	52.		496.	3.422

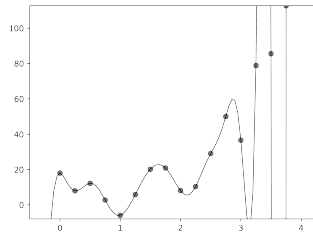




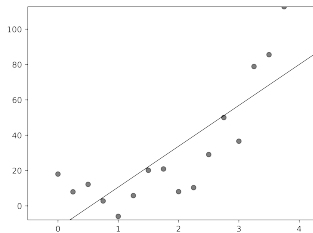
(a) underfit



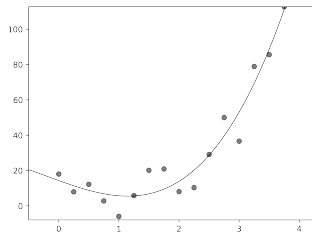
(b) “good” fit



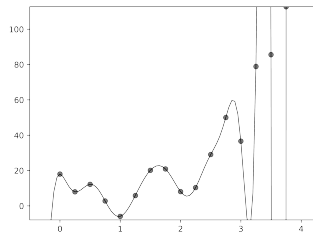
(c) overfit



(a) degree 1



(b) degree 3



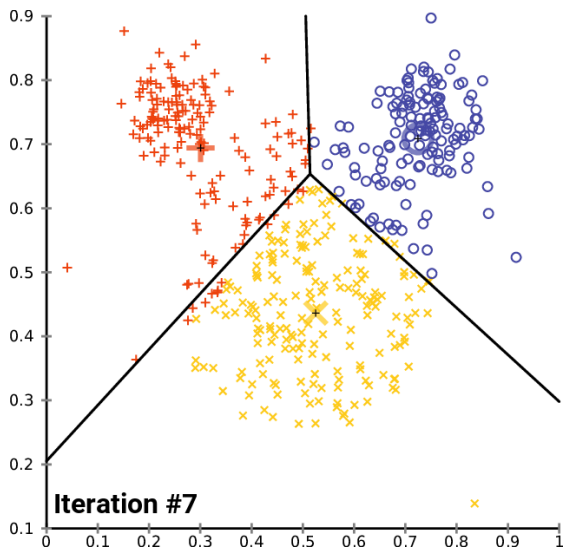
(c) degree 20

unsupervised learning (at a glance)

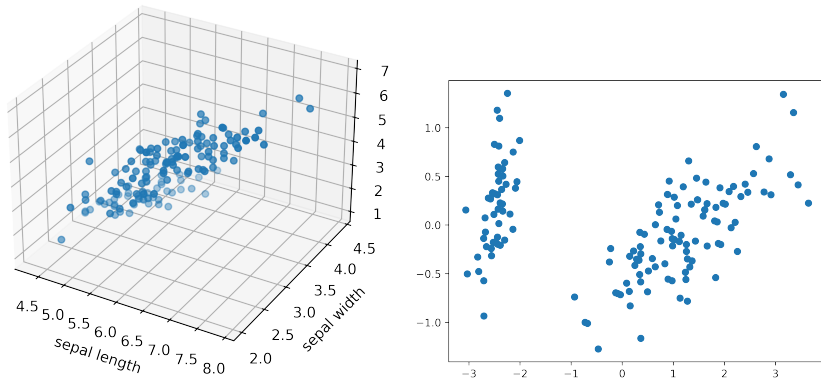
unsupervised learning (at a glance)

extracting patterns from data

clustering



dimensionality reduction



break time, go make friends

Our first “**model**”

Our first “**model**”, is the k -nearest neighbours algorithm

Our first “**model**”, is the k -nearest neighbours algorithm,
a **supervised-learning** algorithm for both **classification** and **regression**.

Notation

Input Features			Targets			
	x_0	x_1	x_2	y_0	y_1	
$\vec{x}^{(0)}$	0.1	1.6	2.4	1.1	2.5	$\vec{y}^{(0)}$
$\vec{x}^{(1)}$	0.6	2	7	0.6	1.4	$\vec{y}^{(1)}$
$\vec{x}^{(2)}$	0.05	3	5	0.4	2.7	$\vec{y}^{(2)}$
$\vec{x}^{(3)}$	0.8	3.4	4.2	0.81	3.4	$\vec{y}^{(3)}$
$\vec{x}^{(4)}$	0.98	3.9	3.7	1.2	2.45	$\vec{y}^{(4)}$

Notation

We denote our dataset as a set of pairs of *features* and *labels / targets*

$$\mathcal{D} = \{(\vec{x}^{(1)}, \vec{y}^{(1)}), (\vec{x}^{(2)}, \vec{y}^{(2)}), \dots, (\vec{x}^{(N)}, \vec{y}^{(N)})\}$$

where N is the number of pairs in the dataset.

Notation

We denote our dataset as a set of pairs of *features* and *labels / targets*

$$\mathcal{D} = \{(\vec{x}^{(1)}, \vec{y}^{(1)}), (\vec{x}^{(2)}, \vec{y}^{(2)}), \dots, (\vec{x}^{(N)}, \vec{y}^{(N)})\}$$

where N is the number of pairs in the dataset.

For some $1 \leq i \leq N$,

$$\vec{x}^{(i)} = (x_0^{(i)}, x_1^{(i)}, x_2^{(i)}, \dots, x_n^{(i)})$$

are the features of instance i

Notation

We denote our dataset as a set of pairs of *features* and *labels / targets*

$$\mathcal{D} = \{(\vec{x}^{(1)}, \vec{y}^{(1)}), (\vec{x}^{(2)}, \vec{y}^{(2)}), \dots, (\vec{x}^{(N)}, \vec{y}^{(N)})\}$$

where N is the number of pairs in the dataset.

For some $1 \leq i \leq N$,

$$\vec{x}^{(i)} = (x_0^{(i)}, x_1^{(i)}, x_2^{(i)}, \dots, x_n^{(i)})$$

are the features of instance i

$$\vec{y}^{(i)} = (y_0^{(i)}, y_1^{(i)}, y_2^{(i)}, \dots, y_m^{(i)})$$

are the labels / targets of instance i .

Toy Example: KNN for Classification

To motivate the KNN model, we will use the following dataset containing measurements of apples and clementines:

#	Diameter (x_0)	Weight (x_1)	label (y)
1	3.2	7.9	<i>Clementine</i>
2	6.4	10.1	<i>Apple</i>
$\vdots$	$\vdots$	$\vdots$	$\vdots$
30	5.3	12.2	<i>Apple</i>

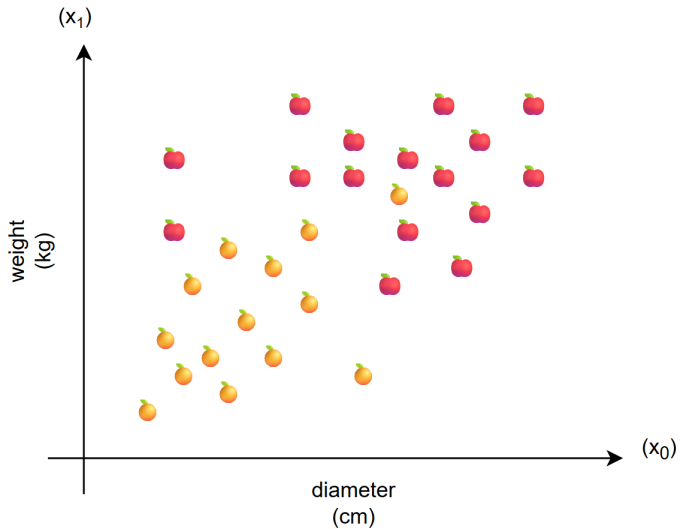
Toy Example: KNN for Classification

To motivate the KNN model, we will use the following dataset containing measurements of apples and clementines:

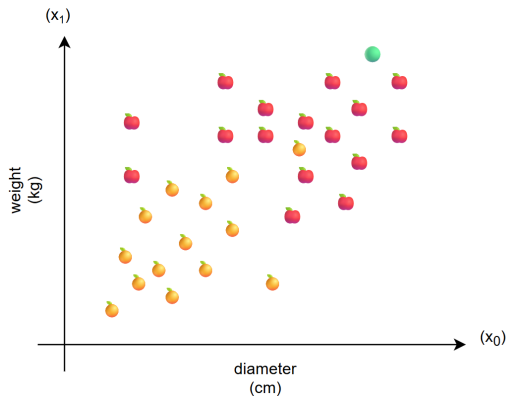
#	Diameter (x_0)	Weight (x_1)	label (y)
1	3.2	7.9	<i>Clementine</i>
2	6.4	10.1	<i>Apple</i>
$\vdots$	$\vdots$	$\vdots$	$\vdots$
30	5.3	12.2	<i>Apple</i>

Test Yourself: What is N ?

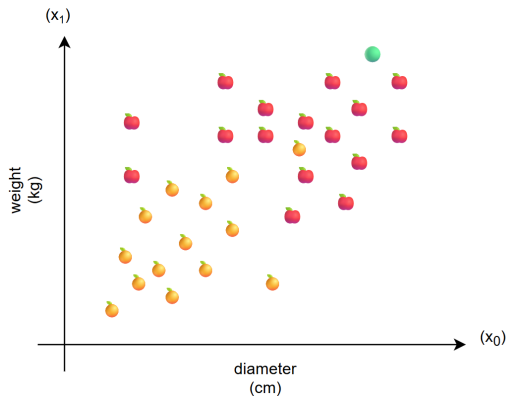
Let's plot this and see what it looks like:



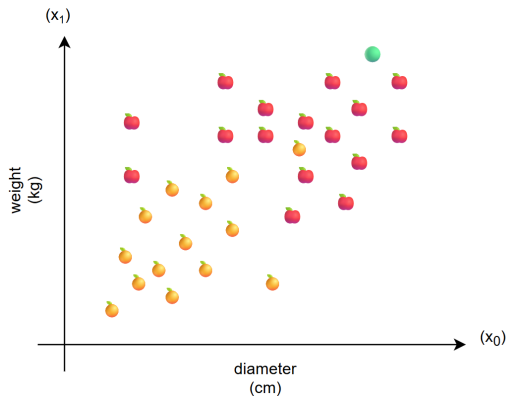
Now suppose we are given the measurements of some new mystery fruit (green circle). Using our existing data, how could we classify it?



Now suppose we are given the measurements of some new mystery fruit (green circle). Using our existing data, how could we classify it?



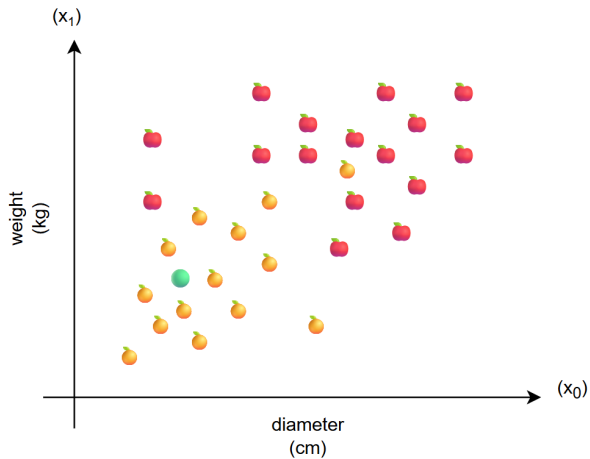
Now suppose we are given the measurements of some new mystery fruit (green circle). Using our existing data, how could we classify it?



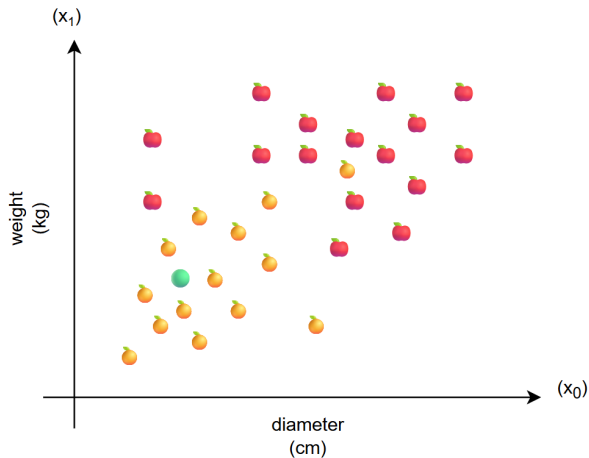
Notice: The measurements of our mystery fruit are closer to those of apples than clementines.

Idea: Classify mystery fruit using the class of the *nearest neighbour* (ie. the closest point in our dataset).

Using our strategy, classify this mystery fruit:

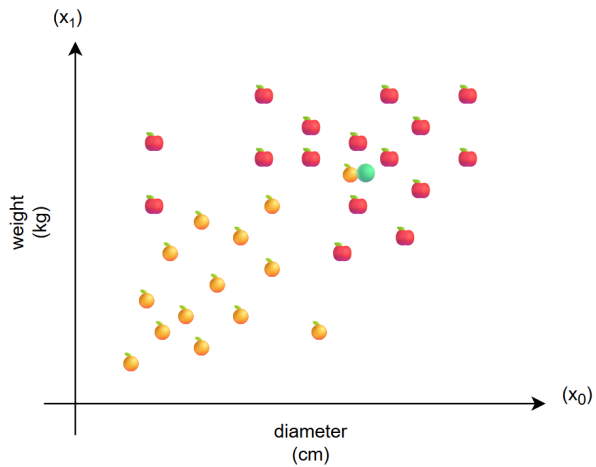


Using our strategy, classify this mystery fruit:

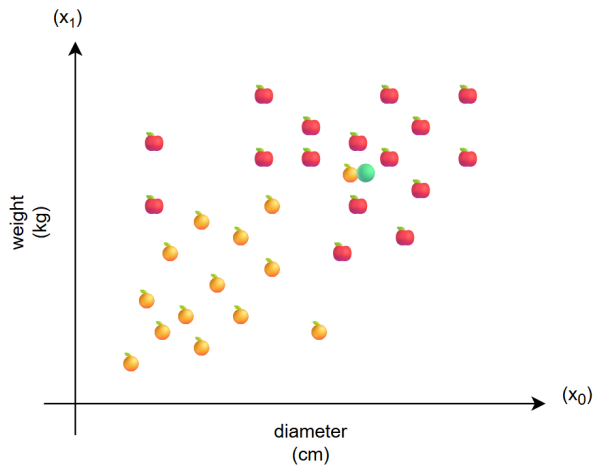


Answer: Clementine

What about this one?



What about this one?

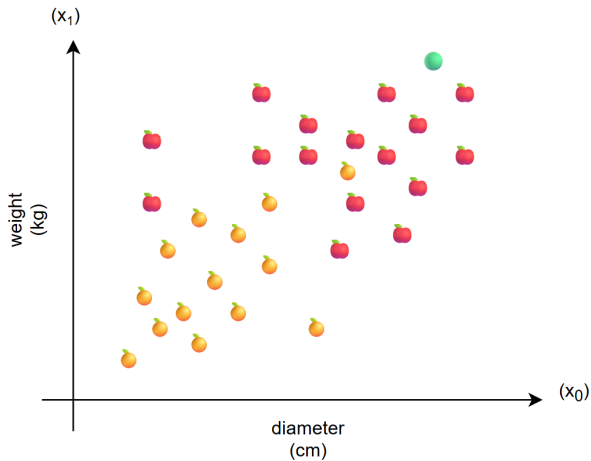


Problem: The closest fruit in our dataset is a clementine, but it *feels* like this should be an apple. We need to adjust our strategy to account for these cases.

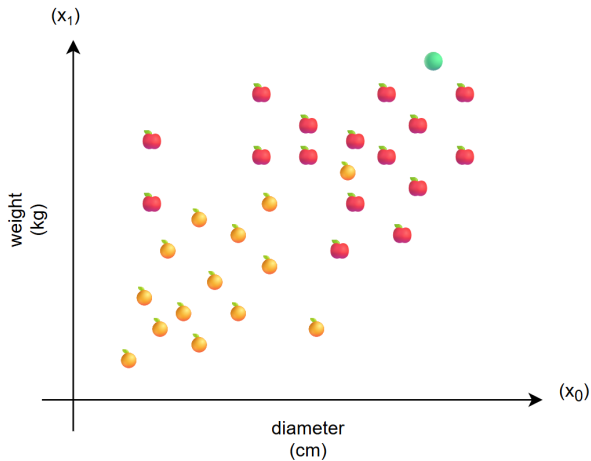
Problem: The closest fruit in our dataset is a clementine, but it *feels* like this should be an apple. We need to adjust our strategy to account for these cases.

Idea: Instead of looking at one point, we can look at multiple points and *pick the most frequent*. We will denote the number of neighbours we look at using k . For example, if $k = 3$, then we classify new points as the class of the 3 nearest neighbours.

Using $k = 3$, how should this point be classified?

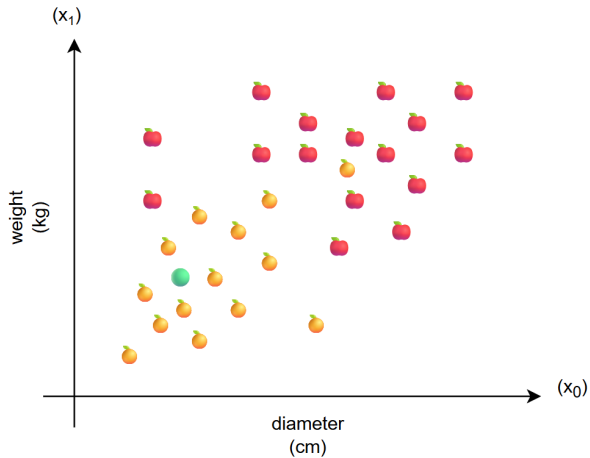


Using $k = 3$, how should this point be classified?

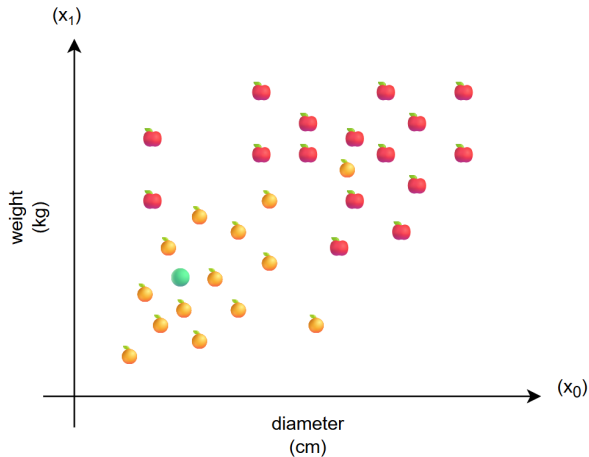


Answer: Apple

And this one?

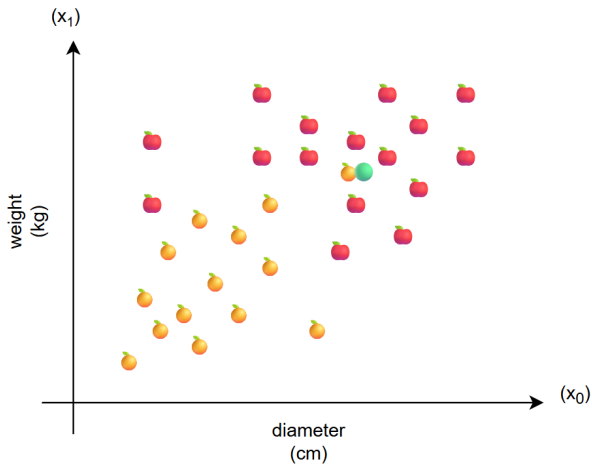


And this one?



Answer: Clementine

And this?



Answer: Apple. Yay, our problem from earlier is solved!

Distance Metrics

Question: For the idea of a “nearest neighbour” to make sense, we need a notion of distance. How should we define distance?

Distance Metrics

Question: For the idea of a “nearest neighbour” to make sense, we need a notion of distance. How should we define distance?

Answer: Any distance metric. Given two points $\vec{x}^{(i)} = (x_1^{(i)}, x_2^{(i)}, \dots, x_n^{(i)})$ and $\vec{x}^{(j)} = (x_1^{(j)}, x_2^{(j)}, \dots, x_n^{(j)})$ we have several options of distance metrics, such as:

Distance Metrics

Question: For the idea of a “nearest neighbour” to make sense, we need a notion of distance. How should we define distance?

Answer: Any distance metric. Given two points $\vec{x}^{(i)} = (x_1^{(i)}, x_2^{(i)}, \dots, x_n^{(i)})$ and $\vec{x}^{(j)} = (x_1^{(j)}, x_2^{(j)}, \dots, x_n^{(j)})$ we have several options of distance metrics, such as:

- Euclidean distance: $d(\vec{x}^{(i)}, \vec{x}^{(j)}) = \sqrt{\sum_{k=1}^n (x_k^{(i)} - x_k^{(j)})^2}$

Distance Metrics

Question: For the idea of a “nearest neighbour” to make sense, we need a notion of distance. How should we define distance?

Answer: Any distance metric. Given two points $\vec{x}^{(i)} = (x_1^{(i)}, x_2^{(i)}, \dots, x_n^{(i)})$ and $\vec{x}^{(j)} = (x_1^{(j)}, x_2^{(j)}, \dots, x_n^{(j)})$ we have several options of distance metrics, such as:

- Euclidean distance: $d(\vec{x}^{(i)}, \vec{x}^{(j)}) = \sqrt{\sum_{k=1}^n (x_k^{(i)} - x_k^{(j)})^2}$
- Manhattan distance: $d(\vec{x}^{(i)}, \vec{x}^{(j)}) = \sum_{k=1}^n |x_k^{(i)} - x_k^{(j)}|$

Distance Metrics: Visualized



Recap (so far)

So far we have seen that the k -nearest neighbours algorithm allows us to *classify* unlabelled data point $\vec{x}^*$ by finding the k -nearest neighbours of $\vec{x}^*$ in our existing *labelled* data.

Recap (so far)

So far we have seen that the k -nearest neighbours algorithm allows us to *classify* unlabelled data point $\vec{x}^*$ by finding the k -nearest neighbours of $\vec{x}^*$ in our existing *labelled* data.

Algorithm: KNN Classification

Inputs: (dataset: $\mathcal{D}$, k , distance metric d , unclassified point $\vec{x}^*$)

Output: Our predicted class for $\vec{x}^*$

Steps

1. For each point $\vec{x}^{(i)} \in \mathcal{D}$, compute the distance $d(\vec{x}^{(i)}, \vec{x}^*)$ and sort them from smallest to largest

Recap (so far)

So far we have seen that the k -nearest neighbours algorithm allows us to *classify* unlabelled data point $\vec{x}^*$ by finding the k -nearest neighbours of $\vec{x}^*$ in our existing *labelled* data.

Algorithm: KNN Classification

Inputs: (dataset: $\mathcal{D}$, k , distance metric d , unclassified point $\vec{x}^*$)

Output: Our predicted class for $\vec{x}^*$

Steps

1. For each point $\vec{x}^{(i)} \in \mathcal{D}$, compute the distance $d(\vec{x}^{(i)}, \vec{x}^*)$ and sort them from smallest to largest
2. Let $\{\vec{x}^{(1)}, \vec{x}^{(2)}, \dots, \vec{x}^{(k)}\}$ be the k -nearest neighbours

Recap (so far)

So far we have seen that the k -nearest neighbours algorithm allows us to *classify* unlabelled data point $\vec{x}^*$ by finding the k -nearest neighbours of $\vec{x}^*$ in our existing *labelled* data.

Algorithm: KNN Classification

Inputs: (dataset: $\mathcal{D}$, k , distance metric d , unclassified point $\vec{x}^*$)

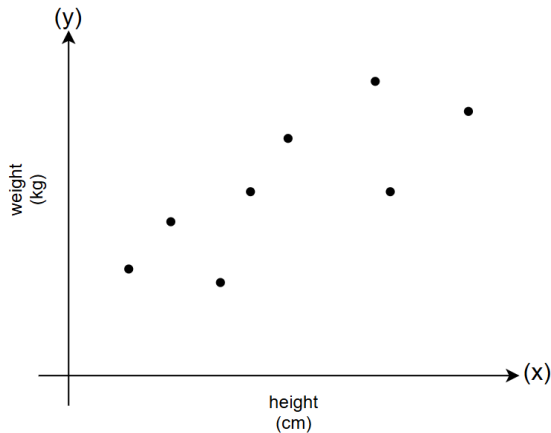
Output: Our predicted class for $\vec{x}^*$

Steps

1. For each point $\vec{x}^{(i)} \in \mathcal{D}$, compute the distance $d(\vec{x}^{(i)}, \vec{x}^*)$ and sort them from smallest to largest
2. Let $\{\vec{x}^{(1)}, \vec{x}^{(2)}, \dots, \vec{x}^{(k)}\}$ be the k -nearest neighbours
3. Set the label y^* to be the most frequent label among the set $\{y^{(1)}, y^{(2)}, \dots, y^{(k)}\}$

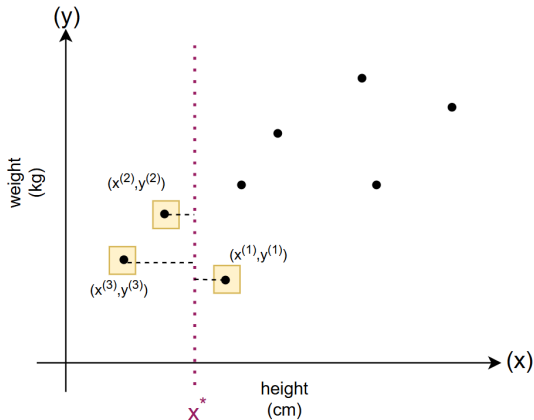
Toy Example: KNN for Regression

Suppose we are predicting someone's weight (y) using their height (x):



Question: How can we predict the weight using a similar strategy to before?

Given a height $\vec{x}^*$ and $k = 3$, we can predict the weight by taking the **average weights** of the three nearest neighbours of $\vec{x}^*$:



- The three nearest neighbours are:
 $x^{(1)}, x^{(2)}, x^{(3)}$
- Their corresponding weights are:
 $y^{(1)}, y^{(2)}, y^{(3)}$
- Predicted weight:

$$y^* = \frac{1}{3}(y^{(1)} + y^{(2)} + y^{(3)})$$

Recap (so far)

We have now also seen that the k -nearest neighbours algorithm allows us to perform regression on an unlabelled data point $\vec{x}^*$ by finding the k -nearest neighbours of $\vec{x}^*$ in our existing *labelled* data.

Recap (so far)

We have now also seen that the k -nearest neighbours algorithm allows us to perform regression on an unlabelled data point $\vec{x}^*$ by finding the k -nearest neighbours of $\vec{x}^*$ in our existing *labelled* data.

Algorithm: KNN Regression

Inputs: (dataset: $\mathcal{D}$, k , distance metric d , unclassified point $\vec{x}^*$)

Output: Our predicted value for $\vec{x}^*$

Steps

1. For each point $\vec{x}^{(i)} \in \mathcal{D}$, compute the distance $d(\vec{x}^{(i)}, \vec{x}^*)$ and sort them from smallest to largest.

Recap (so far)

We have now also seen that the k -nearest neighbours algorithm allows us to perform regression on an unlabelled data point $\vec{x}^*$ by finding the k -nearest neighbours of $\vec{x}^*$ in our existing *labelled* data.

Algorithm: KNN Regression

Inputs: (dataset: $\mathcal{D}$, k , distance metric d , unclassified point $\vec{x}^*$)

Output: Our predicted value for $\vec{x}^*$

Steps

1. For each point $\vec{x}^{(i)} \in \mathcal{D}$, compute the distance $d(\vec{x}^{(i)}, \vec{x}^*)$ and sort them from smallest to largest.
2. Let $\{\vec{x}^{(1)}, \vec{x}^{(2)}, \dots, \vec{x}^{(k)}\}$ be the k -nearest neighbours

Recap (so far)

We have now also seen that the k -nearest neighbours algorithm allows us to perform regression on an unlabelled data point $\vec{x}^*$ by finding the k -nearest neighbours of $\vec{x}^*$ in our existing *labelled* data.

Algorithm: KNN Regression

Inputs: (dataset: $\mathcal{D}$, k , distance metric d , unclassified point $\vec{x}^*$)

Output: Our predicted value for $\vec{x}^*$

Steps

1. For each point $\vec{x}^{(i)} \in \mathcal{D}$, compute the distance $d(\vec{x}^{(i)}, \vec{x}^*)$ and sort them from smallest to largest.
2. Let $\{\vec{x}^{(1)}, \vec{x}^{(2)}, \dots, \vec{x}^{(k)}\}$ be the k -nearest neighbours
3. Set $y^* = \frac{1}{k} \sum_{i=1}^k y^{(i)}$

feature scaling (why?)

feature scaling (why?)

$$d(\vec{x}^{(i)}, \vec{x}^{(j)}) = \sqrt{(x_1^{(i)} - x_1^{(j)})^2 + \dots + (x_n^{(i)} - x_n^{(j)})^2}$$

feature scaling (why?)

$$d(\vec{x}^{(i)}, \vec{x}^{(j)}) = \sqrt{(x_1^{(i)} - x_1^{(j)})^2 + \dots + (x_n^{(i)} - x_n^{(j)})^2}$$

If some feature has values significantly larger than other features, then it might “overcontribute” to the distance and “overshadow” the other features.

feature scaling (problem)

In this case, the income will contribute a lot more to the distance since the values are much larger, this gives it more weight when computing our nearest neighbours.

#	age (x_0)	income (x_1)
1	30	100,000
2	42	40,000
$\vdots$	$\vdots$	$\vdots$
100	23	23,000

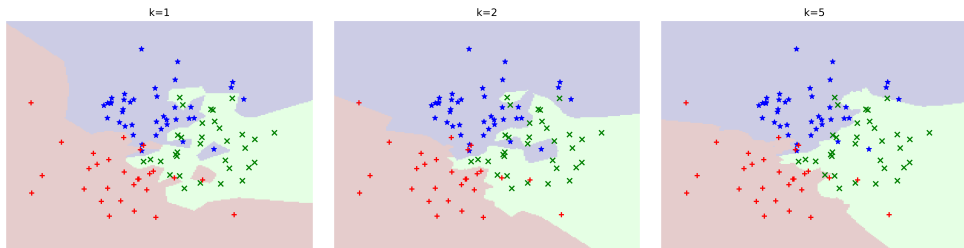
feature scaling (how to do it)

To get all the features on the same scale, we update them using the “standard score” before doing KNN (or before using any other model). To rescale feature j for datapoint i we subtract the mean of feature j across the whole dataset and divide by the standard deviation:

$$x_j'^{(i)} = \frac{x_j^{(i)} - \text{mean}(x_j^{(1)}, x_j^{(2)}, \dots, x_j^{(N)})}{\text{stdev}(x_j^{(1)}, x_j^{(2)}, \dots, x_j^{(N)})}$$

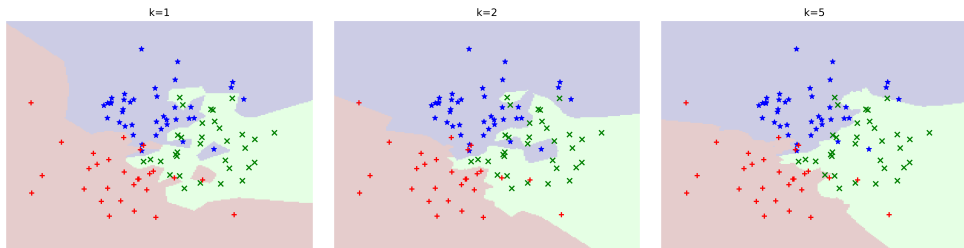
effect of k

The colour of each region tells you how a new point would be classified if it were in that region:



effect of k

The colour of each region tells you how a new point would be classified if it were in that region:



k is what we call a “hyperparameter”. How can we pick k ?

our goal when training a model

We can choose any value of k for KNN, so how do we know which to select?

our goal when training a model

We can choose any value of k for KNN, so how do we know which to select?

Things to keep in mind...

- We build our model based on our dataset $\mathcal{D}$

our goal when training a model

We can choose any value of k for KNN, so how do we know which to select?

Things to keep in mind...

- We build our model based on our dataset $\mathcal{D}$
- Our dataset is a small subset of the total population.

our goal when training a model

We can choose any value of k for KNN, so how do we know which to select?

Things to keep in mind...

- We build our model based on our dataset $\mathcal{D}$
- Our dataset is a small subset of the total population.
 - ▶ Ex: We only had measurements of 30 apples and clementines, but there are millions of them in the world

our goal when training a model

We can choose any value of k for KNN, so how do we know which to select?

Things to keep in mind...

- We build our model based on our dataset $\mathcal{D}$
- Our dataset is a small subset of the total population.
 - ▶ Ex: We only had measurements of 30 apples and clementines, but there are millions of them in the world
- We need to ensure that our model doesn't *overfit* to our dataset
 - ▶ ie. We need to make sure we aren't optimizing the accuracy of our model to fit *our* dataset. It should also work on new data it has never seen. In other words it should *generalize* well.

our goal when training a model

We can choose any value of k for KNN, so how do we know which to select?

Things to keep in mind...

- We build our model based on our dataset $\mathcal{D}$
- Our dataset is a small subset of the total population.
 - ▶ Ex: We only had measurements of 30 apples and clementines, but there are millions of them in the world
- We need to ensure that our model doesn't *overfit* to our dataset
 - ▶ ie. We need to make sure we aren't optimizing the accuracy of our model to fit *our* dataset. It should also work on new data it has never seen. In other words it should *generalize* well.

Result: We should be picking the value of k that generalizes the best.

Our goal is to find the value of k that *generalizes* the best to new data. We will do this by testing various choices of k and picking the best one. This process is known as **hyperparameter tuning**.

hyperparameter tuning at a high level

Hyperparameter tuning is a analagous to studying for an exam:

- When you are studying you are *building your mental model* (knowledge) of the course material

hyperparameter tuning at a high level

Hyperparameter tuning is a analagous to studying for an exam:

- When you are studying you are *building your mental model* (knowledge) of the course material
- The exam tests whether your mental model *generalizes* to new questions

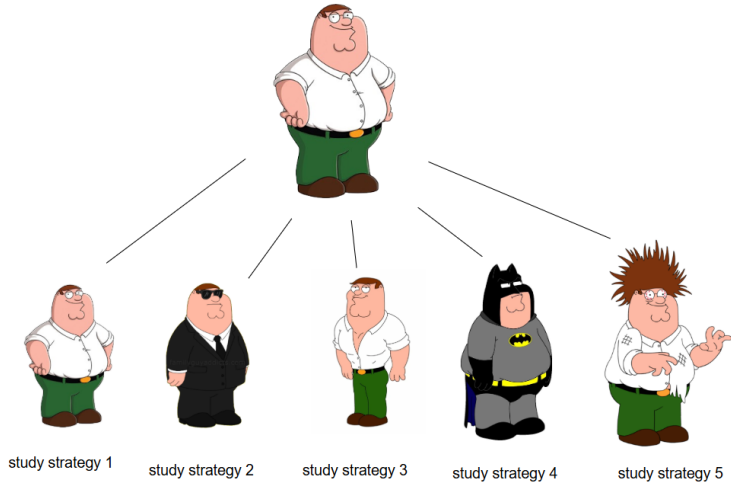
hyperparameter tuning at a high level

Hyperparameter tuning is analogous to studying for an exam:

- When you are studying you are *building your mental model* (knowledge) of the course material
- The exam tests whether your mental model *generalizes* to new questions

In hyperparameter tuning:

- We try a bunch of different values of k (different study strategies)
- We test each model on unseen data to see which one generalizes the best (each study strategy writes the final exam).
- We pick the value of k that performs the best (best study strategy for exam)



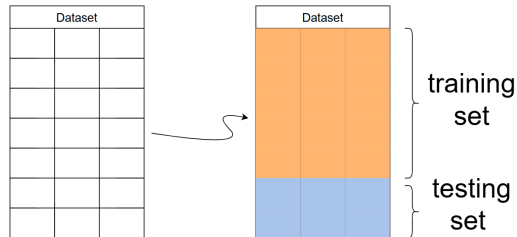
hyperparameter tuning: strategy # 1 - train/test split

Study strategy:

1. Read all the lecture notes
2. Do the exam and pray for the best

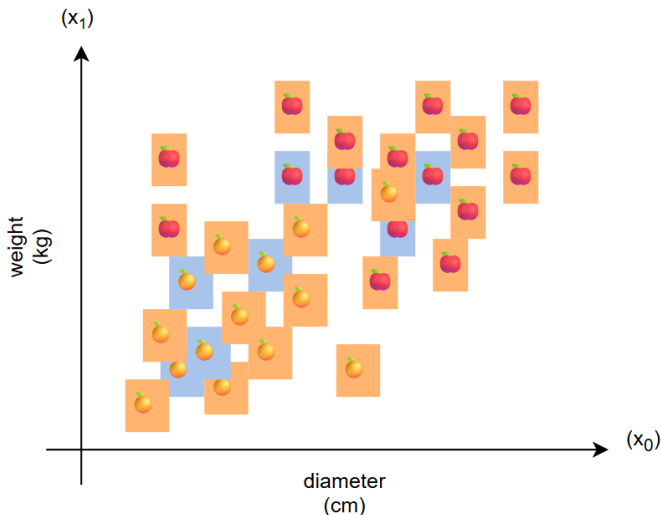
hyperparameter tuning: strategy # 1 - train/test split

- Our training set is used to create (or “train”) the model
- Our testing set is used to test our trained model
- **Note:** The split is done *randomly*



hyperparameter tuning: strategy # 1 - train/test split

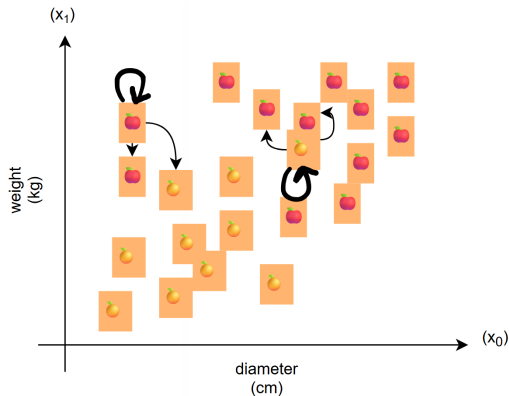
Step 1: Split the data into train (orange) and test (blue) sets:



hyperparameter tuning: strategy # 1 - train/test split

Step 2: Compute accuracy using *training data only* (here we have $k = 3$)

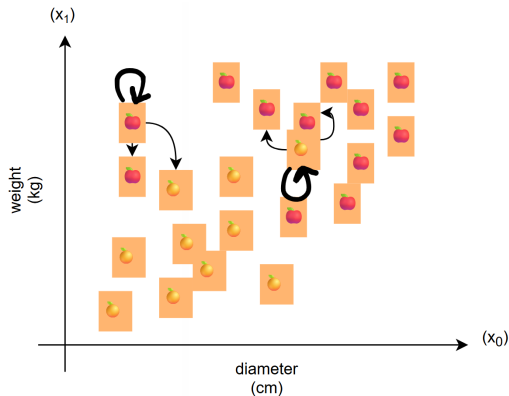
$$\text{accuracy} = \frac{\# \text{ training correctly classified}}{\# \text{ training data points}}$$



hyperparameter tuning: strategy # 1 - train/test split

Step 2: Compute accuracy using *training data only* (here we have $k = 3$)

$$\text{accuracy} = \frac{\# \text{ training correctly classified}}{\# \text{ training data points}}$$

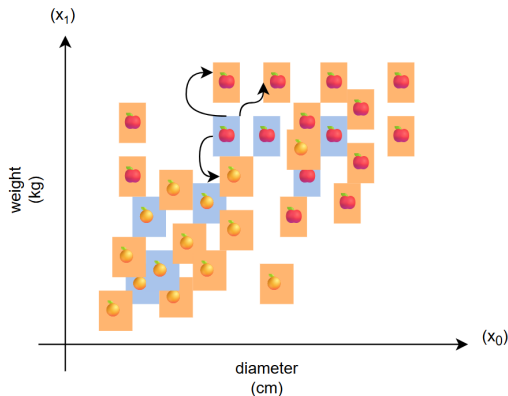


Note: The closest point to a point in the data is itself (as seen in the diagram)!

hyperparameter tuning: strategy # 1 - train/test split

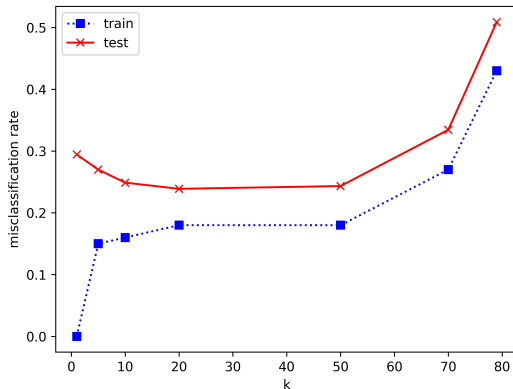
Step 3: Compute accuracy for the test data. Note that the neighbours must be elements of the *training set*! k is same as training, so $k = 3$:

$$\text{accuracy} = \frac{\# \text{ test correctly classified}}{\# \text{ test data points}}$$



hyperparameter tuning: strategy # 1 - train/test split

Step 4: Pick the value of k that results in the best *test accuracy*:



the problem with train test split

Question: Can you spot any bias in this strategy? Is it fair to say that the model that performed best on the test set is truly the model that generalizes the best?

the problem with train test split

Question: Can you spot any bias in this strategy? Is it fair to say that the model that performed best on the test set is truly the model that generalizes the best?

Answer: The problem is that we can “overfit” to the test set. Just because our model is good for *this* random test set doesn’t mean it will do well in general. Just because a student did really well on *this exam* doesn’t mean they’re an expert in the material. They could’ve just been lucky with the material they chose to study. In a sense, the test accuracy *isn’t very meaningful*.

hyperparameter tuning: strategy # 2 - train/test/validation split

Our previous study strategy wasn't great. We can improve it by doing practice problems to test our understanding. This gives us a gauge of which study strategy is the best *before the exam*.

hyperparameter tuning: strategy # 2 - train/test/validation split

Using this improvements we can study in the following way:

1. Read all the lectures notes (each clone tries reading them in a different way)

hyperparameter tuning: strategy # 2 - train/test/validation split

Using this improvements we can study in the following way:

1. Read all the lectures notes (each clone tries reading them in a different way)
2. Test each clone's understanding by doing the practice problems

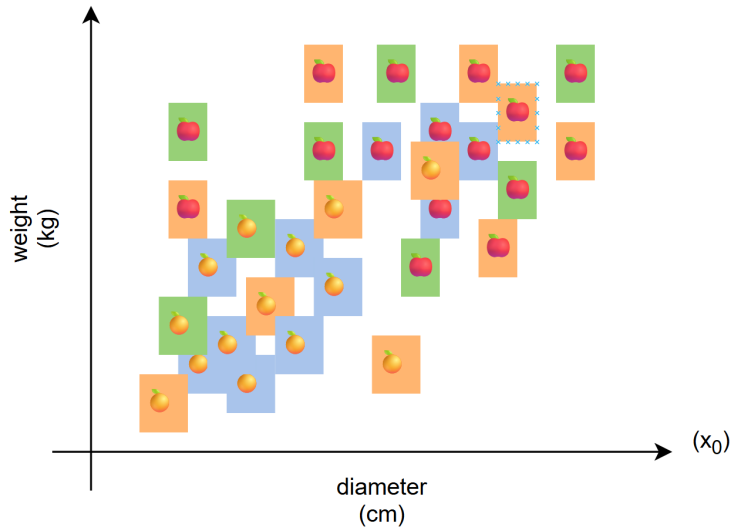
hyperparameter tuning: strategy # 2 - train/test/validation split

Using this improvements we can study in the following way:

1. Read all the lectures notes (each clone tries reading them in a different way)
2. Test each clone's understanding by doing the practice problems
3. Send the clone that performs *the best* on the practice problems to do the exam.

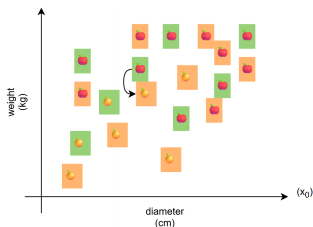
hyperparameter tuning: strategy # 2 - train/test/validation split

Step 1: Split into train (orange), test (blue), and validation (green) sets:

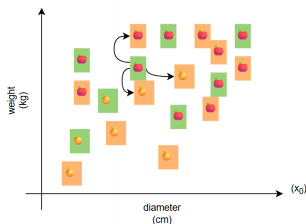


hyperparameter tuning: strategy # 2 - train/test/validation split

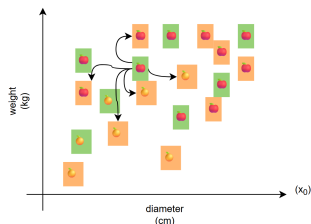
Step 2: Compute validation accuracy for various choices of k and find the one that performs the best:



(a) $k = 1$



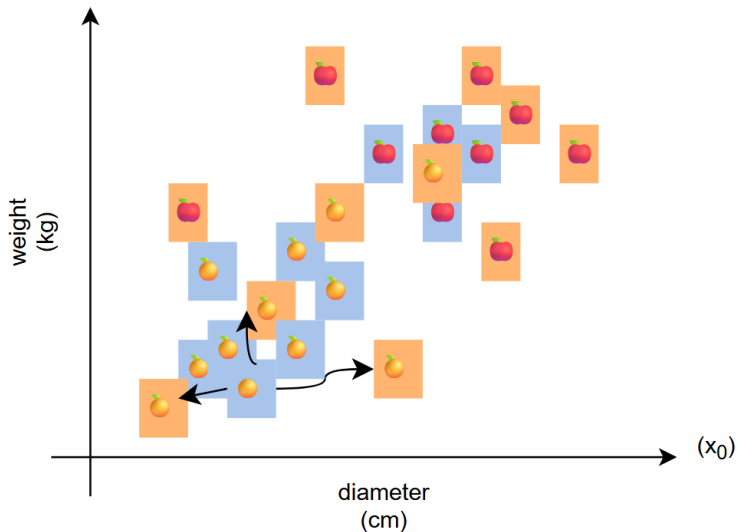
(b) $k = 3$



(c) $k = 5$

hyperparameter tuning: strategy # 2 - train/test/validation split

Step 3: Compute the accuracy of the model on the test set using the *best* k from step 2:

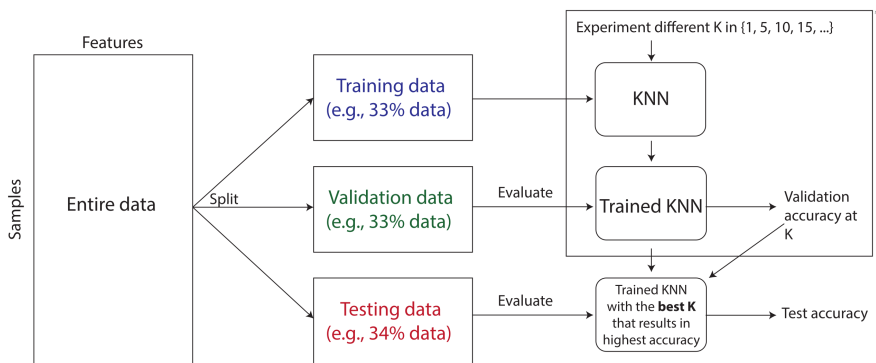


improvements from before

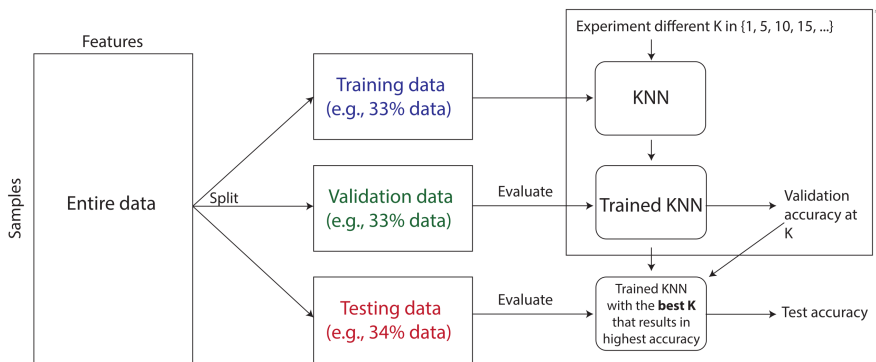
Question: Why is our model's test accuracy more meaningful now?

Answer: Because we picked our model based on the *validation set*, so we had no chance to overfit to the test set.

hyperparameter tuning: strategy # 2 - train/test/validation split



hyperparameter tuning: strategy # 2 - train/test/validation split



Why not pick k using the testing data?

hyperparameter tuning: strategy # 3 - cross validation

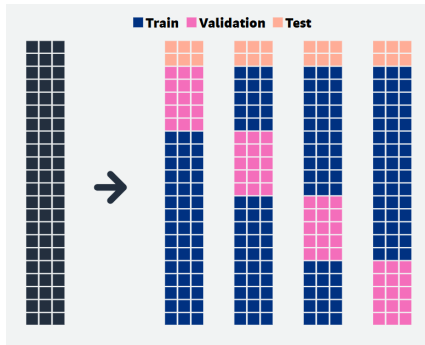
Problem: We want our training data to be a *representative sample* of the data. In our previous two strategies, however, we *randomly* split our data. We might get unlucky in the way we split our data and get a bad sample for training.

Solution: Cross-validation

hyperparameter tuning: strategy # 3 - cross validation

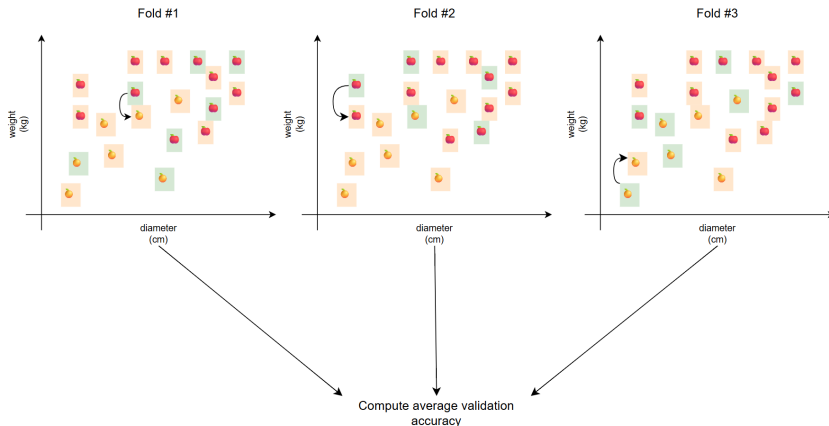
In *k-fold cross validation* we...

1. Randomly split our dataset into k folds of equal size
2. Reserve one fold for validation and the remaining $k - 1$ for training
3. Compute the validation accuracy
4. Repeat k times, using a different fold for validation each time
5. Accuracy = average accuracy across all folds



hyperparameter tuning: strategy # 3 - cross validation

Example: 3-fold cross validation for KNN with $k = 1$ neighbour:



measuring model performance

How do we tell how well our model did?

measuring model performance

How do we tell how well our model did? Well, it depends on the task.

measuring model performance

How do we tell how well our model did? Well, it depends on the task. For today, let's consider the example of binary classification

binary classification

Our model will make mistakes - but there are different kinds of mistakes.

		Model Predictions	
		Apple	Clementine
Actual	Apple	10 (True Positive)	3 (False Negative)
	Clementine	5 (False positive)	7 (True Negative)

binary classification

Our model will make mistakes - but there are different kinds of mistakes.

		Model Predictions	
		Apple	Clementine
Actual	Apple	10 (True Positive)	3 (False Negative)
	Clementine	5 (False positive)	7 (True Negative)

Some mistakes can be more costly than others...

metrics for binary classification

		Model Predictions	
		Apple	Clementine
Actual	Apple	10 (True Positive)	3 (False Negative)
	Clementine	5 (False positive)	7 (True Negative)

$$\text{Accuracy} = \frac{TP+TN}{(TP+FN)+(TN+FP)} \mid \text{Precision} = \frac{TP}{TP+FP} \mid \text{Recall} = \frac{TP}{TP+FN}$$

metrics for binary classification

$$\text{Accuracy} = \frac{TP+TN}{(TP+FN)+(TN+FP)} \mid \text{Precision} = \frac{TP}{TP+FP} \mid \text{Recall} = \frac{TP}{TP+FN}$$

- **Precision:** Of all the times the model predicted positive how often was it actually right?
 - ▶ Important when we want to be sure it never accidentally predicts a true negative to be positive. Eg. Criminal sentencing.

metrics for binary classification

$$\text{Accuracy} = \frac{TP+TN}{(TP+FN)+(TN+FP)} \mid \text{Precision} = \frac{TP}{TP+FP} \mid \text{Recall} = \frac{TP}{TP+FN}$$

- **Precision:** Of all the times the model predicted positive how often was it actually right?
 - ▶ Important when we want to be sure it never accidentally predicts a true negative to be positive. Eg. Criminal sentencing.
- **Recall:** Of all the actual positive cases out there, how many did the model find?
 - ▶ Important when we want to be sure it finds all the positive cases. Eg. Cancer screening.

bibliography & further reading

- Infamous ML textbook (but very dense) - “Probabilistic Machine Learning: An Introduction” by Kevin Murphy (2022)
<https://probml.github.io/pml-book/book1.html>
Suggested readings: **chapter 16**
- KNN explained in 2 minutes -
<https://youtu.be/0p0o5cmgLdE?si=h5ZPLYTe3G8PYQPr>
- Beautiful train/test/validation explanation -
<https://mlu-explain.github.io/train-test-validation/>
- Beautiful cross-validation explanation -
<https://mlu-explain.github.io/cross-validation/>