Explanatory Notes for 6.390

Shaunticlair Ruiz (Current TA)

Fall 2024

Contents

CHAPTER 12

Clustering

12.0.1 Why do clustering?

In chapter 4, we discussed **classification**: sorting data points into different groups, or **classes**.

Example: We might sort animals by **genetics**, or different sub-diseases that need different **treatments**.

Simplifying our data into categories can allow us to do better work, more easily.

This has lots of benefits:

- It could be used to make **decisions**. Sometimes, knowing the class of an object is enough to make a decision, by itself.
- We could use this to understand the structure and **distribution** of our data.
- We could **sort** different types of data to be processed **separately**.

The problem is, this relied on us **knowing** what classes we plan to sort into.

This may seem obvious, but what if we're looking at something **new**? A disease we don't fully understand, or animals we've never seen before? How do we **classify** them?

In the past, we've done this ourselves, giving us lots of useful classifications. But, **computers** allow us to do this in new situations:

- **High-dimensional** datasets, with too much **complex** information for a human to make sense of.
  - **Example:** Looking for patterns in hundreds of genetic factors at the same time.
- Discovering **new classes** **faster** than ever using computers.
  - And thus saving human labor.
- Finding **patterns** in creative ways humans would never think to, especially for really **abstract** problems.

Concept 1

Clustering is like **classification**, where we want to assign things to **classes**: we call them **clusters**.

But, we use it when we **don't know** what groupings we want, so we have to **find** them.

We have some challenges ahead of us, though. How do we decide what things are "similar" or "different"? How do we create new classes, and know that they're meaningful?

12.1 Clustering Formalisms

12.1.1 Unsupervised Learning

The first thing we should note:

This problem is similar to classification, a **supervised** problem.

- It was **supervised** because we knew the **correct** labels for our data in our advance. We just wanted to **teach** it to our computer.

The problem here: we **don't** know the correct labels! In fact, we're making them up as we go.

Because we aren't being "supervised" by a correct answer, we call this **unsupervised learning**.

Concept 2

Clustering is a type of **unsupervised learning**: meaning, we don't have a **correct** answer in advance.

The labels we create are not based on a **known** truth.

The **label** for data point $x^{(i)}$ is written as $y^{(i)}$.

12.1.2 What is clustering?

So, if we don't know **what** our classes are, how do we figure out **which** classes to create? Well, we have to think of what we expect in a class.

Intuitively, we think of a class as a **collection** of things that are **similar** to each other, and more different from other classes.

So, two points in the **same class** are more similar: in RBF, we decided that "more similar" meant "low distance" in the input space.

Remember that input space is where we represent each data point using input variables.

- **Example:** Two people might look more similar if their heights are numerically closer: shorter distance.

Meanwhile, two points in **different classes** are more **distant**: they're further apart in input space.

- **Example:** Two people might look more different if their weights are numerically further: greater distance.

We'll call these "possible classes" that we discover, **clusters**.

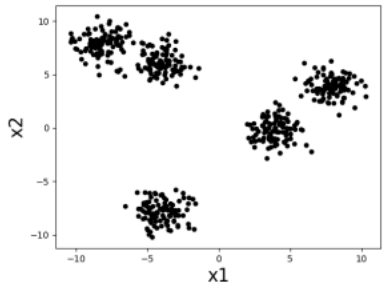
Definition 3

Informally, a **cluster** is a collection of **data points** that are all

- **Near** each other
- **Far** from the other clusters.

We use clusters as our way to discover **new classifications.**

Example: Below, we can visually mark out what looks like 5 distinct **clusters** in input space $(x_1, x_2) \in \mathbb{R}^2$:



This is an informal way to understand clusters, though. If we want to be more precise, we need to ask ourselves questions like:

- What does it mean for points to be "close" or "far"? How are we measuring distance?
- How many clusters do we want?
- How do we evaluate how "good" we are at clustering? Which clusters are closest to what we want?

12.2 The k-means formulations

In this section, we'll introduce a common way to do clustering, called the k**-means approach**.

12.2.1 Defining a cluster: The mean

We need to **define** what makes a "cluster" in order to move forward.

Suppose we have a collection of n points, which we informally think of as a "cluster".

We want the points within a cluster to be as **close together** as possible. So, you might ask, "how far is this point $x^{(i)}$ from the rest of the cluster?"

- How do we measure this? Well, we could average the distance to every other point. That could get time-consuming.
- Instead, could we somehow we could use one point that "**represents**" the whole cluster?
  - With this system, if you want the **distance from the cluster**, we can just use that "representative" as the "position" of our cluster.
  - That way, you only need to compute 1 distance, not $n - 1$ distances.
- This "representative" will be the **average** of all of our points: that way, it'll give us a rough idea of "where" all the points in the cluster are.

This is our **cluster mean**: when we want to know how far a data point is from the rest of the cluster, we'll compare it to the cluster mean.

Definition 4

We want to represent our **cluster** using its **mean**: the **average** of all of the data points in that **cluster**.

- This **cluster mean** will be treated as the "position" of our cluster.

Our goal is for the **cluster mean** to have the **minimum average distance** possible to all of our data points: it's as **close** to our points as we can get.

Example: We describe the "male lifespan" using **life expectancy**: the **average** time a male human lives for. Same for women as well.

12.2.2 k-means

Now, we've created **one** cluster. To extend this to **many** clusters, we just need each cluster to have its **own** mean.

We say that there are k of these clusters: this is why we call this the **k-means formulation**.

How do we decide which point goes in which **cluster**? Well, we want our points to be close to their cluster. So, we'll assign it to the **closest** one.

Concept 5

A **point** is assigned to the **closest cluster mean**.

For a point $x^{(i)}$, the **output** is which **cluster** ("new class") it has been assigned to: $y^{(i)}$.

Once we've successfully clustered using our **algorithm** below, we will find that both of these goals are met:

- Our points are **assigned** to the **closest** cluster mean.
  - This separates **different** clusters of points from each other:
  - If they're closer to different cluster means, they're in different groups.
- The cluster mean is the **average** of all of our points: the **minimum distance** to them.
  - This makes sure our cluster is made up of points that are **similar** to each other:
  - If our point is close to the **mean**, it's probably close to the **other** points in the cluster.

12.2.3 k-means loss

Now, we know what we **want** out of our clusters. But, the problem is, we don't know **how** to get those nice clusters.

So, first, we will have to **assign** our initial "cluster means": often, we **randomly** select some points from our dataset.

Concept 6

We **initialize** our clustering by **randomly** selecting one point to **represent** each cluster, which we call the **cluster mean**.

At first, each point is assigned to the **closest** cluster mean.

But as you'll notice, these points are **not** specially designed clusters! They're just a random **initialization**: we have to **optimize**.

Clarification 7

Notice that, when we **first** select our "cluster means", we don't get them by **averaging** any points: we choose them **randomly**.

That means, at first, is our cluster mean **isn't a true mean**!

Our k-means algorithm is designed to **fix** this problem.

In order to **improve** our clustering, it helps to have a way to measure the **quality** of a clustering: we need a **loss function**.

12.2.4 One-cluster loss

Let's start with just one cluster: what do we want to **minimize**?

Well, we want the points within a cluster to be as **close together** as possible. So, we want to minimize the **distance** to the mean, μ.

To make our function smooth, we'll use **squared distance** instead.

Concept 8

In **k-means loss**, we want to minimize the **square distance** from each point $x^{(i)}$ to the **cluster mean** μ.

$$D_i = \left\| x^{(i)} - \mu \right\|^2 \tag{12.1}$$

We'll add this up for each of the n data points in our cluster.

$$\mathcal{L} = \sum_{i=1}^{n} \left\| x^{(i)} - \mu \right\|^2 \tag{12.2}$$

12.2.5 Building up to k clusters

So, what do we do for each of our k clusters? Well, we can just **add** up the **loss** for them.

We'll use $j \in \{1, 2, 3, ...k\}$ to represent our j^{th} cluster. Each cluster has a mean $\mu^{(j)}$.

$$\overbrace{\mathcal{L}_j}^{\text{Loss for only cluster } j} = \sum_{i=1}^{n} \left\| x^{(i)} - \mu^{(j)} \right\|^2 \tag{12.3}$$

Problem is, we're including **every** point $x^{(i)}$ in **every** cluster! We want a way to filter by **cluster**: we only put one data point in each cluster.

Remember that we **label** clusters the same way we labeled **classes** before:

Notation 9

For a **data point** $x^{(i)}$, its **cluster** is given by

$$y^{(i)} \in \{1, 2, ...k\}$$

Where j represents the j^{th} cluster.

Cluster mean $\mu^{(j)}$ **only** includes points in cluster j. So, when computing **loss**, we **only** want to include data point $x^{(i)}$ when:

$$\overbrace{y^{(i)} = j}^{x^{(i)} \text{ is in cluster } j} \tag{12.4}$$

We'll do this using the following helpful **function**:

Notation 10

The **indicator function** $\mathbb{1}$ tells you whether a statement p is true:

$$\mathbb{1}(p) = \begin{cases} 1 & \text{if } p \\ 0 & \text{otherwise} \end{cases}$$

Combined with our **condition** of matching clusters, this can be useful:

$$\mathbb{1}\left(y^{(i)} = j\right) = \begin{cases} 1 & x^{(i)} \text{ is in cluster } j \\ 0 & x^{(i)} \text{ is not in cluster } j \end{cases} \tag{12.5}$$

If we **multiply** this by our loss, it'll filter for situations where the clusters **match**! We can **eliminate** data points in a different cluster.

$$\mathbb{1}\left(y^{(i)} = j\right)\left\|x^{(i)} - \mu^{(j)}\right\|^2 = \begin{cases} \left\|x^{(i)} - \mu^{(j)}\right\|^2 & x^{(i)} \text{ is in cluster } j \\ 0 & x^{(i)} \text{ is not in cluster } j \end{cases} \tag{12.6}$$

12.2.6 k-mean loss: final form

So, we can **filter** by the data points in our cluster:

$$\mathcal{L}_j = \sum_{i=1}^{n} \overbrace{\mathbb{1}\left(y^{(i)} = j\right)}^{\text{Check cluster}} \cdot \overbrace{\left\|x^{(i)} - \mu^{(j)}\right\|^2}^{\text{Sq. dist from mean}} \tag{12.7}$$

And finally, we add up over many clusters:

$$\mathcal{L} = \sum_{j=1}^{k} \mathcal{L}_j \tag{12.8}$$

Using our equation, we get:

$$\mathcal{L} = \overbrace{\sum_{j=1}^{k}}^{\text{clusters}} \overbrace{\sum_{i=1}^{n}}^{\text{data points}} \overbrace{\mathbb{1}(y^{(i)} = j)}^{\text{Check cluster}} \cdot \overbrace{\left\| x^{(i)} - \mu^{(j)} \right\|^2}^{\text{Dist from mean}}$$

Let's clean that up:

Key Equation 11

The **k-means loss** is given as:

$$\mathcal{L} = \sum_{j=1}^{k} \sum_{i=1}^{n} \mathbb{1}(y^{(i)} = j) \left\| x^{(i)} - \mu^{(j)} \right\|^2$$

Where:

- μ_j is the **cluster mean**: the **average** of the points in the j^{th} cluster.
- $\mathbb{1}(y^{(i)} = j)$ is the **indicator function**: meaning that we only **include** terms where the data point and mean are in the **same cluster**.

12.2.7 Making further use of the indicator function (Optional)

We can actually use our **indicator function** to represent some of our **other** variables:

For example: the **cluster mean** is the average of data points, but **only** those belonging to that cluster.

So, we can use $1(\cdot)$ to **filter** those other data points out:

$$\mu^{(j)} = \frac{1}{N_j} \sum_{j=1}^{k} \overbrace{\mathbb{1}\left(y^{(i)} = j\right)}^{\text{check cluster}} \cdot \overbrace{x^{(i)}}^{\text{data points}} \tag{12.9}$$

And how large is N_j? We can just **count** the number of data points in cluster j:

$$N_j = \sum_{j=1}^{k} \mathbb{1}\left(y^{(i)} = j\right) \tag{12.10}$$

One more loose end: we've been focusing on **square distance** as loss function. We want to

minimize this, but we're doing this over multiple data points.

So, really we want to minimize the **average** of that. This is a very common (and very useful!) property of a distribution called the **variance**.

Definition 12

The **variance** of a dataset is the **average square distance** from the **mean**:

$$\text{Var}[X] = \frac{1}{N}\sum_{i=1}^{N}\left\|x^{(i)} - \mu\right\|^2$$

It tells us how **spread out** our data is.

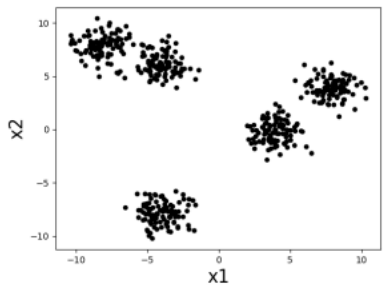
That means, our loss function is meant minimize the total **variance**.

Almost, the factor of 1/N is missing: since this constant won't change much as we improve our clustering, we'll leave it alone.

12.2.8 Initializing the k-means algorithm

Now that we have our **clusters**, **means**, and a **loss** function for evaluating them, we can begin looking for a better **clustering**.

We'll start out with a **dataset** we want to cluster: we'll use the one from the **beginning** of the chapter:



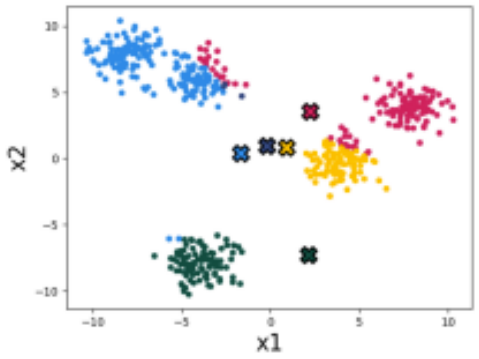
We could cluster this visually, but we want our machine to be able to do it for us.

First, we need to decide on our **number** of clusters. When you can't **visualize** it, this can be **difficult** - how many is too many or too few?

But, for now, we'll **ignore** that problem, and say $k = 5$.

Let's **randomly** assign our initial cluster means, and assign each point to the **closest** cluster:

Above, we suggested selecting a random data point as our cluster mean, but you can also just pick a random position, like we did here.



1) Initial assignments

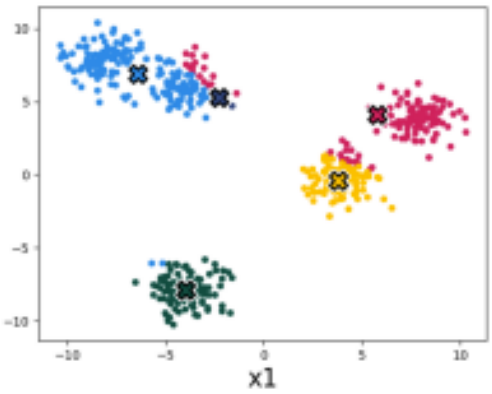
This is our starting point for the algorithm.

12.2.9 First step: moving our cluster means

As we mentioned before, these points aren't **actually** the average of their cluster: you can tell that by looking at it.

We want to **minimize** the variation in our cluster: that's why we're using the mean.

So, let's fix this: we'll take the **average** of all the points in each **cluster**, and **move** the cluster mean to that position.



2) Update means

And now, our cluster means are closer to all our data points!

Concept 13

One way **minimize** the **distance** between the **cluster mean** and its **data points** is:

- Take the **average** of all the points in the cluster, and **reassign** the cluster mean to that average.

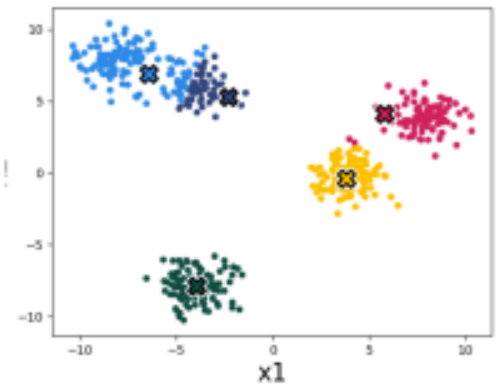
12.2.10 Second step: Reassign data points

We've **improved** our model by moving the cluster mean.

The problem is, we originally **assigned** every point to the **closest** cluster mean.

If the cluster means **move**, then some points might be closer to a **different** cluster now!

If so, we can **improve** our clustering further by reassigning points to the cluster they're **closest** to!



3) Update assignments

Concept 14

Another way **minimize** the **distance** between the **cluster mean** and its **data points** is:

- After the **means** have been **moved**, **reassign** the **data points** to whichever mean is **closest**.

12.2.11 The cycle continues

But wait - now that we've changed the points in each cluster, our cluster mean might not be the **true** average!

So, we can, again, improve our loss by taking the average of each cluster, and moving the cluster mean.

This creates a cycle that continues until we **converge** on our final answer.

Concept 15

Together, both of our steps for **improving** our clusters create a **cycle** of **optimization**:

- **Moving** our cluster mean **changes** which point should go in each cluster.
- **Reassigning** points to different clusters **changes** our cluster mean.

12.2.12 The k-means algorithm

These two steps make up the **bulk** of our algorithm:

Definition 16

The **k-means algorithm** uses the following steps:

- First, we **randomly** choose our **initial** cluster means.

Then, we **cycle** through the following two steps:

- **Reassign points** to the cluster mean they're closest to.
- **Move** each **cluster mean** to the average of all the points in that cluster.

Until our clusters means **stop** changing.

When we run our algorithm on the above dataset, we get:

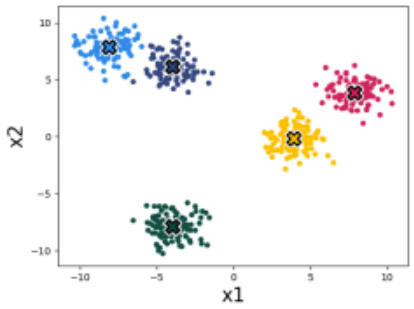


Figure 6.3: Converged result.

Note that, our cycle **works** because changing **either** cluster mean or point assignments allows you to further **improve** the **other** step.

So, if we're **not** changing one of them, the other one won't **change** either: the cycle is **broken**, and we can **stop**.

This is our termination condition!

Another nice fact: it can be shown that this algorithm does **converge** to a local minimum!

Concept 17

The **k-means algorithm** is guaranteed to **converge** to a **local minimum**.

12.2.13 Pseudocode

K-MEANS$(k, \tau, \{x^{(i)}\}_{i=1}^{n})$

```
μ, y = randinit        #Random initialization
for t = 1 to τ         #Begin cycling

    y_old = y          #Keep track of last step

    for i = 1 to n
        y^(i) = arg min_j ||x^(i) − μ^(j)||_2^2      #Reassign data point to closest mean

    for j = 1 to k
        μ^(j) = (1/N_j) Σ_{i=1}^{n} 1(y^(i) = j) x^(i)      #Move cluster mean to average of cluster

    if 1(y = y_old)
        break          #If nothing has changed, then the cycle is done. Terminate

return μ, y
```

12.2.14 Using gradient descent: minimizing distance to μ

We can also use **gradient descent** to solve this problem!

We want to **minimize** our loss $\mathcal{L}$, and we do this by **adjusting** our cluster means $\mu^{(j)}$ until they're in the **best** position.

> **Concept 18**
> We can solve the **k-means problem** using **gradient descent**!

So, we want to **optimize** $\mathcal{L}$ using μ:

$$\mathcal{L}(\mu) = \sum_{i=1}^{n} \mathbb{1}\left(y^{(i)} = j\right) \left\| x^{(i)} - \mu^{(j)} \right\|^2 \tag{12.11}$$

Rather than dealing with the indicator function $1(\cdot)$, we could instead just consider whichever μ is closest: **minimum** distance.

$$\overbrace{\min_{j}}^{\text{Minimizing}} \overbrace{\left\| x^{(i)} - \mu^{(j)} \right\|^2}^{\text{distance}} \tag{12.12}$$

This **automatically** assigns every point to the closest **cluster** before we get our loss! So, all we need to worry about is μ_j.

Notation 19

Instead of using an **indicator function** $\mathbb{1}(p)$, we can represent **cluster assignment** another way: using the **function** $\min_j$.

It can give **minimum distance** from $x^{(i)}$ to one of the cluster means: it picks the **closest** mean.

This **automatically** assigns the point to the **closest** cluster, making our job easier.

$$\mathcal{L}(\mu) = \sum_{i=1}^{n} \overbrace{\min_j}^{\text{Nearest cluster}} \left\| x^{(i)} - \mu^{(j)} \right\|^2 \tag{12.13}$$

Now, we can do gradient descent using $\frac{\partial L(\mu)}{\partial \mu}$.

We move our means until they're minima!

$L(\mu)$ is **mostly** smooth, except when the cluster assignment of a **point** changes. So, it's usually smooth **enough** to do gradient descent.

12.2.15 Getting labels

Once we've finished gradient descent, and we've **minimized** our loss, we can get our **labels**.

The "min" function gives the **output** value that we get by minimizing. In this case, average **squared distance** from the cluster mean: the **loss**.

Meanwhile, argmin gives us the **input** value that gives us the minimum output. In this case, the **choice of cluster means** that gives the minimum distance.

So, arg min gives us the cluster closest to each point: that's our **label**!

We can use this notation to get our **labels**.

Notation 20

After **optimizing** μ, our **labels** are given by:

$$y^{(i)} = \arg\min_j \left\| x^{(i)} - \mu^{(j)} \right\|^2$$

Using gradient descent can give us a **local** minimum, but our surface is not fully **convex**: so, we don't necessarily get a **global** minimum.

Even though individual terms of squared distance may be convex, adding min terms may not be convex.

Of course, this is also true for the k-means algorithm.

12.3 How to evaluate clustering algorithms

The biggest problem with clustering algorithms is that they're **unsupervised**: this makes it much harder to know if we've gotten a **good** result.

This is partly because our **loss** function doesn't necessarily tell us if clustering is **useful**, or represents the data **accurately**.

It just tells us if our points are **close** to their cluster **mean**. That doesn't always mean the clustering is **good**.

Example: Imagine **every** single point in the dataset gets its **own** cluster mean. The **distance** to the cluster mean would be 0 (low loss), but this isn't very **useful**!

This isn't useful because nothing has changed: we've gone from having n separate data points, to having... n separate clusters.

Clarification 21

The k**-means loss function** does **not** tell us if we have a good and **useful** clustering or not.

It only tells us if the points in our clusters are **close** to their **cluster mean**.

This can help us make **better** clusters, but that does not mean they are **good** or what we **want**.

Without having "true" labels, we have to find other ways to **verify** our approach.

We'll do two things to **approach** this problem:

- We'll look at some of the ways our **algorithm** can go wrong (or right).
- Then, we'll find **better** ways to evaluate our clusterings than just looking at the **loss**.

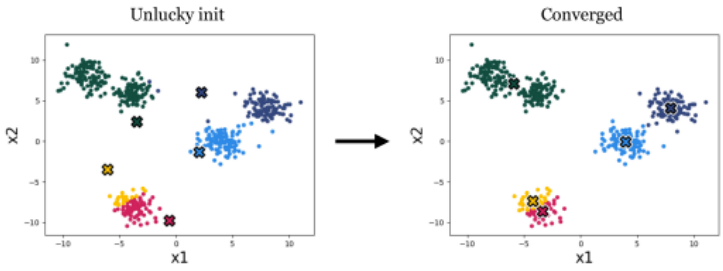
But, always remember that a "good" clustering is partly **subjective**, and depends on what you **want** to accomplish.

12.3.1 Initialization

The first problem we have is related to something we mentioned at the end of the last section: k-means is not **convex**.

That means we can find **local** minima that are not the **global** minimum: our **initialization** (our **starting** clusters) can affect whether we end up in a useful minimum.

The reason why is, mathematically, the same as when we first introduced the idea of a local minimum.



In this example, notice that we ended up with convergence on some very **bad** clusters: the bottom cluster is split in **half**!

The easiest way to resolve this is to run k-means multiple times with different initializations.

Other techniques exist, but this is the simplest one.

Concept 22

Getting an **unlucky initialization** can result in **clusters** that aren't **useful**.

We try to **solve** this by running our algorithm **multiple times**.

12.3.2 Choice of k

One important question we decided to **ignore** earlier was: **how many** clusters should we pick in advance?

Especially for **complex** data, we **don't know** how many natural clusters there will be.

But our number of clusters matter: because it's a parameter determines **how** our learning algorithm runs (rather than being chosen *by* the algorithm), it's a **hyperparameter**:

Concept 23

Our **number of clusters** k is a **hyperparameter**.

And, choosing too high *or* too low can both be **problematic**:

- If we set k too **high**, then we have more clusters than actually **exist**.
  - This can cause us to **split** real clusters in half, or find **patterns** that don't exist.
  - In a way, this resembles a kind of **overfitting**: we try to **closely** match the data, but end up fitting **too closely** and not **generalizing** well: **estimation error**.
  - **Example:** The **extreme** case looks like the example we mentioned **before**: when labeling animals, we could make... a different **species** for every single instance of **any** animal we find.

That doesn't sound very helpful.

- If we set k too **low**, we don't have **enough** clusters to represent our data.

- This means some clusters will be **lumped together** as a single thing: we **lose** some information.
- In this case, it's **impossible** to cluster everything in the way that would make the most **sense**: we have **structural error**.
- **Example:** Let's say we wanted to **sort** fish, birds, and mammals into **two** categories: we might just **divide** them into "flies" and "doesn't fly".

That's some information, but often not enough!

Concept 24

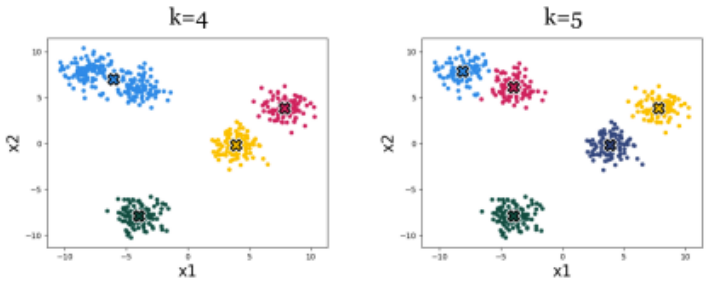
When choosing k (our **number of clusters**), we can cause **problems** by picking an inappropriate **value**:

- **Too many** clusters (large k) can cause **overfitting** and **estimation error**: we find patterns we don't want.
- **Not enough** clusters (small k) causes **structural error**: it prevents us from correctly **separating** data.

12.3.3 Subjectivity of k

Not only is it hard to choose a "good" value of k, what a good value of k is can really depend on your opinion, and what you know about reality.

For example, consider the following example:



Which of these two clusterings is more accurate?

Should the top left be **one** cluster, or **two**? It's hard to say!

Even if you're **sure**, you might **disagree** with others, or find that the best one depends on your **needs**.

So not only can k values be too high or too low, they can also be **debatably** better or worse!

Concept 25

The **best** choice of **clustering** is not entirely objective: it can depend on your **opinion**, or how you plan to **use** the clustering.

What do we mean by, what we're "**using**" the clustering for? We'll get into that later, but in short: we might use **clusters** to make sense of **information**, or to make better **decisions**.

Different clusterings might be good when you want a different kind of understanding.

Example: The understanding you get from high-level comparisons (plants vs animals vs bacteria) is different from low-level comparison (cats vs dogs).

12.3.4 Hierarchical Clustering

That last example reveals something: not all types of groups are the same! Some are much broader than others, for example.

If two types of groups are different, then why do we only have to have one type in our clustering? We don't have to restrict ourselves to a single k.

Instead, we could treat some groups as inside of other groups: we call this a *hierarchy*, because some groups are "higher" on the scale.

Definition 26

Hierarchical Clustering is when we cluster at multiple different **levels**.

Some groups are **high-level**, or **coarse**: they are groupings that contains **more elements**: items in the same group can be more **different**.

Some groups are **low-level**, or **fine**: they are groupings that contain **fewer elements**: items in the same group have to be very **similar**.

Example: Categorizing living things is done using hierarchical clustering: some groupings **contain** other groupings.

The top row of clusters are more **coarse**, while the bottom row is more **fine**.

We can **split** our groupings into **smaller** and smaller ones, to be as **fine-grained** as we need. Useful!

12.3.5 k-means in feature space

One **important** consideration: when working with **feature** representations, we found that sometimes, feature transformations made it **easier** to **classify** data.

Often, it wasn't even possible without those transformations!

Clustering is very **similar**, so, could we do the same here? It turns out, we **can**!

Rather than directly clustering in **input** space x, we can **process** our data using features, and then cluster in **feature** space $\phi(x)$.

Concept 27

Feature transformations can be used to make it easier to **accurately** cluster our data in a **meaningful** way.

There are some **other** reasons to do feature transformations, though: imagine that our data is **stretched** out along axis x_1, but not x_2.

x_1 distances would be **larger** in general: it would contribute more to our distance metric! We could correct for this by **standardizing** our data: **scaling down** the more stretched axis.

This would be a **feature** transformation, and would make it **easier** to do our **clustering**.

12.3.6 Solutions: Validation

Now, we start trying to answer the **question**: how do we **check** whether we have a **good** clustering?

Well, first, we can check for a **poor fit** (or overfitting) using new, **held-out** testing data: do we get **low loss** on that testing data?

If we **don't**, then our clusters definitely aren't **representative** of the overall **dataset**: they don't **generalize** to new data.

Concept 28

If our clusters give **large testing loss**, then they aren't **generalizing** well, and are probably **not representative** of the overall distribution.

So, we already know our clusters **don't fit the distribution**.

12.3.7 Solutions: Consistency

But, just like for classification/regression **validation**, we don't only run our algorithm **one time**: we'll run it **many** times, with different training and testing sets.

We can't **just** use the loss, though: having **more** clusters could make our error lower, without making a better clustering, for example.

Another thought: we're trying to find some patterns **inherent** in the data. The idea is: if the pattern we're finding is **real**, we should find a similar pattern **each time**!

So, we look to see if our clusters are **consistent** when we generate them using different training data: if they **aren't**, then it's possible we're not finding the "**real**" patterns in the data.

Different training data from the same distribution, of course.

Concept 29

If our **clusters** accurately **reflect** the underlying classes of data, then we should expect some **consistency** of which clusters we **generate** by running k-means many times.

If our clusters aren't **consistent**, then we might doubt if any of them especially reflect the **distribution**, rather than **noise**.

If our clusters are **consistent**, then we're probably seeing something about the **real** dataset.

If it was based on random noise, then the odds of getting matching results would be really low!

12.3.8 Solutions: Ground Truth

But, even if we're getting something **consistent**, that doesn't mean we're seeing the patterns that **matter**.

One way to **check** this is, if we have some idea of what the "**true**" clustering looks like for just a few data points, we can compare those results to ours.

We call this "real" clustering the "ground truth".

Definition 30

In machine learning, the **ground truth** is what we know about the "real world".

In general, we want our models to be able to **reproduce** this reality: it is the data that we tend to **trust** the most, if it is gathered correctly.

That way, we can use a very **small** amount of **supervision** to get an idea of whether our clustering is on the **right track**.

12.3.9 Applications: Visualization and Interpretability

We've discussed some ways to **abstractly** test whether our clustering might be **accurate** the data.

But, when it comes down to it, often, the **quality** of a clustering is based on how **useful** it is. So, what sorts of **uses** does clustering **have**?

Well, we're organizing our data into **groups**: this **simplifies** how we look at our data. And when we can **look** at our data, we can better **understand** what's going on.

In short: clustering allows humans to more easily make sense of data.

Concept 31

One of the the main **goals** of **clustering** is to make it easier for humans to **understand** the data.

This happens in two ways:

- We can **visualize** the data: we can **see** it, and more easily use our **intuition** to make sense of it.
- We can **interpret** our data: by seeing what sorts of **groupings** we create, we learn about the **structure** of the data.

So, machine learning experts judge partly based on how well a clustering **helps** them **achieve** these two goals.

Evaluating clusterings is **subjective** for exactly this reason: what is **good** "visually", or is the **best** "interpretation" of data, is often up to **debate**.

So, **human** judgement is important for this type of **problem**.

12.3.10 Applications: Downstream Tasks

Finally, there's one more way to think about clustering that is more **practical**, and closer to **objective**.

We use clustering to **sort** different data points that need different processing: this can make our model more **effective**, since different parts of the dataset may work better with different **treatment**.

Example: We could train a different regression model on each cluster: this can create a more accurate model.

We call this next problem a **downstream application**.

Definition 32

A **downstream application** is a **problem** that relies on a **different** process to make its work better or easier.

In this case, **clustering** has **downstream applications** that can **take advantage** of the **structure** that clustering reveals.

- These "applications" rely on clustering for this improvement.

If our clustering is **good**, we would expect it to **improve** the performance of downstream tasks.

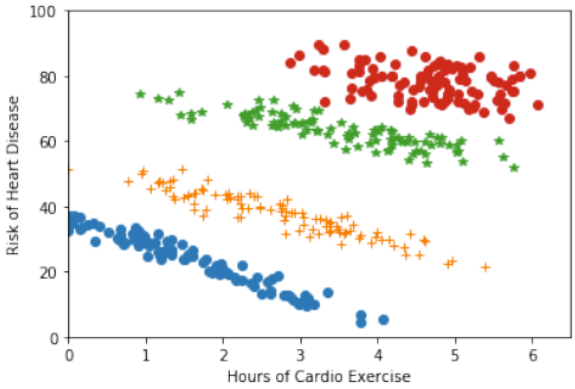
Concept 33

We can indirectly **evaluate** a **clustering algorithm** based on how **successful** the **downstream application** is.

If it **improves** the performance of a downstream application, we could say it works **well**.

12.3.11 A benefit of clustering

One advantage for downstream applications is, there might be patterns that are more obvious if you only look at a related segment of the data. For example:



If we take the data as a whole (**no** clustering), we would draw a **positive** regression: it seems that exercise and heart disease increase **together**. That doesn't make sense!

But, if we divide it into **clusters**, based on age, we see a **negative** relationship: each individual group experiences **benefits** from exercise.

This particular issue is called **Simpson's Paradox**.

Definition 34

Simpson's Paradox is when a **trend** that appears in groups of data either **vanishes** or **reverses** when we look at all the data **together**.

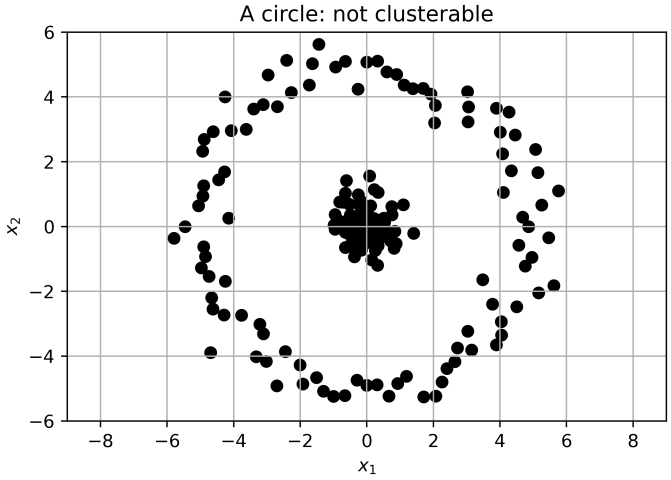
It shows that sometimes, **patterns** that we see may reflect how we're **looking** at the data.

Rest assured, you don't need to know this paradox by **name**. But it's important to **understand** possible problems like it: it'll help you make more responsible judgements in the future.

12.3.12 Weaknesses of k-means

There are some **weaknesses** to k-means clustering. Some patterns that a human eye can see, aren't easily **clustered** by our algorithms.

We can see this with a few **examples**:



This data clearly seems to have a pattern. But does it have "clusters"?

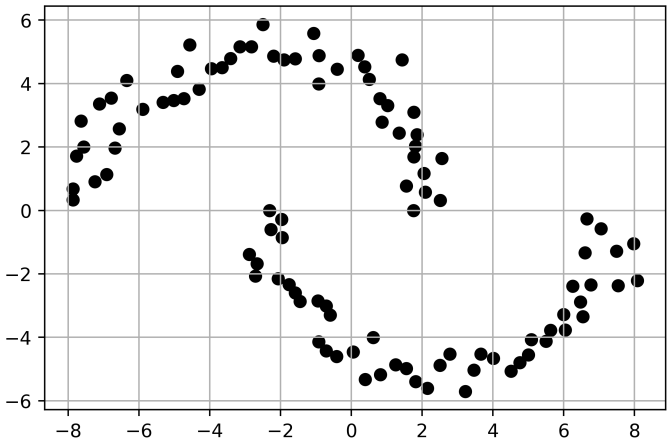
This example can't be effectively **clustered**: and yet, most people would agree that the "outer ring" should be **one** cluster, while the "inner circle" should be **another**.

Assuming we (correctly) place one cluster mean in the **center**, there's **nowhere** we can put our other cluster mean to be **closest** to all of the **outer** points, but **not** the inner points.

We might be able to **resolve** this using a **feature** transformation. But, the problem remains.

For example, we could have a feature represent the radius! But then, we would still struggle with a ring not centered on the origin. Or, two rings with different centers.

Another example works for clusters that aren't very centralized:



This data can't be clustered either!

The edge of one cluster is too close to the other: we can't easily create a good pair of cluster means for each semi-circle.

These sorts of cases are often approached with either

- Attempts to directly visualize them
- Feature transformations
- Other algorithms not discussed here

12.4 Terms

- Clustering
- Unsupervised Learning
- Cluster
- Cluster mean
- k-means problem
- k-means loss
- Initialization
- Indicator Function
- Variance (Optional)
- k-means algorithm
- Hierarchical Clustering
- Consistency
- Ground Truth
- Visualization
- Interpretability
- Downstream Application
- Simpson's Paradox (Optional)