DataSci 510

lesson 8

unsupervised learning

PROFESSIONAL & CONTINUING EDUCATION
UNIVERSITY of WASHINGTON

today's agenda

- unsupervised learning
- supervised vs unsupervised
- k-means clutering
- k-means vs. k-nearest neighbor
- k-means assumptions
- where k-means fails

unsupervised learning

- the goal is to **find structure** in the data
- look at **unlabeled data** and find general patterns
- more subjective and difficult to evaluate and interpret, and hence it is far **less common** than supervised learning
- **clustering** is the most common example
    - k-means clustering
    - variable clustering / dimensionality reduction
    - word clouds (kind of)

supervised vs unsupervised

- let's take **anomaly detection** as an example, such as **fraud detection**, **inrusion detection**, **health monitoring**

1. if we have historical data where past anomalies are **labeled**, we can use **supervised learning** to predict future anomalies
2. if the data is **unlabeled**, we can use **unsupervised learning** such as **k-means** to detect clusters that look suspicious

- **no free lunch**: labeling data can be expensive and time-consuming, but so is examining and interpreting clusters

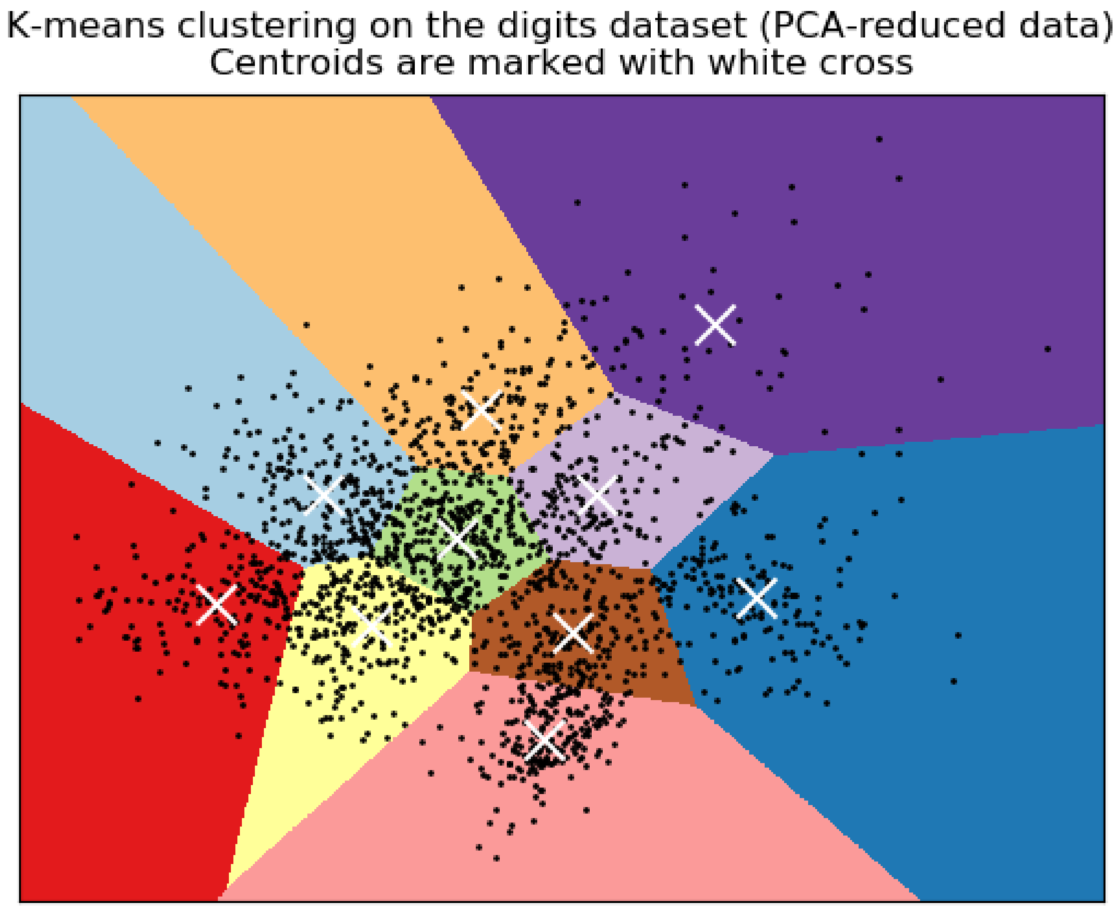
k-means characteristics

- there are **no labels**: k-means is **unsupervised**
- clusters are a **construct** we create, not something set in stone
- clusters can be hard to interpret
  - lots of **gray areas** when comparing clusters
  - k-means provides **hard clusters** but **soft clusters** are better
- there is a **supervised learning** algorithm that is very similar to k-means in how it works, called **k-nearest neighbors**
  - unlike k-means, it can be easily **evaluated**

k-means clutering

- here we chose $k = 10$
- we have two **numeric features**
- the white crosses are **cluster centroids**
- the colors show cluster **assignments**

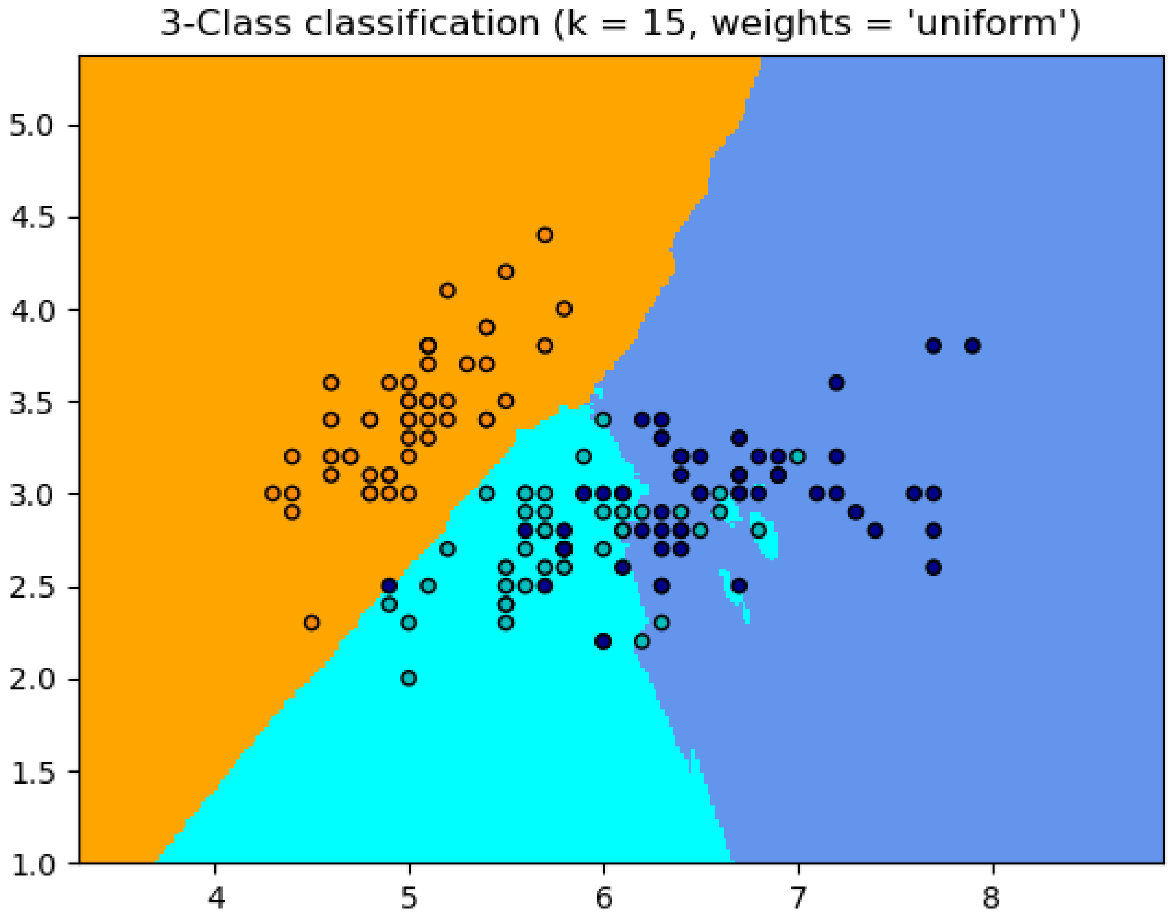
image source: [scikit-learn.org]



k-nearest neighbor

- k is the number of **neighbors** to consider
- colors show the **labels**
- the colors of regions show **decision boundaries**
- larger k makes decision boundary **smoother**

image source: [scikit-learn.org]



notebook time

we return to the lecture later

k-means algorithm implementation

1. start with k random **centroids** in the **feature space**, preferably spread out well
2. calculate the **Euclidean distance** of every row to each of the k centroids
3. **assign** each row to whichever centroid it is closest to
4. **recalculate** cluster centroids
5. **repeat** steps 2 through 4 until results stabilize

k-means assumption

- **Euclidean distance** means that
  - categorical data must be represented numerically
  - numeric data must be **normalized**
- we want to maximize variability **between clusters**
  - i.e. cluster centroids should be far away from each other
- we want to minimize variability **within clusters**
  - i.e. points belonging to the same cluster should be close to the centroid of their cluster

numeric distance metrics

Let $u = (u_1, u_2, \ldots, u_n)$ and $v = (v_1, v_2, \ldots, v_n)$.

Euclidean and Manhattan distance are part of a larger family called Minkowski distance:

- the **Euclidean distance** $\mathrm{dist}(u, v) = \sum_{i=0}^{n} (u_i - v_i)^2$ is also called the **L2-metric**
- the **Manhattan distance** $\mathrm{dist}(u, v) = \sum_{i=0}^{n} \left| u_i - v_i \right|$ is also called the **L1-metric**

Cosine similarity is a measure of the angle made between the vectors u and v. It is a good choice when directionality matters more than position (e.g. word vectors):

- the **cosine similarity** is given by $\mathrm{dist}(u, v) = \frac{u \cdot v}{||u|| \cdot ||v||}$ where the product at the numerator is a **dot product** but the product in the denominator is a simple product

categorical distance metrics

- the **Hamming distance** lines up strings (same length) and counts the number of positions that don't match
- the **Levenshtein distance** (also called the **edit distance**) between any two strings is the total cost of the number of insertions (costs 1), deletions (costs 1), substitutions (costs 2) needed to convert one string into the other
- the **Jaccard index** measures the size of the **intersection** of characters divided by size of the **union** of characters: $J(A, B) = 1 - \frac{|A \cap B|}{|A \cup B|}$, for example $J(\mathrm{beer}, \mathrm{bear}) = 1 - \frac{3}{4}$

discussion

The **weighted Jaccard index** is based on the minimum number of times (denoted m_i) that a letter appears in either word and the number of times it appears in both words **combined** (denoted M_i). It is given by:

$$J'(A, B) = 1 - \frac{\sum m_i}{\sum M_i}$$

What is $J'(\text{beer}, \text{bear})$?

break time

discussion

- in the next slide, you are presented with 3 different situations where k-means **didn't work as intended**
- look at the scatter plot and do your best to **explain** why k-means didn't work as intended in each situation
- propose an **approach** for what to do to **avoid** getting in such a trap
- even though the examples are 2 dimensional, your approach should work even when we have more than 2 features and **cannot** rely on **data visualization**

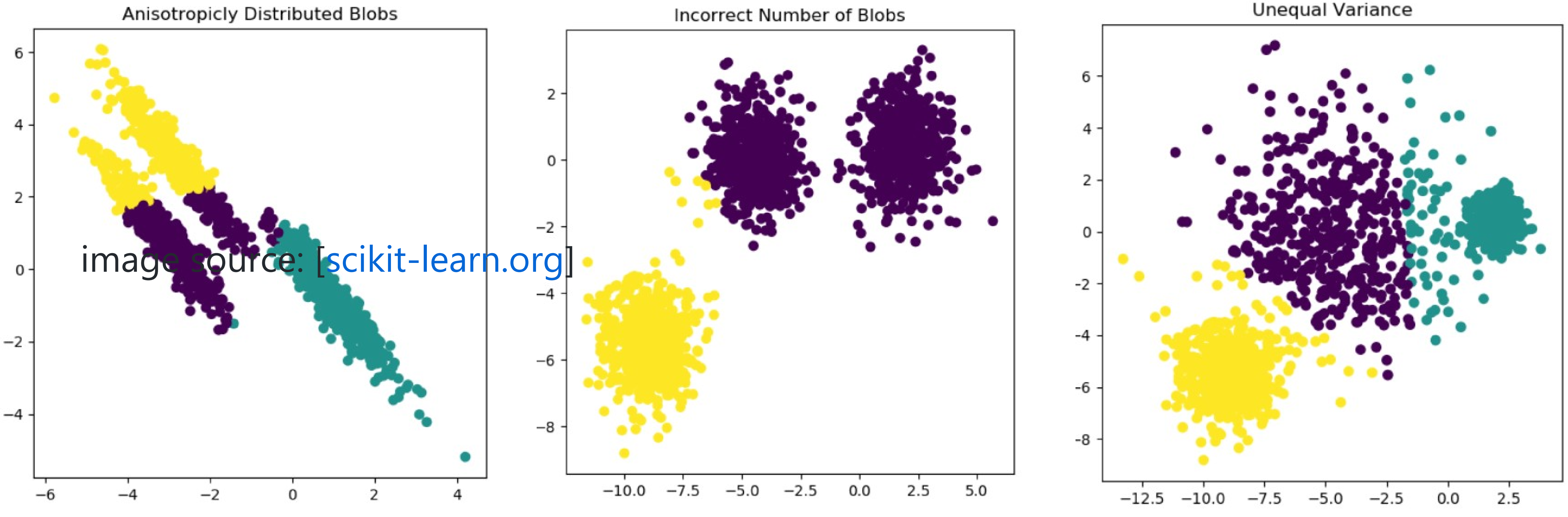
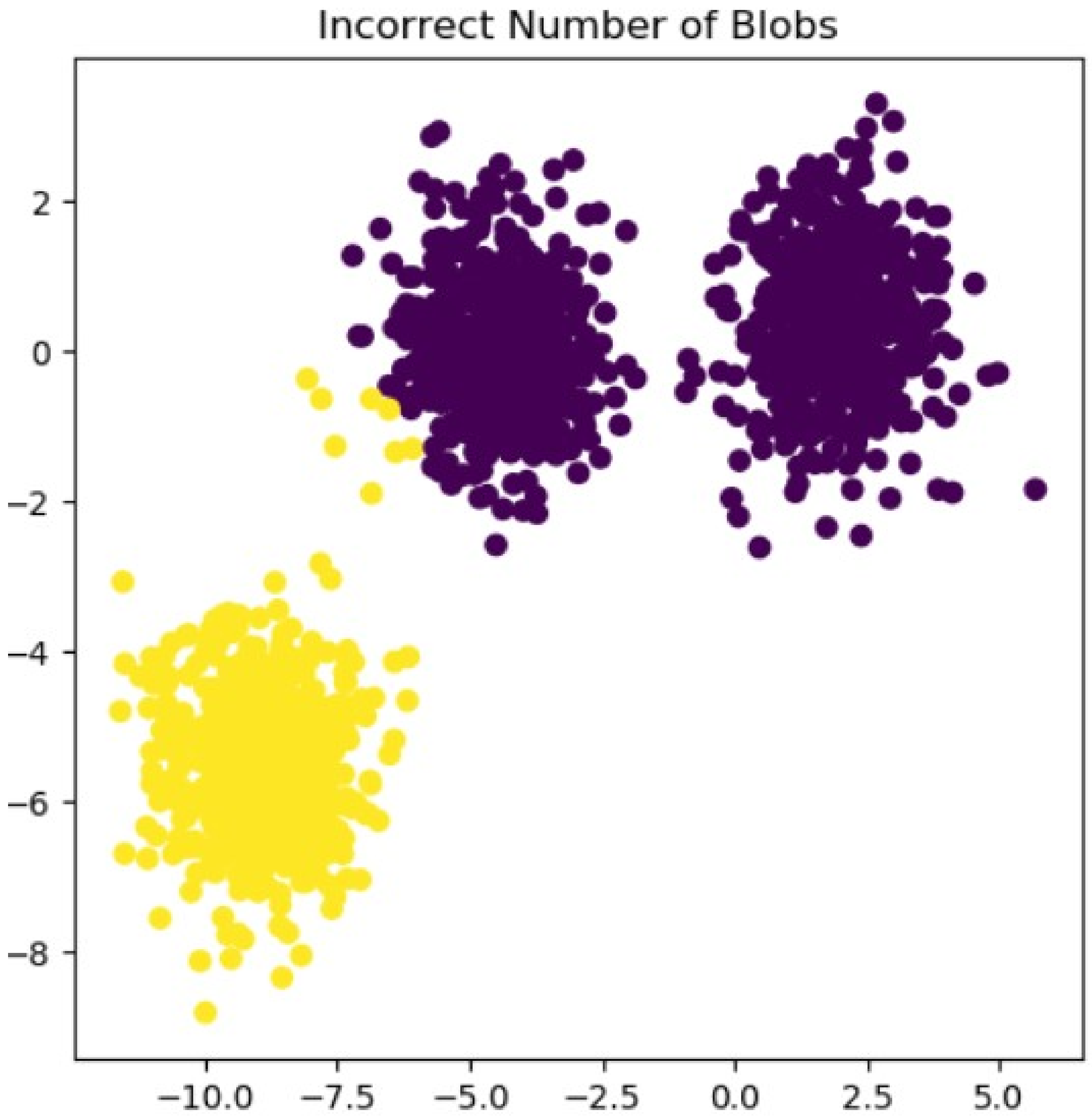


image source: [scikit-learn.org]

k-means fail # 1

- we are too **conservative** in our choice of k
- we can catch this by **increasing** k and noticing a big drop in within-cluster **variability**

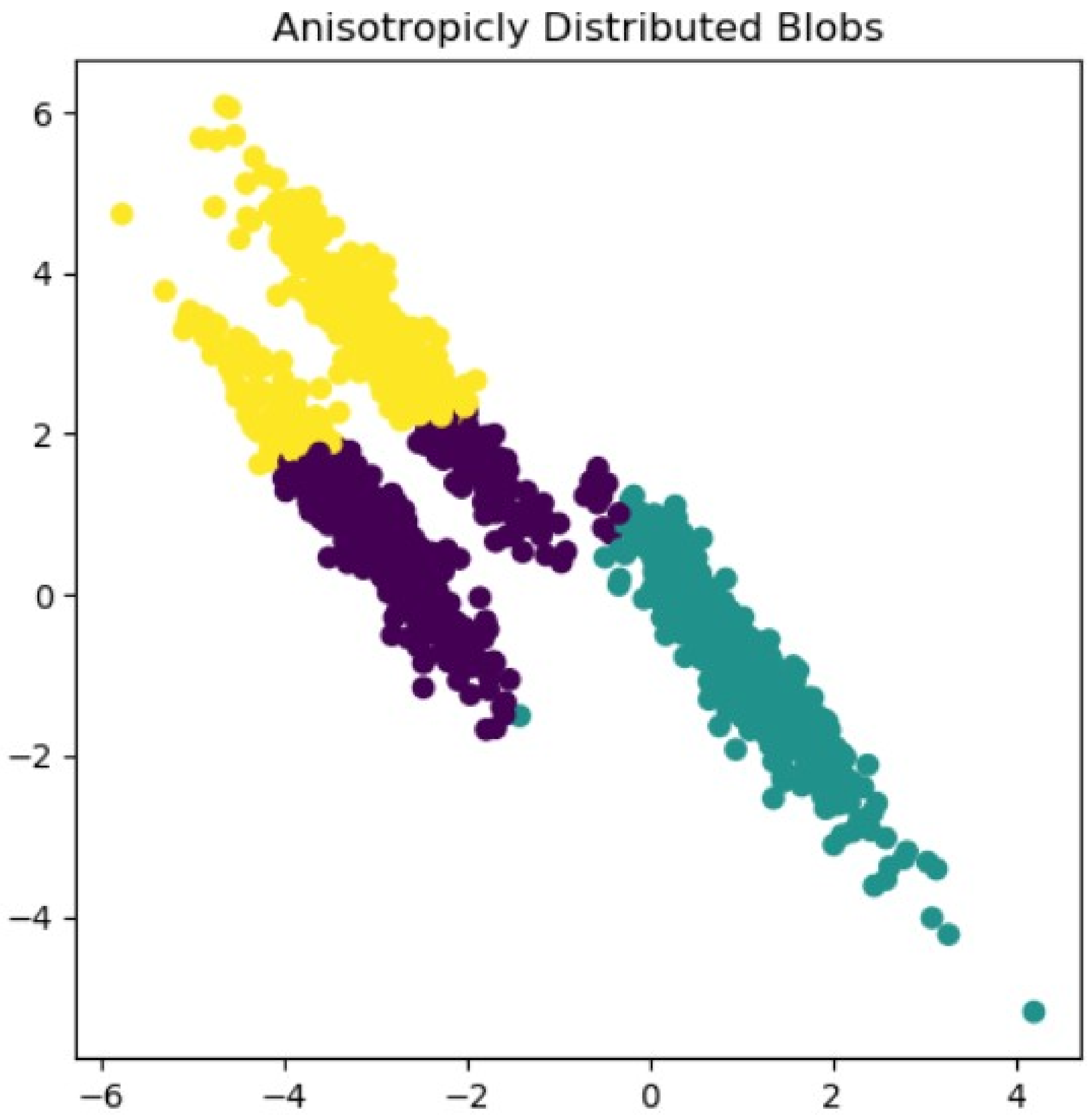
image source: [scikit-learn.org]



k-means fail # 2

- data distributions follow slanted shapes
- avoid this by excluding **highly correlated** features
- we can try certain **transformations**, e.g. rotation or PCA

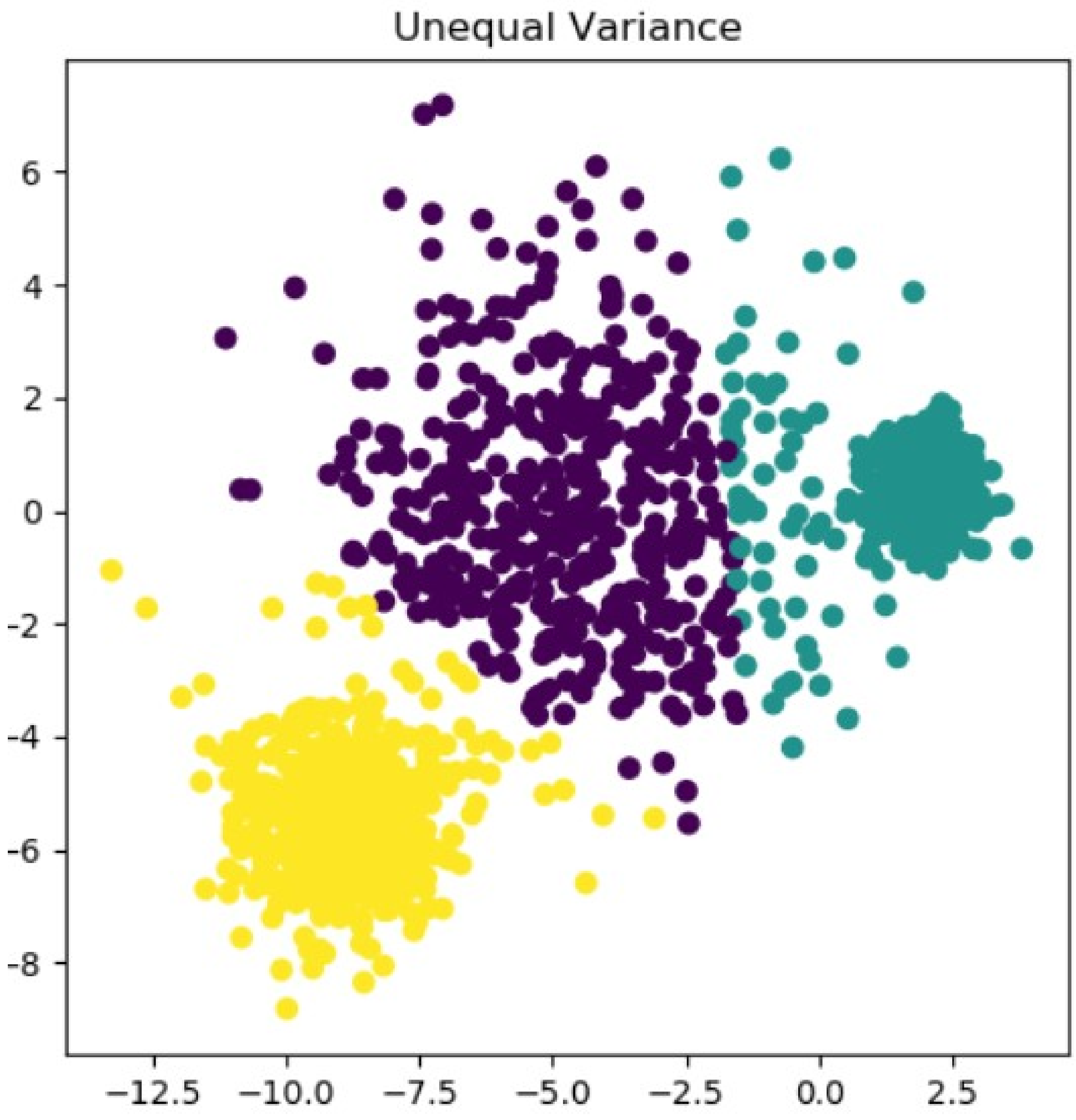
image source: [scikit-learn.org]



k-means fail # 3

- the middle cluster looks like it should own more of the points around it
- this is a **tough** one
- reruning with different seeds can help identify **borderline points**

image source: [scikit-learn.org]



the end