Lecture 5

Clustering with k-means

COURSE MATH 637: Mathematical Techniques in Data Science

WEBSITE https://jacobcalvert.com/teaching/math637fall2026/

INSTRUCTOR Jacob Calvert (calvert@udel.edu)

OFFICE HOURS MW 2:45pm–3:45pm, Ewing Hall 509

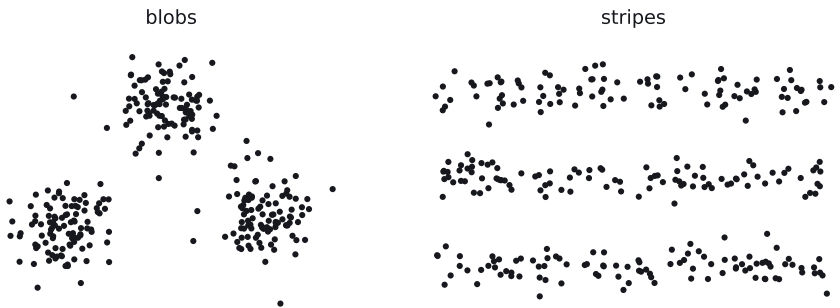
Announcements & Reminders

- Homework 1 is due before class on Sep 14
- Friday's meeting will be a lecture on density estimation
- I'll return Quiz 1 on Friday

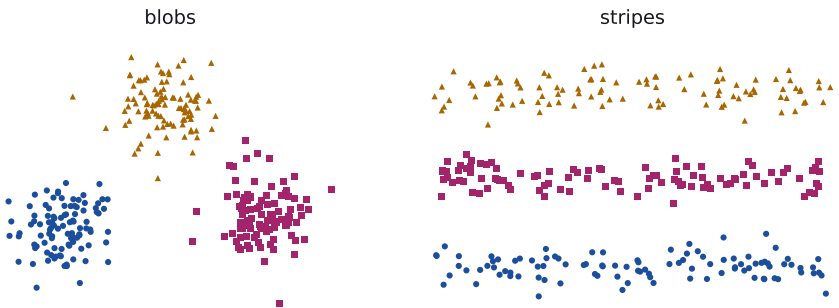
Plan for today

• Intro to clustering	5 min
• The k-means objective	20 min
• Lloyd's algorithm	15 min
• Example: color quantization	5 min
• Issues with k-means	8 min
• Surprise!	2 min

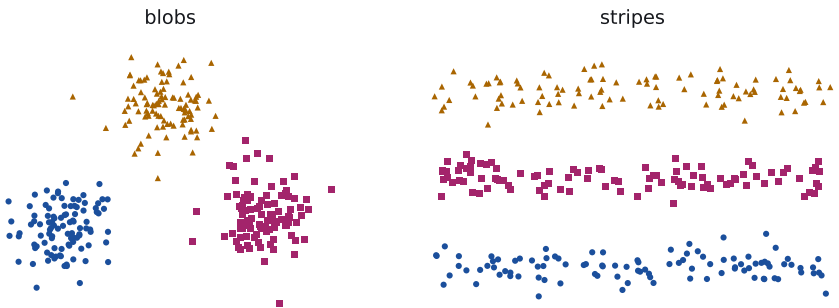
How would you assign these points to k groups?



How would you assign these points to k groups?



How would you assign these points to k groups?



Question

What criteria did you use? How would you generalize these criteria to high-dimensional data? How would you automate finding the groups?

Definition: Clustering

- A clustering is simply a partition of points $x_1, \dots, x_n \in \mathbb{R}^p$

Definition: Clustering

- A clustering is simply a partition of points $x_1, \ldots, x_n \in \mathbb{R}^p$
- One way to formalize: a function c that, given index i, returns the label of the cluster to which x_i is assigned

Definition: Clustering

- A clustering is simply a partition of points $x_1, \ldots, x_n \in \mathbb{R}^p$
- One way to formalize: a function c that, given index i, returns the label of the cluster to which x_i is assigned
- E.g., if x_i is assigned to cluster j, then we would write

$$c(i) = j$$

Definition: Clustering

- A clustering is simply a partition of points $x_1, \dots, x_n \in \mathbb{R}^p$
- One way to formalize: a function c that, given index i, returns the label of the cluster to which x_i is assigned
- E.g., if x_i is assigned to cluster j, then we would write

$$c(i) = j$$

- The # of clusters k is usually fixed ahead of time, so $c(i) \in \{1, \dots, k\}$

Definition: Clustering

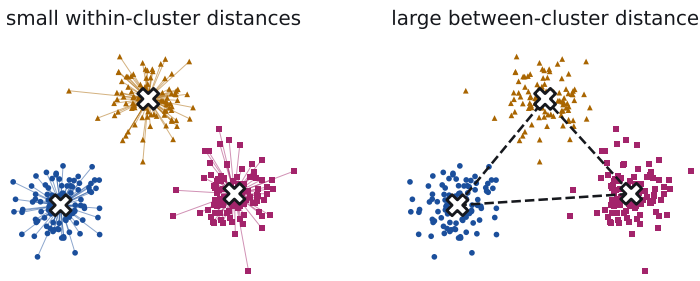
- A clustering is simply a partition of points $x_1, \dots, x_n \in \mathbb{R}^p$
- One way to formalize: a function c that, given index i, returns the label of the cluster to which x_i is assigned
- E.g., if x_i is assigned to cluster j, then we would write

$$c(i) = j$$

- The # of clusters k is usually fixed ahead of time, so $c(i) \in \{1, \dots, k\}$
- The j-th cluster C_j consists of the indices assigned label j:

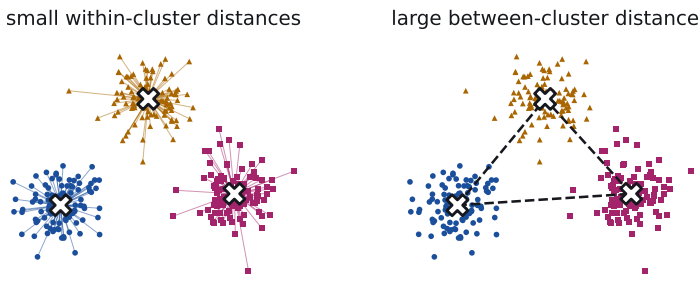
$$C_j = \{i : c(i) = j\}.$$

Criteria: What makes a good clustering?



Idea: Intuitively, we want similar examples to be assigned to the same cluster. But similarity is in the eye of the beholder!

Criteria: What makes a good clustering?



Idea: Similar within group, dissimilar between groups. These are competing demands, as examples of $k = 1$ and $k = n$ groups show.

Some common applications of clustering

- Customer segmentation. Every customer is a vector of purchase counts. Groups are "typical customers," and each customer is labeled with a group.
- Anomaly detection. Group financial transactions, use outliers to detect fraud.
- Image segmentation. Every pixel is a point in $\mathbb{R}^3$ or $\mathbb{R}^5$ (color, plus position). Groups correspond to regions of the image.

> **Note:** In every case the output is: a small list of representative points and a label of which representative point each example is most like.

Plan for today

- Intro to clustering — 5 min
- The k-means objective — 20 min
- Lloyd's algorithm — 15 min
- Example: color quantization — 5 min
- Issues with k-means — 8 min
- Surprise! — 2 min

Basic idea of k-means

- Represent each cluster of points by one of k centers

$$\mu_1, \dots, \mu_k \in \mathbb{R}^p$$

Basic idea of k-means

- Represent each cluster of points by one of k centers

$$\mu_1, \dots, \mu_k \in \mathbb{R}^p$$

- Assign example x_i to nearest center, i.e., pick $c(i)$ to minimize

$$\|x_i - \mu_{c(i)}\|^2$$

Basic idea of k-means

- Represent each cluster of points by one of k centers

$$\mu_1, \dots, \mu_k \in \mathbb{R}^p$$

- Assign example x_i to nearest center, i.e., pick $c(i)$ to minimize

$$\|x_i - \mu_{c(i)}\|^2$$

- Freedom to choose both

  centers $\mu_1, \dots, \mu_k$ and assignments $c(1), \dots, c(n)$

  makes the optimization difficult

Problem: k-means objective

Problem (k-means)

Given examples $x_1, \dots, x_n \in \mathbb{R}^p$ and a number of clusters k, solve

$$\underset{c,\, \mu_1, \dots, \mu_k}{\arg\min} \quad J(c, \mu) \;=\; \sum_{i=1}^{n} \left\| x_i - \mu_{c(i)} \right\|^2,$$

over all assignments $c : \{1, \dots, n\} \to \{1, \dots, k\}$ and all $\mu_1, \dots, \mu_k \in \mathbb{R}^p$.

Each group gets a representative point μ_j, and J adds up the squared distance from every example to its own group's representative.

> **Note:** This asks only that points be close to their own center.

How should we optimize?

Brute force? Suppose we tried every assignment. The number of ways to split n points into k nonempty groups grows rapidly with n and k:

$n = 6, \quad k = 2$	31
$n = 100, \quad k = 3$	$\approx 8.6 \times 10^{46}$
$n = 1000, \quad k = 10$	a number with 994 digits

How should we optimize?

Brute force? Suppose we tried every assignment. The number of ways to split n points into k nonempty groups grows rapidly with n and k:

$n = 6, \quad k = 2$	31
$n = 100, \quad k = 3$	$\approx 8.6 \times 10^{46}$
$n = 1000, \quad k = 10$	a number with 994 digits

Something else? In fact, there is no efficient algorithm for exactly minimizing J!

> **In the literature:** Minimizing J exactly is NP-hard even for $k = 2$ in high dimension (Aloise et al., 2009).

Key observation: Easy if either c or μ is fixed!

$$J(c, \mu) = \sum_{i=1}^{n} \| x_i - \mu_{c(i)} \|^2$$

- If assignments c are fixed, pick centroids μ_j to be the means
- If centroids μ_j are fixed, assign each point to nearest one

Key observation: Easy if either c or μ is fixed!

$$J(c, \mu) = \sum_{i=1}^{n} \| x_i - \mu_{c(i)} \|^2$$

- If assignments c are fixed, pick centroids μ_j to be the means
- If centroids μ_j are fixed, assign each point to nearest one

Idea: Alternate between these easier sub-problems!

Step 1: Fix the centers

With $\mu_1, \ldots, \mu_k$ held fixed,

$$J(c, \mu) \;=\; \sum_{i=1}^{n} \left\| x_i - \mu_{c(i)} \right\|^2$$

is a sum in which the i-th term involves only $c(i)$.

- so the terms can be minimized separately, one point at a time
- the best choice for x_i is the nearest center: $c(i) = \arg\min_j \|x_i - \mu_j\|^2$

> **Key idea:** What would normally be a search over roughly k^n assignments is n separate searches over k centers. This is why k-means is practical.

Step 2: Fix the assignment

With c held fixed, J splits into k pieces that share no variables:

$$J(c, \mu) \;=\; \sum_{j=1}^{k} \underbrace{\sum_{i \in C_j} \|x_i - \mu_j\|^2}_{\text{involves only } \mu_j} .$$

So minimize each piece separately. Each has the same form:

Problem

Given $y_1, \ldots, y_m \in \mathbb{R}^p$, which single point μ minimizes $\sum_{\ell=1}^{m} \|y_\ell - \mu\|^2$?

The answer is the mean

Lemma

Let $y_1, \dots, y_m \in \mathbb{R}^p$ have mean $\bar{y} = \frac{1}{m}\sum_\ell y_\ell$. Then for every $\mu \in \mathbb{R}^p$,

$$\sum_{\ell=1}^{m} \|y_\ell - \mu\|^2 \;=\; \sum_{\ell=1}^{m} \|y_\ell - \bar{y}\|^2 \;+\; m\|\bar{y} - \mu\|^2.$$

Proof. Write $y_\ell - \mu = (y_\ell - \bar{y}) + (\bar{y} - \mu)$ and expand:

$$\|y_\ell - \mu\|^2 = \|y_\ell - \bar{y}\|^2 + 2\,(y_\ell - \bar{y})^{\mathsf{T}}(\bar{y} - \mu) + \|\bar{y} - \mu\|^2.$$

Sum over ℓ. The middle terms add to $2\big(\sum_\ell (y_\ell - \bar{y})\big)^{\mathsf{T}}(\bar{y} - \mu) = 0$, because $\sum_\ell (y_\ell - \bar{y}) = 0$ by definition of $\bar{y}$. □

The answer is the mean

Lemma

Let $y_1, \dots, y_m \in \mathbb{R}^p$ have mean $\bar{y}$. Then for every $\mu \in \mathbb{R}^p$,

$$\sum_{\ell=1}^{m} \|y_\ell - \mu\|^2 \;=\; \sum_{\ell=1}^{m} \|y_\ell - \bar{y}\|^2 \;+\; m\|\bar{y} - \mu\|^2.$$

- the first term does not depend on μ, and the second is never negative
- so the best center of a group is its mean, and that group then costs its total squared spread about the mean

Note: The mean is the answer because the loss is a *squared* distance. Measure error by $\sum_\ell \|y_\ell - \mu\|_1$ instead and the answer is the coordinate-wise median.

Why does k-means also spread clusters out?

Let $\bar{x}$ be the mean of all n points, with the centers at the group means.

Proposition

$$\underbrace{\sum_{i=1}^{n} \|x_i - \bar{x}\|^2}_{\text{total spread, fixed}} = \underbrace{\sum_{j=1}^{k} \sum_{i \in C_j} \|x_i - \mu_j\|^2}_{\text{within groups} = J} + \underbrace{\sum_{j=1}^{k} |C_j| \cdot \|\mu_j - \bar{x}\|^2}_{\text{between groups}}.$$

Proof. Apply the Lemma to each group C_j with $\mu = \bar{x}$, then sum over j. □

Key idea: The left side does not depend on the grouping. So making J small is exactly pushing the group centers away from the overall mean.

Plan for today

• Intro to clustering	5 min
• The k-means objective	20 min
• Lloyd's algorithm	15 min
• Example: color quantization	5 min
• Issues with k-means	8 min
• Surprise!	2 min

Algorithm: Lloyd's algorithm

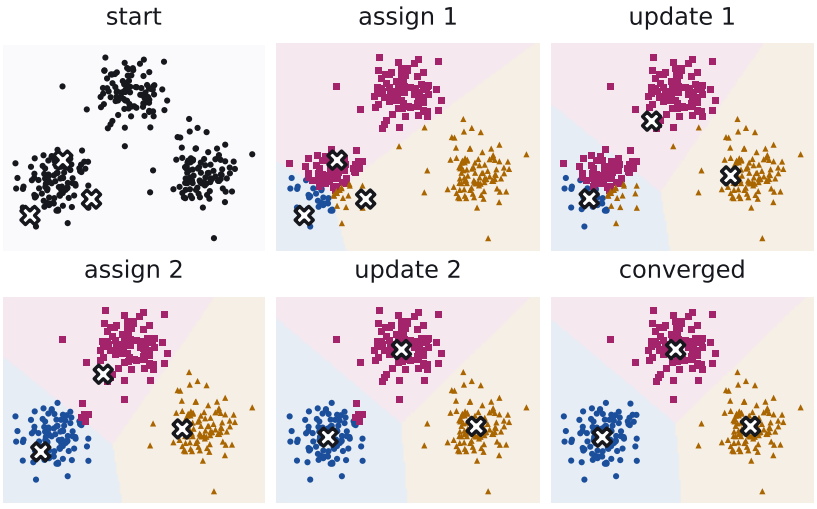
Definition (Lloyd's algorithm)

Choose starting centers $\mu_1, \dots, \mu_k$. Repeat until the assignment stops changing:

1. Assignment step. For each i, set $c(i) = \arg\min_j \|x_i - \mu_j\|^2$.
2. Update step. For each j, set $\mu_j = \frac{1}{|C_j|} \sum_{i \in C_j} x_i$.

- Step 1 is the exact minimizer over c with μ fixed
- Step 2 is the exact minimizer over μ with c fixed
- $\implies$ Neither step can ever increase J!

Lloyd's algorithm in action



Why the algorithm stops

Theorem

If “ties” are broken by a fixed rule, then Lloyd’s algorithm reaches an assignment that no longer changes, after finitely many iterations.

Proof.

- Each step is an exact minimization on either c or μ, so J never increases.

□

Why the algorithm stops

Theorem

If “ties” are broken by a fixed rule, then Lloyd’s algorithm reaches an assignment that no longer changes, after finitely many iterations.

Proof.

- Each step is an exact minimization on either c or μ, so J never increases.
- If some point changes group, the assignment step *strictly* decreases J. So no assignment can be visited twice.

□

Why the algorithm stops

Theorem

If “ties” are broken by a fixed rule, then Lloyd’s algorithm reaches an assignment that no longer changes, after finitely many iterations.

Proof.

- Each step is an exact minimization on either c or μ, so J never increases.
- If some point changes group, the assignment step *strictly* decreases J. So no assignment can be visited twice.
- There are finitely many assignments, so the algorithm must stop.

□

What the theorem does not promise

- It does not say the algorithm finds the smallest possible J
- It does not bound how far from the smallest J we end up
- It does not bound the number of iterations by a polynomial in n. In fact, there are data sets in the plane needing exponentially many.

> **Note:** So Lloyd's algorithm is a heuristic, and its starting point matters.

Algorithm: k-means++ initialization

Definition (k-means++, Arthur and Vassilvitskii, 2007)

Pick μ_1 uniformly at random from the data. Having chosen $\mu_1, \dots, \mu_{j-1}$, let $D(x) = \min_{\ell<j} \|x - \mu_\ell\|$ and pick $\mu_j = x_i$ with probability proportional to $D(x_i)^2$.

- points that are farther from existing centers are likelier candidates for the next center

Algorithm: k-means++ initialization

Definition (k-means++, Arthur and Vassilvitskii, 2007)

Pick μ_1 uniformly at random from the data. Having chosen $\mu_1, \dots, \mu_{j-1}$, let $D(x) = \min_{\ell<j} \|x - \mu_\ell\|$ and pick $\mu_j = x_i$ with probability proportional to $D(x_i)^2$.

- points that are farther from existing centers are likelier candidates for the next center
- algorithm is random, so the centers can still be badly spread, but this is rare

Algorithm: k-means++ initialization

Definition (k-means++, Arthur and Vassilvitskii, 2007)

Pick μ_1 uniformly at random from the data. Having chosen $\mu_1, \dots, \mu_{j-1}$, let $D(x) = \min_{\ell<j} \|x - \mu_\ell\|$ and pick $\mu_j = x_i$ with probability proportional to $D(x_i)^2$.

- points that are farther from existing centers are likelier candidates for the next center
- algorithm is random, so the centers can still be badly spread, but this is rare
- the initial value of the objective J is guaranteed to satisfy

$$\mathbb{E}[J] \leq 8(\ln k + 2)\, J_{min}$$

In practice

```python
from sklearn.cluster import KMeans

km = KMeans(n_clusters=3, n_init=10).fit(X)
km.labels_           # the assignment c(i)
km.cluster_centers_  # the centers mu_1, ..., mu_k
km.inertia_          # the best J over the 10 runs
```

- `init='k-means++'` is the default

In practice

```python
from sklearn.cluster import KMeans

km = KMeans(n_clusters=3, n_init=10).fit(X)
km.labels_           # the assignment c(i)
km.cluster_centers_  # the centers mu_1, ..., mu_k
km.inertia_          # the best J over the 10 runs
```

- `init='k-means++'` is the default
- `n_init=10` repeats the random initialization ten times and returns the best result

In practice

```python
from sklearn.cluster import KMeans

km = KMeans(n_clusters=3, n_init=10).fit(X)
km.labels_            # the assignment c(i)
km.cluster_centers_   # the centers mu_1, ..., mu_k
km.inertia_           # the best J over the 10 runs
```

- `init='k-means++'` is the default
- `n_init=10` repeats the random initialization ten times and returns the best result
- `km.inertia_` is the value of the objective

Plan for today

• Intro to clustering	5 min
• The k-means objective	20 min
• Lloyd's algorithm	15 min
• Example: color quantization	5 min
• Issues with k-means	8 min
• Surprise!	2 min

Example: Color quantization

A still from *The Life and Death of Colonel Blimp* (1943)

Example: Color quantization

The image has 1280×936 pixels. Each pixel has red, green, and blue values $=$ a point in $\{0, \ldots, 255\}^3$. So we need

$$1280 \times 936 \text{ pixels} \times 3 \text{ bytes per pixel} = 3{,}594{,}240 \text{ bytes}$$

to store it.

Example: Color quantization

The image has 1280×936 pixels. Each pixel has red, green, and blue values $=$ a point in $\{0, \ldots, 255\}^3$. So we need

$$1280 \times 936 \text{ pixels} \times 3 \text{ bytes per pixel} = 3{,}594{,}240 \text{ bytes}$$

to store it.

> **Idea:** What if we instead cluster the pixels into $k = 16$ groups and replace each pixel by the nearest centroid (a representative color)?

Example: Color quantization

The image has 1280×936 pixels. Each pixel has red, green, and blue values $=$ a point in $\{0, \ldots, 255\}^3$. So we need

$$1280 \times 936 \text{ pixels} \times 3 \text{ bytes per pixel} = 3{,}594{,}240 \text{ bytes}$$

to store it.

> **Idea:** What if we instead cluster the pixels into $k = 16$ groups and replace each pixel by the nearest centroid (a representative color)?

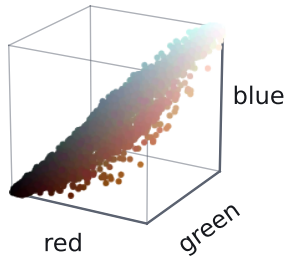
With 16 groups, each pixel needs 0.5 bytes to name its group, and the 16 representative colors cost 48 bytes. Stored this way:

$$1280 \times 936 \text{ pixels} \times 0.5 \text{ bytes per pixel} + 48 \text{ bytes} = 599{,}088 \text{ bytes}$$

Example: Pixels as points

the photograph

its pixels, as points

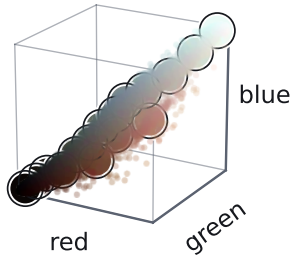


$1{,}198{,}080$ points in the "cube" $\{0, \ldots, 255\}^3$, concentrated along its diagonal.

Example: Pixels as points

the photograph

its pixels, as points



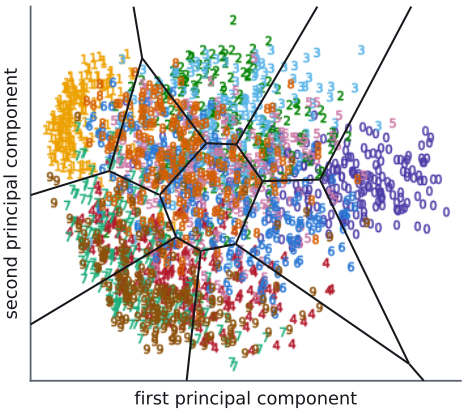
Each pixel replaced by the nearest of 16 representative points.

Example: Image compressed by a factor of six

original, 108,909 colors

16 colors, chosen by k-means

Combining techniques: Do PCA, then cluster



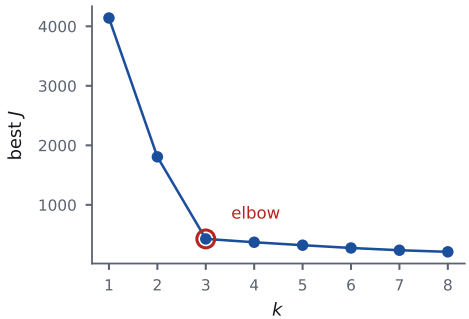
The 2000 handwritten digits from hw1, each a point in $\mathbb{R}^{784}$.

- keep the top two principal components capturing 17.6% of the variation
- run k-means++ with $k = 10$ on those two coordinates
- lines show the cluster boundaries

Plan for today

- Intro to clustering — 5 min
- The k-means objective — 20 min
- Lloyd's algorithm — 15 min
- Example: color quantization — 5 min
- Issues with k-means — 8 min
- Surprise! — 2 min

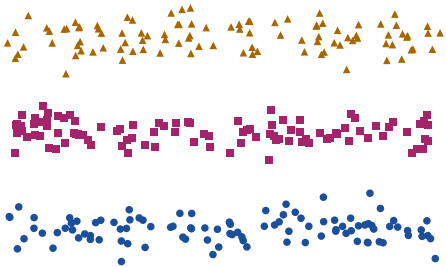
How do you choose k?



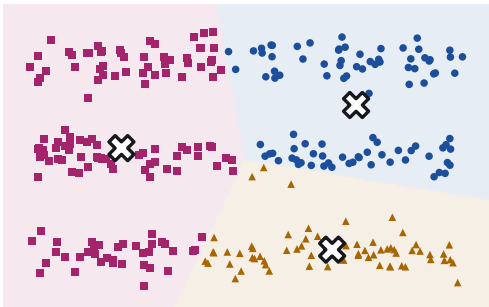
- J decreases every time k goes up, and reaches 0 at $k = n$
- So "choose k to minimize J" always answers $k = n$
- Usual practice is to plot J against k and look for an elbow

Failure 1: Stripes with $k = 3$

the grouping we wanted

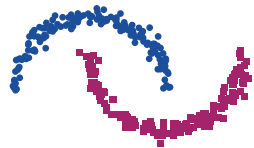


what *k*-means returns

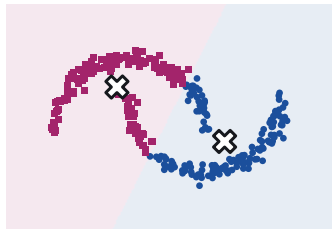


Failure 2: Non-convex clusters

the grouping we wanted



what *k*-means returns



Proposition

Fix centers $\mu_1, \ldots, \mu_k$. The set of points assigned to μ_j is an intersection of $k-1$ half-spaces, hence convex.

Failure 3: No measure of confidence

The output is a label $c(i)$ for every point, and nothing else.

- a point sitting halfway between two centers gets a label, with no note attached
- a point far from every center gets a label too
- there is no way to ask “how sure are we about this one?”

Key idea: k-means has an objective, but we have no model of the process generating the data, hence we cannot provide a measure of confidence.

What is missing

Suppose instead we wrote down a description of how the data were generated: a recipe that says how to produce a point at random, with unknown quantities in it.

- "how sure are we?" becomes a probability we can compute
- "groups of different sizes" becomes a number in the recipe
- "groups that are long and thin" becomes a matrix in the recipe
- "which k?" becomes a comparison between two recipes

Coming up: Next time we write down that recipe and fit it. The algorithm we get looks very much like Lloyd's algorithm, and working out exactly why comes after that.

Plan for today

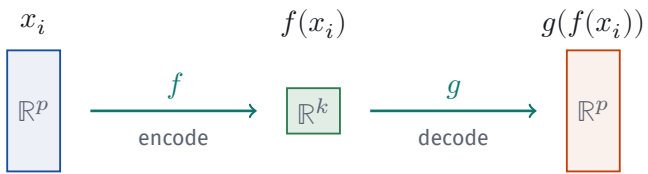
• Intro to clustering	5 min
• The k-means objective	20 min
• Lloyd's algorithm	15 min
• Example: color quantization	5 min
• Issues with k-means	8 min
• Surprise!	2 min

Recall: Dimension reduction

Problem (Dimension reduction)

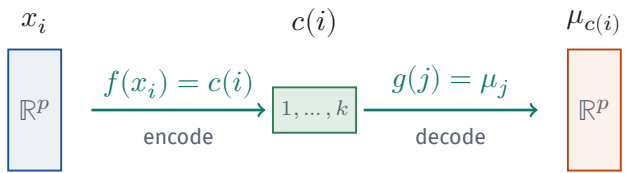
Given examples $x_1, \dots, x_n \in \mathbb{R}^p$ and a dimension $k < p$, solve

$$\arg\min_{f \in \mathcal{F},\, g \in \mathcal{G}} \sum_{i=1}^{n} \| x_i - g(f(x_i)) \|^2.$$



Surprise: It's (sort of) dimension reduction

Take $f(x_i) = c(i) \in \{1, \dots, k\}$ and $g(j) = \mu_j$. Then $g(f(x_i)) = \mu_{c(i)}$, and the objective above is J.



Key idea:

$$\sum_{i=1}^{n} \| x_i - g(f(x_i)) \|^2 = \sum_{i=1}^{n} \| x_i - \mu_{c(i)} \|^2 = J$$

Backup: the code

```
X  = np.load("mnist2000.npz")["images"]
X  = X.reshape(2000, -1)              # one row per digit
Xc = X - X.mean(axis=0)               # center the pixels

U, s, Vt = np.linalg.svd(Xc, full_matrices=False)
Z = Xc @ Vt[:2].T                     # the top two scores

km = KMeans(10, init="k-means++", n_init=10).fit(Z)

plt.scatter(Z[:, 0], Z[:, 1], c=km.labels_, s=2)
```

`Vt[:2].T` is V_2, so `Z` is the score matrix XV_2 from last lecture — everything above the `KMeans` line is PCA.

Backup: why the regions are convex

Proposition

Fix centers $\mu_1, \dots, \mu_k$. The set of points assigned to μ_j is $\{x : \|x - \mu_j\| \le \|x - \mu_\ell\|$ for all $\ell\}$, an intersection of $k - 1$ half-spaces, hence convex.

Proof. $\|x - \mu_j\|^2 \le \|x - \mu_\ell\|^2$ expands to $2\, x^{\mathsf{T}}(\mu_\ell - \mu_j) \le \|\mu_\ell\|^2 - \|\mu_j\|^2$, which is linear in x. Each such condition is a half-space, and an intersection of half-spaces is convex. □

> **Note:** With equality the condition describes the perpendicular bisector of the segment from μ_j to μ_ℓ. The regions are the cells of a Voronoi diagram.

Backup: changing the loss

The two steps of Lloyd's algorithm came from two facts: nearest-center assignment minimizes a sum of squared distances, and the mean minimizes squared distance to a set. Change the loss and both facts change.

loss	best center of a group	name
$\sum_i \|x_i - \mu\|_2^2$	the mean	k-means
$\sum_i \|x_i - \mu\|_1$	coordinatewise median	k-medians
$\sum_i d(x_i, \mu)$, μ a data point	the best data point	k-medoids

> **Note:** k-medoids needs only a table of pairwise dissimilarities, not coordinates, so it applies where there is no sensible notion of an average.