

# CPSC 340: Machine Learning and Data Mining

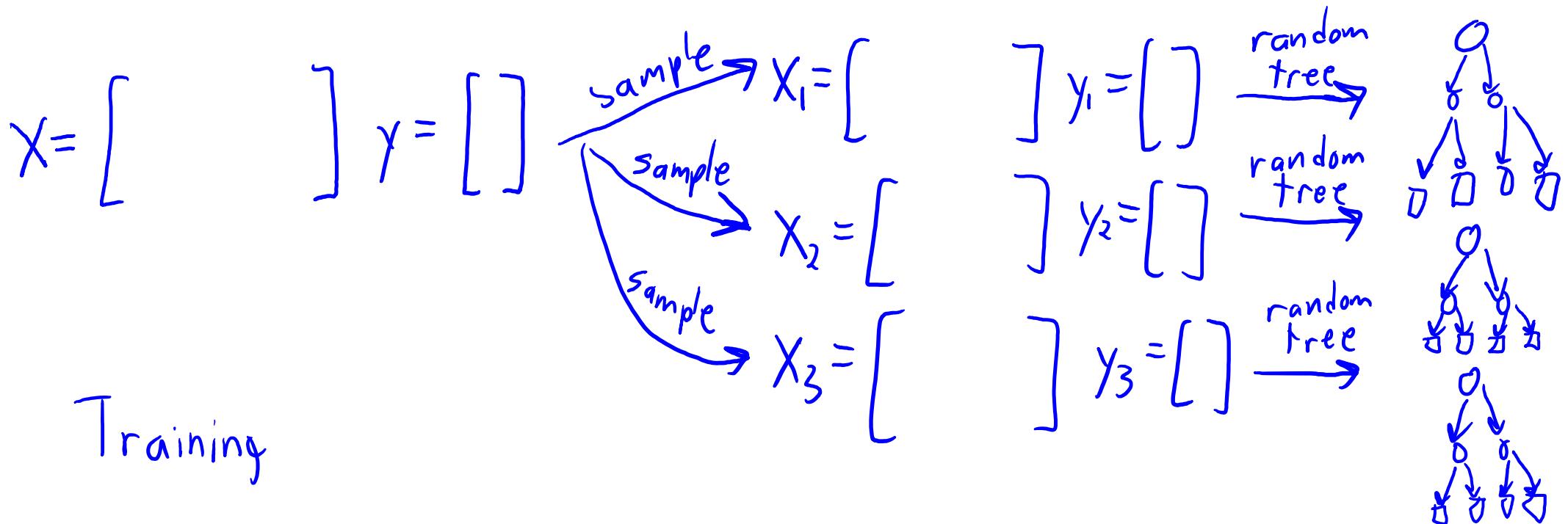
K-Means Clustering  
Spring 2022 (2021W2)

# Admin

- Assignment 2 is out
  - Due Friday of next week. It's long – start early
  - Keep an eye on Piazza and/or commits on the site for updates/fixes
- Midterm
  - Date/time posted on course website

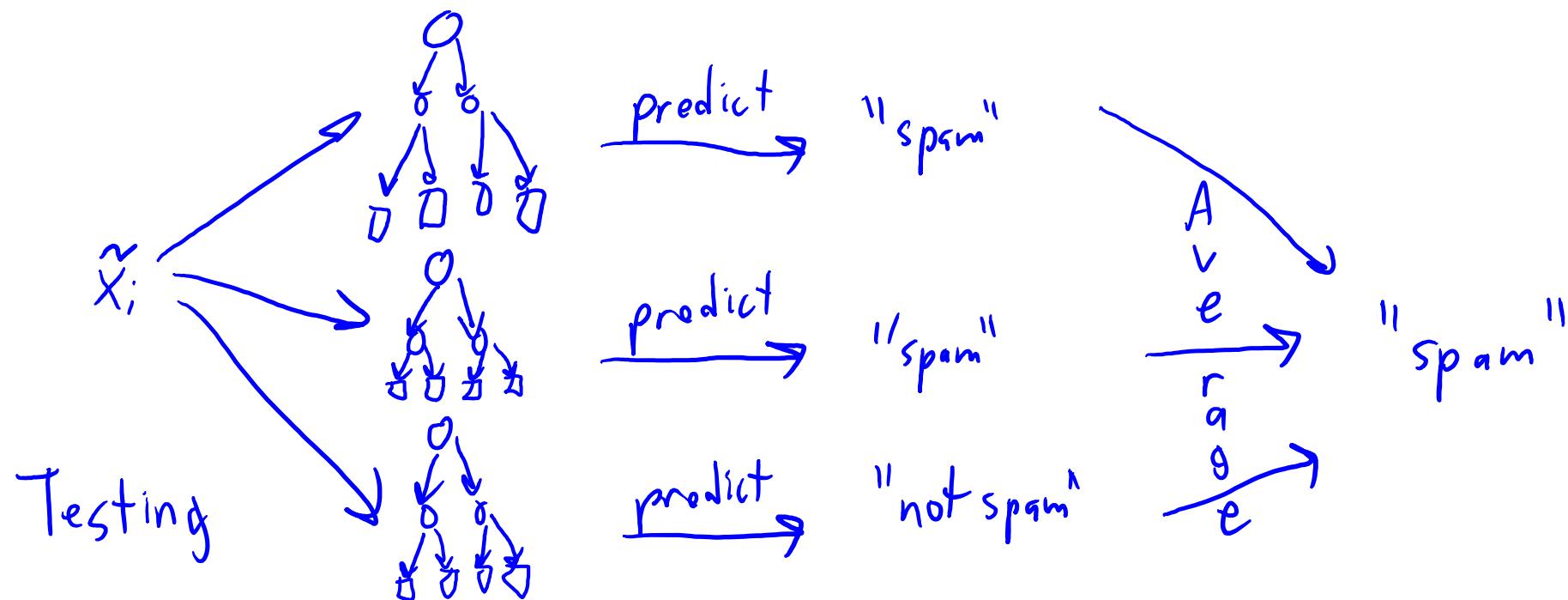
# Last Time: Random Forests

- Random forests are an **ensemble** method.
  - Takes vote among deep **random trees** fit to **bootstrapped samples** of data
  - Randomization **encourages** errors of different trees to be independent.



# Last Time: Random Forests

- Random forests are an ensemble method.
  - Takes vote among deep random trees fit to bootstrapped samples of data.
  - Randomization encourages errors of different trees to be independent.



# Random Forest Ingredient 2: Random Trees

- For each split in a random tree model:
  - Randomly sample a small number of possible features (typically  $\sqrt{d}$ ).
  - Only consider these random features when searching for the optimal rule.

Random tree 1:

- sample (milk, Granges)

$$\text{milk} > 0.5$$

Random tree 2:

- sample (egg, lactase)

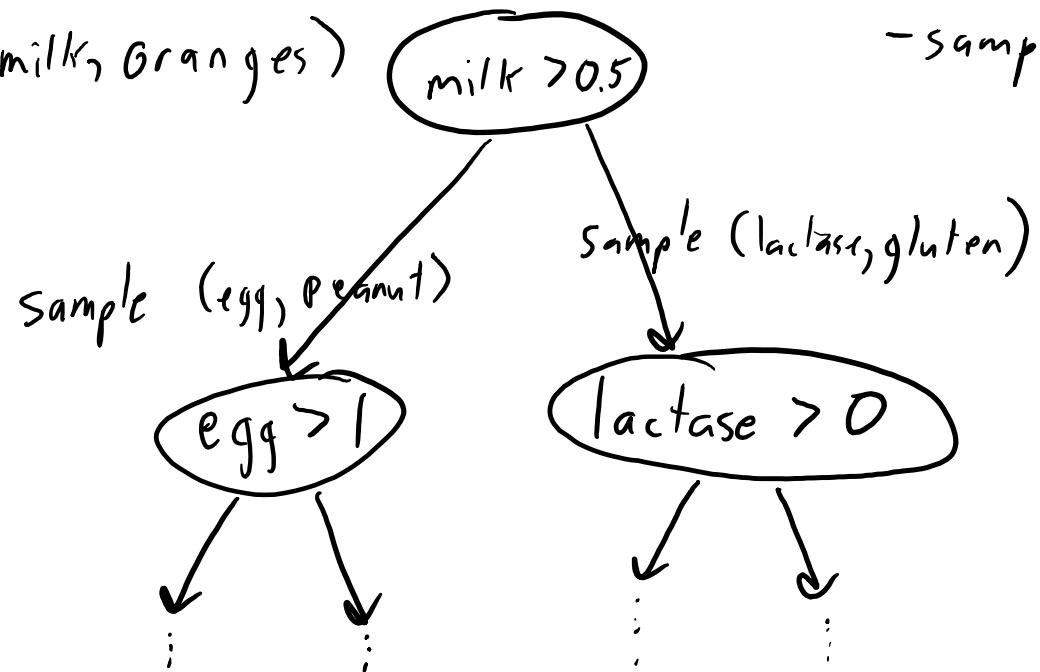
$$\text{egg} > 0$$

# Random Forest Ingredient 2: Random Trees

- For each split in a random tree model:
  - Randomly sample a small number of possible features (typically  $\sqrt{d}$ ).
  - Only consider these random features when searching for the optimal rule.

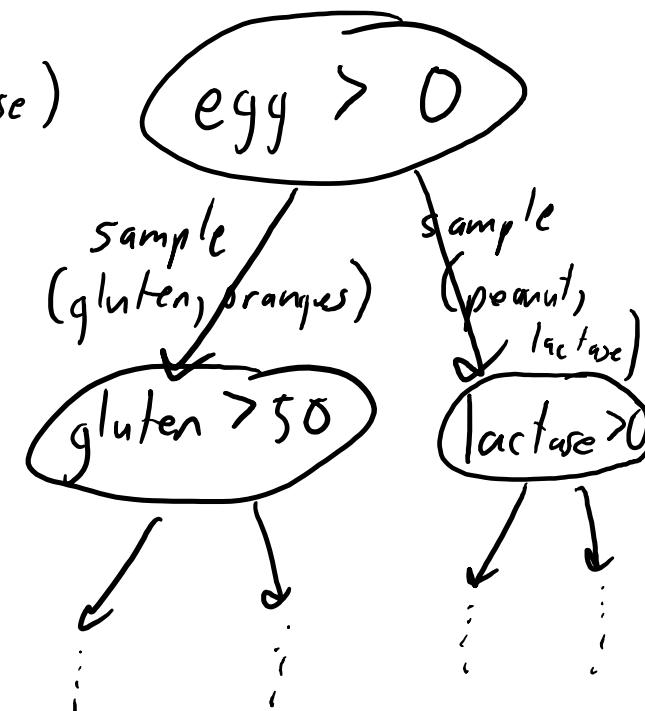
Random tree 1:

- sample (milk, Granges)



Random tree 2:

- sample (egg, lactase)



# Random Forest Ingredient 2: Random Trees

- For each split in a random tree model:
  - Randomly sample a small number of possible features (typically  $\sqrt{d}$ ).
  - Only consider these random features when searching for the optimal rule.
- Splits will tend to use different features in different trees.
  - They will still overfit, but hopefully errors will be more independent.
- So the vote tends to have a much lower test error.
- Empirically, random forests are one of the “best” classifiers.
- Fernandez-Delgado et al. [2014]:
  - Compared 179 classifiers on 121 datasets.
  - Random forests are most likely to be the best classifier.

# Beyond Voting: Model Averaging

- Voting is a special case of “**averaging**” ensemble methods.
  - Where we somehow “**average**” the **predictions** of different models.
- Other averaging:
  - For “regression” (where  $y_i$  is continuous), take average  $y_i$  predictions:

$$\hat{y}_i = \frac{\hat{y}_{i1} + \hat{y}_{i2} + \hat{y}_{i3}}{3}$$

- With probabilistic classifiers, take the average probabilities:

$$p(y_i=1 | x_i) = \frac{1}{3} p_1(y_i=1 | x_i) + \frac{1}{3} p_2(y_i=1 | x_i) + \frac{1}{3} p_3(y_i=1 | x_i)$$

- And there are variations where some classifiers get more weight (see bonus):

$$p(y_i=1 | x_i) = \frac{1}{5} p_1(y_i=1 | x_i) + \frac{3}{5} p_2(y_i=1 | x_i) + \frac{1}{5} p_3(y_i=1 | x_i)$$

# Types and Goals of Ensemble Methods

- Remember the fundamental trade-off:
  1.  $E_{\text{train}}$ : How small you can make the training error.  
vs.
  2.  $E_{\text{approx}}$ : How well training error approximates the test error.
- Goal of ensemble methods is that meta-classifier:
  - Does much better on one of these than individual classifiers.
  - Doesn't do too much worse on the other.
- This suggests two types of ensemble methods:
  1. **Averaging**: improves approximation error of classifiers with high  $E_{\text{approx}}$ .
    - This is the point of “voting”.
  2. **Boosting**: improves training error of classifiers with high  $E_{\text{train}}$ .
    - Covered later in course.

# End of Part 1: Key Concepts

- Fundamental ideas:
  - Training vs. test error (memorization vs. learning).
  - IID assumption (examples come independently from same distribution).
  - Golden rule of ML (test set should not influence training).
  - Fundamental trade-off (between training error vs. approximation error).
  - Validation sets and cross-validation (can approximate test error)
  - Optimization bias (we can overfit the training set and the validation set).
  - Decision theory (we should consider costs of predictions).
  - Parametric vs. non-parametric (whether model size depends on ‘n’).
  - No free lunch theorem (there is no “best” model).

# End of Part 1: Key Concepts

- We saw 3 ways of “learning”:
  - Searching for rules.
    - Decision trees (greedy recursive splitting using decision stumps).
  - Counting frequencies.
    - Naïve Bayes (probabilistic classifier based on conditional independence).
  - Measuring distances.
    - K-nearest neighbours (non-parametric classifier with universal consistency).
- We saw 2 generic ways of improving performance:
  - Encouraging invariances with data augmentation.
  - Ensemble methods (combine predictions of several models).
    - Random forests (averaging plus randomization to reduce overfitting).

# Unsupervised Learning

- Major kingdom of ML
  - Supervised learning
  - Unsupervised learning
  - Reinforcement learning

# Supervised Learning



Tree



Bird



???



Tree



Bird



???



Tree



Bird



???

# Unsupervised Learning

- agent learns patterns in the input even though no feedback is provided

# Unsupervised Learning



# Unsupervised Learning



# Unsupervised Learning



# Unsupervised Learning



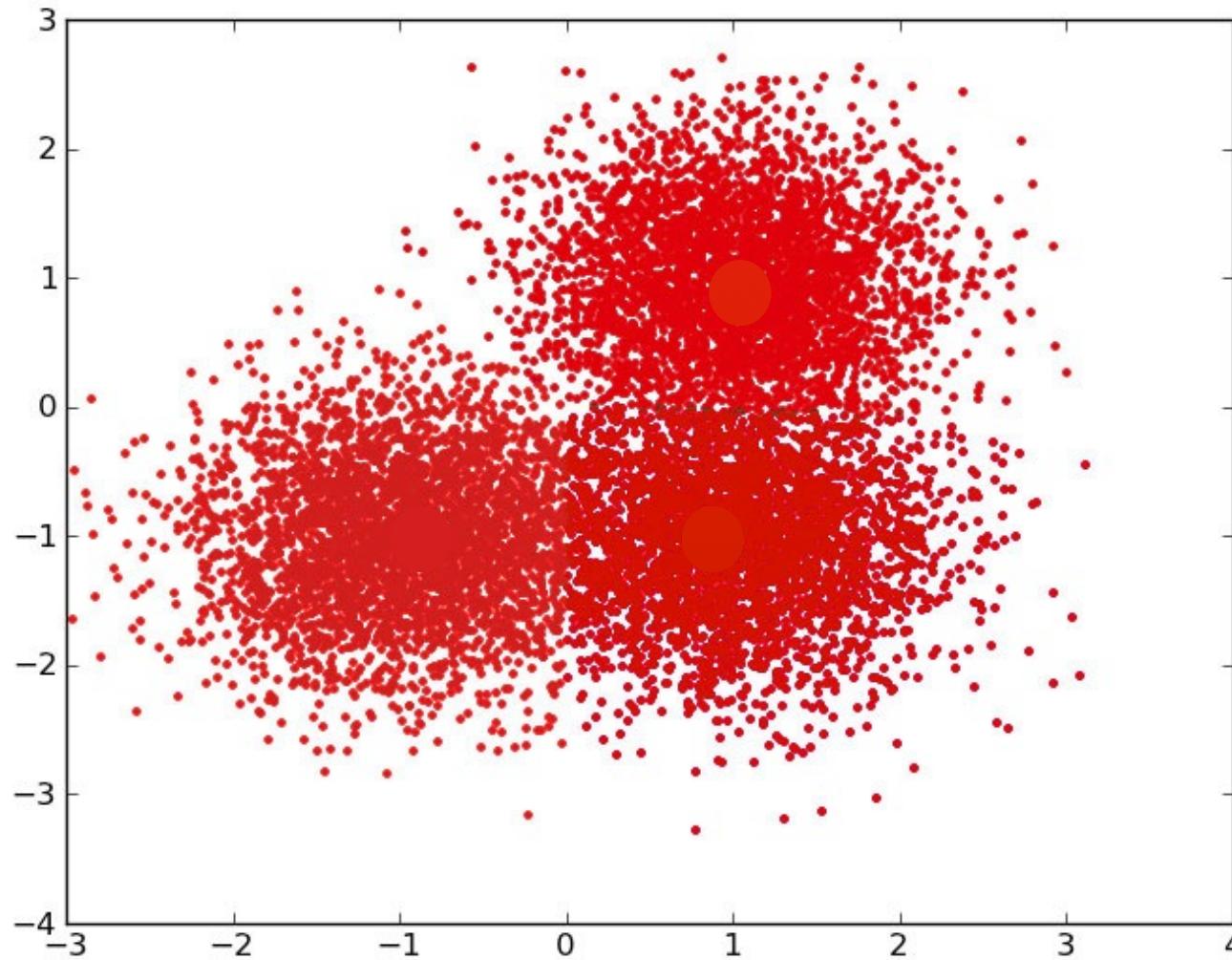
wiseGEEK



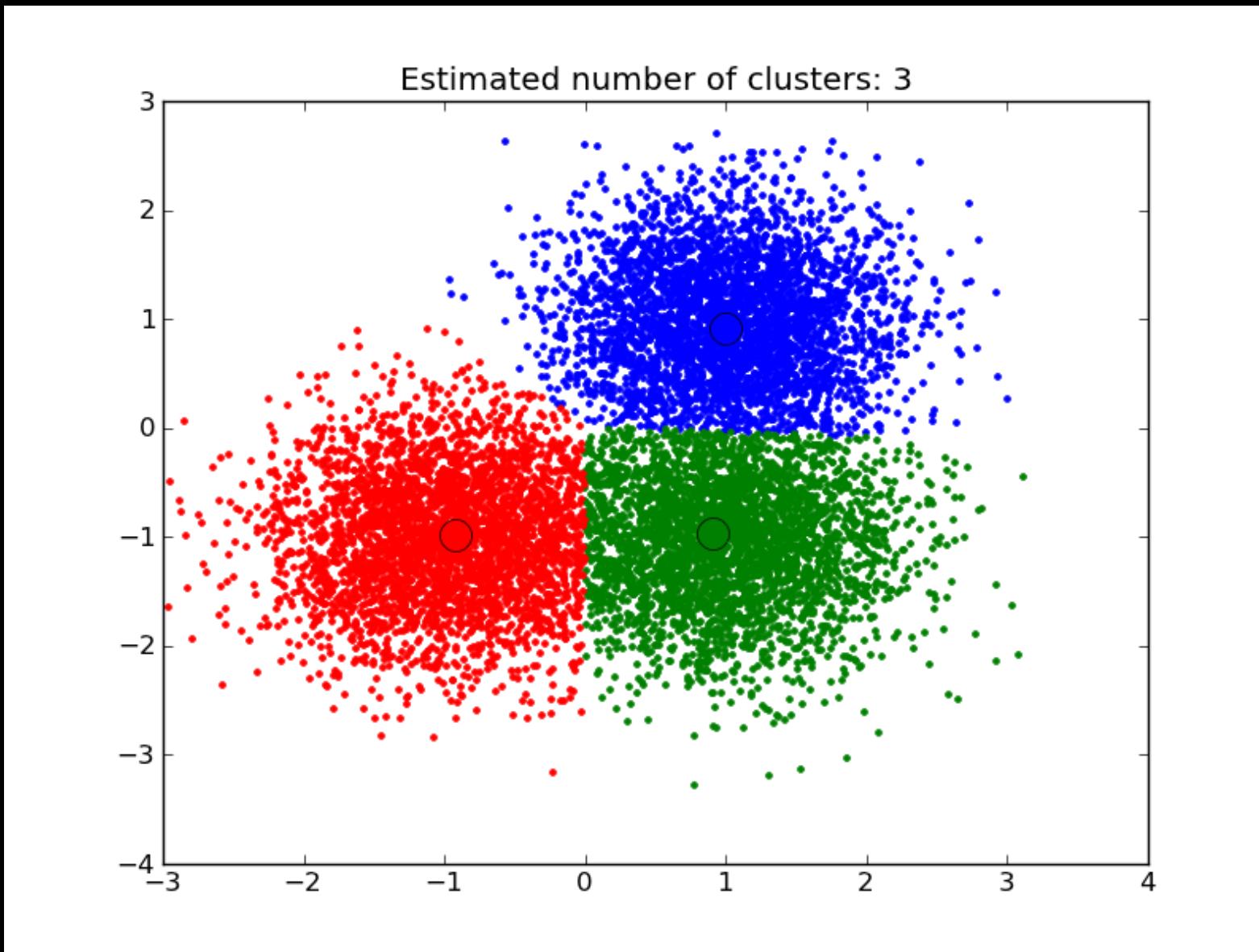
# Unsupervised Learning



# Unsupervised Learning



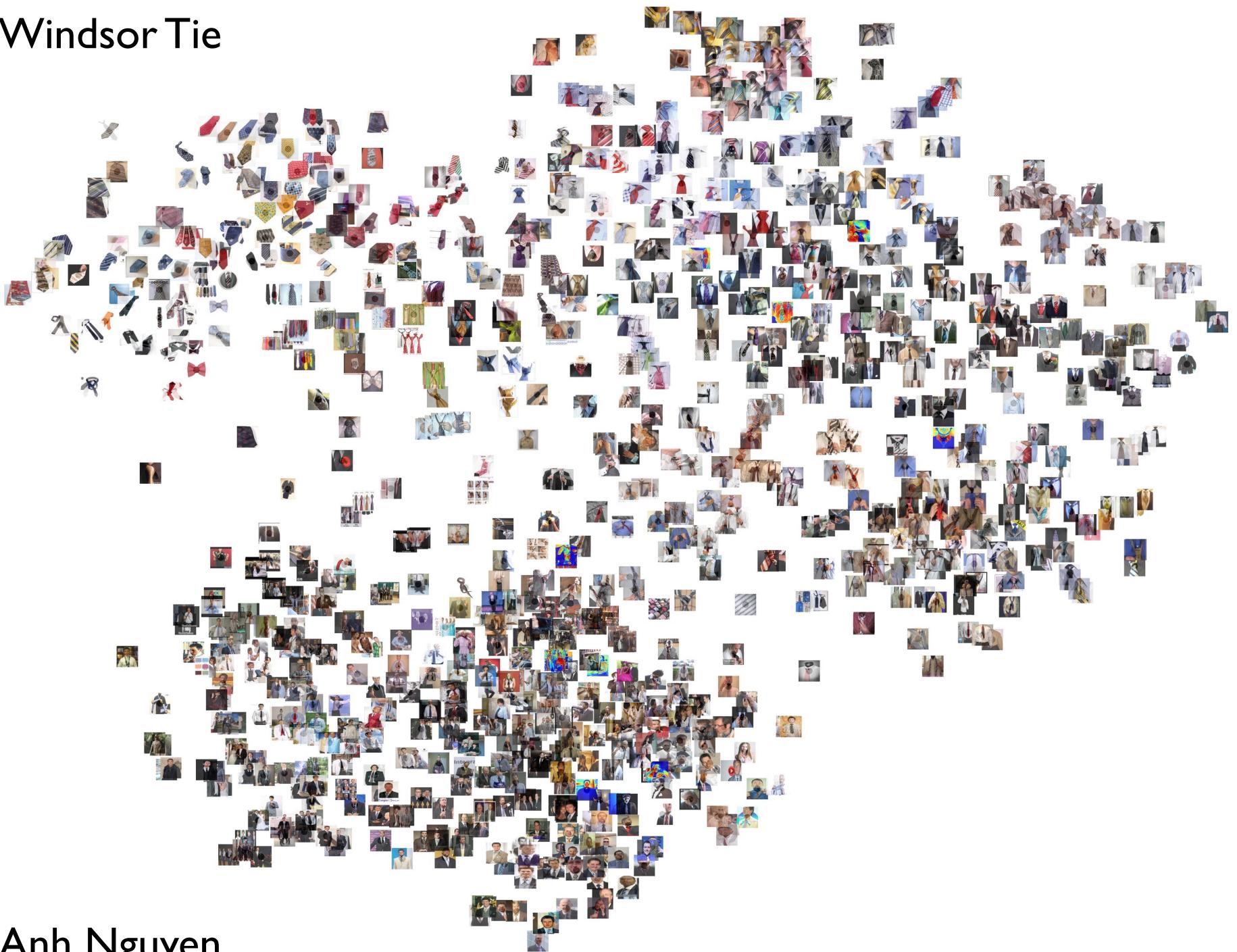
# Unsupervised Learning



# Unsupervised Learning



# Windsor Tie

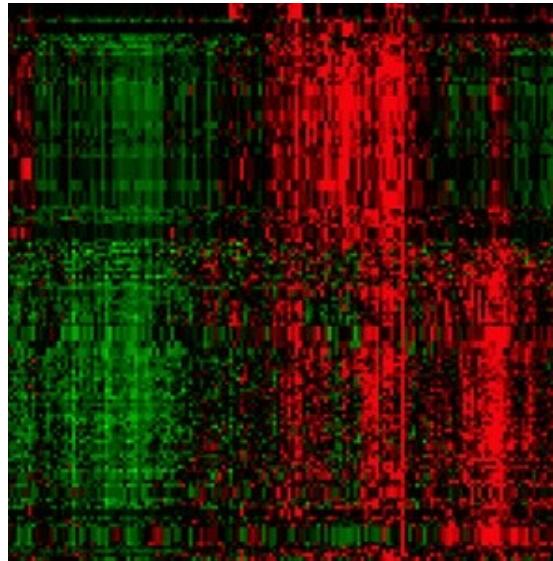


Anh Nguyen

# Application: Classifying Cancer Types

- “I collected gene expression data for 1000 different types of cancer cells, can you tell me the different classes of cancer?”

$X =$



- We are not given the class labels  $y$ , but want **meaningful labels**.
- An example of **unsupervised learning**.

# Unsupervised Learning

- Supervised learning:
  - We have features  $x_i$  and class labels  $y_i$ .
  - Write a program that produces  $y_i$  from  $x_i$ .
- Unsupervised learning:
  - We **only have  $x_i$  values**, but no explicit target labels.
  - You want to do “something” with them.
- Some unsupervised learning tasks:
  - Outlier detection: Is this a ‘normal’  $x_i$ ?
  - Similarity search: Which examples look like this  $x_i$ ?
  - Association rules: Which  $x^j$  occur together?
  - Latent-factors: What ‘parts’ are the  $x_i$  made from?
  - Data visualization: What does the high-dimensional  $X$  look like?
  - Ranking: Which are the most important  $x_i$ ?
  - Clustering: What types of  $x_i$  are there?

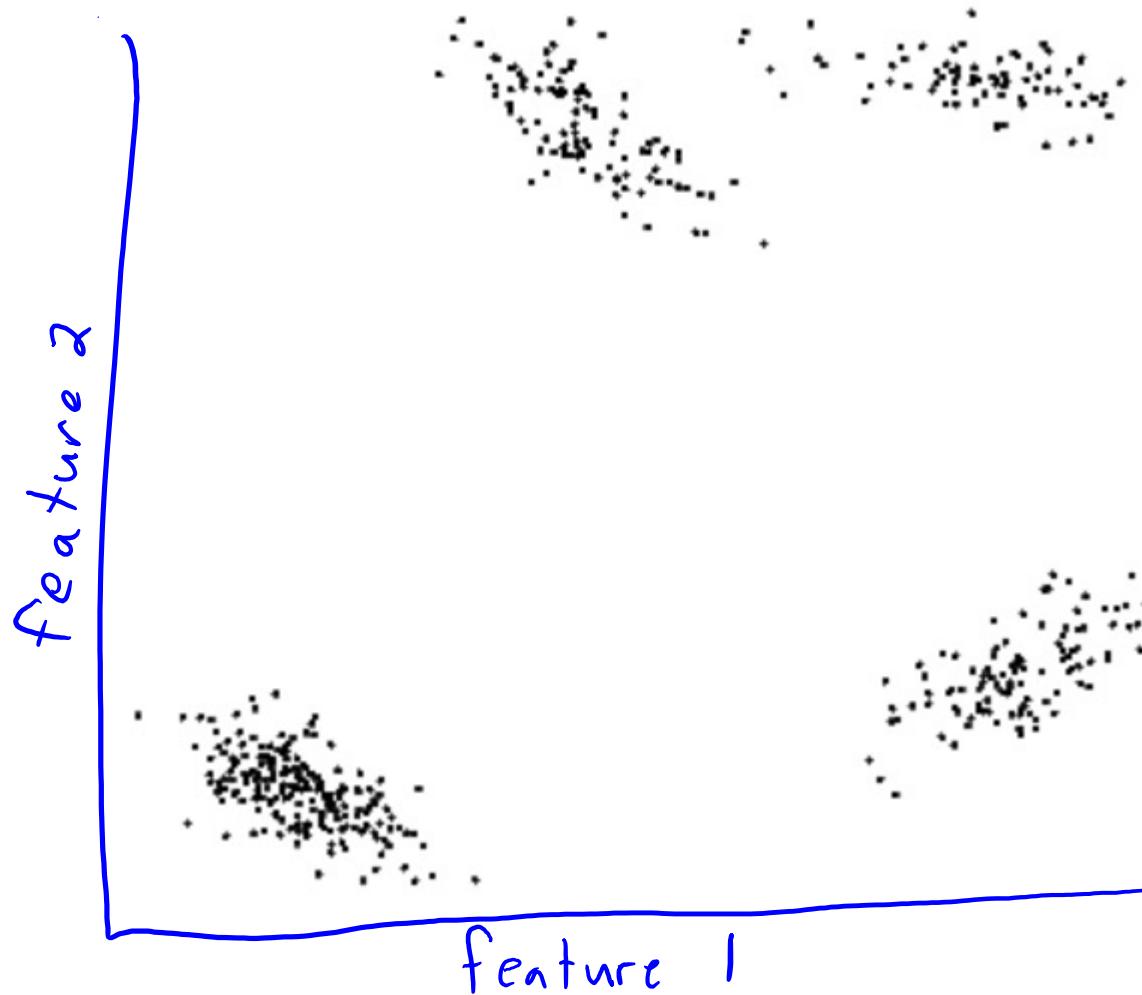
# Clustering

- Clustering:
  - Input: set of examples described by features  $x_i$ .
  - Output: an assignment of examples to ‘groups’.
- Unlike classification, we are not given the ‘groups’.
  - Algorithm must discover groups.
- Example of groups we might discover in e-mail spam:
  - ‘Lucky winner’ group.
  - ‘Weight loss’ group.
  - ‘I need your help’ group.
  - ‘Solve  $<x>$  with this one weird trick’ group.

# Clustering Example

Input: data matrix 'X'.

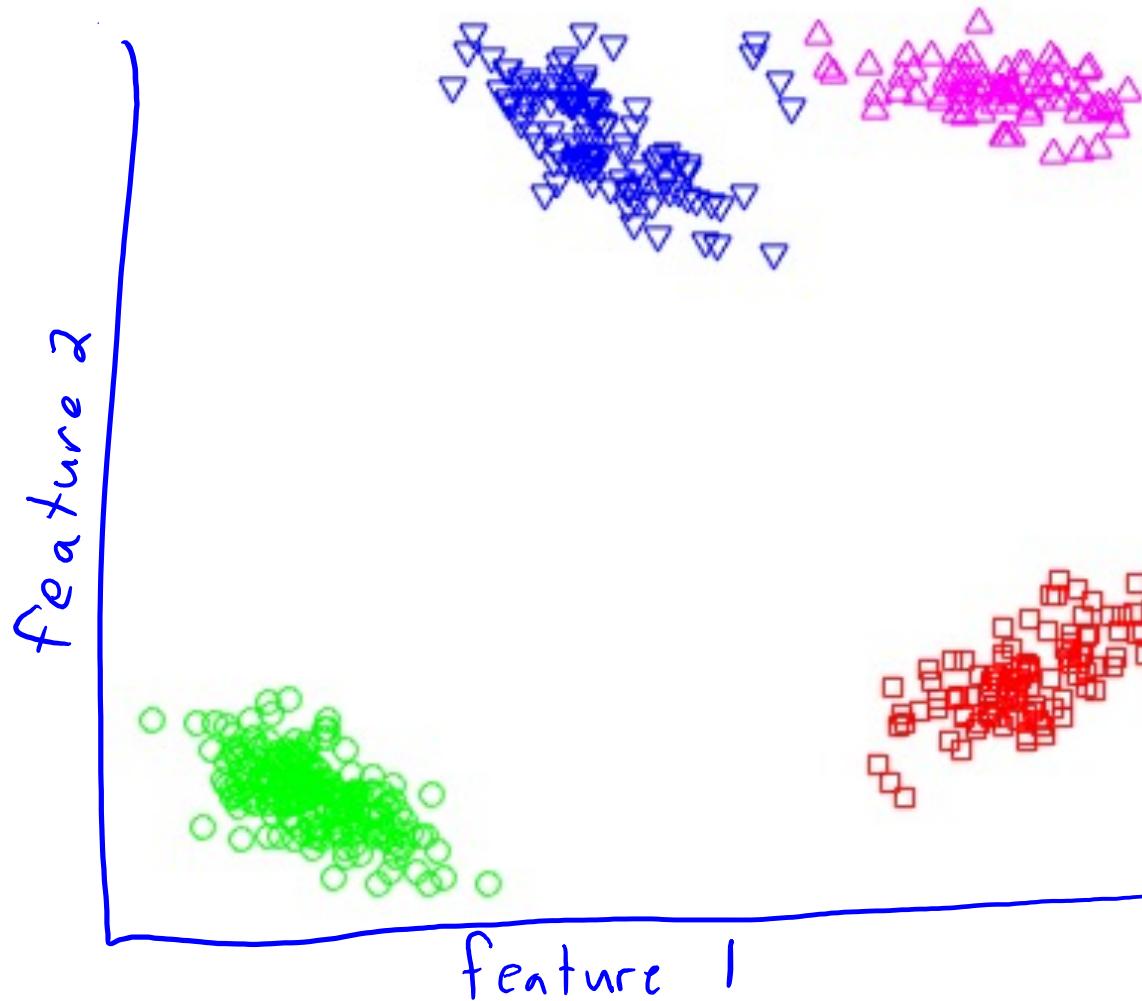
$$X = \begin{bmatrix} -9.0 & -7.3 \\ -10.9 & -9.0 \\ 13.7 & 19.3 \\ 13.8 & 20.4 \\ 12.8 & 20.6 \\ \vdots & \vdots \end{bmatrix}$$



# Clustering Example

Input: data matrix 'X'.

$$X = \begin{bmatrix} -9.0 & -7.3 \\ -10.9 & -9.0 \\ 13.7 & 19.3 \\ 13.8 & 20.4 \\ 12.8 & 20.6 \\ \vdots & \vdots \end{bmatrix}$$



Output: clusters  $\hat{y}$ .

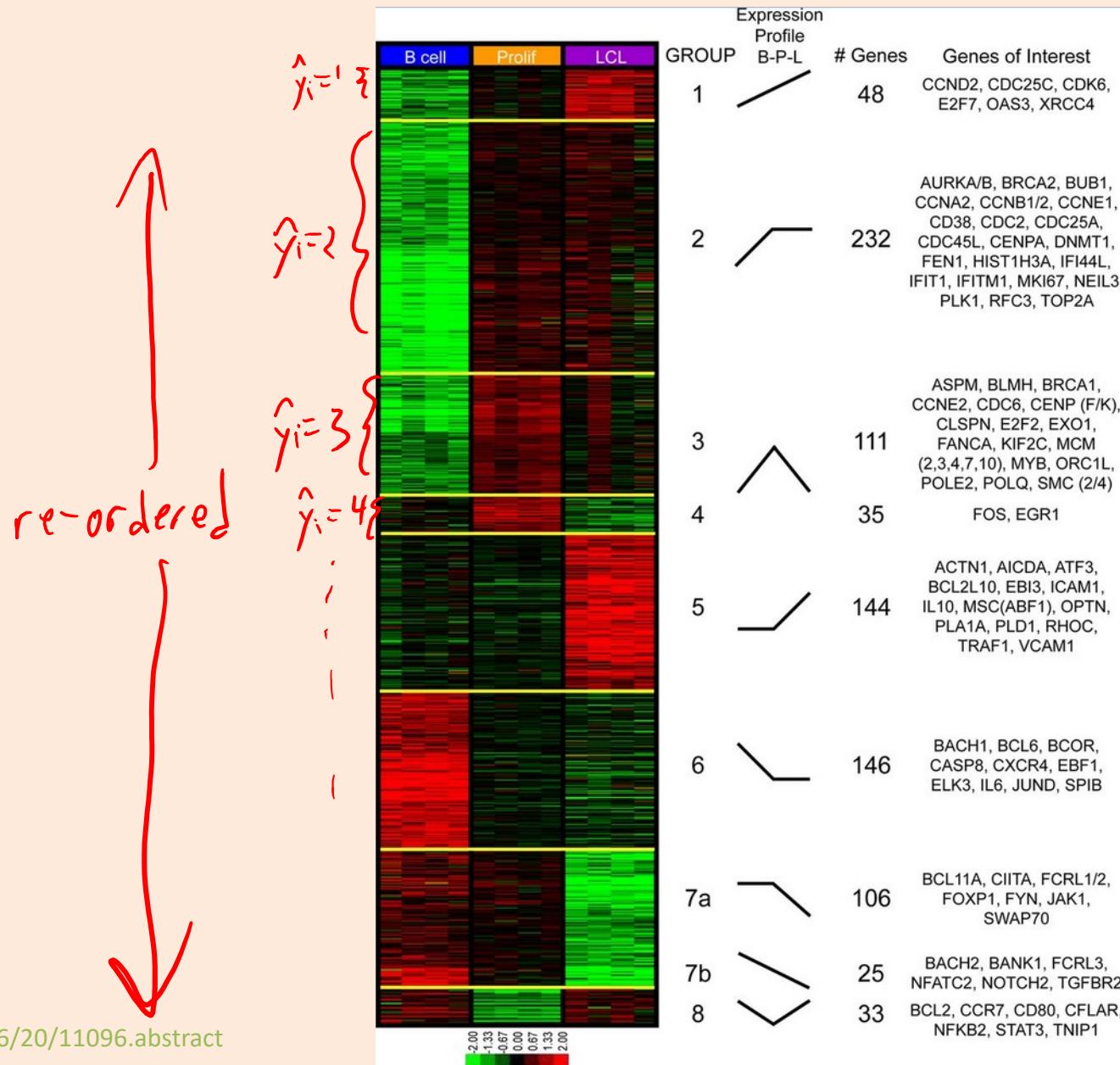
$$\hat{y} = \begin{bmatrix} 2 \\ 2 \\ 3 \\ 3 \\ 1 \\ \vdots \end{bmatrix}$$

# Data Clustering

- General goal of clustering algorithms:
  - Examples in the same group should be ‘similar’.
  - Examples in different groups should be ‘different’.
- But the ‘best’ clustering is hard to define:
  - We don’t have a test error.
  - Generally, there is no ‘best’ method in unsupervised learning.
    - So there are lots of methods: we’ll focus on important/representative ones.
- Why cluster?
  - You could want to know what the groups are.
  - You could want to find the group for a new example  $x_i$ .
  - You could want to find examples related to a new example  $x_i$ .
  - You could want a ‘prototype’ example for each group.

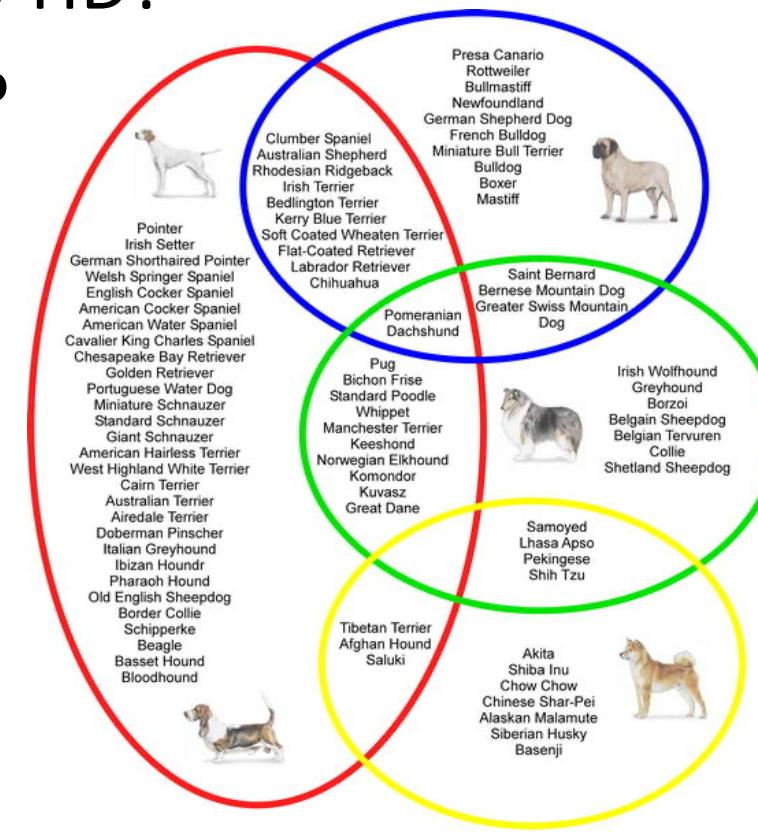
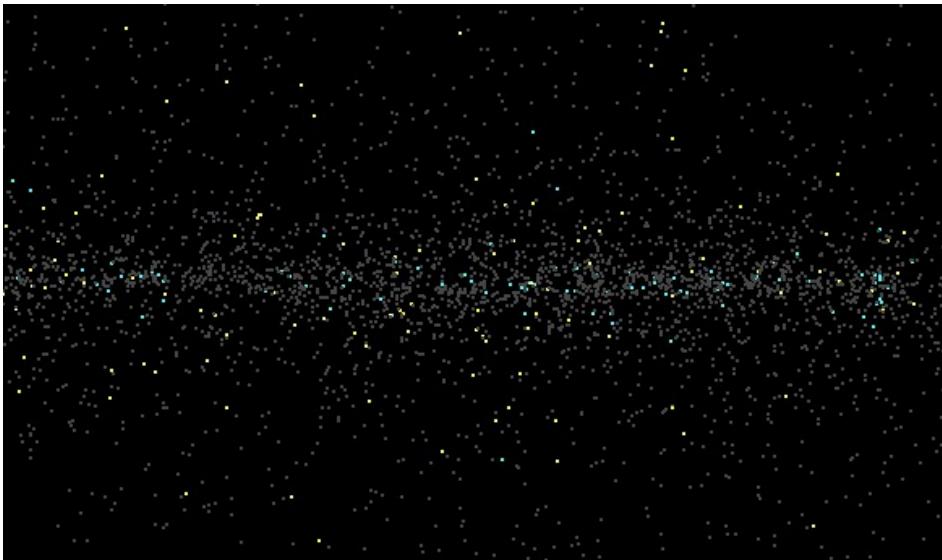
# Clustering of Epstein-Barr Virus

bonus!



# Other Clustering Applications

- NASA: what types of stars are there?
- Biology: are there sub-species?
- Documents: what kinds of documents are on my HD?
- Commercial: what kinds of customers do I have?



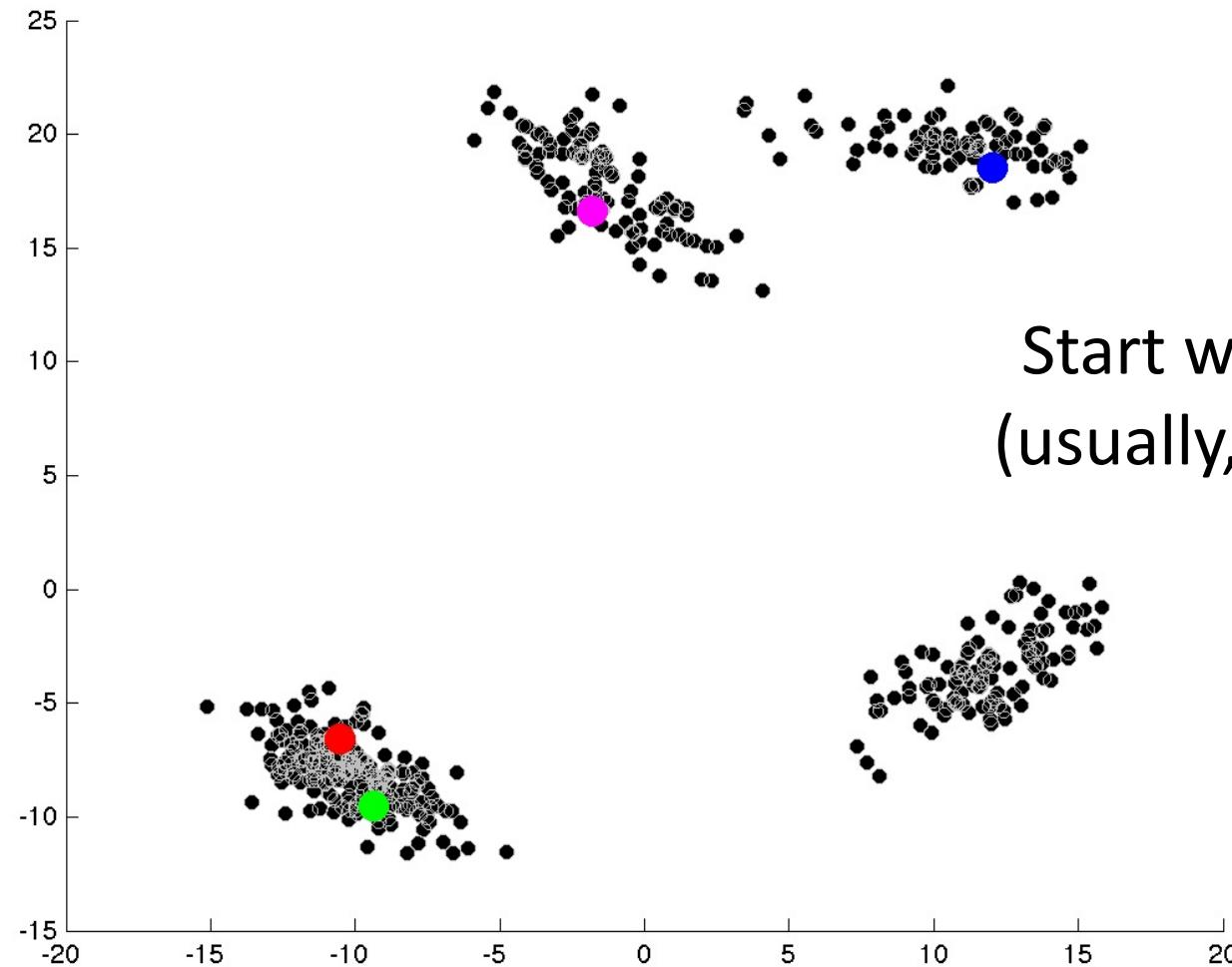
# K-Means

- Most popular clustering method is k-means.
- Input:
  - The number of clusters ‘k’ (hyper-parameter).
  - Initial guess of the center (the “mean”) of each cluster.
- Algorithm:
  - Assign each  $x_i$  to its closest mean.
  - Update the means based on the assignment.
  - Repeat until convergence.

# K-Means Example

Input: data matrix 'X'.

$$X = \begin{bmatrix} -9.0 & -7.3 \\ -10.9 & -9.0 \\ 13.7 & 19.3 \\ 13.8 & 20.4 \\ 12.8 & 20.6 \\ \vdots & \vdots \end{bmatrix}$$

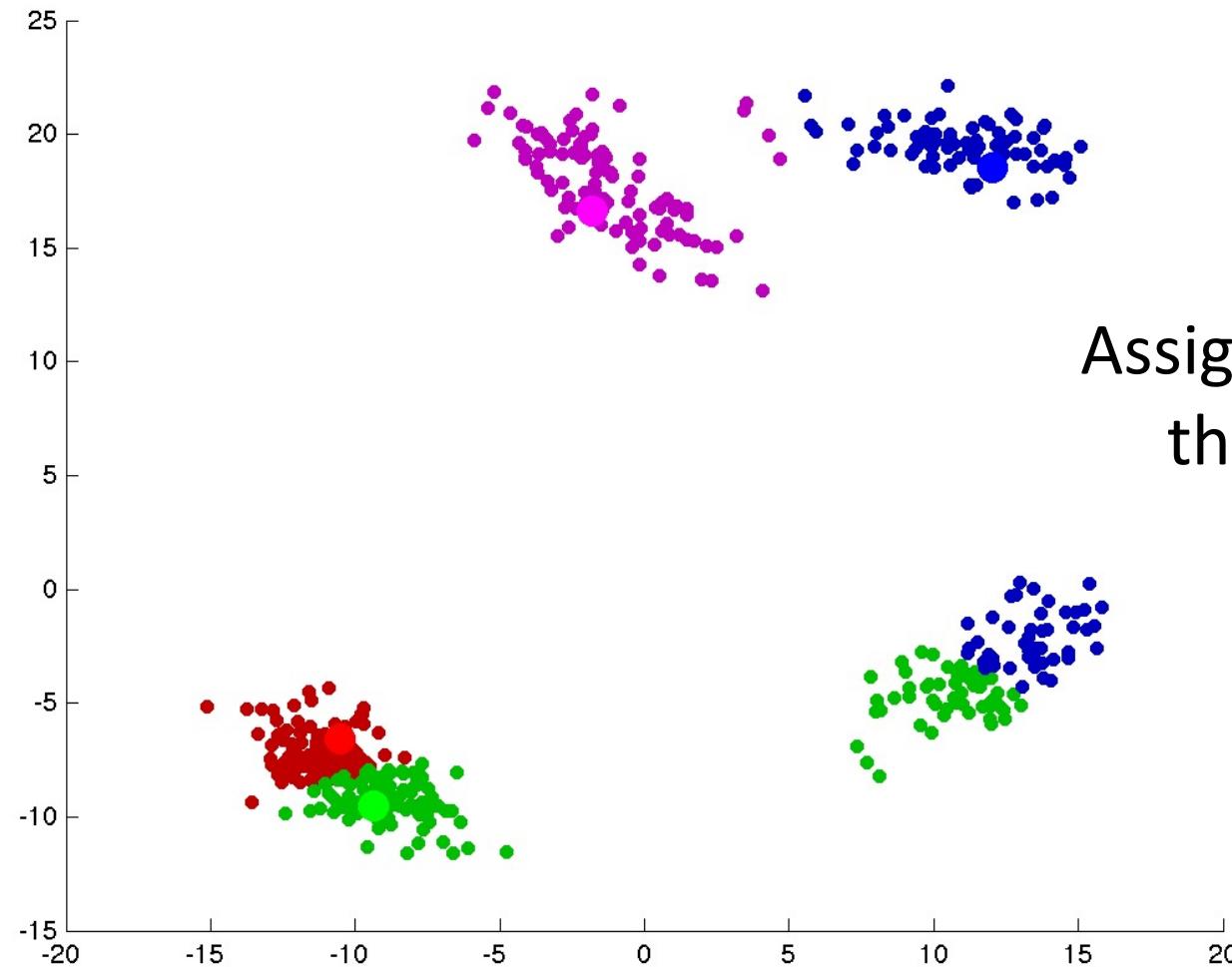


Start with 'k' initial 'means'  
(usually, random data points)

# K-Means Example

Input: data matrix 'X'.

$$X = \begin{bmatrix} -9.0 & -7.3 \\ -10.9 & -9.0 \\ 13.7 & 19.3 \\ 13.8 & 20.4 \\ 12.8 & 20.6 \\ \vdots & \vdots \end{bmatrix}$$

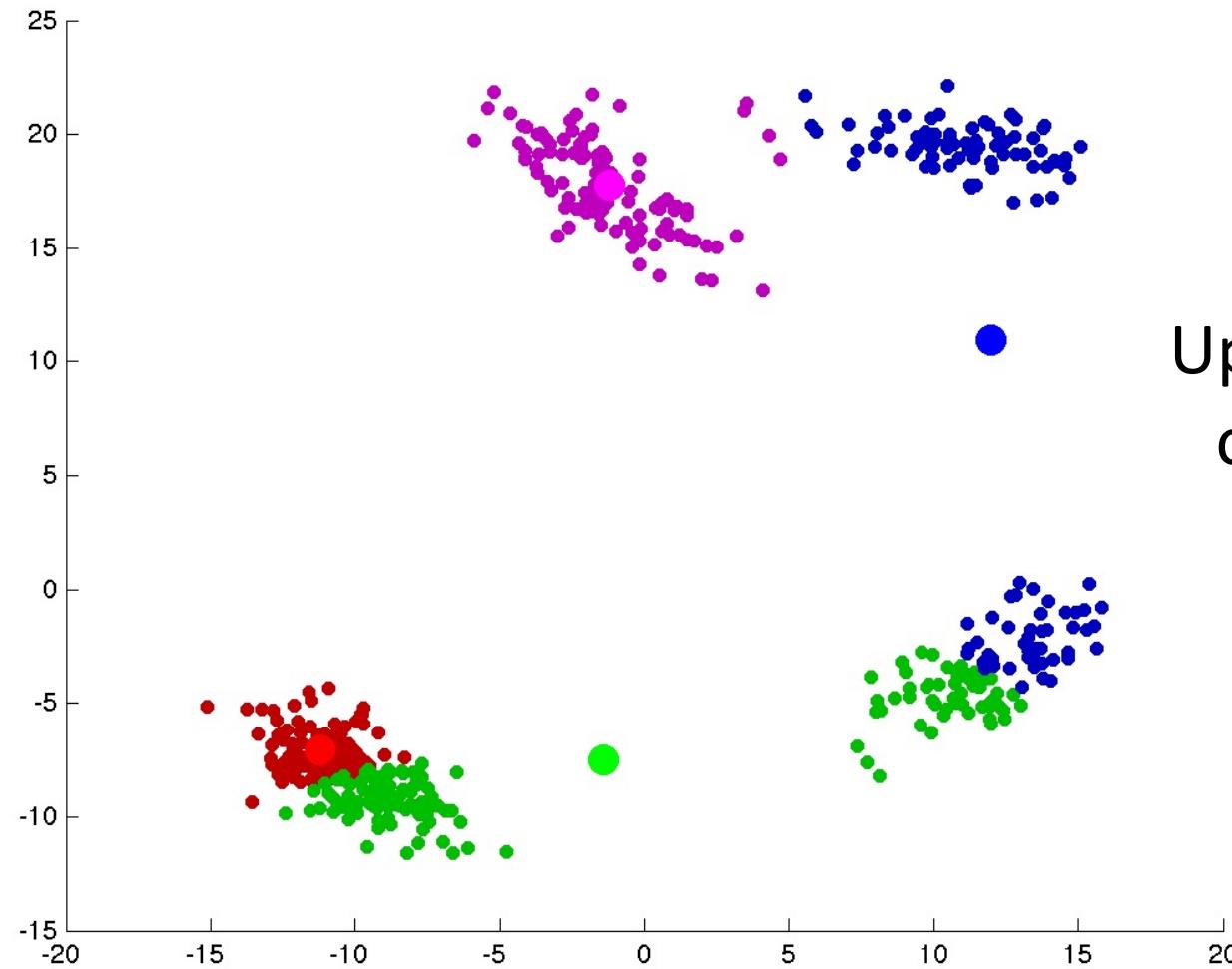


Assign each example to  
the closest mean.

# K-Means Example

Input: data matrix 'X'.

$$X = \begin{bmatrix} -9.0 & -7.3 \\ -10.9 & -9.0 \\ 13.7 & 19.3 \\ 13.8 & 20.4 \\ 12.8 & 20.6 \\ \vdots & \vdots \end{bmatrix}$$

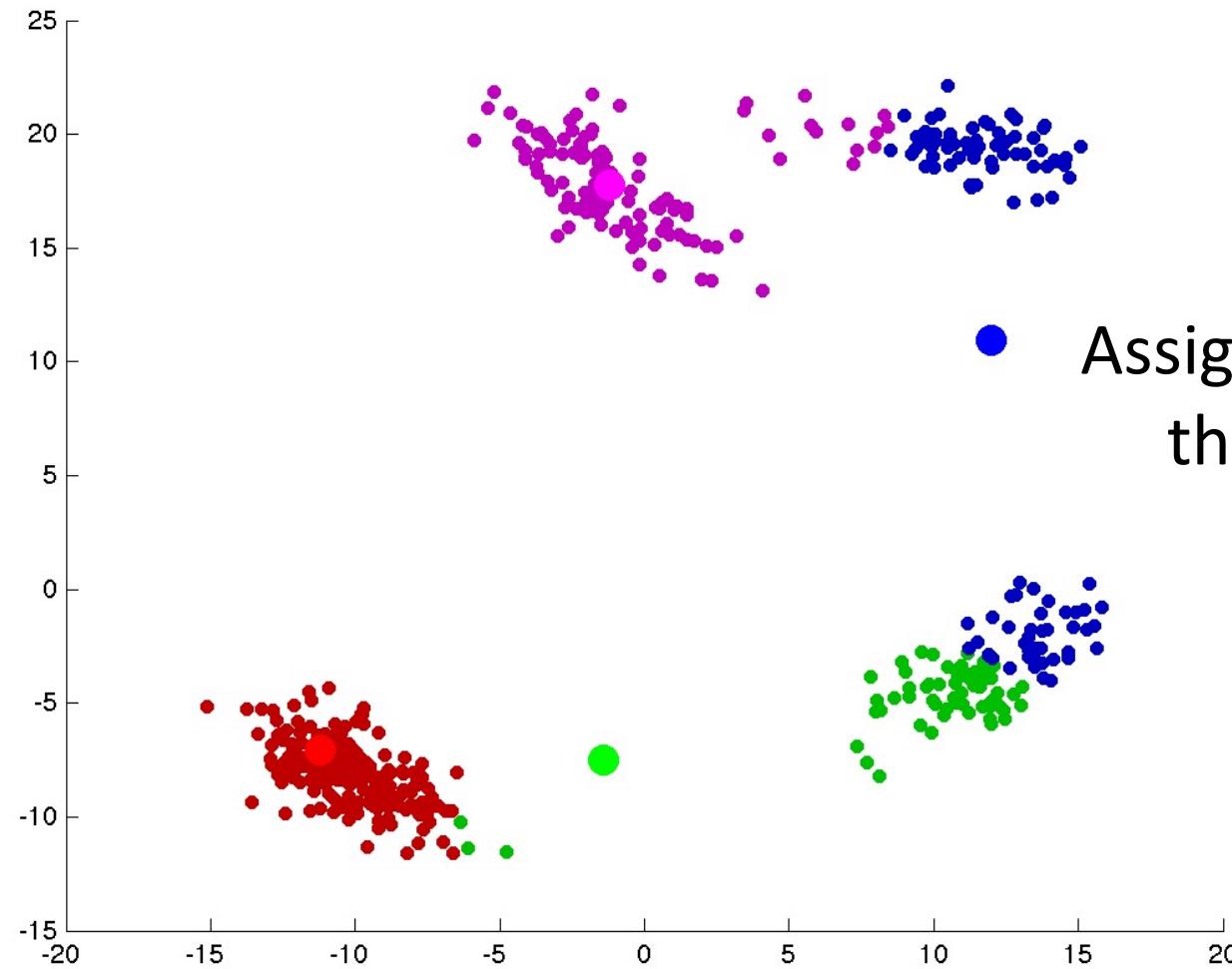


Update the mean  
of each group.

# K-Means Example

Input: data matrix 'X'.

$$X = \begin{bmatrix} -9.0 & -7.3 \\ -10.9 & -9.0 \\ 13.7 & 19.3 \\ 13.8 & 20.4 \\ 12.8 & 20.6 \\ \vdots & \vdots \end{bmatrix}$$

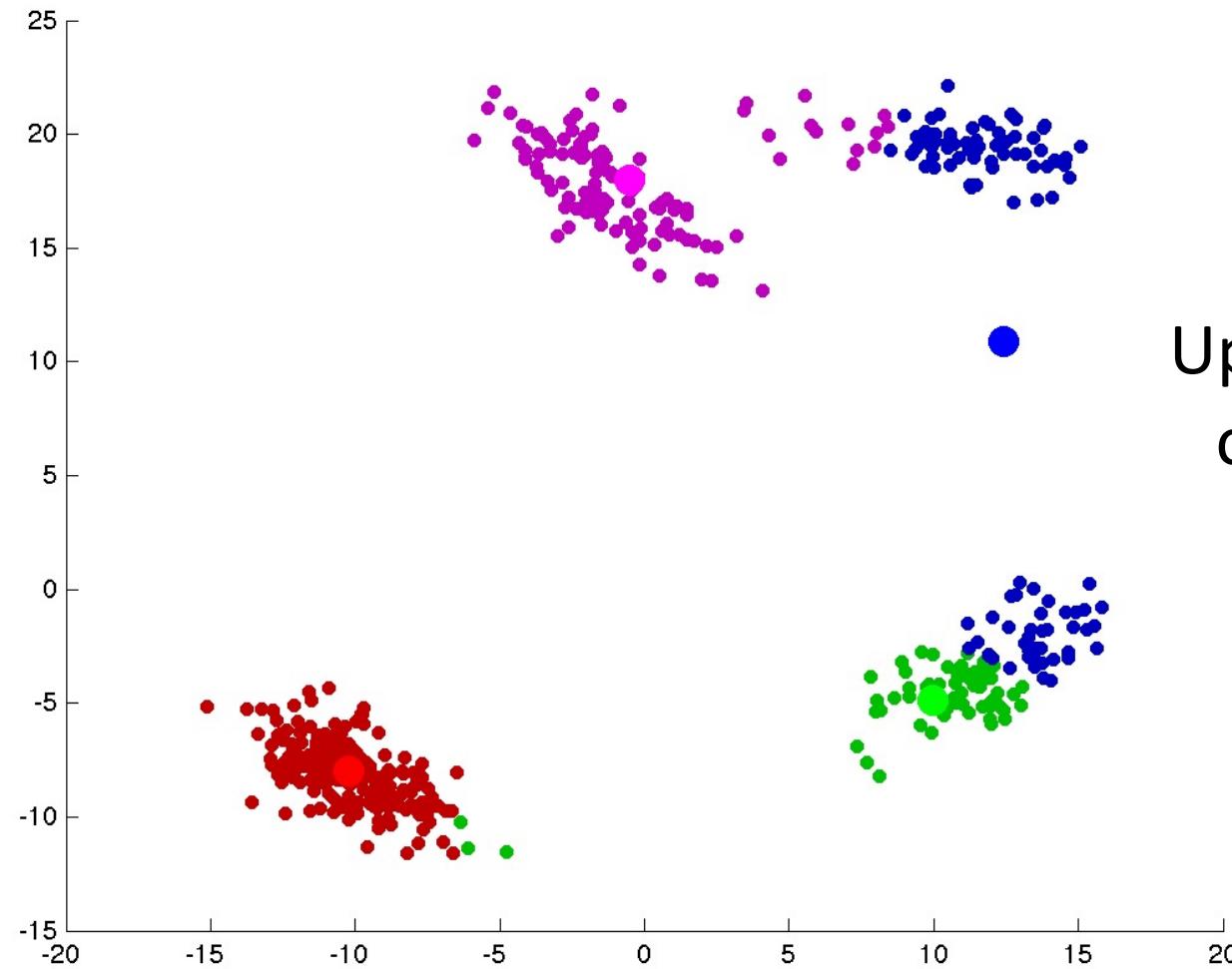


- Assign each example to the closest mean.

# K-Means Example

Input: data matrix 'X'.

$$X = \begin{bmatrix} -9.0 & -7.3 \\ -10.9 & -9.0 \\ 13.7 & 19.3 \\ 13.8 & 20.4 \\ 12.8 & 20.6 \\ \vdots & \vdots \end{bmatrix}$$

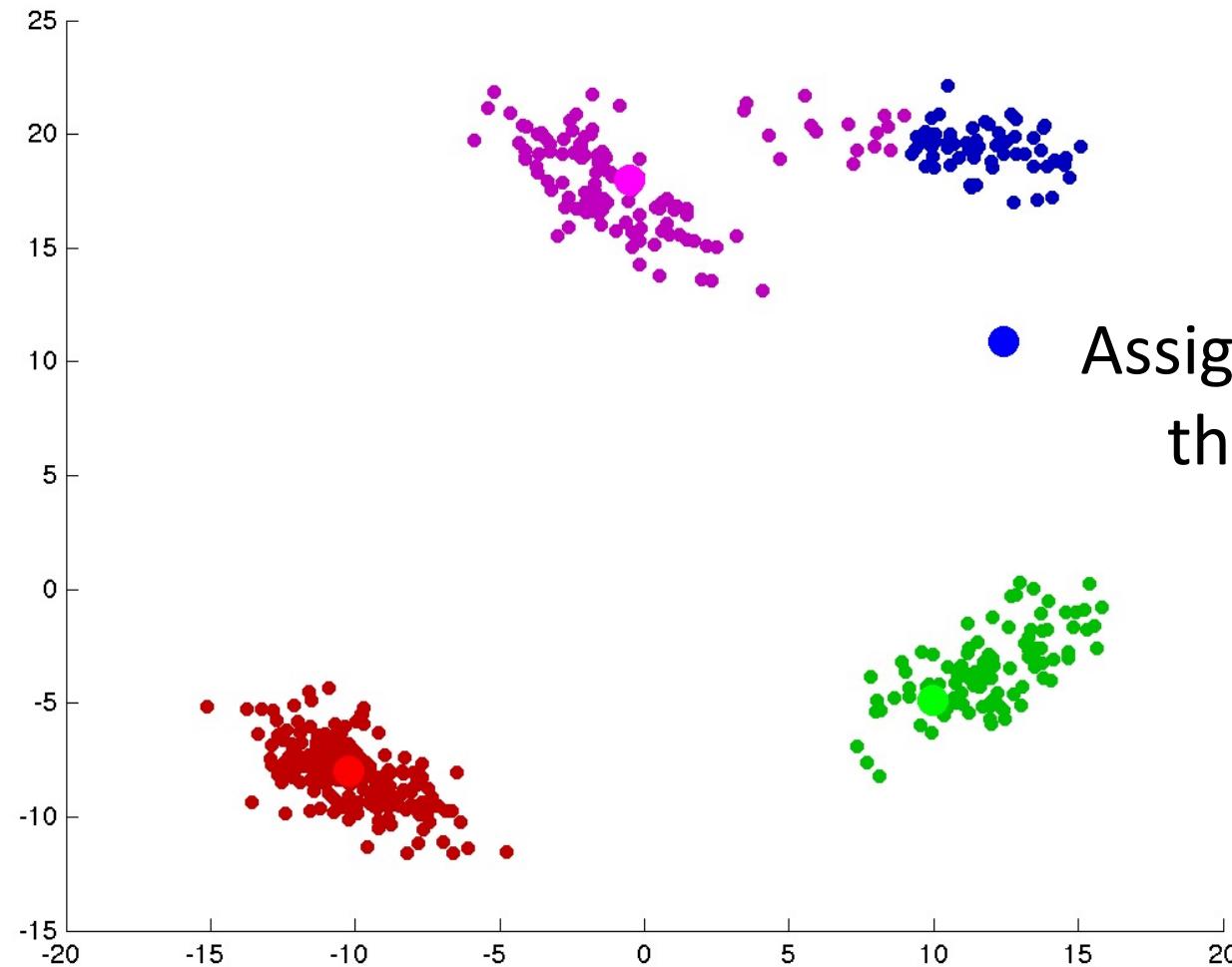


Update the mean  
of each group.

# K-Means Example

Input: data matrix 'X'.

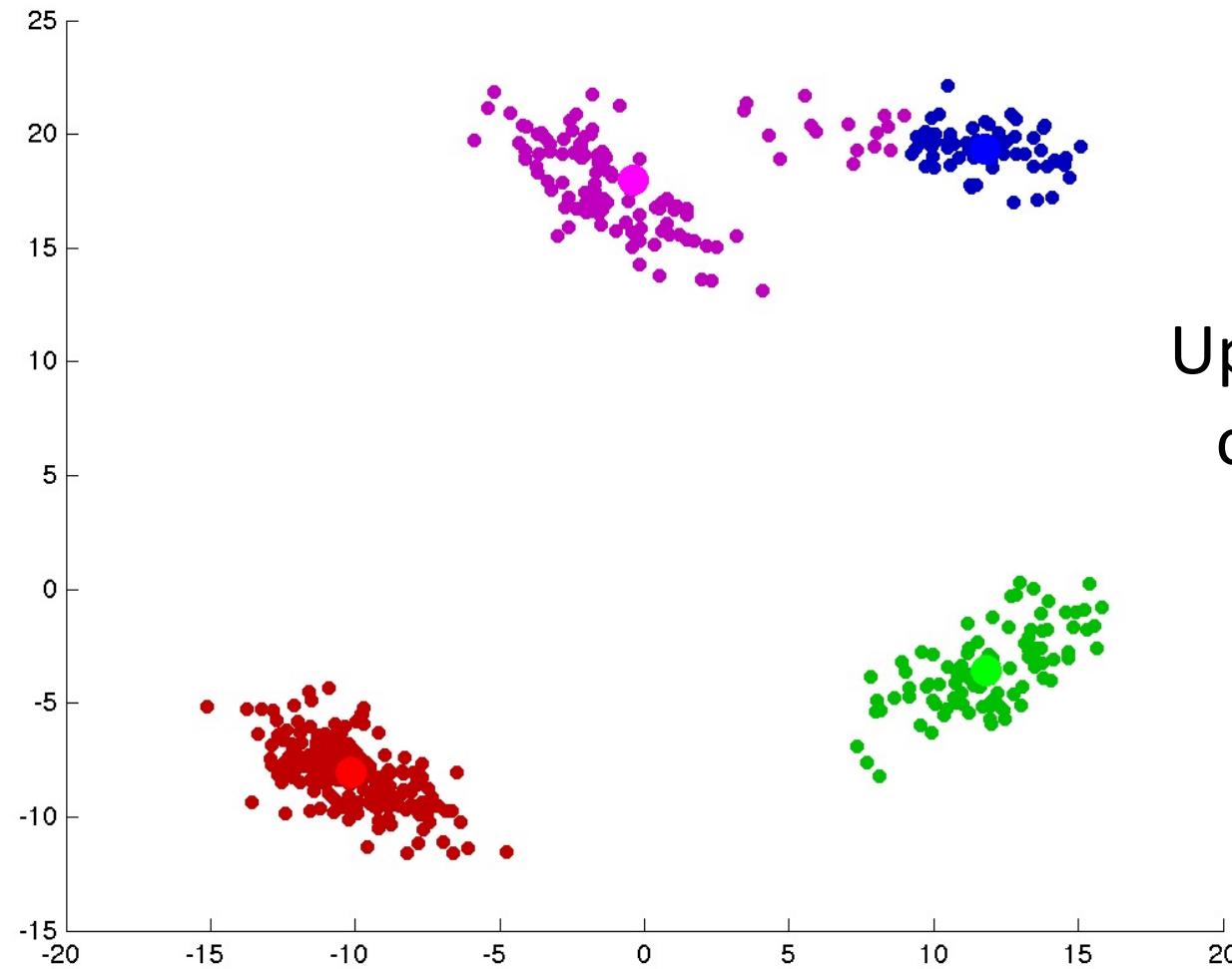
$$X = \begin{bmatrix} -9.0 & -7.3 \\ -10.9 & -9.0 \\ 13.7 & 19.3 \\ 13.8 & 20.4 \\ 12.8 & 20.6 \\ \vdots & \vdots \end{bmatrix}$$



# K-Means Example

Input: data matrix 'X'.

$$X = \begin{bmatrix} -9.0 & -7.3 \\ -10.9 & -9.0 \\ 13.7 & 19.3 \\ 13.8 & 20.4 \\ 12.8 & 20.6 \\ \vdots & \vdots \end{bmatrix}$$

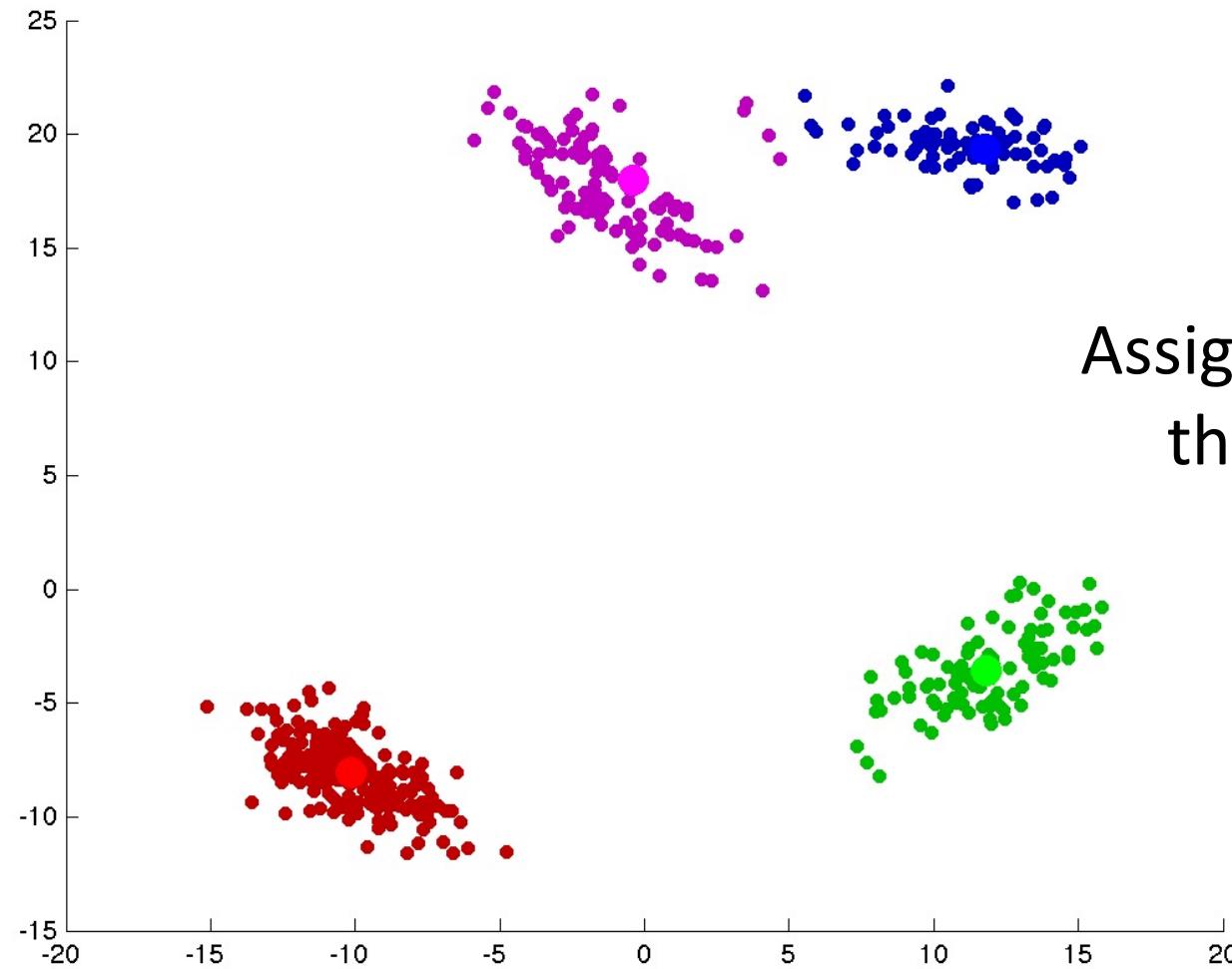


Update the mean  
of each group.

# K-Means Example

Input: data matrix 'X'.

$$X = \begin{bmatrix} -9.0 & -7.3 \\ -10.9 & -9.0 \\ 13.7 & 19.3 \\ 13.8 & 20.4 \\ 12.8 & 20.6 \\ \vdots & \vdots \end{bmatrix}$$

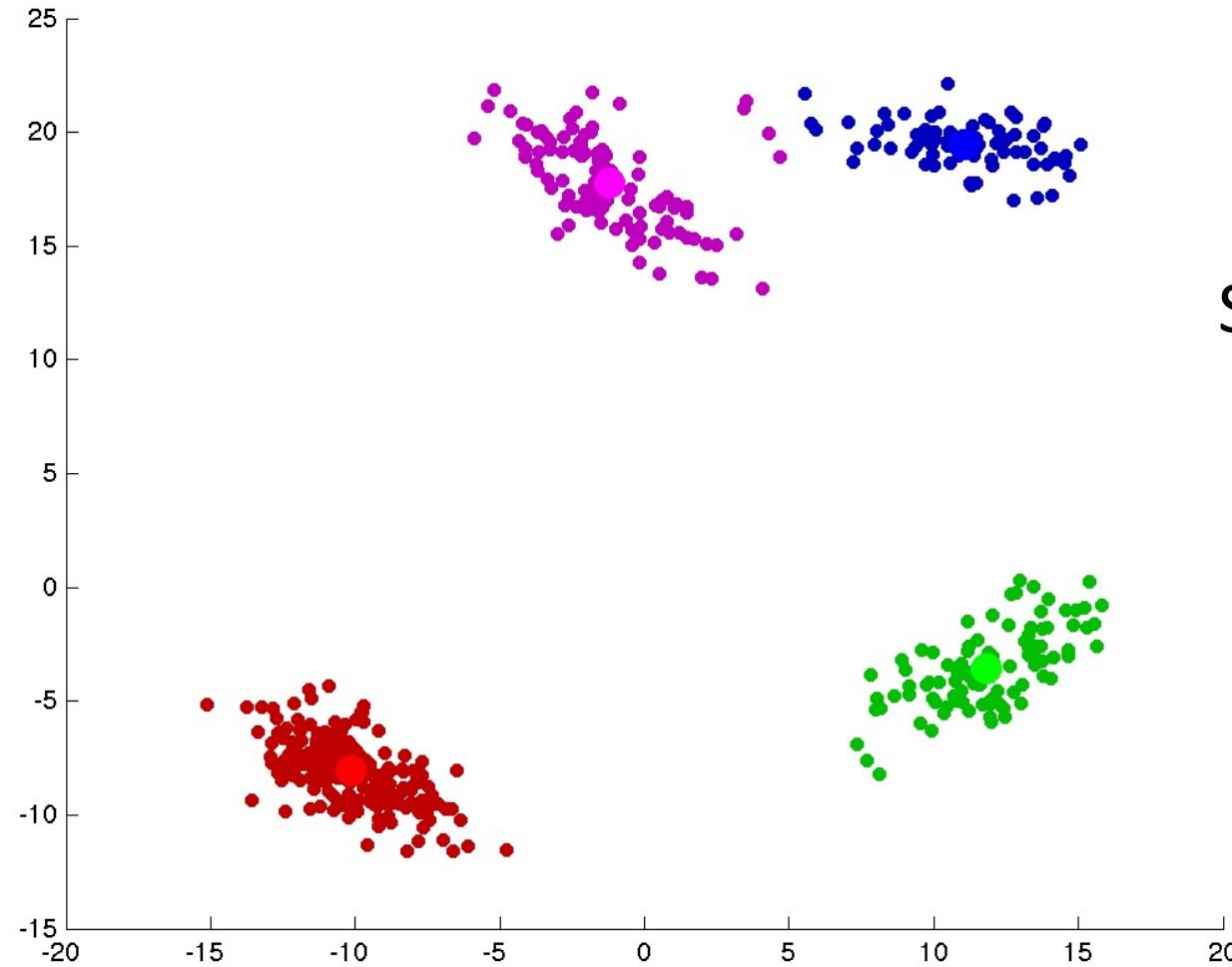


Assign each example to  
the closest mean.

# K-Means Example

Input: data matrix 'X'.

$$X = \begin{bmatrix} -9.0 & -7.3 \\ -10.9 & -9.0 \\ 13.7 & 19.3 \\ 13.8 & 20.4 \\ 12.8 & 20.6 \\ \vdots & \vdots \end{bmatrix}$$

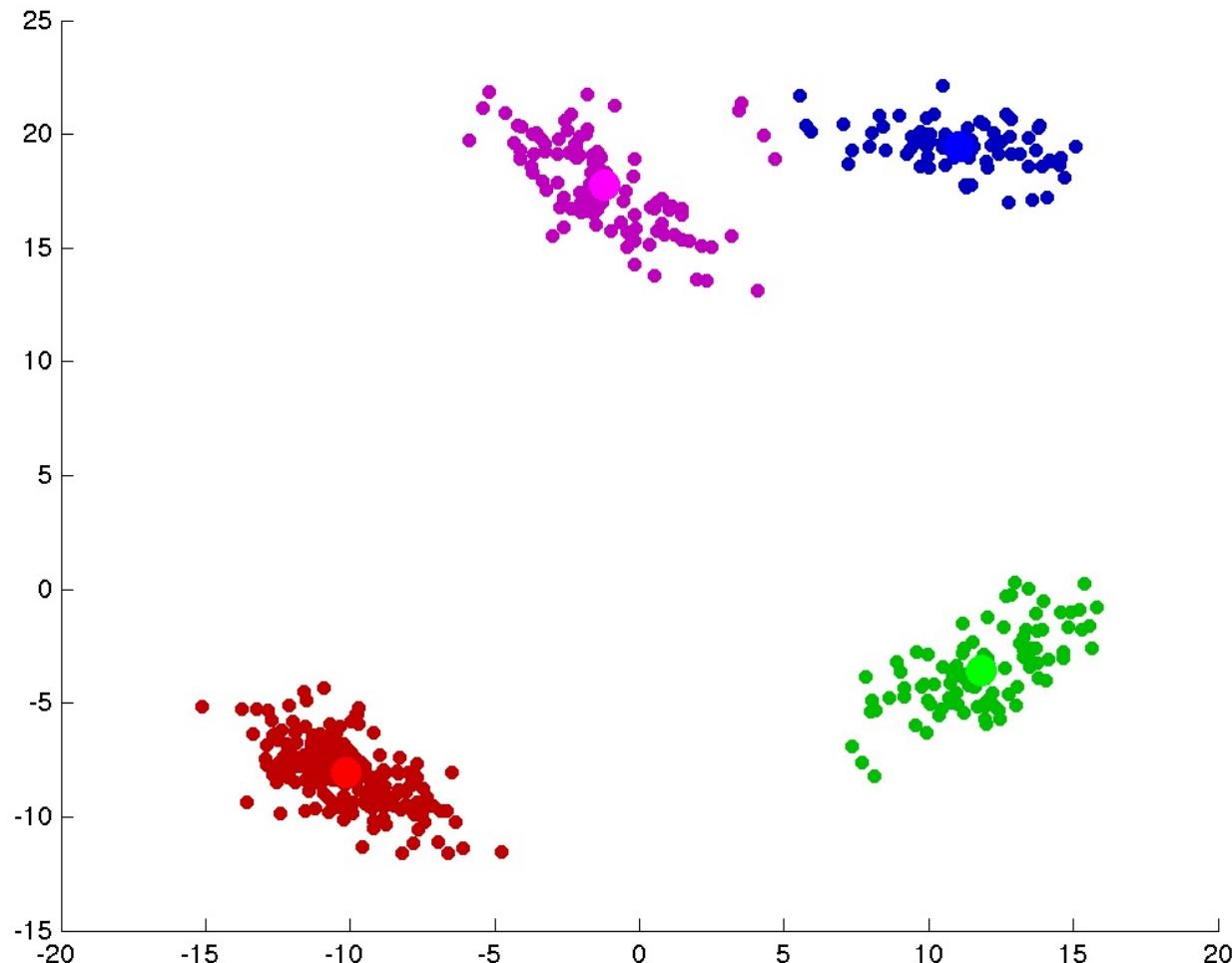


Stop if no examples  
change groups.

# K-Means Example

Input: data matrix 'X'.

$$X = \begin{bmatrix} -9.0 & -7.3 \\ -10.9 & -9.0 \\ 13.7 & 19.3 \\ 13.8 & 20.4 \\ 12.8 & 20.6 \\ \vdots & \vdots \end{bmatrix}$$



Interactive demo:

<https://www.naftaliharris.com/blog/visualizing-k-means-clustering>

Output:

- Clusters ' $\hat{y}$ '.
- Means 'W'.

$$\hat{y} = \begin{bmatrix} 2 \\ 2 \\ 3 \\ 3 \\ 1 \\ \vdots \end{bmatrix}$$

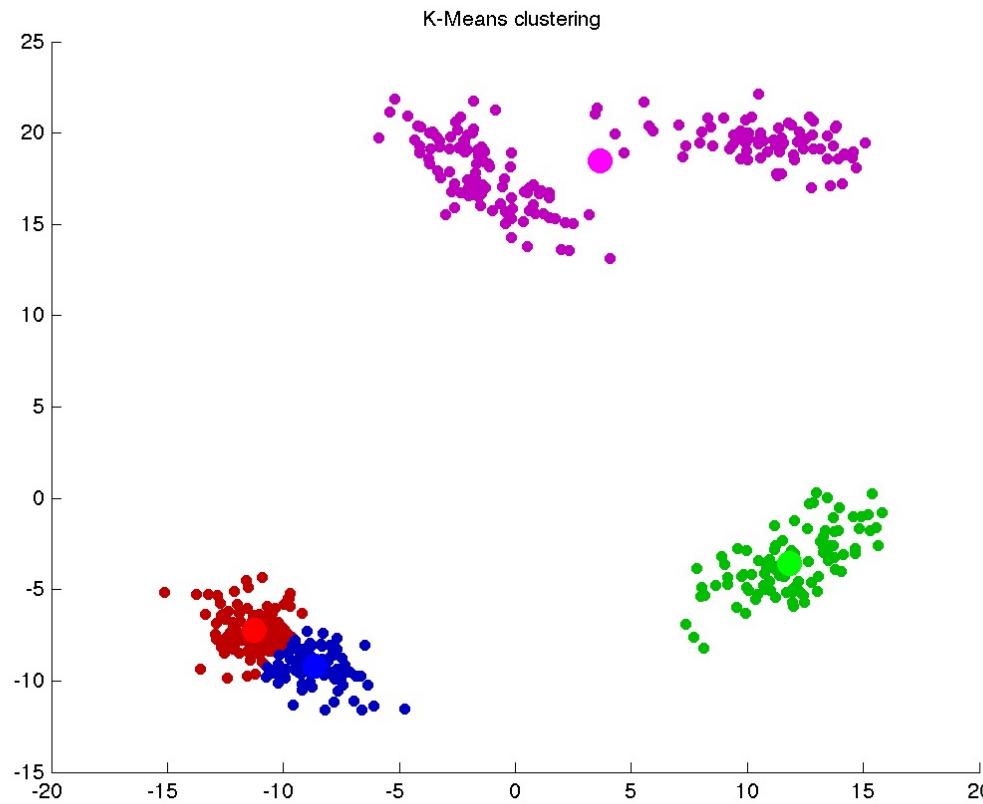
$$W = \begin{bmatrix} -1.2 & 17.8 \\ -10.2 & -8.0 \\ 11.0 & 19.5 \\ 11.8 & -3.6 \end{bmatrix}$$

$k$   
 $d$

# K-Means Issues

- Guaranteed to converge when using Euclidean distance.
- Given a new test example:
  - Assign it to the nearest mean to cluster it.
- Assumes you know number of clusters ‘k’.
  - Lots of heuristics to pick ‘k’, none satisfying:
    - [https://en.wikipedia.org/wiki/Determining\\_the\\_number\\_of\\_clusters\\_in\\_a\\_data\\_set](https://en.wikipedia.org/wiki/Determining_the_number_of_clusters_in_a_data_set)
- Each example is assigned to one (and only one) cluster:
  - No possibility for overlapping clusters or leaving examples unassigned.
- It may converge to sub-optimal solution...

# K-Means Clustering with Different Initialization



- Classic approach to dealing with sensitivity to initialization: **random restarts**.
  - Try several different random starting points, choose the “best”.
- See bonus slides for a more clever approach called k-means++.

# KNN vs. K-Means

- Don't confuse KNN classification and k-means clustering:

Property	KNN Classification	K-Means Clustering
Task	Supervised learning (given $y_i$ )	Unsupervised learning (no given $y_i$ ).
Meaning of 'k'	Number of neighbours to consider (not number of classes).	Number of clusters (always consider single nearest mean).
Initialization	No training phase.	Training that is sensitive to initialization.
Model complexity	Model is complicated for small 'k', simple for large 'k'.	Model is simple for small 'k', complicated for large 'k'.
Parametric?	Non-parametric: - Stores data 'X'	Parametric (for 'k' not depending on 'n') - Stores means 'W'

# What is K-Means Doing?

- We can interpret K-means steps as minimizing an objective:
  - Total sum of squared distances from each example  $x_i$  to its center  $w_{\hat{y}_i}$ :

$$f(w_1, w_2, \dots, w_k, \hat{y}_1, \hat{y}_2, \dots, \hat{y}_n) = \sum_{i=1}^n \|w_{\hat{y}_i} - x_i\|^2$$

Cluster of example 'i',  
 $\hat{y}_i \in \{1, 2, \dots, k\}$

- The k-means steps:
  - Minimize 'f' in terms of the  $\hat{y}_i$  (update cluster assignments).
  - Minimize 'f' in terms of the  $w_c$  (update means).
- Termination of the algorithm follows because:
  - Each step does not increase the objective.
  - There are a finite number of assignments to k clusters.

$$W = \left[ \begin{array}{c} w_1 \\ w_2 \\ \vdots \\ w_k \end{array} \right ] \Bigg \}^k_d$$

# What is K-Means Doing?

- We can interpret K-means steps as minimizing an objective:
  - Total sum of squared distances from each example  $x_i$  to its center  $w_{\hat{y}_i}$ :

$$f(w_1, w_2, \dots, w_k, \hat{y}_1, \hat{y}_2, \dots, \hat{y}_n) = \sum_{i=1}^n \|w_{\hat{y}_i} - x_i\|^2$$

- The k-means steps:
  - Minimize ‘f’ in terms of the  $\hat{y}_i$  (update cluster assignments).
  - Minimize ‘f’ in terms of the  $w_c$  (update means).
- Use final (post-convergence) ‘f’ score to choose between initializations (fixed ‘k’).
- Need to change  $w_c$  update under other distances:
  - L1-norm: set  $w_c$  to median (“k-medians”, see bonus).

Cluster of example ‘i’,  
 $\hat{y}_i \in \{1, 2, \dots, K\}$

$$W = \left[ \begin{array}{c} w_1 \\ w_2 \\ \vdots \\ w_k \end{array} \right] \Bigg\}^k$$

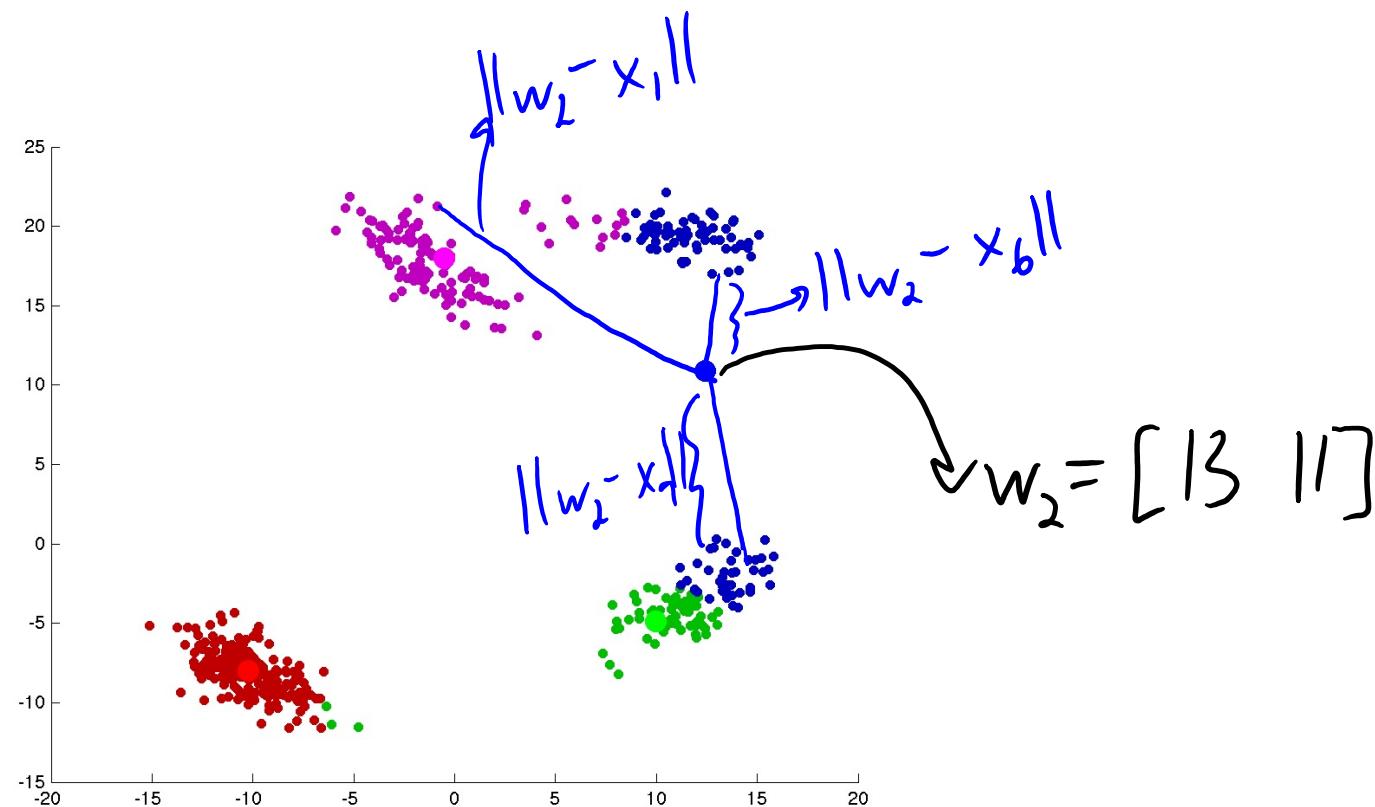
$d$

# Cost of K-means

- Bottleneck is calculating distance from each  $x_i$  to each mean  $w_c$ :

$$\|w_c - x_i\|^2 = \sum_{j=1}^d (w_{cj} - x_{ij})^2$$

*d-dimensional vector*



# Cost of K-means

- Bottleneck is calculating distance from each  $x_i$  to each mean  $w_c$ :

$$\|w_c - x_i\|^2 = \sum_{j=1}^d (w_{cj} - x_{ij})^2$$

*d-dimensional vector*

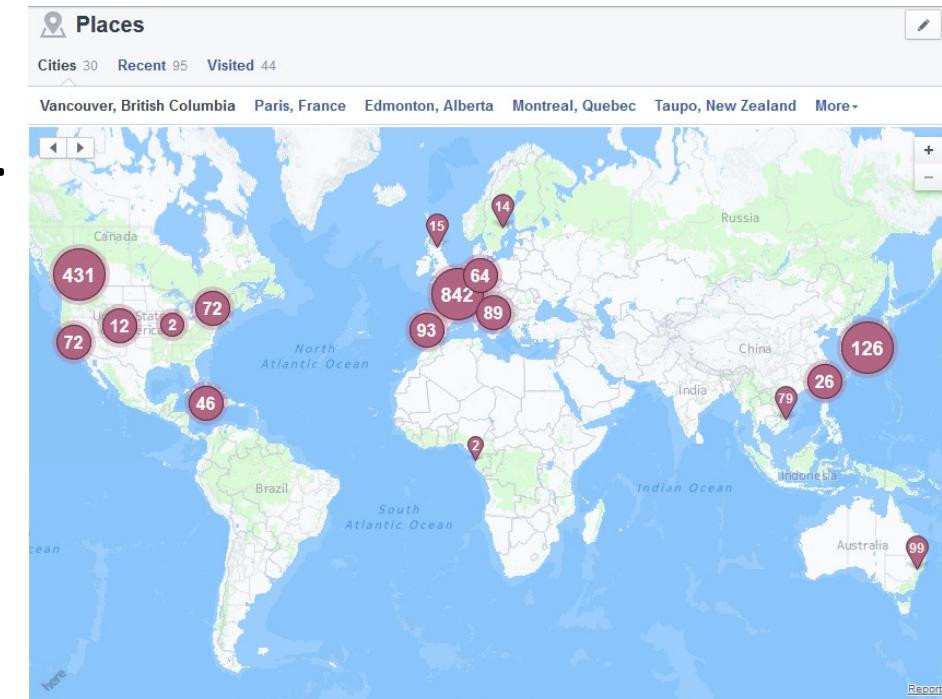
- Each time we do this costs  $O(d)$ .
- We need to compute distance from ‘n’ examples to ‘k’ clusters.
- Total cost of assigning examples to clusters is  $O(ndk)$ .
  - Fast if  $k$  is not too large.
- Updating means is cheaper:  $O(nd)$ .

$$w_c = \frac{1}{n_c} \sum_{i \in C} x_i$$

*Object in cluster.*  
*Loop over objects in cluster.*  
*Number of objects in cluster 'c'*

# Vector Quantization

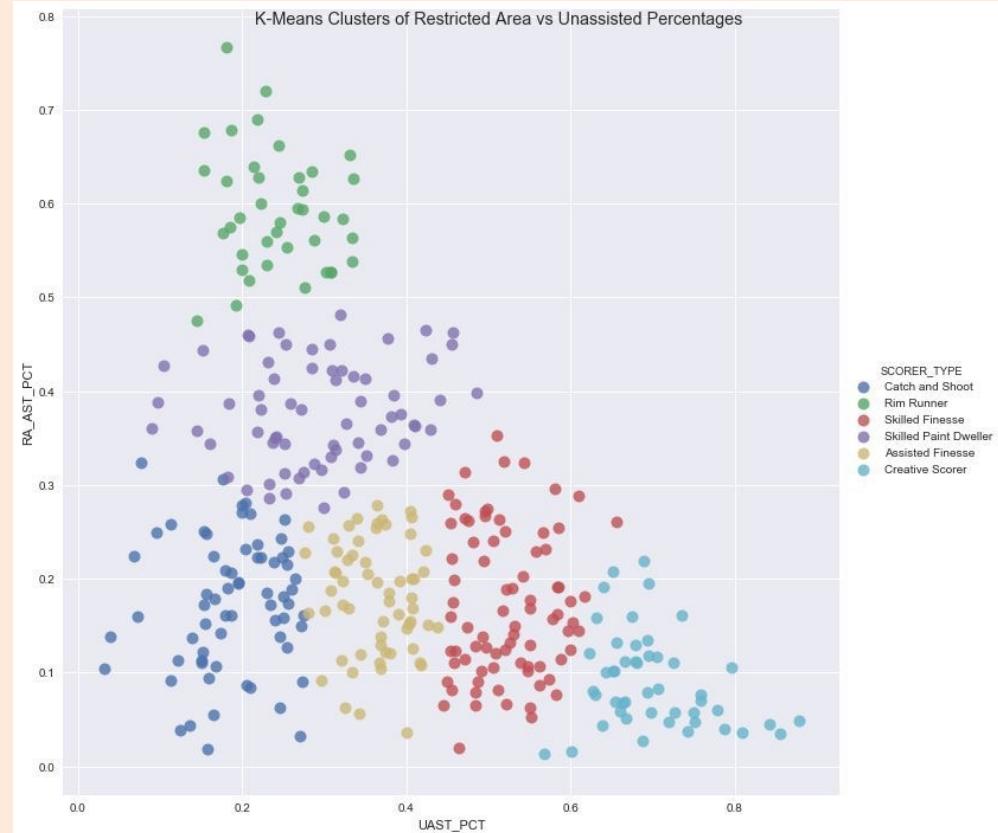
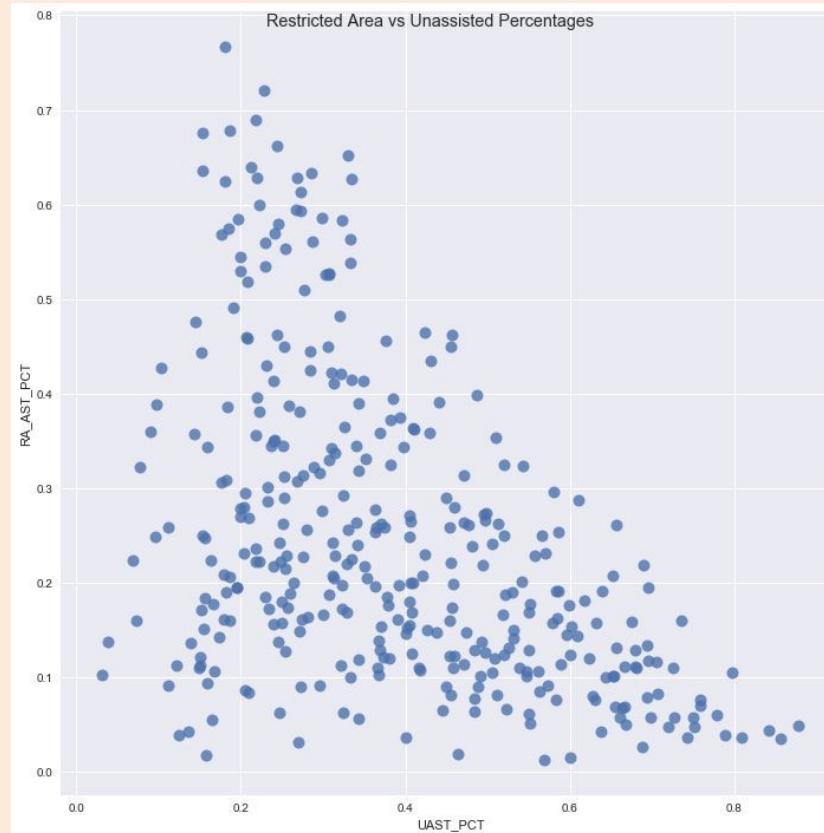
- K-means originally comes from signal processing.
- Designed for **vector quantization**:
  - Replace examples with the mean of their cluster (“prototype”).
- Example:
  - Facebook places: 1 location summarizes many.
  - What sizes of clothing should I make?



bonus!

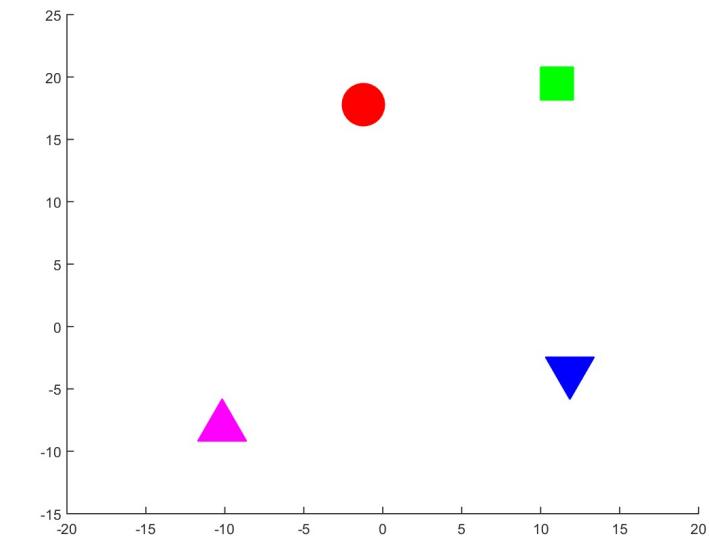
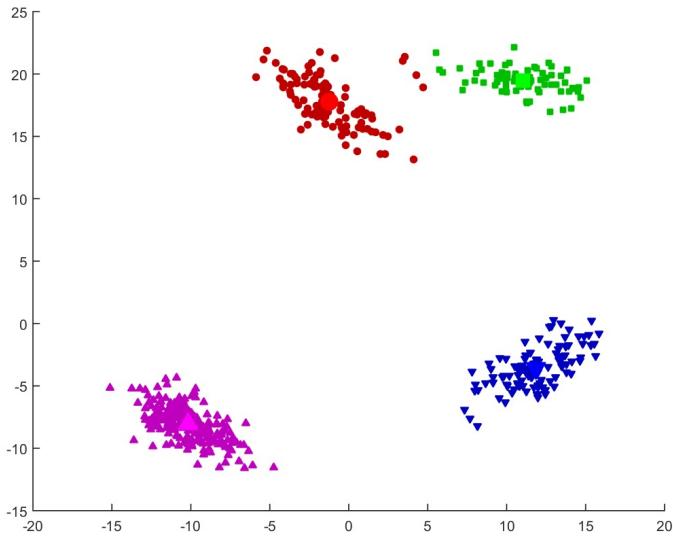
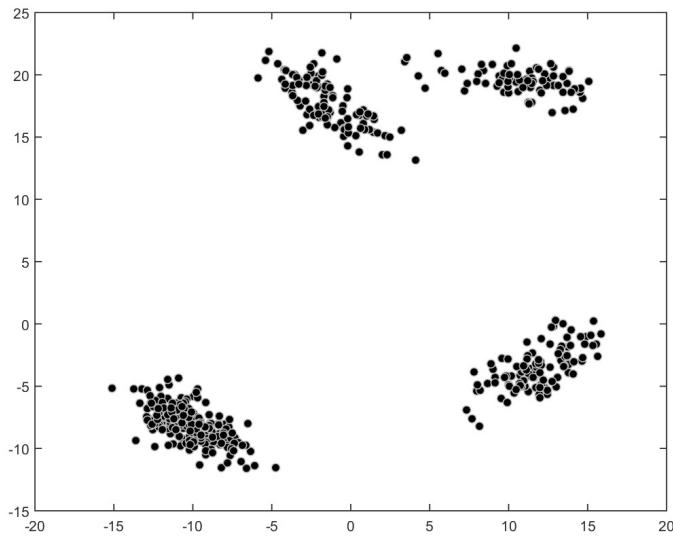
# Vector Quantization for Basketball Players

- Clustering NBA basketball players based on shot type/percentage:



- The “prototypes” (means) give offensive styles (like “catch and shoot”).

# Vector Quantization Example



$$X = \begin{bmatrix} -9.0 & -7.3 \\ -10.9 & -9.0 \\ 13.7 & 19.3 \\ 13.8 & 20.4 \\ 12.8 & 20.6 \\ \vdots & \vdots \end{bmatrix} \quad n \times d$$

*Rnn k-means*

$$W = \begin{bmatrix} -1.2 & 17.8 \\ -10.2 & -8.0 \\ 11.0 & 19.5 \\ 11.8 & -3.6 \end{bmatrix}$$

$\hat{y} = \begin{bmatrix} 2 \\ 2 \\ 3 \\ 3 \\ 1 \end{bmatrix}$

*these mean values are the prototypes.*

*Assume each example can be approximated by prototype*

*Approximate objects with means.*

$$X = \begin{bmatrix} -10.2 & -8.0 \\ -10.2 & -8.0 \\ 11.0 & 19.5 \\ 11.0 & 19.5 \\ -1.2 & 17.8 \\ \vdots & \vdots \end{bmatrix}$$

bonus!

# (Bad) Vector Quantization in Practice

- Political parties can be thought as a form of vector quantization:



- Hope is that parties represent what a cluster of voters want.
  - With larger ‘k’ more voters have a party that closely reflects them.
  - With smaller ‘k’, parties are less accurate reflections of people.

# Summary

- **Random forests:** bagging of deep randomized decision trees.
  - One of the best “out of the box” classifiers.
- **Type of ensemble methods:**
  - “Boosting” (covered later in course) reduces  $E_{\text{train}}$  and “averaging” reduces  $E_{\text{approx}}$ .
- **Unsupervised learning:** fitting data without explicit labels.
- **Clustering:** finding ‘groups’ of related examples.
- **K-means:** simple iterative clustering strategy.
  - Fast but sensitive to initialization.
- **Vector quantization:**
  - Compressing examples by replacing them with the mean of their cluster.
- **Next time:**
  - non-parametric clustering.

bonus!

# Extremely-Randomized Trees

- **Extremely-randomized trees** add an extra level of randomization:
  1. Each tree is fit to a bootstrap sample.
  2. Each split only considers a random subset of the features.
  3. **Each split only considers a random subset of the possible thresholds.**
- So instead of considering up to ‘n’ thresholds,  
only consider 10 or something small.
  - Leads to different partitions so potentially more independence.

bonus!

# Bayesian Model Averaging

- Recall the key observation regarding ensemble methods:
  - If models overfit in “different” ways, averaging gives better performance.
- But should all models get equal weight?
  - E.g., decision trees of different depths, when lower depths have low training error.
  - E.g., a random forest where one tree does very well (on validation error) and others do horribly.
  - In science, research may be fraudulent or not based on evidence.
- In these cases, naïve averaging may do worse.

bonus!

# Bayesian Model Averaging

- Suppose we have a set of 'm' probabilistic binary classifiers  $w_j$ .
- If each one gets equal weight, then we predict using:

$$p(y_i | x_i) = \frac{1}{m} p(y_i | w_1, x_i) + \frac{1}{m} p(y_i | w_2, x_i) + \dots + \left(\frac{1}{m}\right) p(y_i | w_m, x_i)$$

- Bayesian model averaging treats model ' $w_j$ ' as a random variable:  
 $w_j \perp x_i$  Assume

$$p(y_i | x_i) = \sum_{j=1}^m p(y_i, w_j | x_i) = \sum_{j=1}^m p(y_i | w_j, x_j) p(w_j | x_j) = \sum_{j=1}^m p(y_i | w_j, x_j) p(w_j)$$

- So we should weight by probability that  $w_j$  is the correct model:
  - Equal weights assume all models are equally probable.

bonus!

# Bayesian Model Averaging

- Can get better weights by conditioning on training set:

$$p(w_j | X, y) \propto p(y | w_j, X) p(w_j | X) = p(y | w_j, X) p(w_j)$$

- The ‘likelihood’  $p(y | w_j, X)$  makes sense:
  - We should give more weight to models that predict ‘y’ well.
  - Note that hidden denominator penalizes complex models.
- The ‘prior’  $p(w_j)$  is our ‘belief’ that  $w_j$  is the correct model.
- **This is how rules of probability say we should weigh models.**
  - The ‘correct’ way to predict given what we know.
  - But it makes some people unhappy because it is subjective.

Again, assuming  
 $w_j | X$

bonus!

# What is K-Means Doing?

- How are k-means steps decreasing this objective?

$$f(w_1, w_2, \dots, w_k, \hat{y}_1, \hat{y}_2, \dots, \hat{y}_n) = \sum_{i=1}^n \|w_{\hat{y}_i} - x_i\|^2$$

- If we just write as function of a particular  $\hat{y}_i$ , we get:

$$f(\hat{y}_i) = \|w_{\hat{y}_i} - x_i\|^2 + (\text{constant})$$

- The “constant” includes all other terms, and doesn’t affect location of min.
- We can minimize in terms of  $\hat{y}_i$  by setting it to the ‘c’ with  $w_c$  closest to  $x_i$ .

bonus!

# What is K-Means Doing?

- How are k-means steps decreasing this objective?

$$f(w_1, w_2, \dots, w_k, \hat{y}_1, \hat{y}_2, \dots, \hat{y}_n) = \sum_{i=1}^n \|w_{\hat{y}_i} - x_i\|^2$$

- If we just write as function of a particular  $w_{cj}$  we get:

$$f(w_{cj}) = \underbrace{\sum_{i \in C} \sum_{j=1}^d (w_{cj} - x_{ij})^2}_{\text{set of examples with } \hat{y}_i = c} + (\text{constant})$$

- Derivative is given by:  $f'(w_{cj}) = 2 \sum_{i \in C} (w_{cj} - x_{ij})$

- Setting equal to 0 and solving for  $w_{cj}$  gives:  $\sum_{i \in C} w_{cj} = \sum_{i \in C} x_{ij}$  or  $w_{cj} * n_c = \sum_{i \in C} x_{ij}$  or  $w_{cj} = \frac{1}{n_c} \sum_{i \in C} x_{ij}$

bonus!

# K-Medians Clustering

- With other distances k-means may not converge.
  - But we can make it converge by changing the updates so that they are minimizing an objective function.
- E.g., we can use the L1-norm objective:  $\sum_{i=1}^n \|w_{y_i} - x_i\|_1$
- Minimizing the L1-norm objective gives the ‘k-medians’ algorithm:
  - Assign points to clusters by finding “mean” with smallest L1-norm distance.
  - Update ‘means’ as median value (dimension-wise) of each cluster.
    - This minimizes the L1-norm distance to all the points in the cluster.
- This approach is more robust to outliers.

↑ k-means will put a cluster here.

bonus!

# What is the “L1-norm and median” connection?

- Point that minimizes the sum of squared L2-norms to all points:

$$f(w) = \sum_{i=1}^n \|w - x_i\|^2$$

- Is given by the **mean** (just take derivative and set to 0):

$$w = \frac{1}{n} \sum_{i=1}^n x_i$$

- Point that minimizes the sum of L1-norms to all points:

$$f(w) = \sum_{i=1}^n \|w - x_i\|_1$$

- Is given by the **median** (derivative of absolute value is +1 if positive and -1 if negative, so any point with half of points larger and half of points smaller is a solution).

bonus!

# K-Medoids Clustering

- A disadvantage of k-means in some applications:
  - The **means might not be valid data** points.
  - May be important for vector quantization.
- E.g., consider bag of words features like [0,0,1,1,0].
  - We have words 3 and 4 in the document.
- A mean from k-means might look like [0.1 0.3 0.8 0.2 0.3].
  - What does it mean to have 0.3 of word 2 in a document?
- Alternative to k-means is **k-medoids**:
  - Same algorithm as k-means, except the means must be data points.
  - Update the means by finding example in cluster minimizing squared L2-norm distance to all points in the cluster.

bonus!

# K-Means Initialization

- K-means is fast but **sensitive to initialization.**
- Classic approach to initialization: **random restarts.**
  - Run to convergence using different random initializations.
  - Choose the one that minimizes average squared distance of data to means.
- Newer approach: **k-means++**
  - Random initialization that **prefers means that are far apart.**
  - Yields **provable bounds** on expected approximation ratio.

bonus!

# K-Means++

- Steps of k-means++:

1. Select initial mean  $w_1$  as a random  $x_i$ .

2. Compute distance  $d_{ic}$  of each example  $x_i$  to each mean  $w_c$ .

$$d_{ic} = \sqrt{\sum_{j=1}^d (x_{ij} - w_{cj})^2} = \|x_i - w_c\|_2$$

3. For each example 'i' set  $d_i$  to the distance to the closest mean.

$$d_i = \min_c \{ d_{ic} \}$$

4. Choose next mean by sampling an example 'i' proportional to  $(d_i)^2$ .

$$p_i \propto d_i^2 \Rightarrow p_i = \frac{d_i^2}{\sum_{j=1}^n d_j^2}$$

Can be  
done in  
 $O(n)$ .

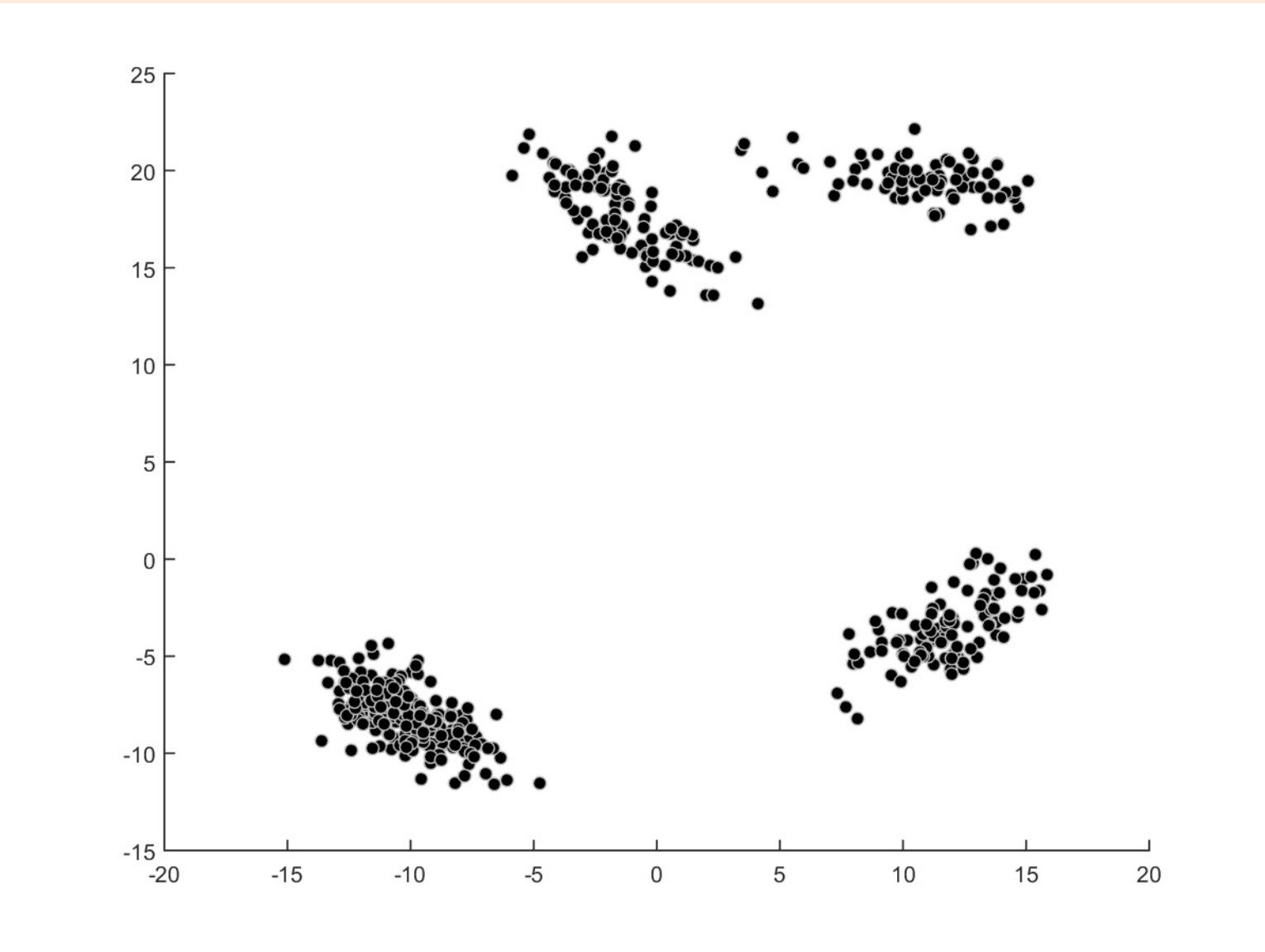
"probability that we  
choose  $x_i$  as next mean"

5. Keep returning to step 2 until we have k-means.

- Expected approximation ratio is  $O(\log(k))$ .

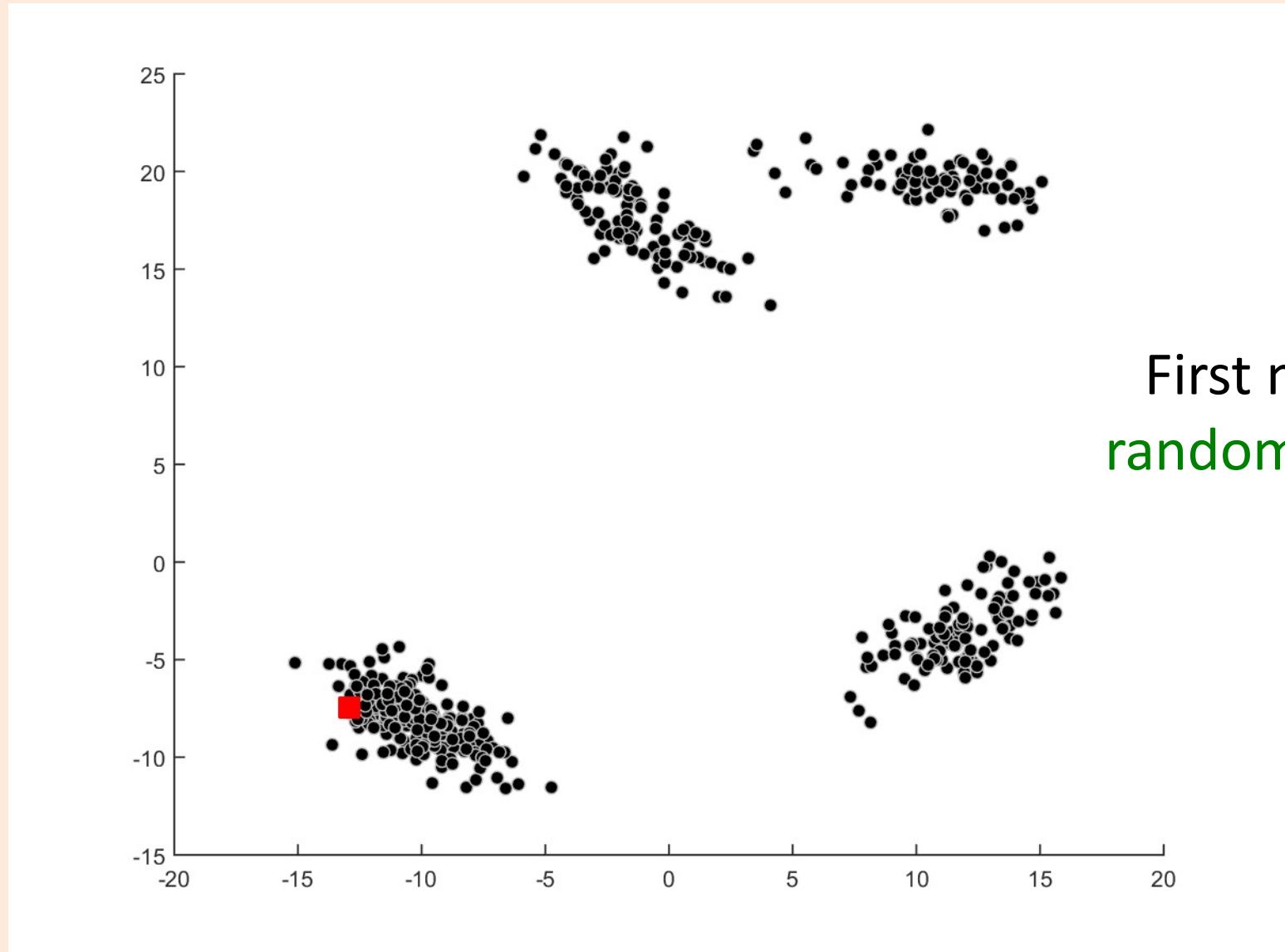
bonus!

# K-Means++



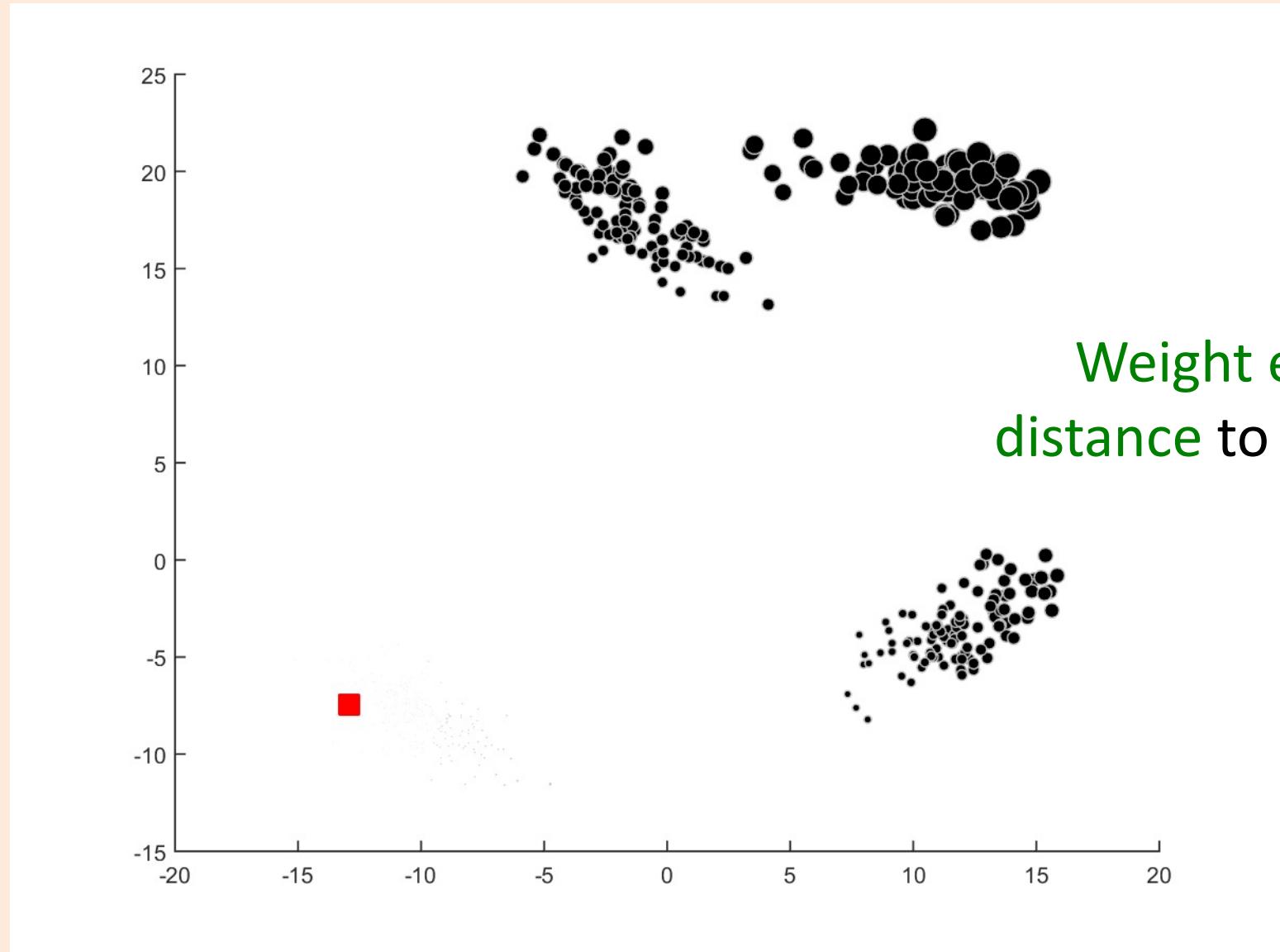
bonus!

# K-Means++



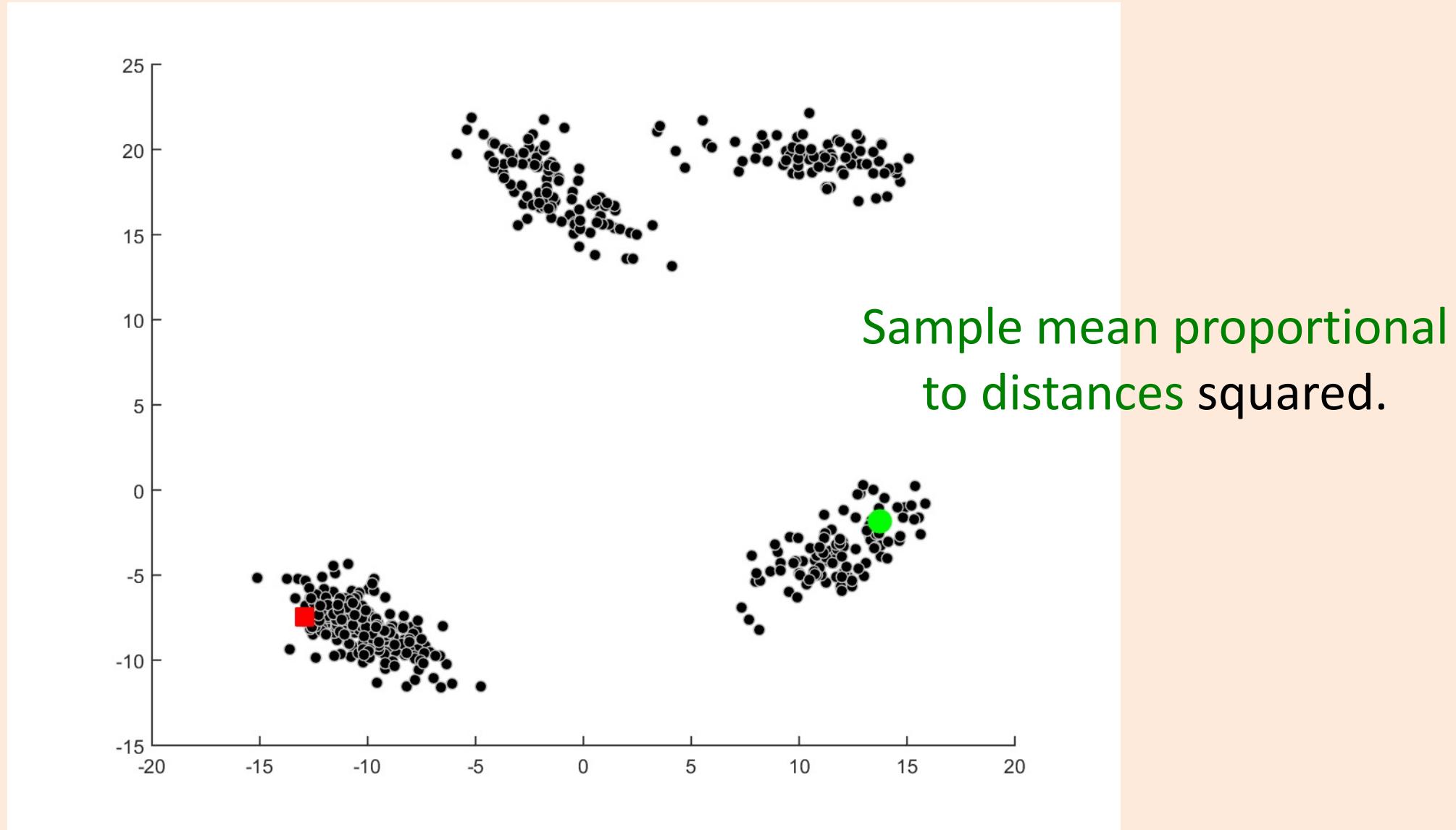
bonus!

# K-Means++



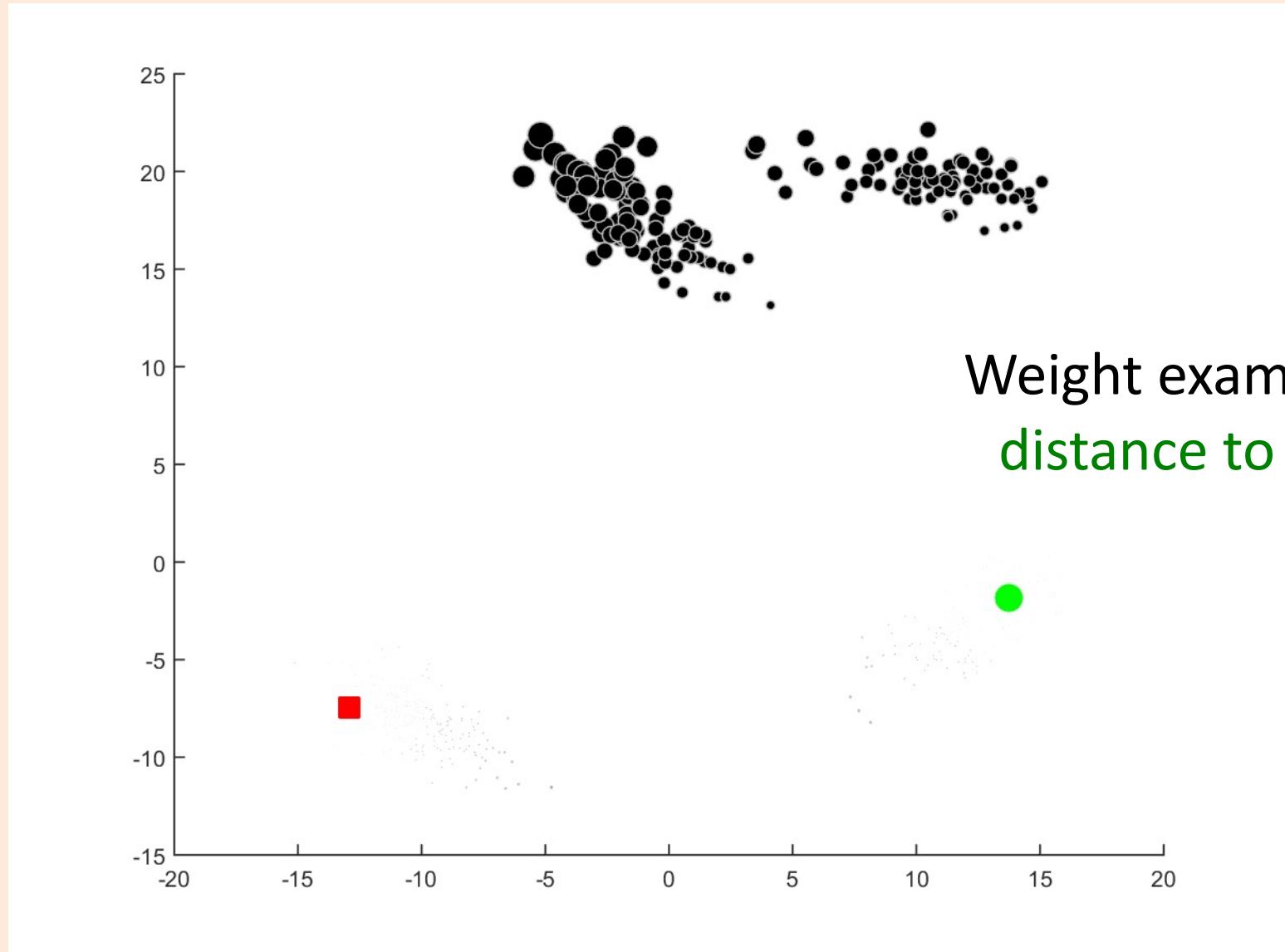
bonus!

# K-Means++



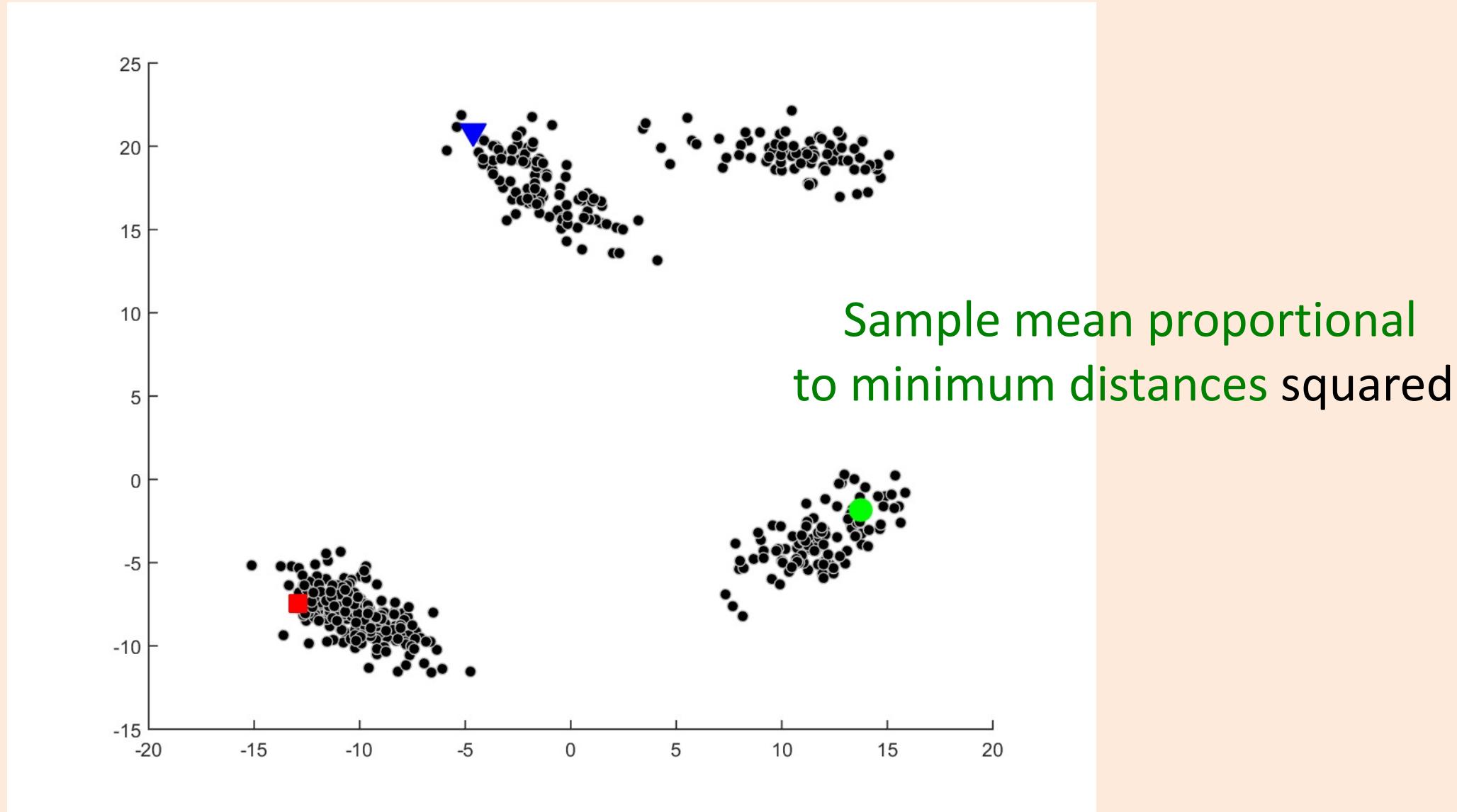
bonus!

# K-Means++



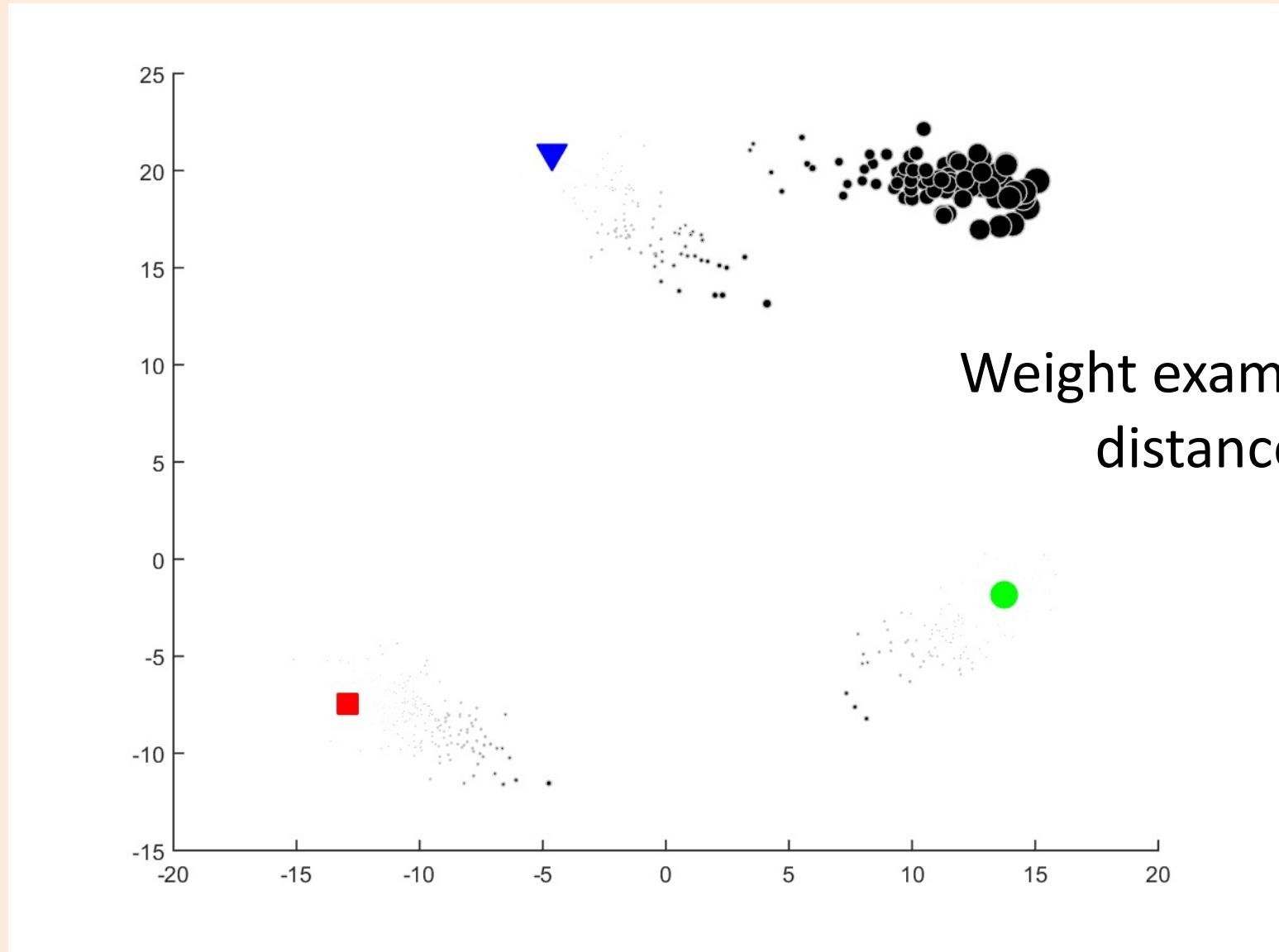
bonus!

# K-Means++



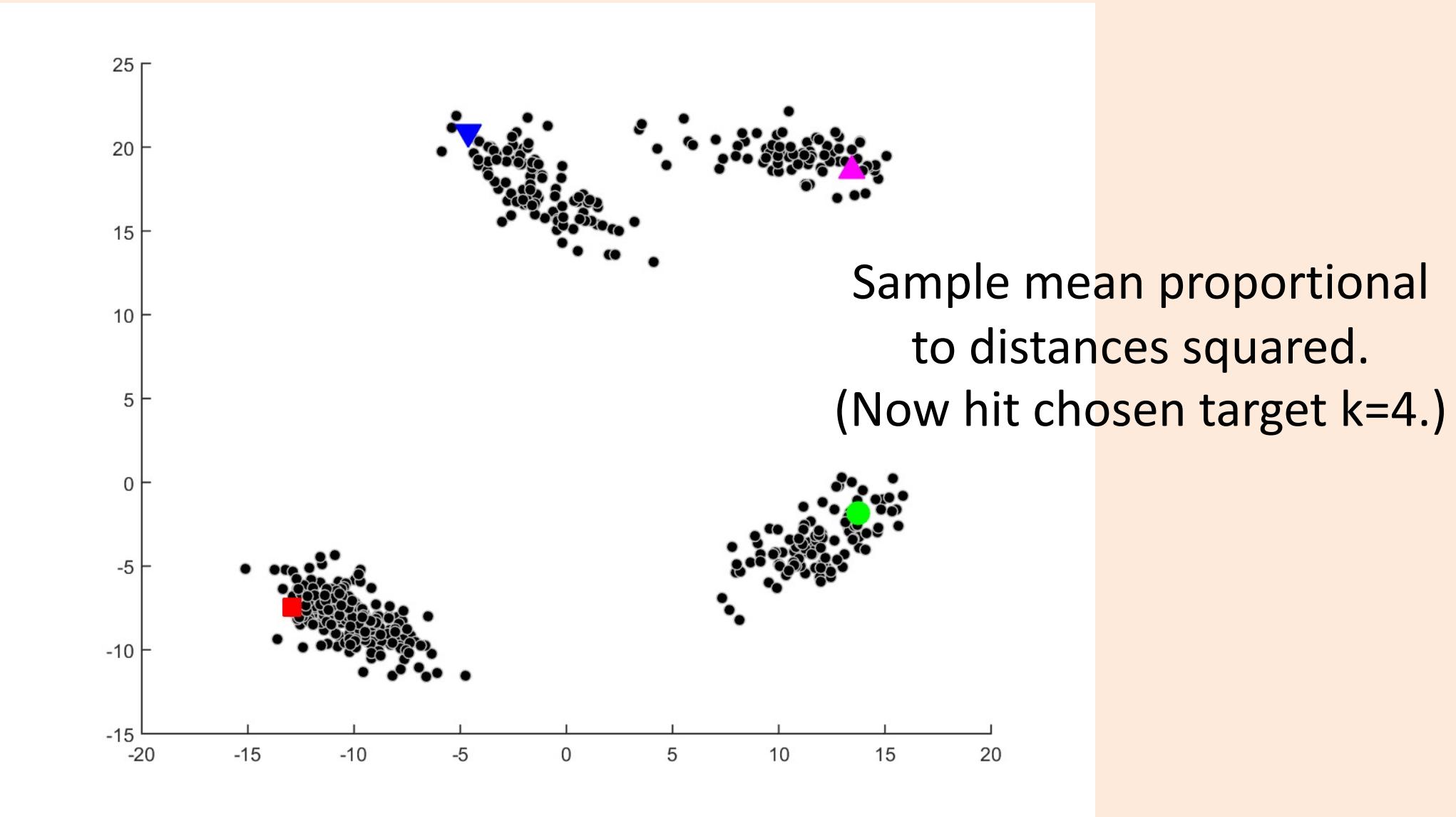
bonus!

# K-Means++



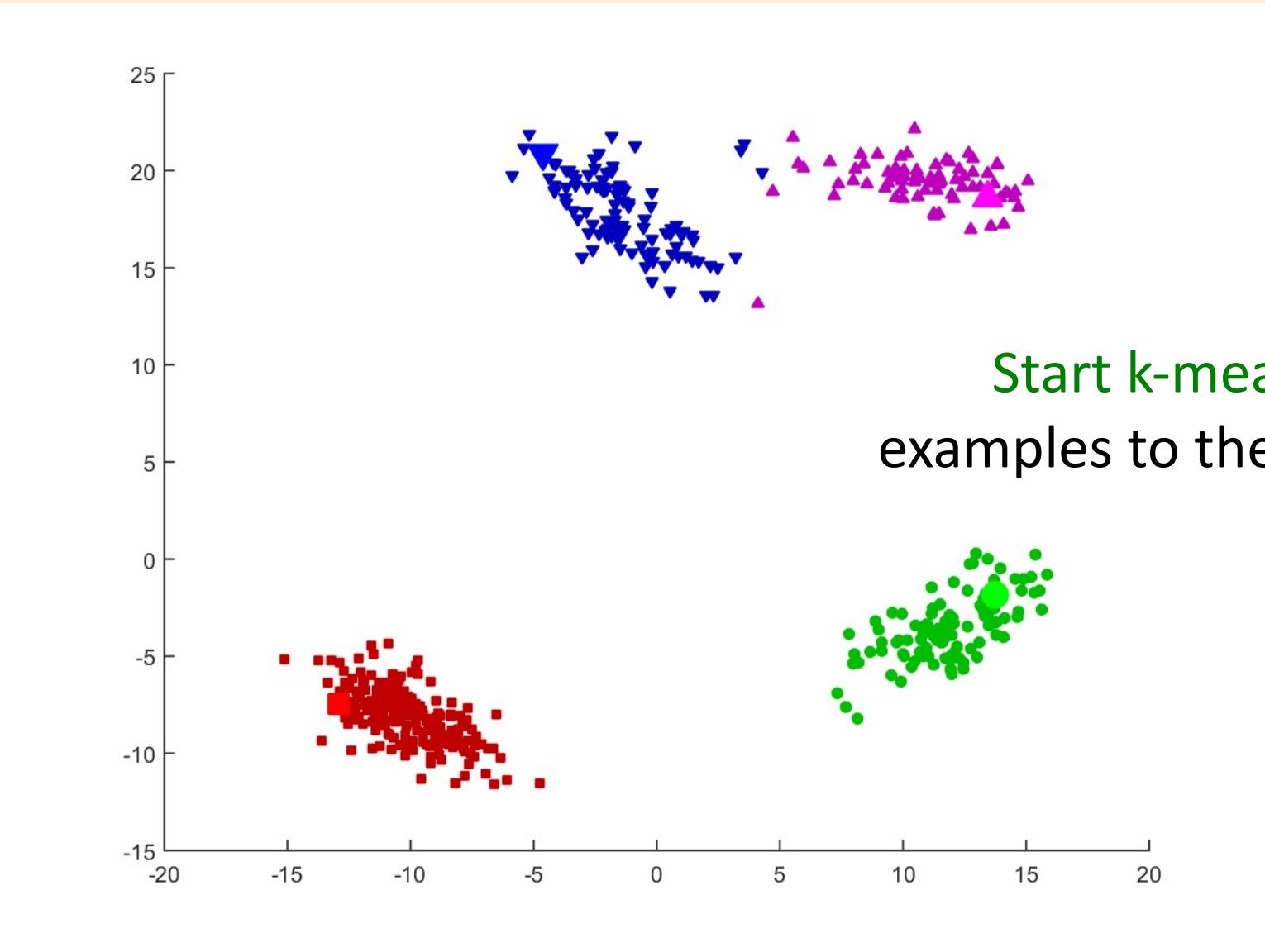
bonus!

# K-Means++



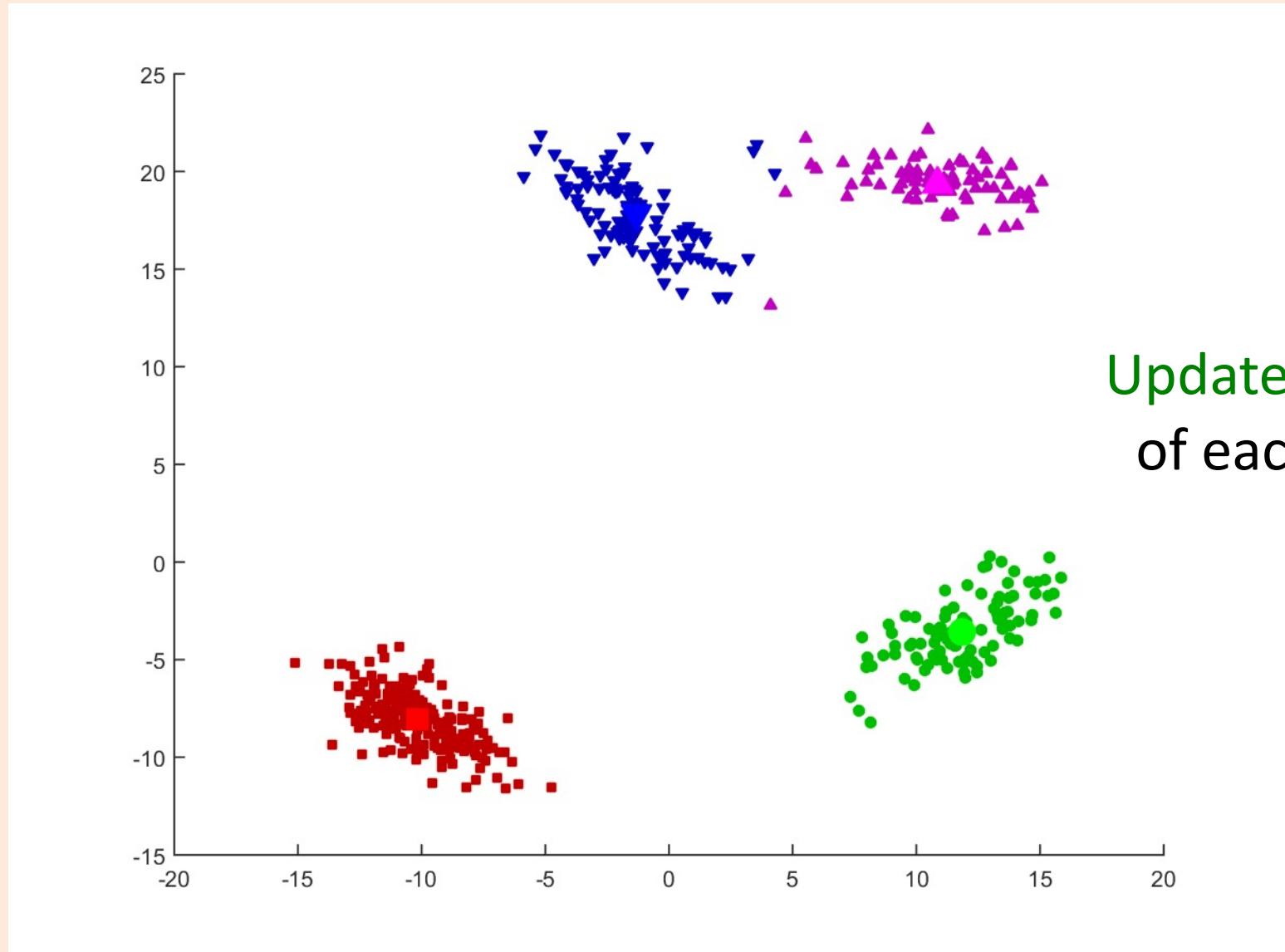
bonus!

# K-Means++



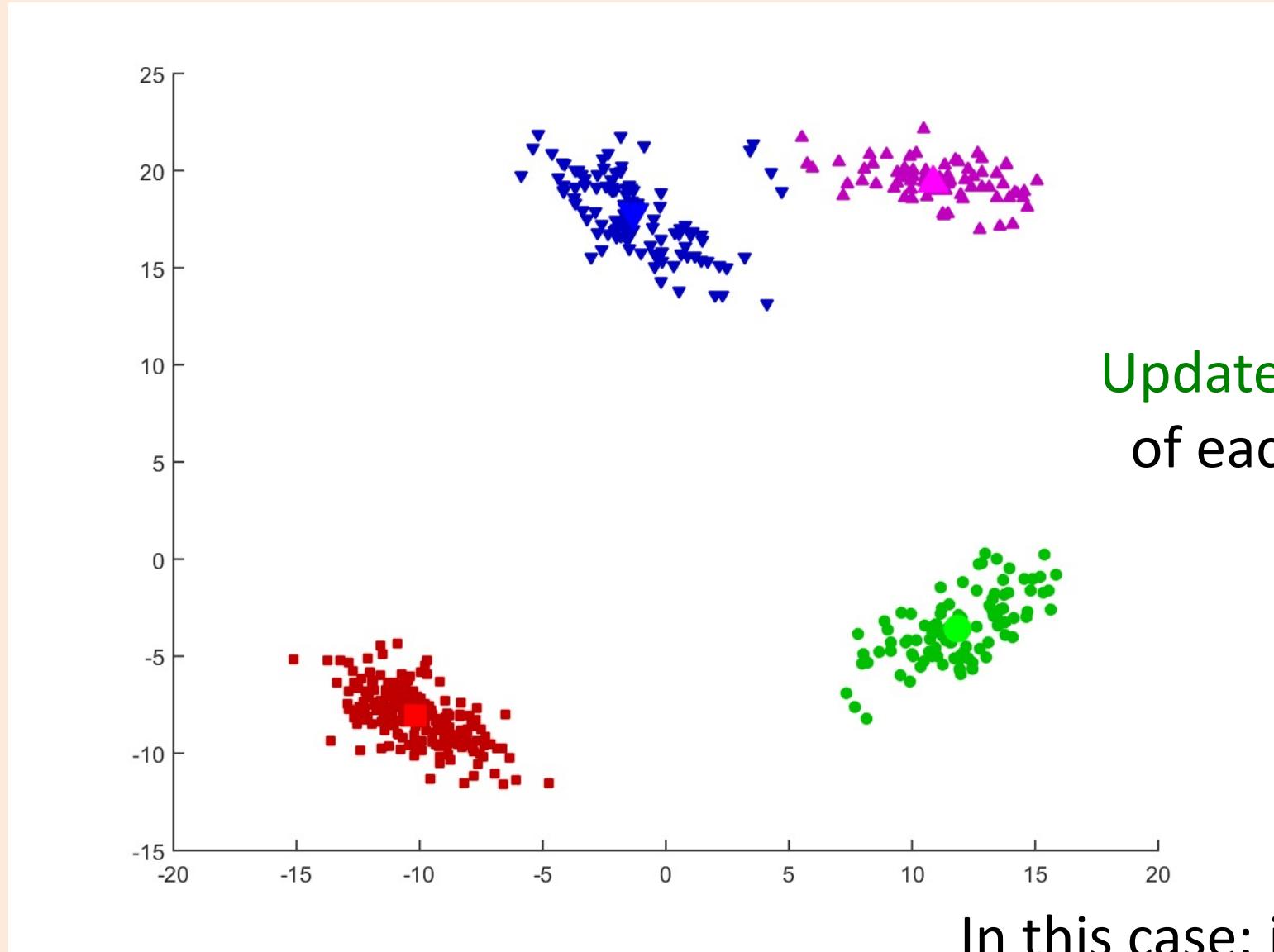
bonus!

# K-Means++



bonus!

# K-Means++



bonus!

# Discussion of K-Means++

- Recall the objective function k-means tries to minimize:

$$f(w, c) = \sum_{i=1}^n \|x_i - w_{c(i)}\|_2^2$$

↑ all means      ↑ all assignments

- The initialization of 'W' and 'c' given by k-means++ satisfies:

$$\mathbb{E}[f(w, c)] \leq O(\log(k))$$

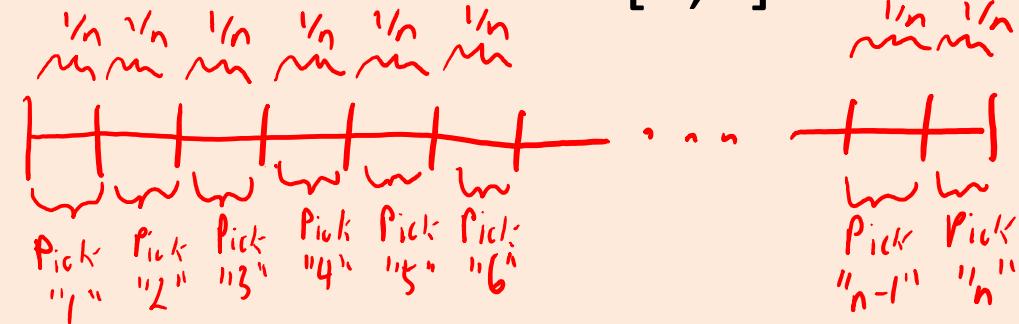
↑ expectation over random samples      ↓ f(w\*, c\*) "Best" mean and clustering according to objective.

- Get good clustering with high probability by re-running.
- However, there is no guarantee that  $c^*$  is a good clustering.

bonus!

# Uniform Sampling

- Standard approach to generating a random number from  $\{1, 2, \dots, n\}$ :
  1. Generate a uniform random number 'u' in the interval  $[0, 1]$ .
  2. Return the largest index 'i' such that  $u \leq i/n$ .
- Conceptually, this divides interval  $[0, 1]$  into 'n' equal-size pieces:



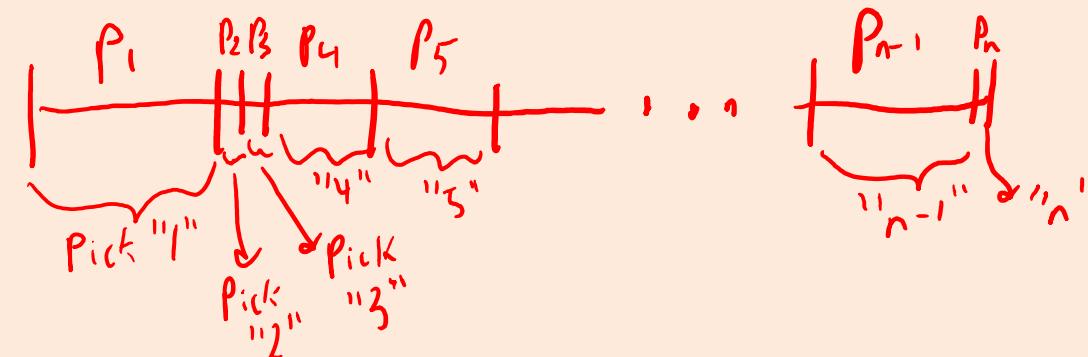
- This assumes  $p_i = 1/n$  for all 'i'.

↑ probability of picking number 'i'.

bonus!

# Non-Uniform Sampling

- Standard approach to generating a random number for general  $p_i$ .
  1. Generate a uniform random number 'u' in the interval [0,1].
  2. Return the largest index 'i' such that  $u \leq \sum_{j=1}^i p_j$
- Conceptually, this divides interval [0,1] into non-equal-size pieces:



- Can sample from a generic discrete probability distribution in  $O(n)$ .
- If you need to generate 'm' samples:
  - Cost is  $O(n + m \log(n))$  with binary search and storing cumulative sums.

bonus!

# How many iterations does k-means take?

- Each update of the ' $\hat{y}_i$ ' or ' $w_c$ ' does not increase the objective 'f'.
- And there are  $k^n$  possible assignments of the  $\hat{y}_i$  to 'k' clusters.
- So within  $k^n$  iterations you cannot improve the objective by changing  $\hat{y}_i$ , and the algorithm stops.
- Tighter-but-more-complicated “smoothed” analysis:
  - <https://arxiv.org/pdf/0904.1113.pdf>

bonus!

# Vector Quantization: Image Colors

- Usual RGB representation of a pixel's color: three 8-bit numbers.
  - For example, [241 13 50] = ■.
  - Can apply k-means to find set of prototype colours.



Original:  
(24-bits/pixel)

$$X = \begin{bmatrix} & \\ & \end{bmatrix} \quad \text{# pixels}$$

(all 3  
8-bit)

Run k-means with  
 $2^6$  clusters:

Average red, green,  
and blue values in  
cluster 1.

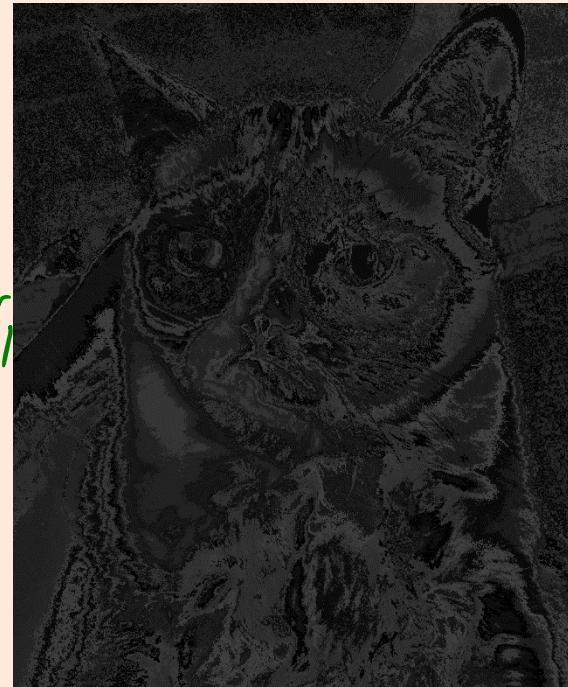
$$W = \begin{bmatrix} & \\ & \end{bmatrix} \quad 2^6$$

3      64 colours

K-means predictions:  
(6-bits/pixel)

$$\hat{Y} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 2 \\ 2 \\ 1 \\ 1 \\ 3 \end{bmatrix} \quad \text{cluster for each pixel}$$

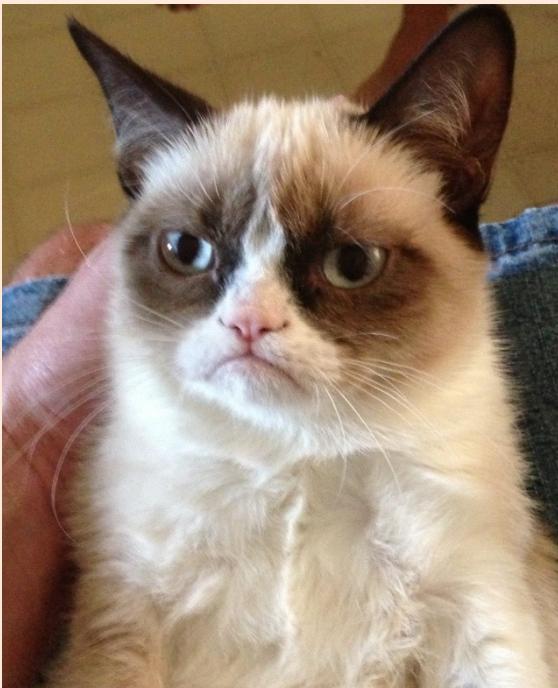
1 6-bit number  
which refers to one of  $2^6$  colours.



bonus!

# Vector Quantization: Image Colors

- Usual RGB representation of a pixel's color: three 8-bit numbers.
  - For example, [241 13 50] = ■.
  - Can apply k-means to find set of prototype colours.



Original:  
(24-bits/pixel)

$$X = \begin{bmatrix} & \\ & \end{bmatrix} \quad \text{# pixels} \quad \text{(all } 3 \text{ 8-bit)}$$

Run k-means with  
 $2^6$  clusters:

Average red, green,  
and blue values in  
cluster 1.

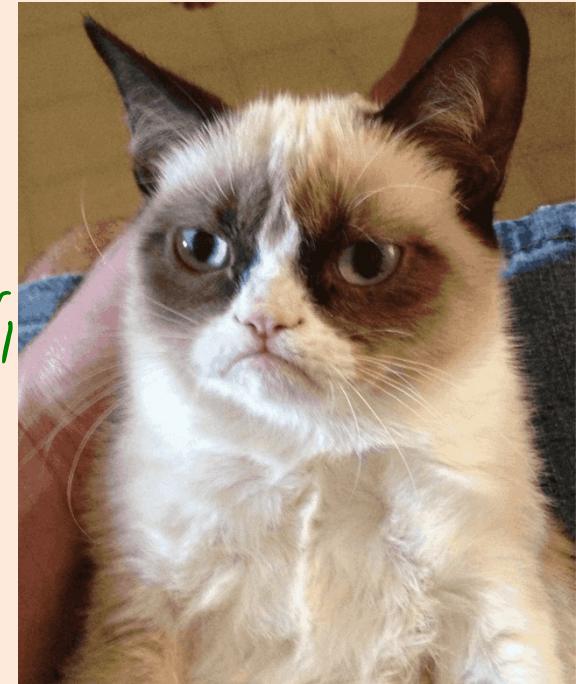
$$W = \begin{bmatrix} & \\ & \end{bmatrix} \quad 2^6 \quad \text{64 colours}$$

K-means predictions:  
(6-bits/pixel)

$$\hat{Y} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 2 \\ 2 \\ 1 \\ 1 \\ 3 \end{bmatrix} \quad \text{1 6-bit number which refers to one of } 2^6 \text{ colours.}$$

cluster for each pixel

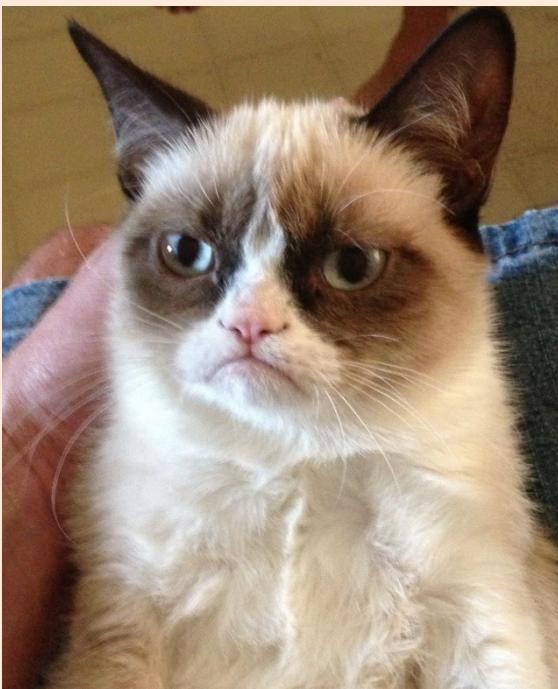
Replace cluster with mean:



bonus!

# Vector Quantization: Image Colors

- Usual RGB representation of a pixel's color: three 8-bit numbers.
  - For example, [241 13 50] = ■.
  - Can apply k-means to find set of prototype colours.



Original:  
(24-bits/pixel)

$$X = \begin{bmatrix} & \\ & \end{bmatrix} \quad \text{# pixels}$$

3  
(all 8-bit)

Run k-means with  
 $2^6$  clusters:

Average red, green,  
and blue values in  
cluster 1.

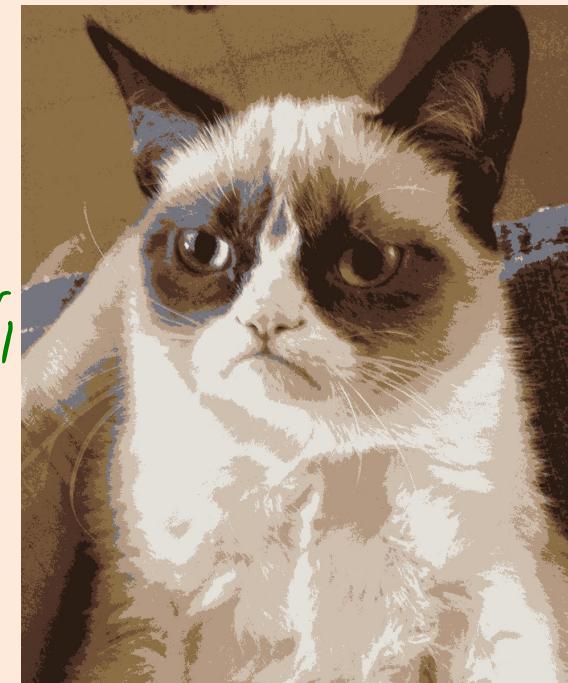
$$W = \begin{bmatrix} & \\ & \end{bmatrix} \quad \text{64 colours}$$

3

K-means predictions:  
(3-bits/pixel)

$$\hat{Y} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 2 \\ 2 \\ 1 \\ 1 \\ 3 \end{bmatrix} \quad \text{1 3-bit number  
which refers to one of } 2^3 \text{ colours.}$$

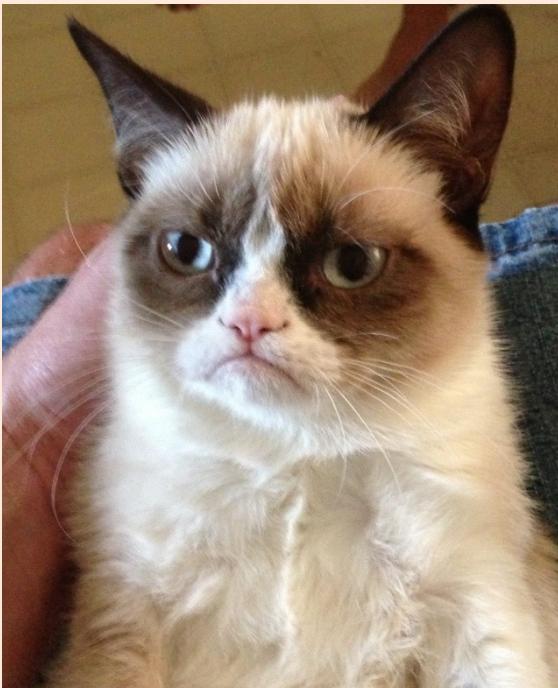
Replace cluster with mean:



bonus!

# Vector Quantization: Image Colors

- Usual RGB representation of a pixel's color: three 8-bit numbers.
  - For example, [241 13 50] = ■.
  - Can apply k-means to find set of prototype colours.



Original:  
(24-bits/pixel)

$$X = \begin{bmatrix} & \\ & \end{bmatrix} \quad \text{# pixels}$$

(all 3  
8-bit)

Run k-means with  
 $2^6$  clusters:

Average red, green,  
and blue values in  
cluster 1.

$$W = \begin{bmatrix} & \\ & \end{bmatrix}$$

$$2^2 = 4$$

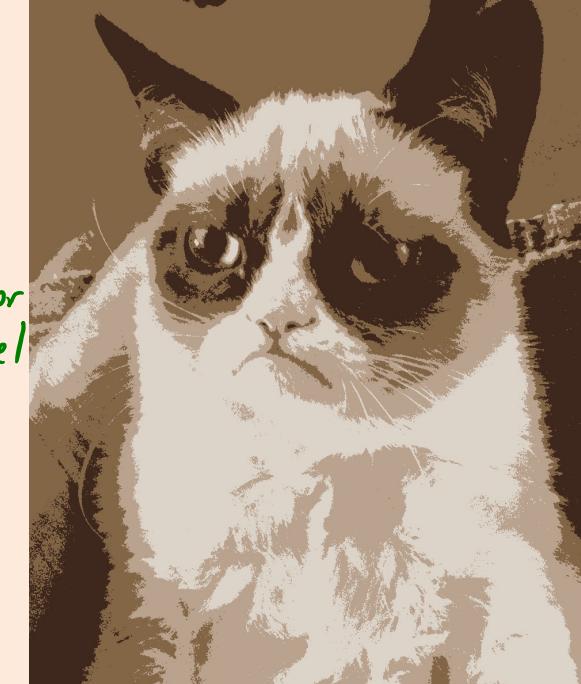
K-means predictions:  
(2-bits/pixel)

$$\hat{Y} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 2 \\ 2 \\ 1 \\ 1 \\ 3 \end{bmatrix}$$

cluster for  
each pixel

1 2-bit number  
which refers to  
one of  $2^2$  colours.

Replace cluster with mean:



bonus!

# Vector Quantization: Image Colors

- Usual RGB representation of a pixel's color: three 8-bit numbers.
  - For example, [241 13 50] = ■.
  - Can apply k-means to find set of prototype colours.



Original:  
(24-bits/pixel)

$$X = \begin{bmatrix} & \\ & \end{bmatrix} \quad \text{# pixels} \quad \text{(all } 3 \text{ 8-bit)}$$

Run k-means with  
 $2^6$  clusters:

Average red, green,  
and blue values in  
cluster 1.

$$W = \begin{bmatrix} & \\ & \end{bmatrix} \quad \text{2}^1 = 2$$

K-means predictions:  
(1-bit/pixel)

$$\hat{y} = \begin{bmatrix} 1 \\ 2 \\ 2 \\ 2 \\ 1 \\ 1 \\ 1 \end{bmatrix} \quad \text{1 1-bit number  
which refers to  
one of 2 colours.}$$

cluster for each pixel

Replace cluster with mean:

