CSC 480/580 Principles of Machine Learning

03 Geometry & Nearest Neighbors

Jason Pacheco

Department of Computer Science

Outline

- Nearest neighbor methods for supervised learning

- Clustering and the k-means algorithm: a first taste of unsupervised learning

Nearest neighbors for supervised learning

Motivation

Example Given student course survey data, predict whether Alice likes Algorithms course

Idea Find a student ``similar'' to Alice & has taken Algorithm course before, say Jeremy

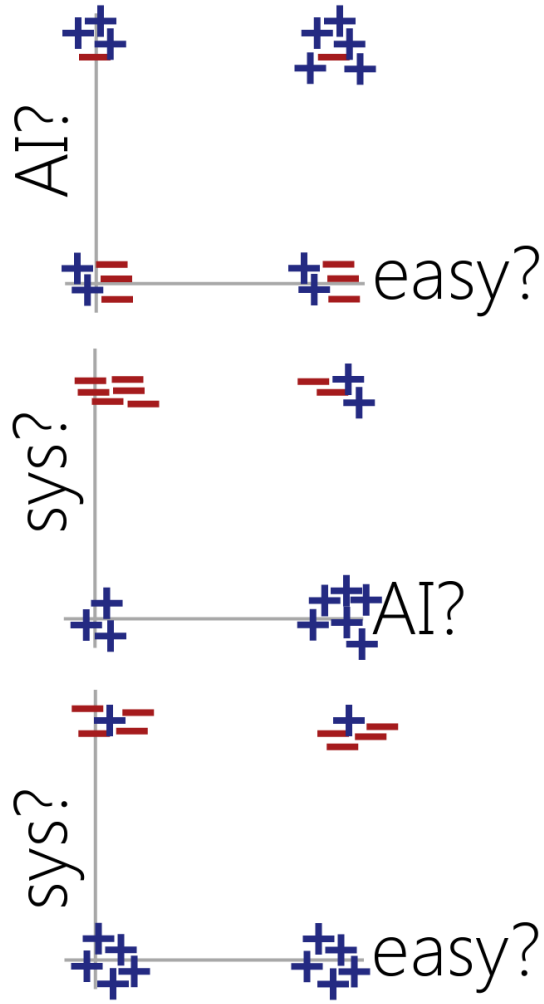
- If Jeremy likes Algorithms, then Alice is also likely to have the same preference.
- Or even better, find *several* similar students

- Prediction = mapping inputs to outputs
- Inputs = *features* that can be viewed as points in some space (possibly high-dimensional)
- “Similarity” = “distance” in feature space
- Suggests a **geometric** view of data

Example: Course Recommendation

Rating	Easy?	AI?	Sys?	Thy?	Morning?
+2	y	y	n	y	n
+2	y	y	n	y	n
+2	n	y	n	n	n
+2	n	n	n	y	n
+2	n	y	y	n	y
+1	y	y	n	n	n
+1	y	y	n	y	n
+1	n	y	n	y	n
0	n	n	n	n	y
0	y	n	n	y	y
0	n	y	n	y	n
0	y	y	y	y	y
-1	y	y	y	n	y
-1	n	n	y	y	n
-1	n	n	y	n	y
-1	y	n	y	n	y
-2	n	n	y	y	n
-2	n	y	y	n	y
-2	y	n	y	n	n
-2	y	n	y	n	y

Features



ML begins by mapping data to feature vectors

Represented as points in 5-dimensional space for this example

That's too many dimensions to plot…so we look at 2D projections…

Bottom line: in real-world data, examples with similar features likely have similar labels

Measuring nearest neighbors

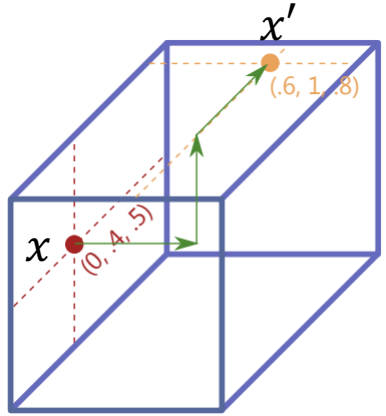
- Oftentimes convenient to work with feature $x \in \mathrm{R}^d$ $\sqrt{(0-.6)^2+(.4-1)^2+(.5-.8)^2} = \sqrt{0.81} = 0.9$
- Distances in R^d:

notation $x(f)$: $x = (x(1), \ldots, x(d))$

  - (popular) Euclidean distance $d_2(x, x') = \sqrt{\sum_{f=1}^{d} \left(x(f) - x'(f)\right)^2}$
  - Manhattan distance $d_1(x, x') = \sum_{f=1}^{d} |x(f) - x'(f)|$

$|0-.6| + |.4-1| + |.5-.8| = 1.5$

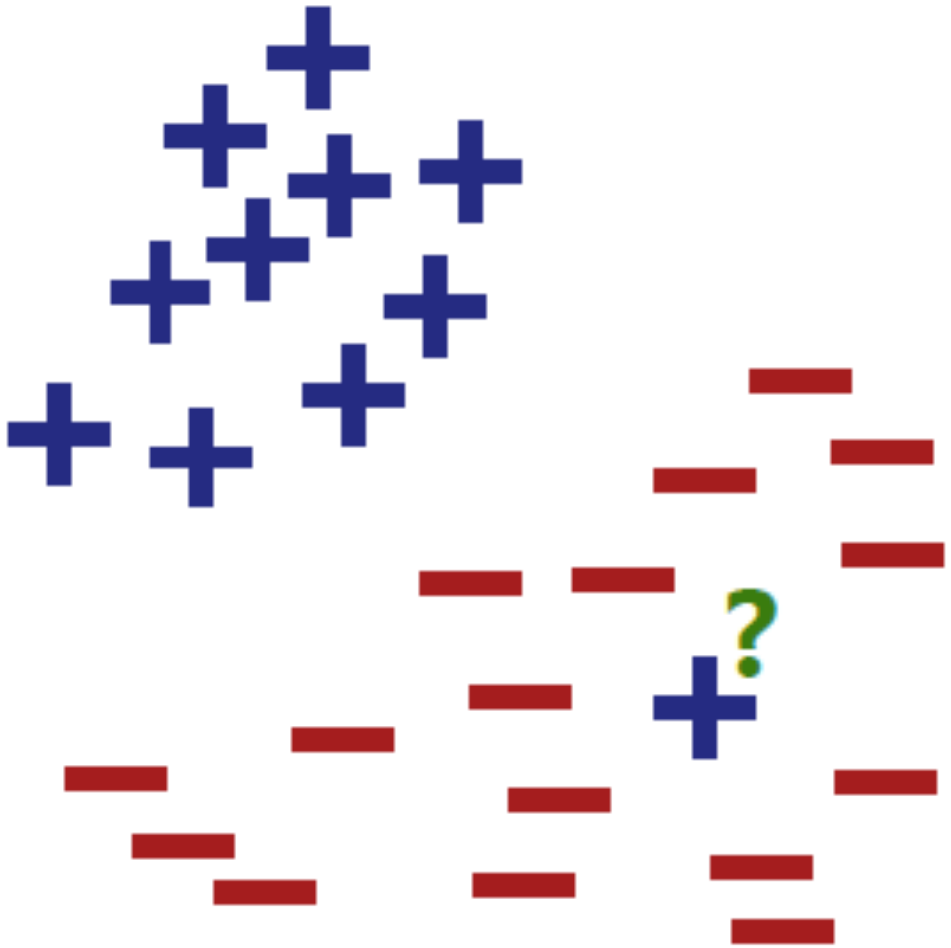
  - If we shift a feature, would the distance change?
  - What about scaling a feature?



- How to extract features as **real values**?
  - Boolean features: {Y, N} -> {0,1}
  - Categorical features: {Red, Blue, Green, Black}
    - Convert to {1, 2, 3, 4}?
    - Better: one-hot encoding: (1,0,0,0), .., (0,0,0,1) (IsRed?/isGreen?/isBlue?/IsBlack?)

Not good – imposes implicit bias e.g. Red is closer to Blue than Green

Nearest Neighbor Classification



Test example ? will be classified as + but should be –

Problem: predicting using a nearest neighbor's label can be sensitive to noisy data

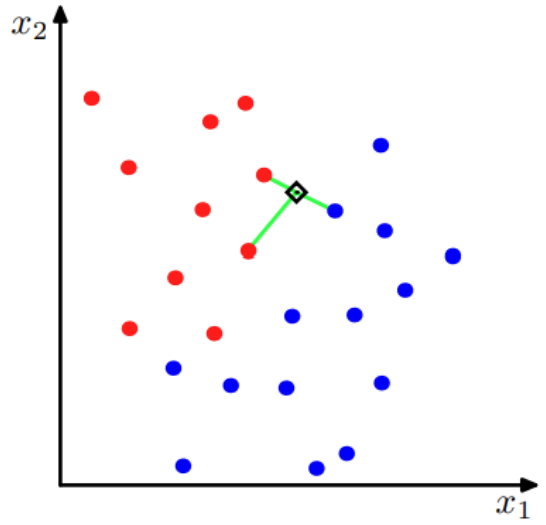
How to mitigate this?

k-nearest neighbors (k-NN): main concept

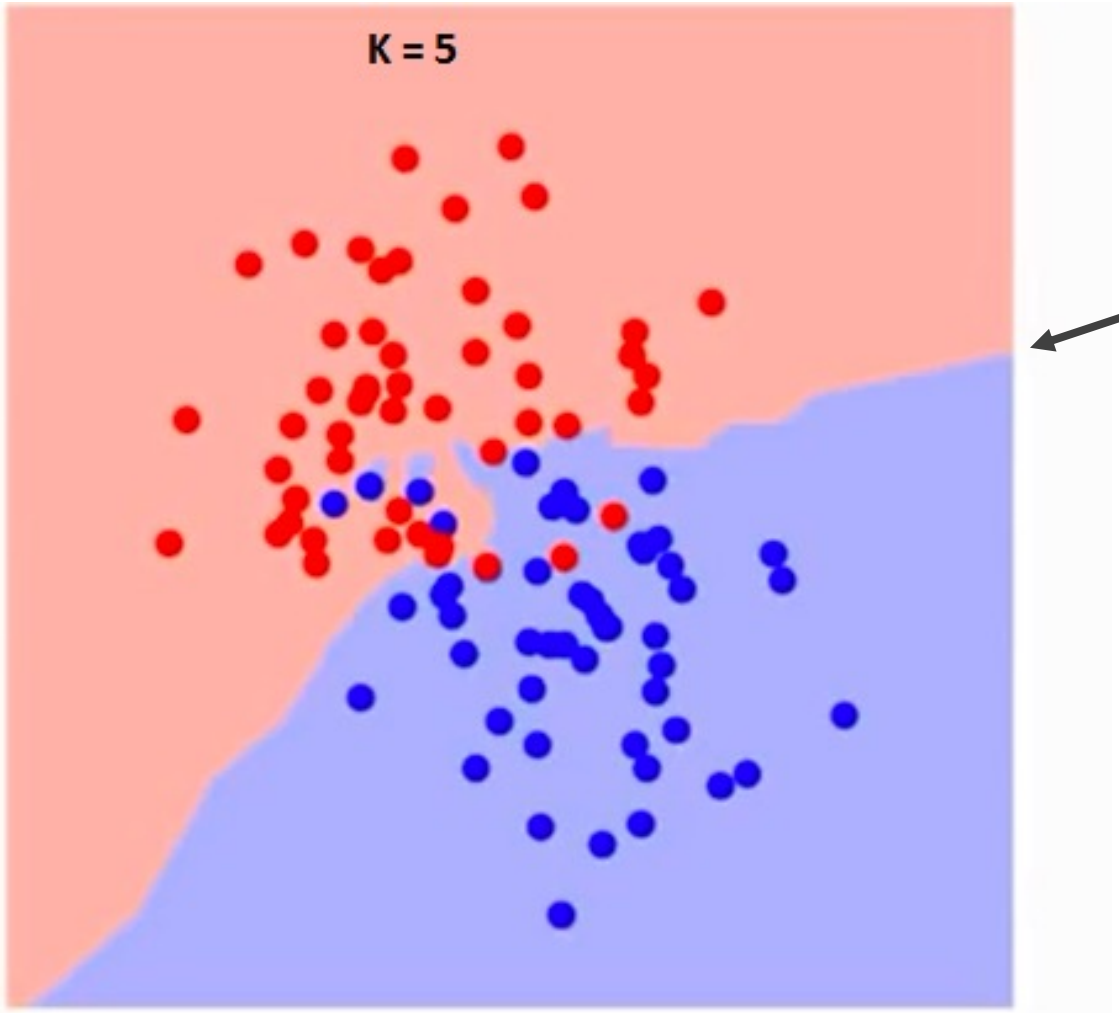
- Training set: $S = \{ (x_1, y_1), \dots, (x_m, y_m)\}$

- **<u>Inductive bias</u>**: given test example x, its label should resemble the labels of **nearby points**

- k-NearestNeighbor(input: x)
  - find the k nearest points to x from S; call their indices $N(x)$
  - output:
    - (classification) the majority vote of $\{y_i : i \in N(x)\}$
    - (regression) the average of $\{y_i : i \in N(x)\}$



k-NN classification example



decision boundary

k-NN classification: pseudocode

- Training is trivial: store the training set
- Test (when $y \in \{-1, +1\}$):

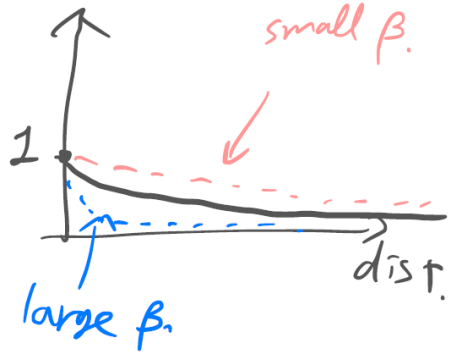
list →
append to list →
sort by distance →
Majority vote of $\{y_i : i \in N(x)\}$ →

Algorithm 3 KNN-PREDICT($\mathbf{D}$, K, $\hat{\boldsymbol{x}}$)

```
S ← [ ]
for n = 1 to N do
    S ← S ⊕ ⟨d(x_n, x̂), n⟩        // store distance to training example n
end for
S ← SORT(S)                        // put lowest-distance objects first
ŷ ← 0
for k = 1 to K do
    ⟨dist,n⟩ ← S_k                 // n this is the kth closest data point
    ŷ ← ŷ + y_n                    // vote according to the label for the nth training point
end for
return SIGN(ŷ)                     // return +1 if ŷ > 0 and −1 if ŷ < 0
```

- Time complexity (assuming distance calculation takes $O(d)$ time)
  - $O(m\,d\ + m \log m\ + k\ ) = O\big(m(d\ + \log m)\big)$
- Faster nearest neighbor search: k-d trees, locality sensitive hashing

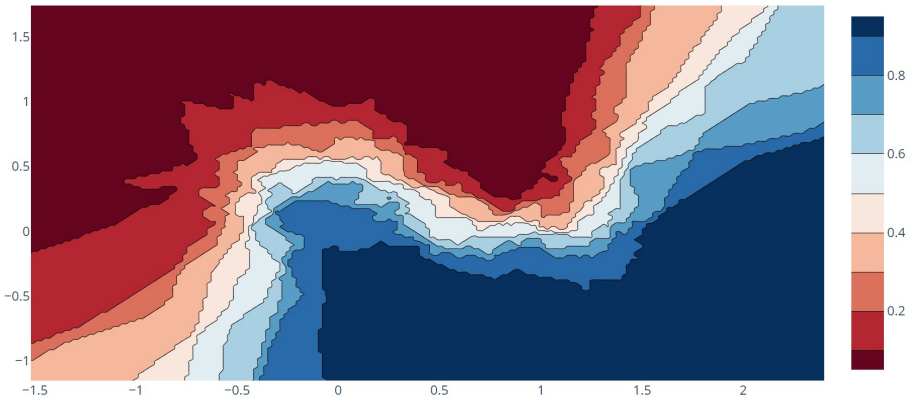
Variations



- Classification
  - Recall the majority vote rule: $\hat{y} = \underset{y \in \{1,\ldots,C\}}{\operatorname{argmax}} \sum_{i \in N(x)} 1\{y_i = y\}$
  - Soft weighting nearest neighbors: $\hat{y} = \underset{y \in \{1,\ldots,C\}}{\operatorname{argmax}} \sum_{i=1}^{m} w_i \, 1\{y_i = y\}$,

    where $w_i = \exp(-\beta \, d(x, x_i))$, or $\frac{1}{1+d(x,x_i)^{\beta}}$

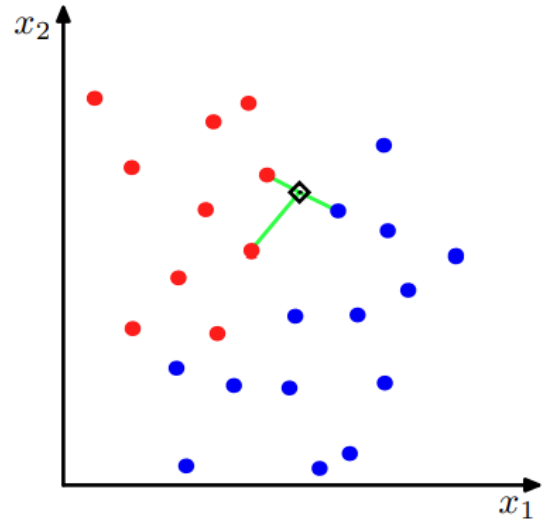
- Class probability estimates
  - $\hat{P}(Y = y \mid x) = \frac{1}{k} \sum_{i \in N(x)} 1\{y_i = y\}$
  - Useful for label uncertainty quantification



  - Application: safety-critical classification (e.g. medical diagnosis)

Hyperparameter tuning in k-NN

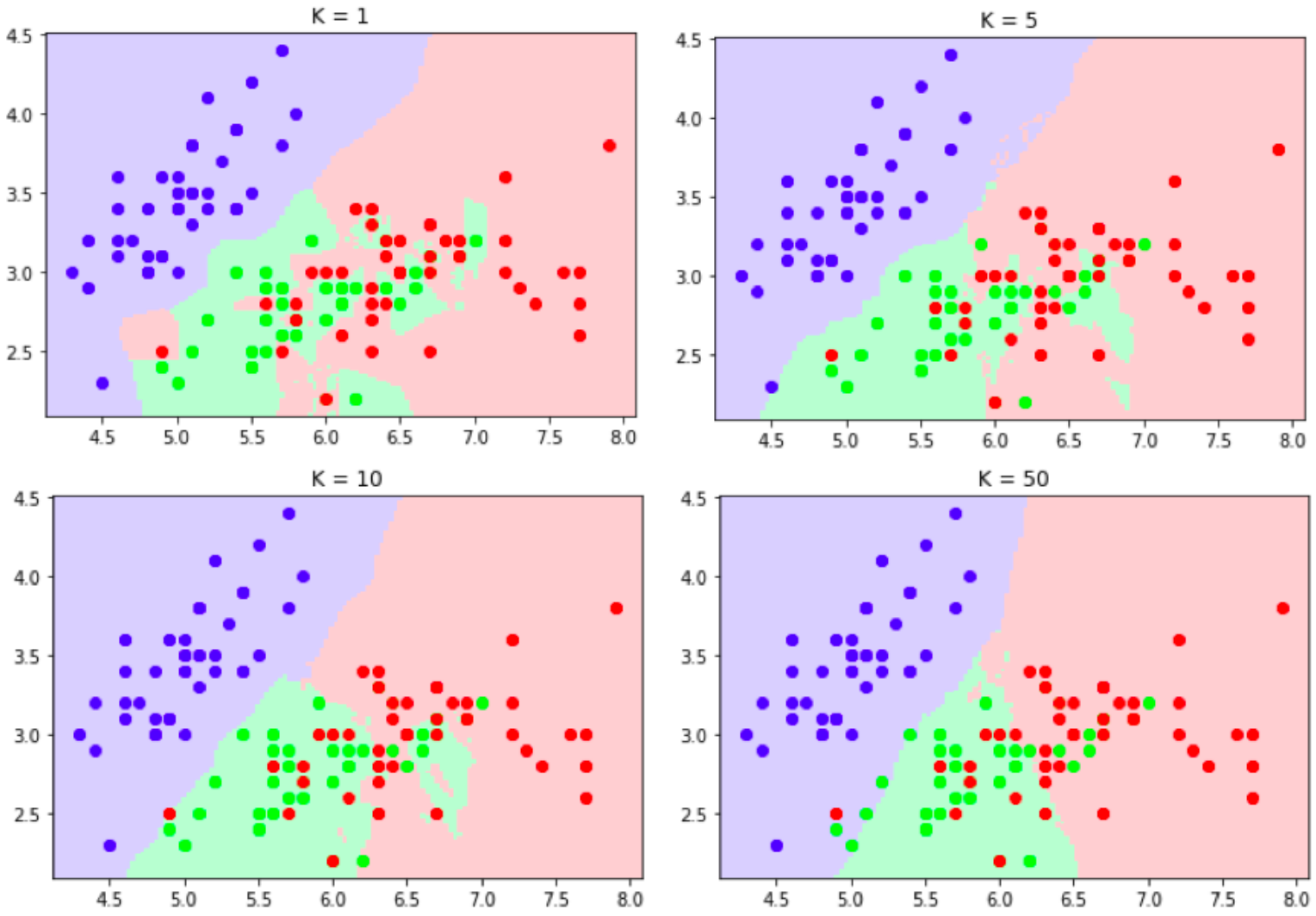
- Q: What are the hyperparameters for k-NN?
  - k, # of neighbors used for prediction



- $k = 1$:
  - Training error = 0, overfitting
- $k = m$:
  - Output a constant (majority class) prediction, underfitting

- From last lecture: can use hold-out validation set to perform hyperparameter tuning, i.e., choose k
  - We'll see this in our HW1

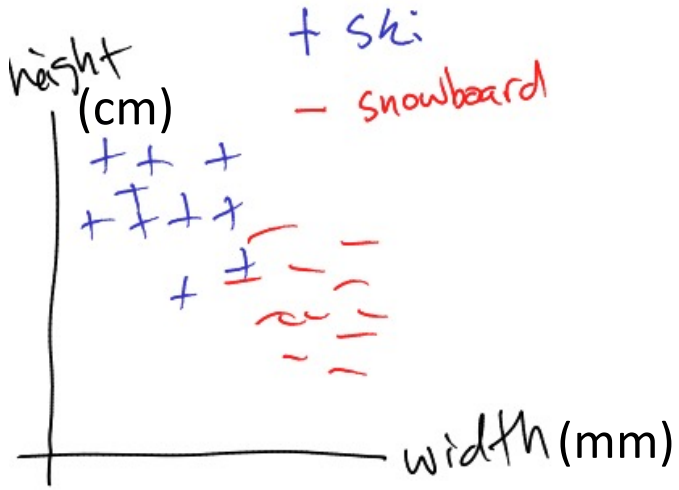
Hyperparameter tuning in k-NN



Feature issue 1: scaling

- Features having different scales can be problematic

- Ex: ski vs. snowboard classification

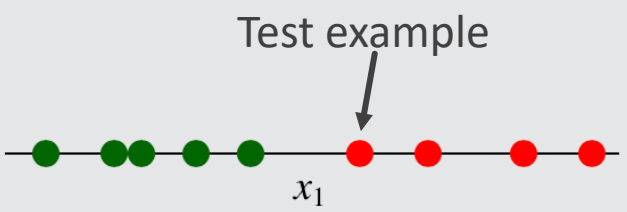


feature ‘width’ becomes less useful.. Larger error rate

- One solution: feature standardization (later in the course)

Feature issue 2: distracting features

here's a case in which there is one relevant feature x_1 and a 1-NN rule classifies each instance correctly



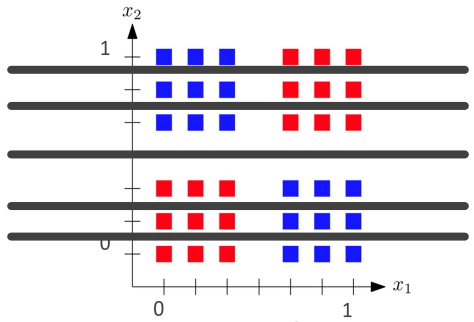
consider the effect of an irrelevant feature x_2 on distances and nearest neighbors

Informativeness scores

- Recall: did we filter out irrelevant features in training decision trees? If so, how?
- To address this in NN methods: feature selection (later in the course)

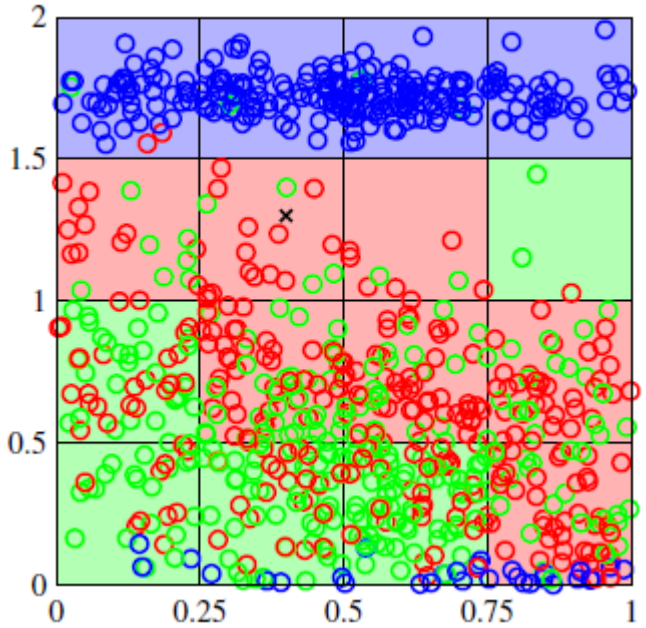
Comparison (feature $x \in \mathrm{R}^d$)

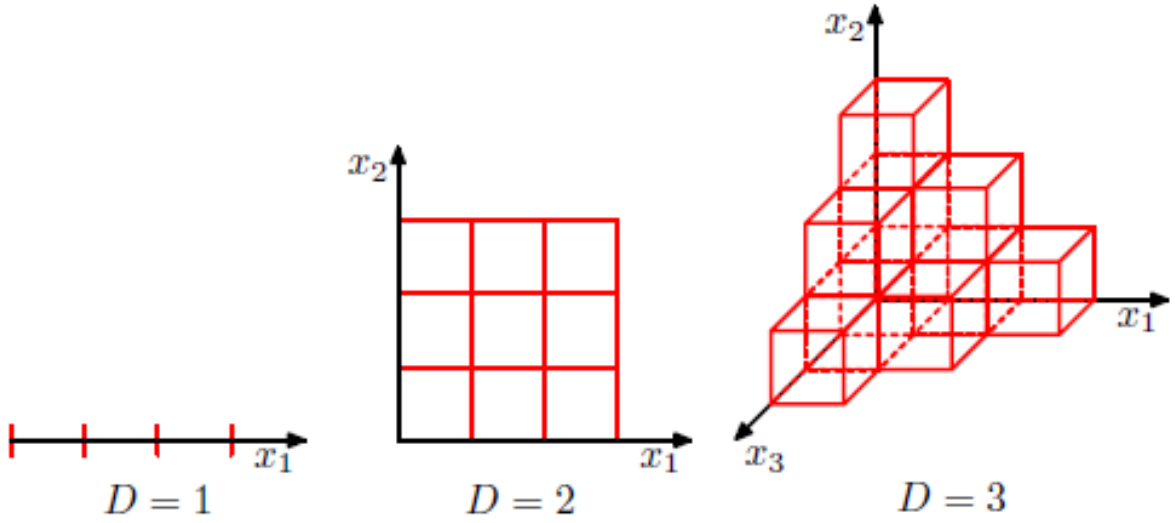
	Decision Tree	k-NN
• Interpretability	High	Medium (example-based)
• Sensitivity to irrelevant features	Low	High
• training time (m=training set size)	$O(\#\text{nodes} \cdot d \cdot (m + m \log m))$	0
• test time per example	$O(\text{depth})$	$O\big(m(d + \log m)\big)$



Curse of Dimensionality

Divide space into regular cells of equal side length





Number of required cells grows exponentially in dimension!

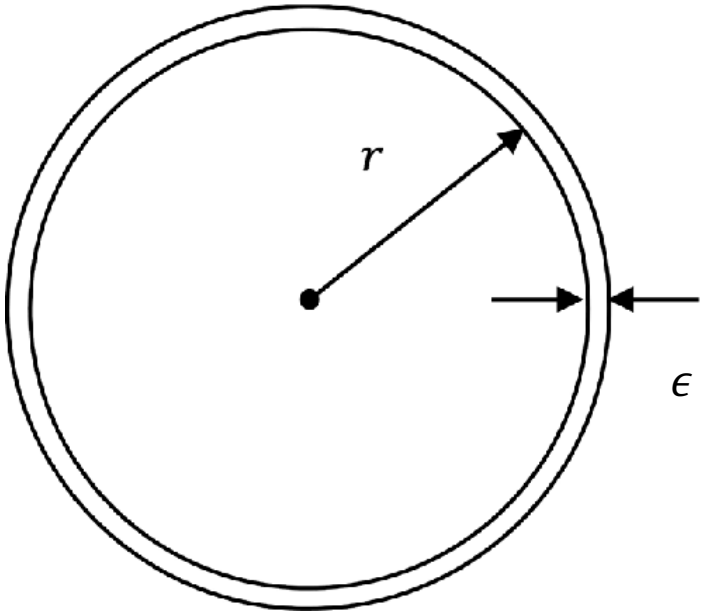
Implications for high-dimensional data:

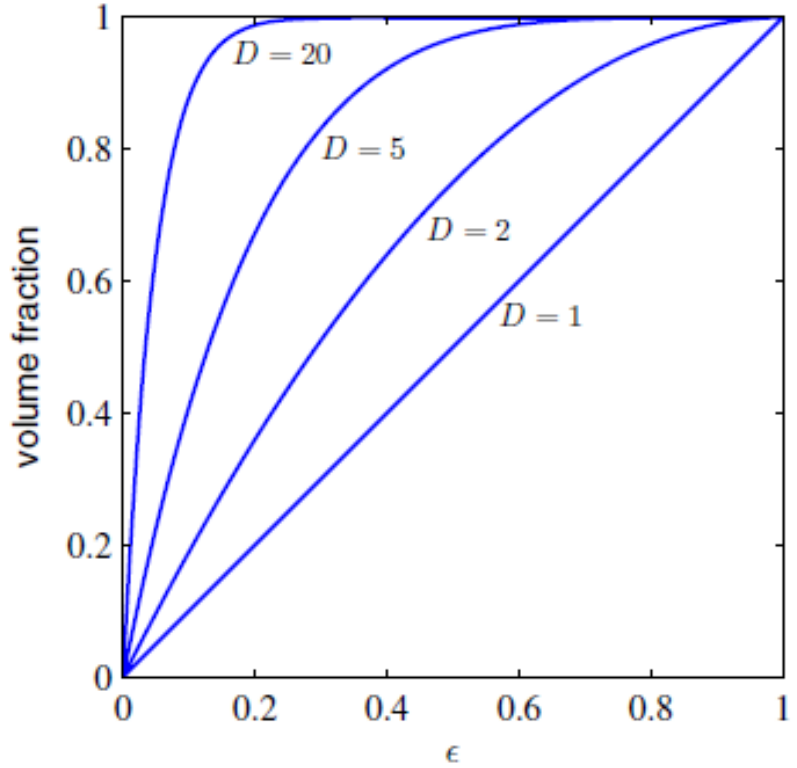
- Data uniformly distributed in space may occupy cells *sparsely*
- Nearest neighbors may actually be far away => k-NN classifier may not perform very well

Curse of Dimensionality – Distance Weirdness

- Consider *D*-dimensional hypersphere of radius *r=1*
- What is the fraction of volume within shell of width ϵ?

$$1 - (1 - \epsilon)^D$$

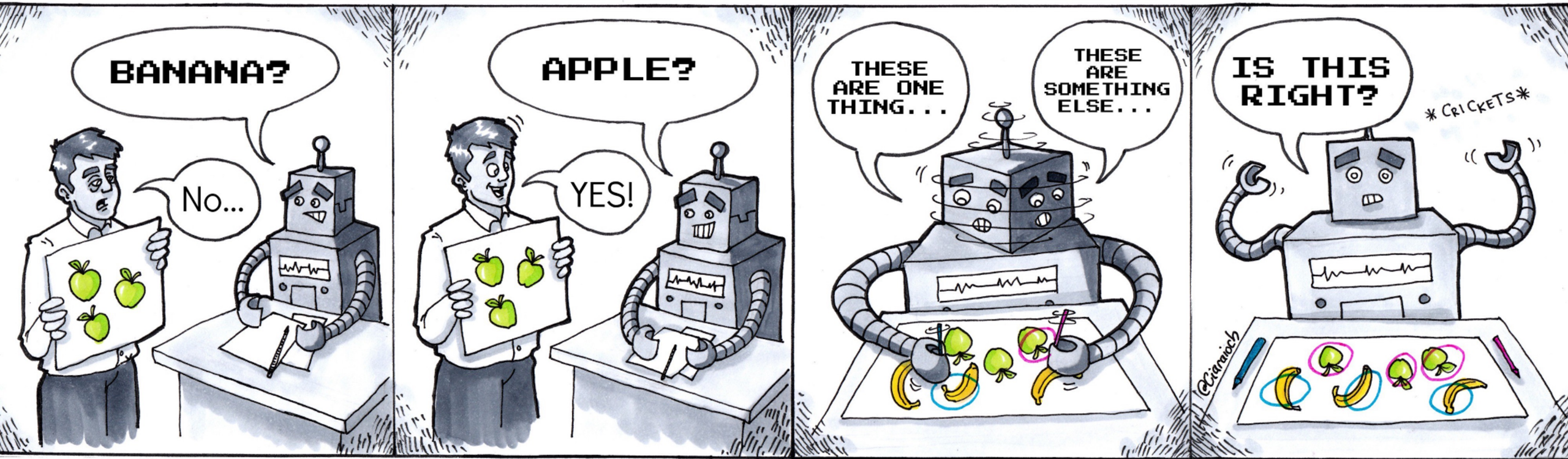




- Total volume of hypersphere concentrates onto shell at the surface!

Dasgupta, "Strange Effects in High Dimensions", 2010

Clustering and k-means algorithm: a first taste of unsupervised learning



Supervised Learning

Unsupervised Learning

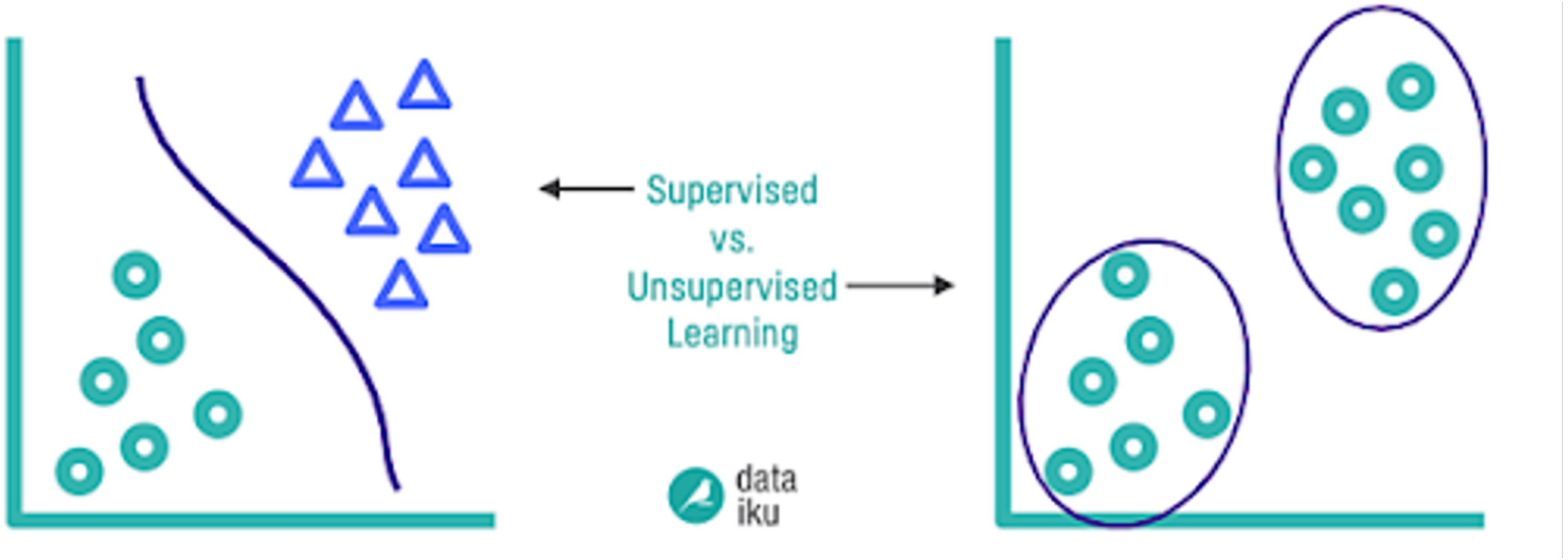
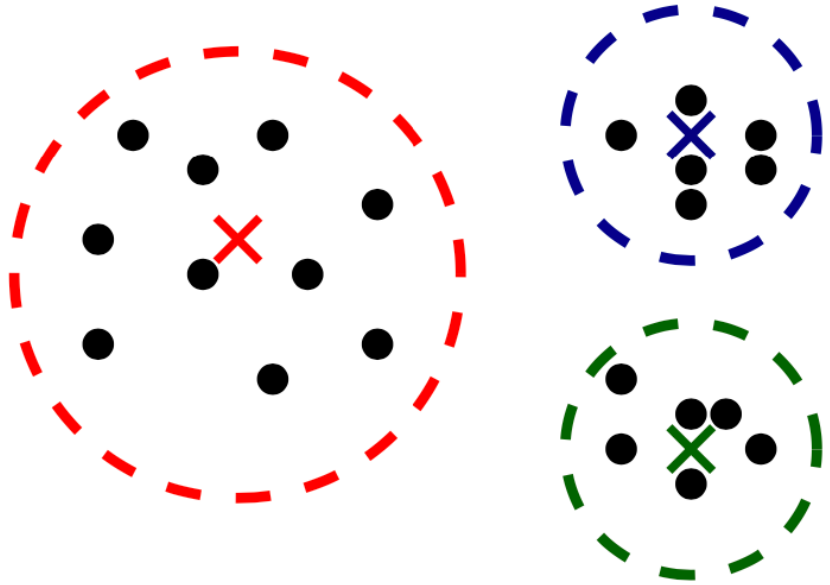


Image from Dataiku

Clustering



- Input: k: the number of clusters (hyperparameter)

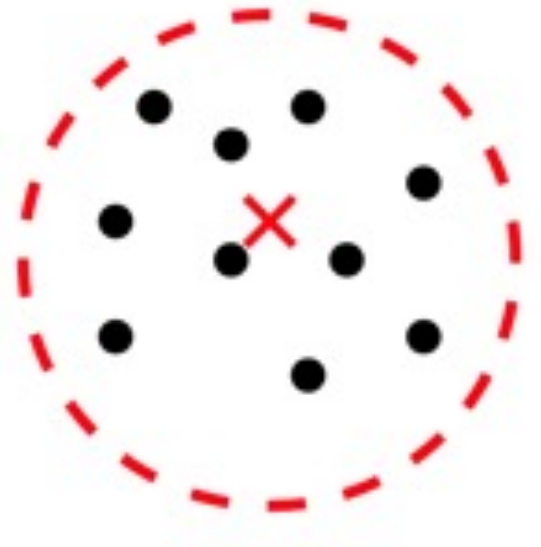
  dataset: $S = \{x_1, \dots, x_n\}$

- Output:
  - partition $\{G_i\}_{i=1}^{k}$ s.t. $S = \cup_i G_i$ (disjoint union).
  - often, we also obtain 'centroids' – representatives for each group

- Q: what would be a reasonable definition of centroids?

Centroid of a point set

- Intuition: a centroid c of point set $S = \{z_1, \dots, z_n\}$ should be close to all points in that set

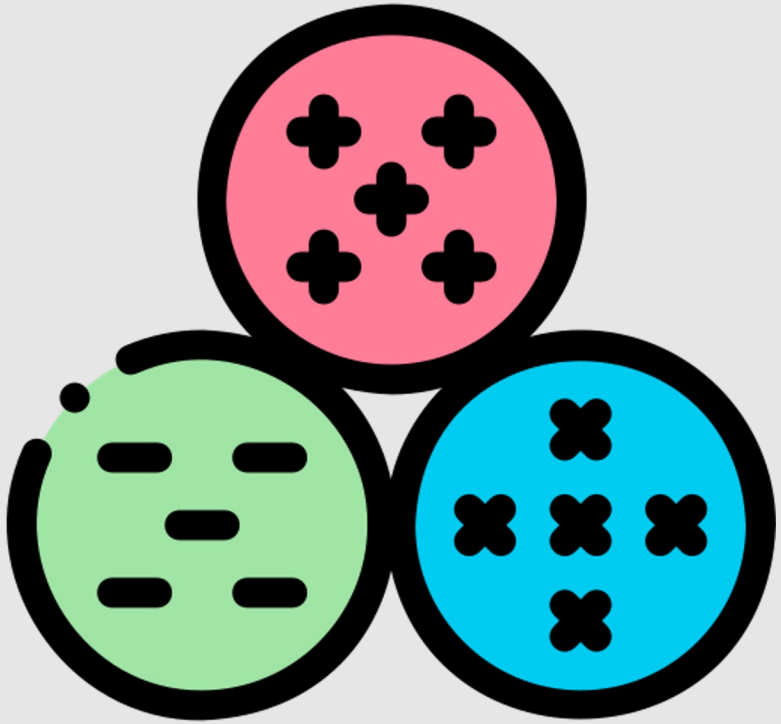


- A reasonable definition: c minimizes the sum of squared distances to points in S:

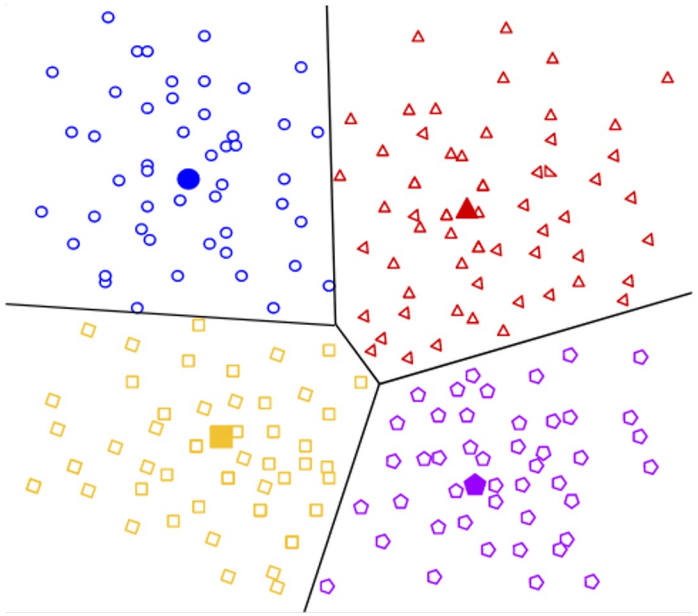
$$c = \underset{w \in \mathbb{R}^d}{\operatorname{argmin}} \|z_1 - w\|^2 + \cdots + \|z_n - w\|^2$$

- When $d = 1$: $c = \frac{1}{n}\sum_{i=1}^{n} z_i$ (*) = their "center of gravity"
- Fact: (*) is still true for general d

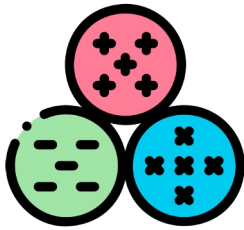
K-means algorithm [Lloyd'82]: Intuition



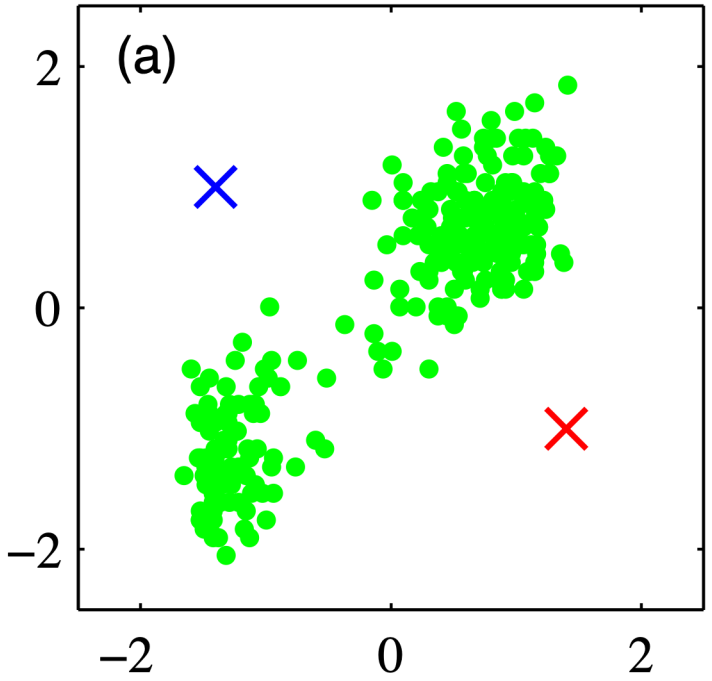
K-means algorithm



- Initialize Cluster Centroids
- Until Convergence:
  - **Cluster Assignment:** for each point, assign it to the cluster with the nearest centroid
  - **Recompute Centroid:** for each cluster, recompute its centroid to be the cluster mean



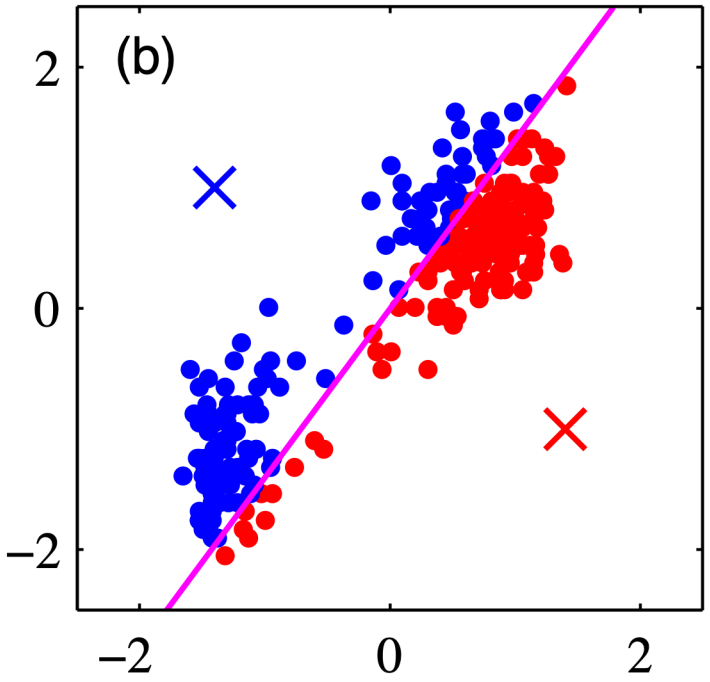
Initialization



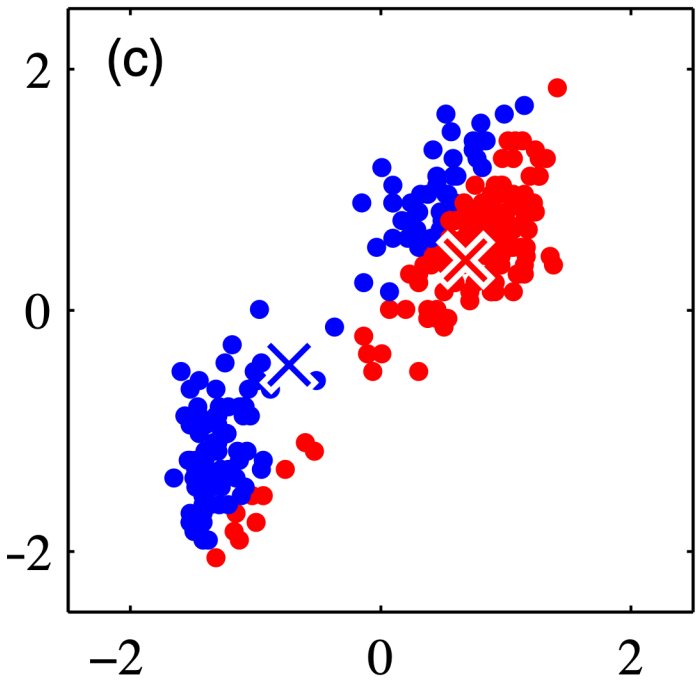
Random initialization of centroids c_1 and c_2

What would the cluster assignment look like given these centroids?

Iteration 1

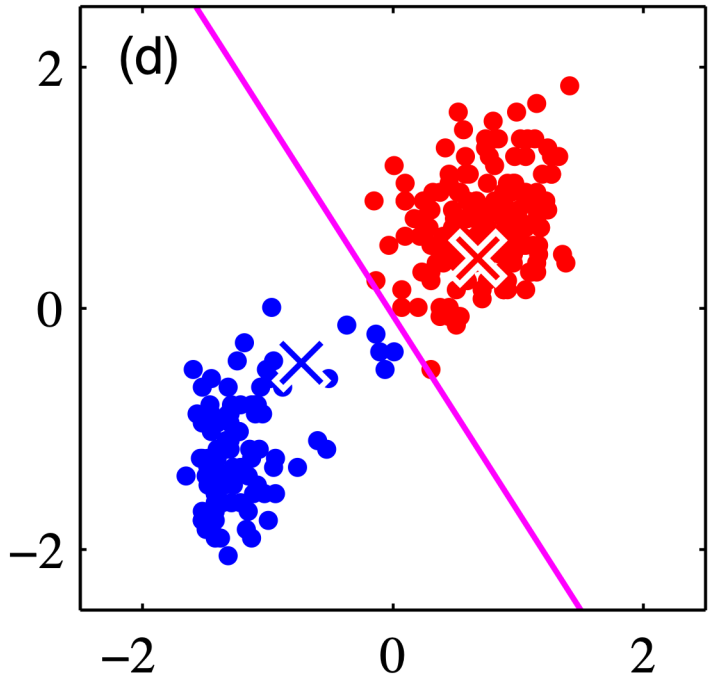


(A) update the cluster assignments

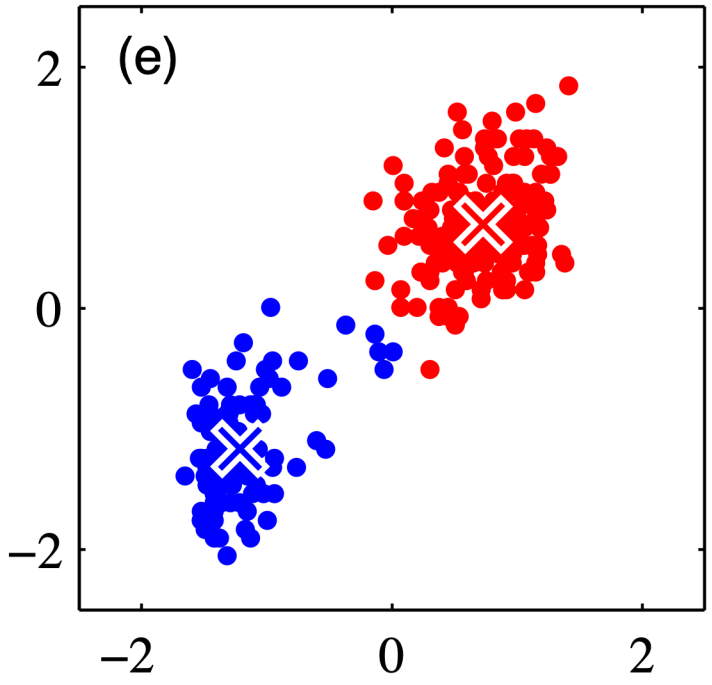


(B) Update the centroids $\{c_j\}$

Iteration 2

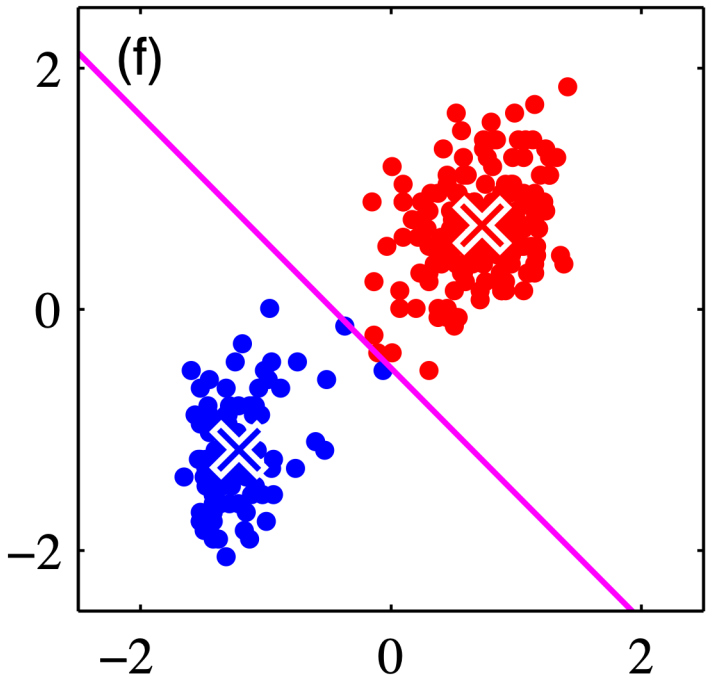


(A) update the cluster assignments

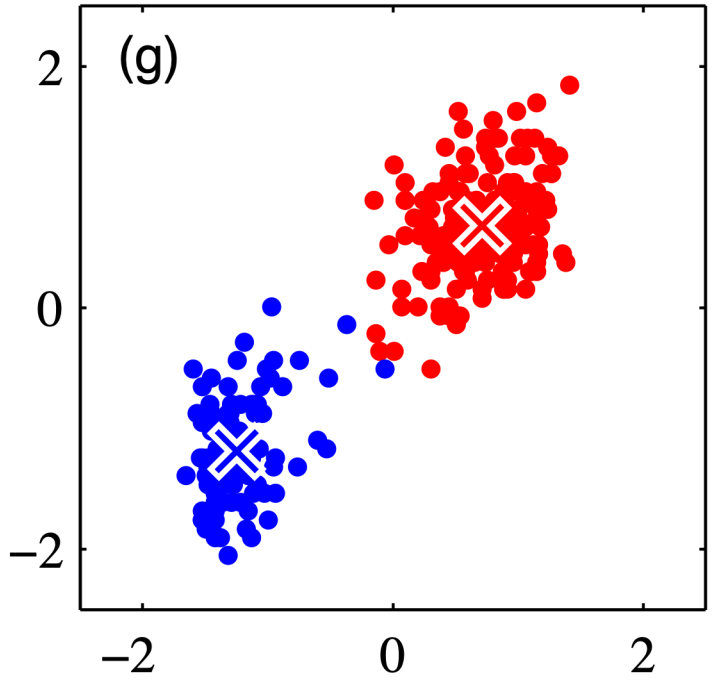


(B) Update the centroids $\{c_j\}$

Iteration 3

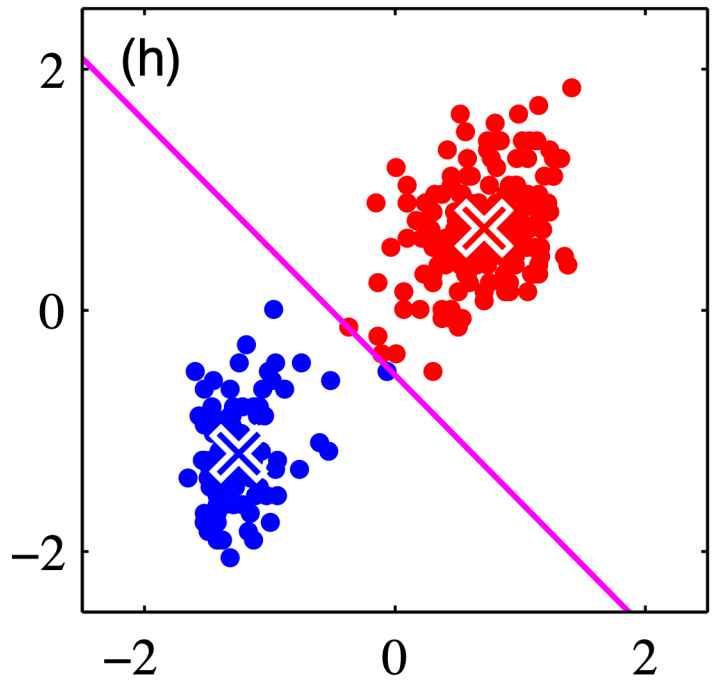


(A) update the cluster assignments

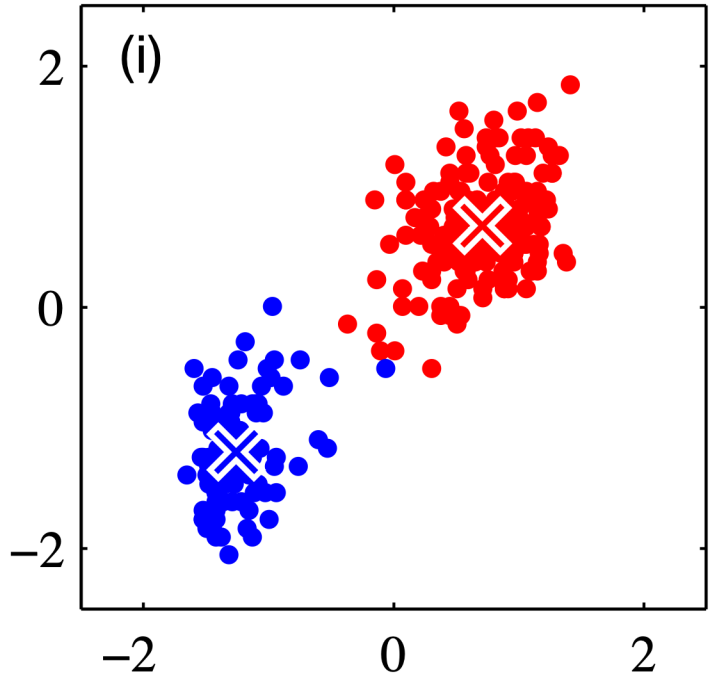


(B) Update the centroids $\{c_j\}$

Iteration 4

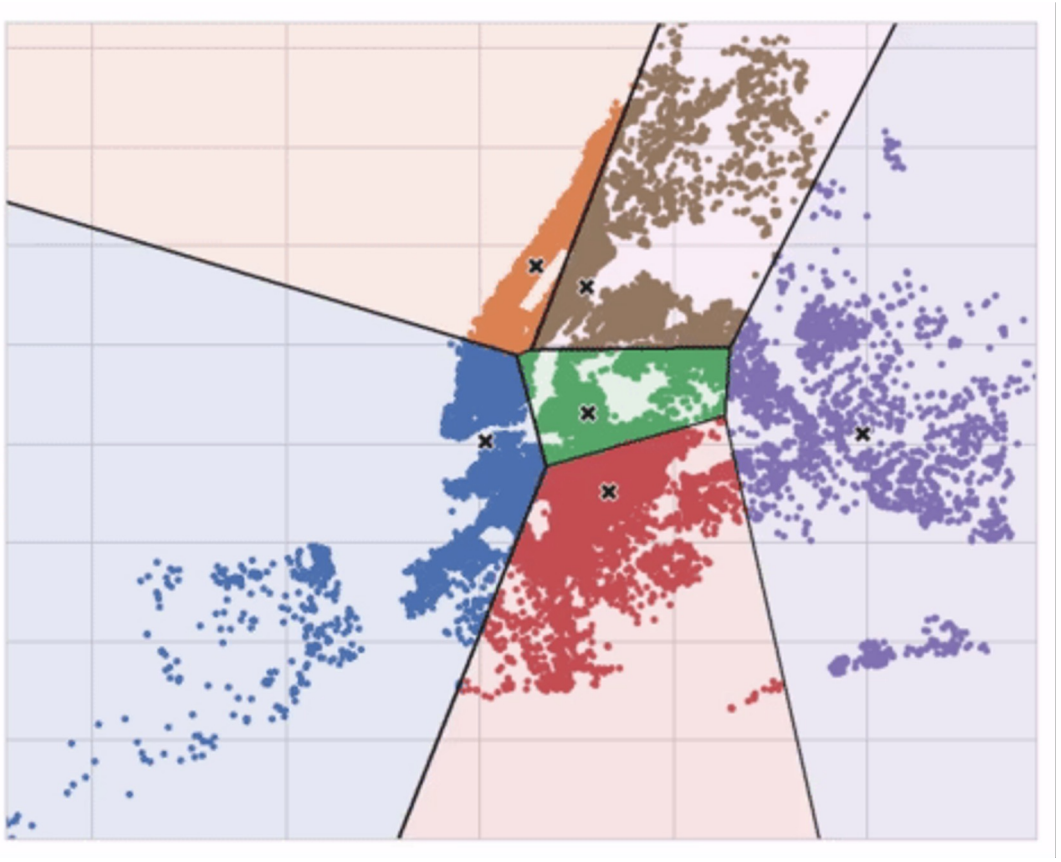


(A) update the cluster assignments



(B) Update the centroids $\{c_j\}$

Iterating until Convergence



Animation from Kaggle

Promise of Convergence

Clustering's approximation error to example n

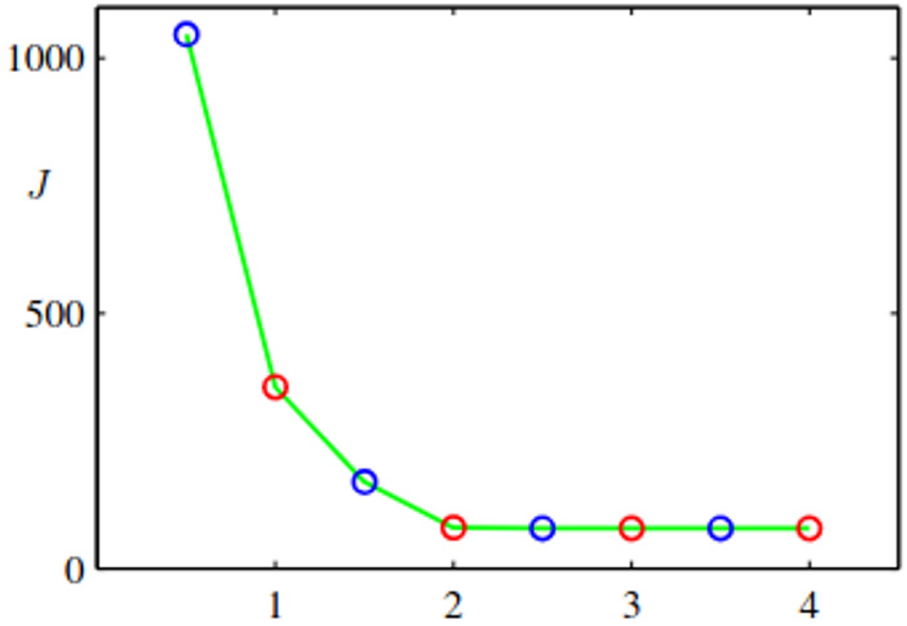
$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2$$

=1 if x_n is assigned to cluster k
=0 otherwise

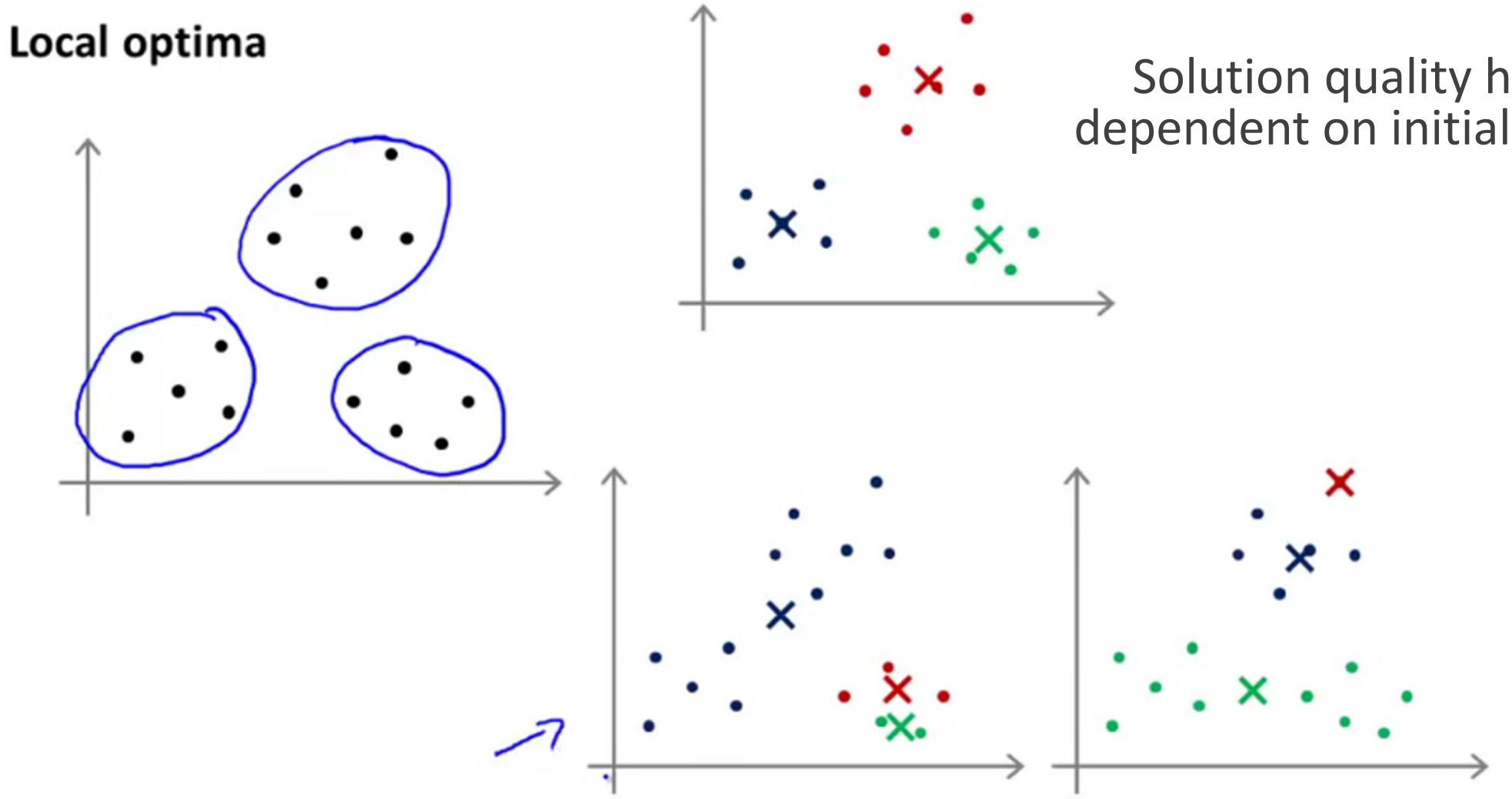
Location of centroid k

But, may converge to a local rather than global minimum of J.

Solution quality highly dependent on initialization!



Plot of the cost function J after each cluster assignment step (blue points) and recompute centroid step (red points)



Solution quality highly dependent on initialization!

Image from Andrew NG Coursera Machine Learning Course

Next lecture (2/9)

- Linear classification; the Perceptron algorithm

- Assigned reading: CIML Chap. 4