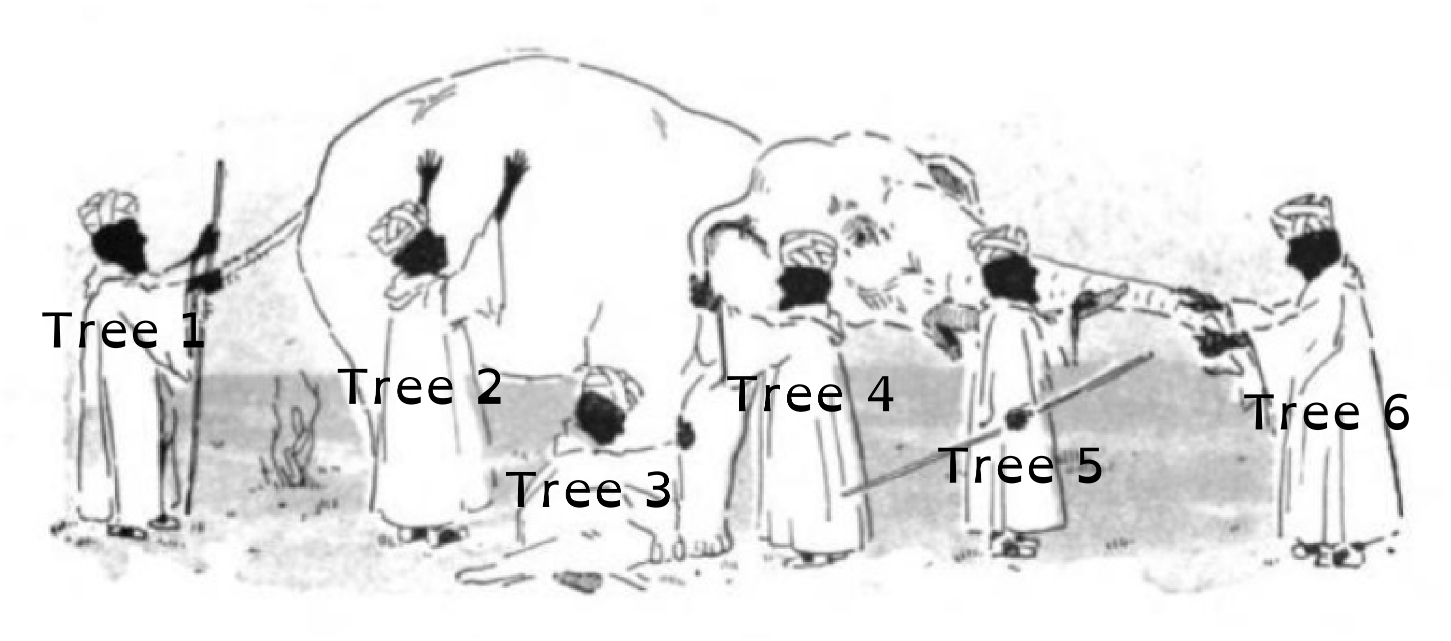
CPSC 340: Machine Learning and Data Mining

K-Means Clustering

Summer 2021

In This Lecture

- Random Forest (15 minutes)
- Unsupervised Learning Intro (15 minutes)
- K-Means Clustering (15 minutes)



Coming Up Next

RANDOM FORESTS

Random Forests

unbounded depth

- Random forests take vote from a set of deep decision trees.
  - Tend to be one of the best "out of the box" classifiers.
    - Often close to the best performance of any method on the first run.
  - And predictions are very fast.

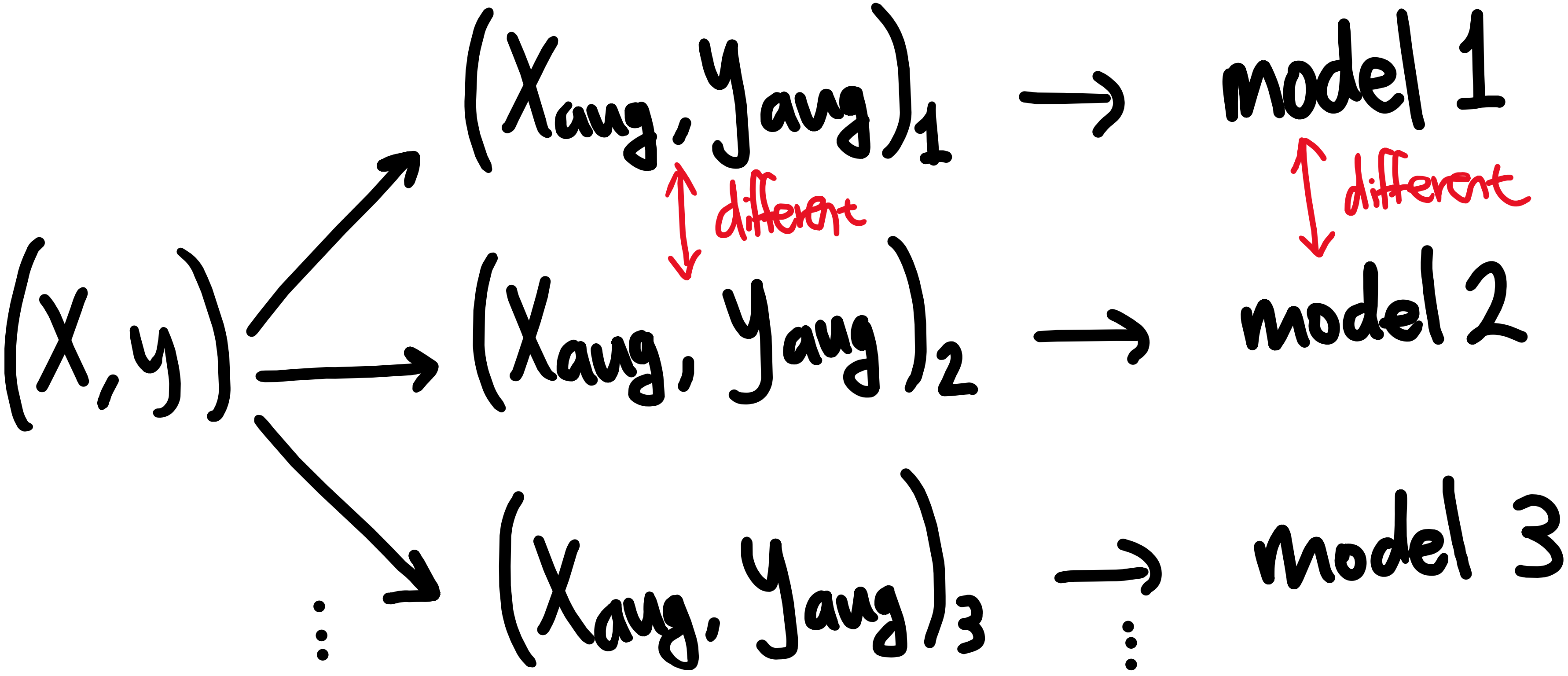
- Do deep decision trees make independent errors?
  - No: with the same training data you'll get the same decision tree.

- Two key ingredients in random forests:
  - Bootstrap sampling: online data augmentation
  - Random trees: decision trees but with randomness

- Special case of "bagging":
  - Bootstrapped aggregation

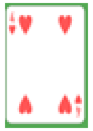
Bootstrap Sampling

- Idea: give each model a unique training data
  - i.e. make each model behave uniquely
  - i.e. help them make independent errors

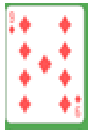


Bootstrap Sampling

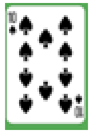
- Start with a standard deck of 52 cards:
  1. Sample a random card:
     (put it back and re-shuffle)



  2. Sample a random card:
     (put it back and re-shuffle)

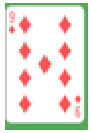


  3. Sample a random card:
     (put it back and re-shuffle)

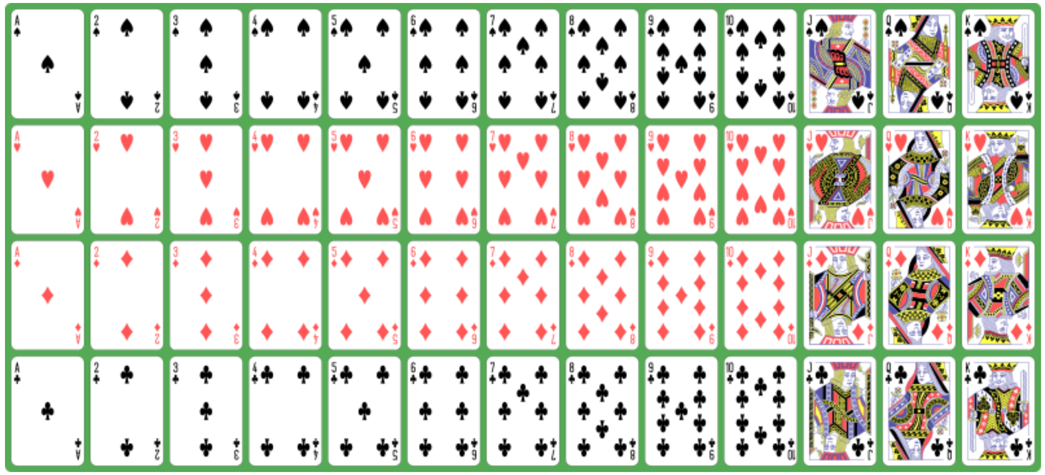


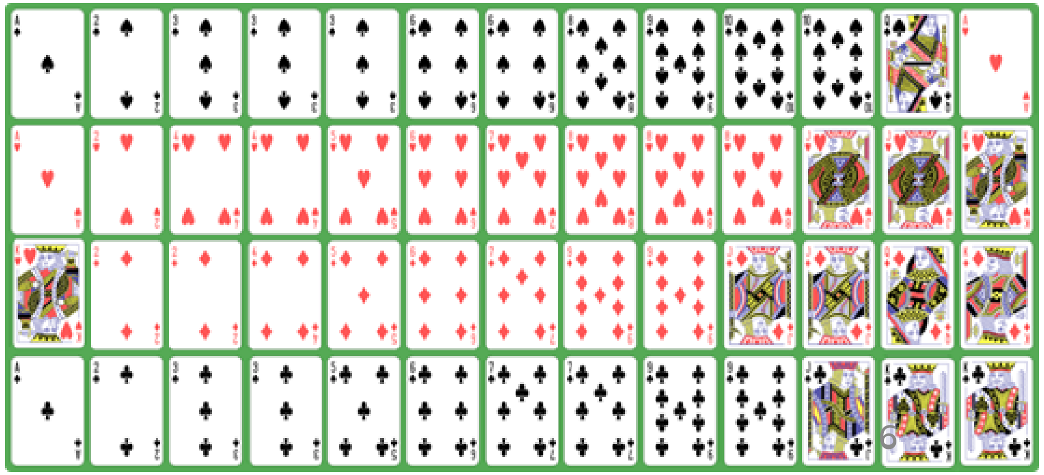
  – …

  52. Sample a random card:
      (which may be a repeat)



- Makes a new deck of the 52 samples:
- aka “bootstrap sample”

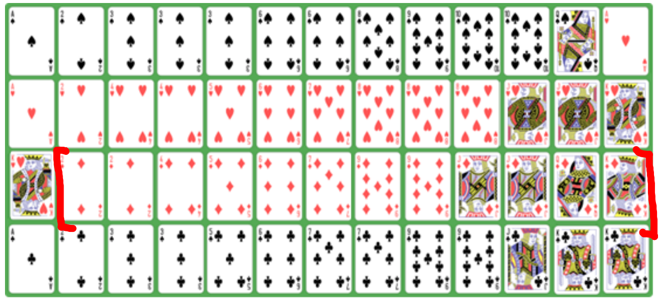




Q: Does this deck look like my original deck?

https://commons.wikimedia.org/wiki/File:English_pattern_playing_cards_deck.svg

Bootstrap Sampling



- No, it doesn't look like the original deck
- Some cards will be duplicated____________, and some cards will be missing__________.
  - Calculations on the bootstrap sample will give different results than original data.
- Yes, it looks like the original deck
- The bootstrap sample roughly maintains trends:
  - Roughly 25% of the cards will be diamonds.
  - Roughly 3/13 of the cards will be "face" cards.
  - There will be roughly four "10" cards.
- Common use: compute a statistic based on several bootstrap samples.
  - Gives you an idea of how the statistic varies as you vary the data.

Random Forest Ingredient 2: Random Trees

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

Random tree 1:

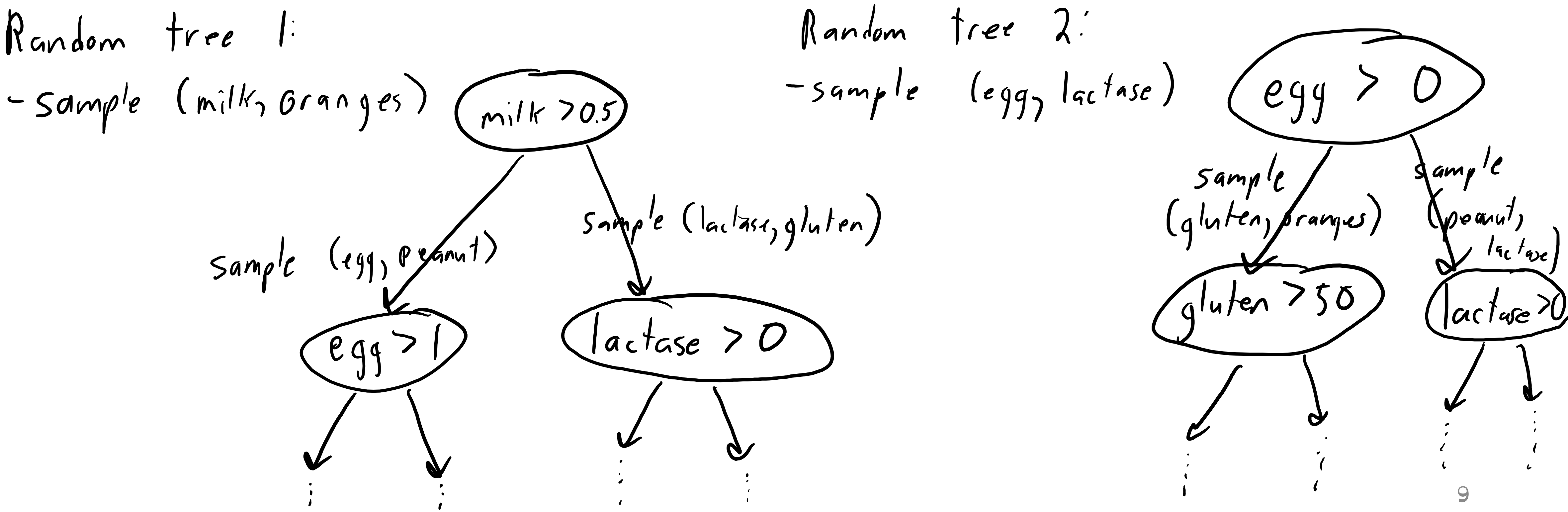
- sample (milk, oranges) milk > 0.5

Random tree 2:

- sample (egg, lactase) egg > 0

Random Forest Ingredient 2: Random Trees

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



Random Forest Ingredient 2: Random Trees

infinite depth

- Choosing a small number of random features will make the search space ____________ unique for each tree
- They will still overfit to the bootstrapped sample (WHY?)
  - but hopefully errors will be more independent.
    - We made both training data AND search space unique to each tree!

- Recall: independent errors => improved ensemble performance
  - 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:

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

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

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

Types and Goals of Ensemble Methods

- Remember the fundamental trade-off:
  1. E_{train}: How small you can make the training error.

     vs.

  2. E_{approx}: How well training error approximates the test error.

Q: How are $\underbrace{E_{train}}_{\text{high}}$ and $\underbrace{E_{approx}}_{\text{low}}$ for a single random tree?

Q: How are $\underbrace{E_{train}}_{\text{low}}$ and $\underbrace{E_{approx}}_{\sim\text{low}}$ for a random forest?

Types and Goals of Ensemble Methods

- Goal of ensemble methods is that our meta-classifier:
  - Does much better on E_{train} or E_{approx} than individual classifiers.
    - Random forest has much better Etrain (A2)
  - Doesn't do too much worse on the other.
    - How can random forest have better E_{approx} with super low E_{train}?

- Two types of ensemble methods:
  1. Averaging: improves approximation error of classifiers with high E_{approx}.
     - This is the point of "voting".
  2. Boosting: improves training error of classifiers with high E_{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).

Part 2: Unsupervised Learning

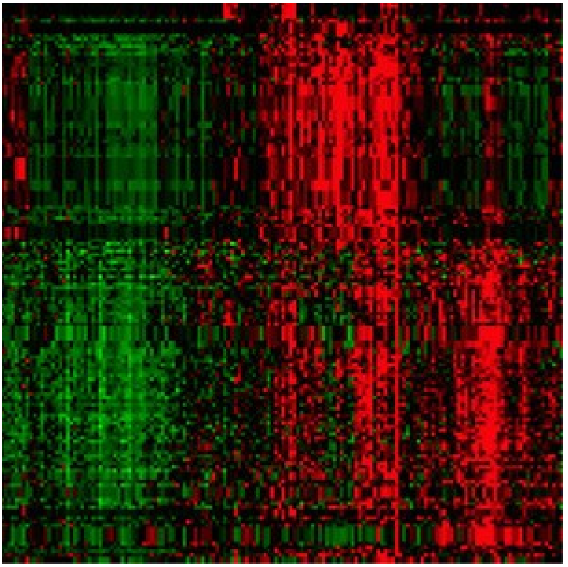
Coming Up Next

UNSUPERVISED LEARNING INTRO

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.

https://corelifesciences.com/human-long-non-coding-rna-expression-microarray-service.html

Unsupervised Learning

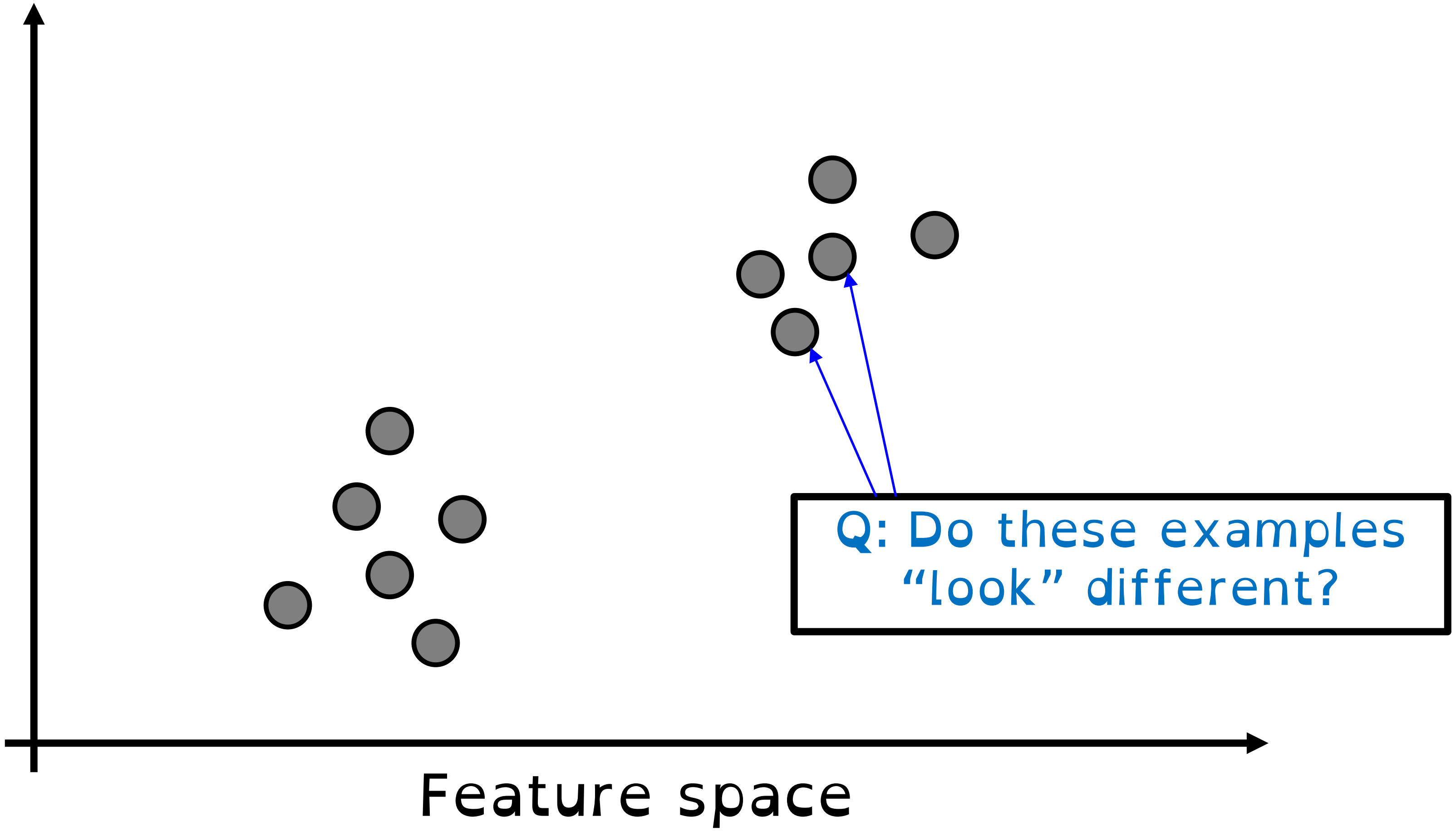
- Supervised learning:
  - We have features x_i and class labels y_i.
  - Write a program that produces y_i from x_i.
- Unsupervised learning:
  - We only have x_i values, but no explicit target labels.
  - You want to do “something” with them.

Unsupervised Learning

- 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

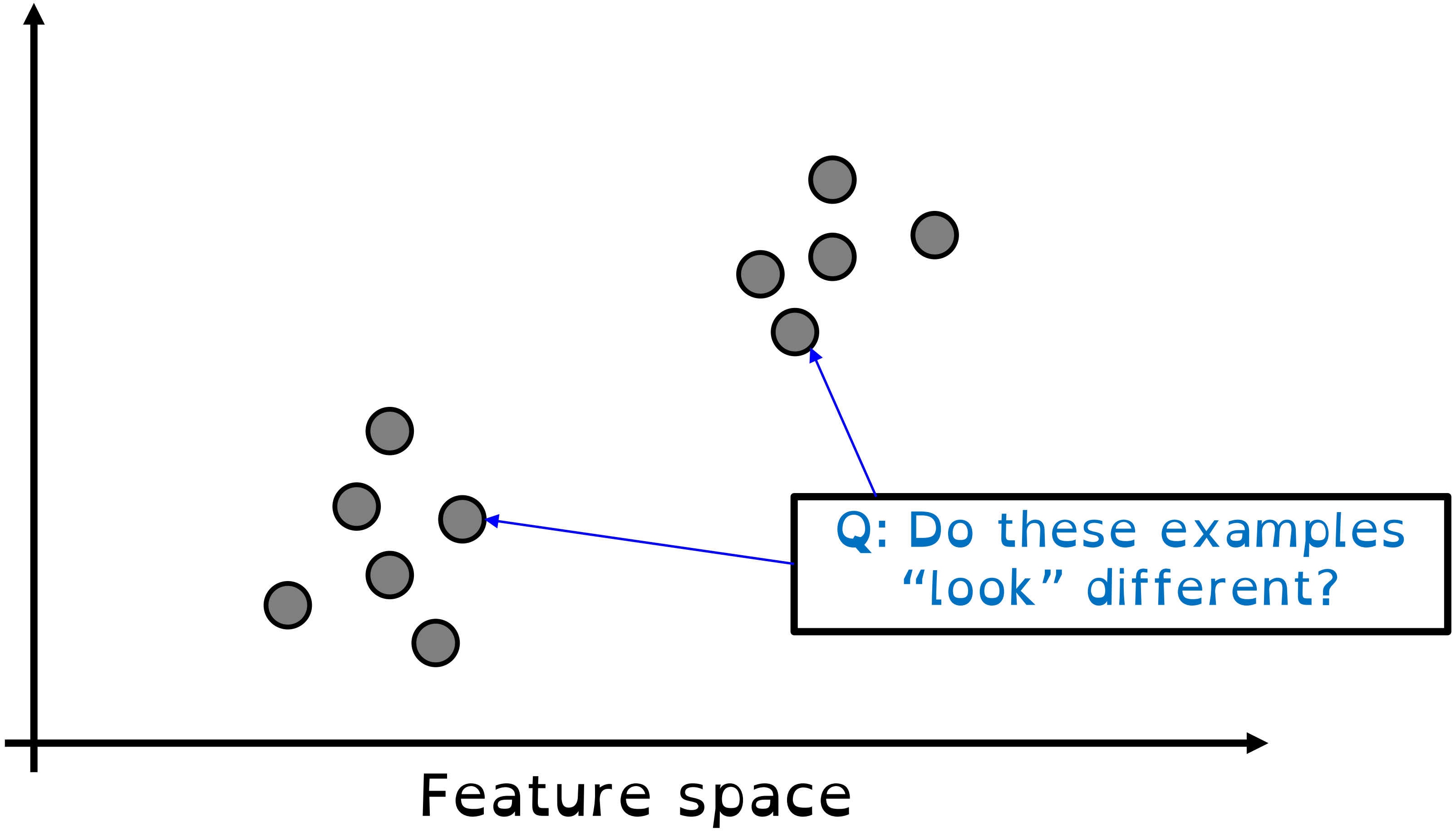
- Assumption: different-looking examples belong to different groups



Q: Do these examples “look” different?

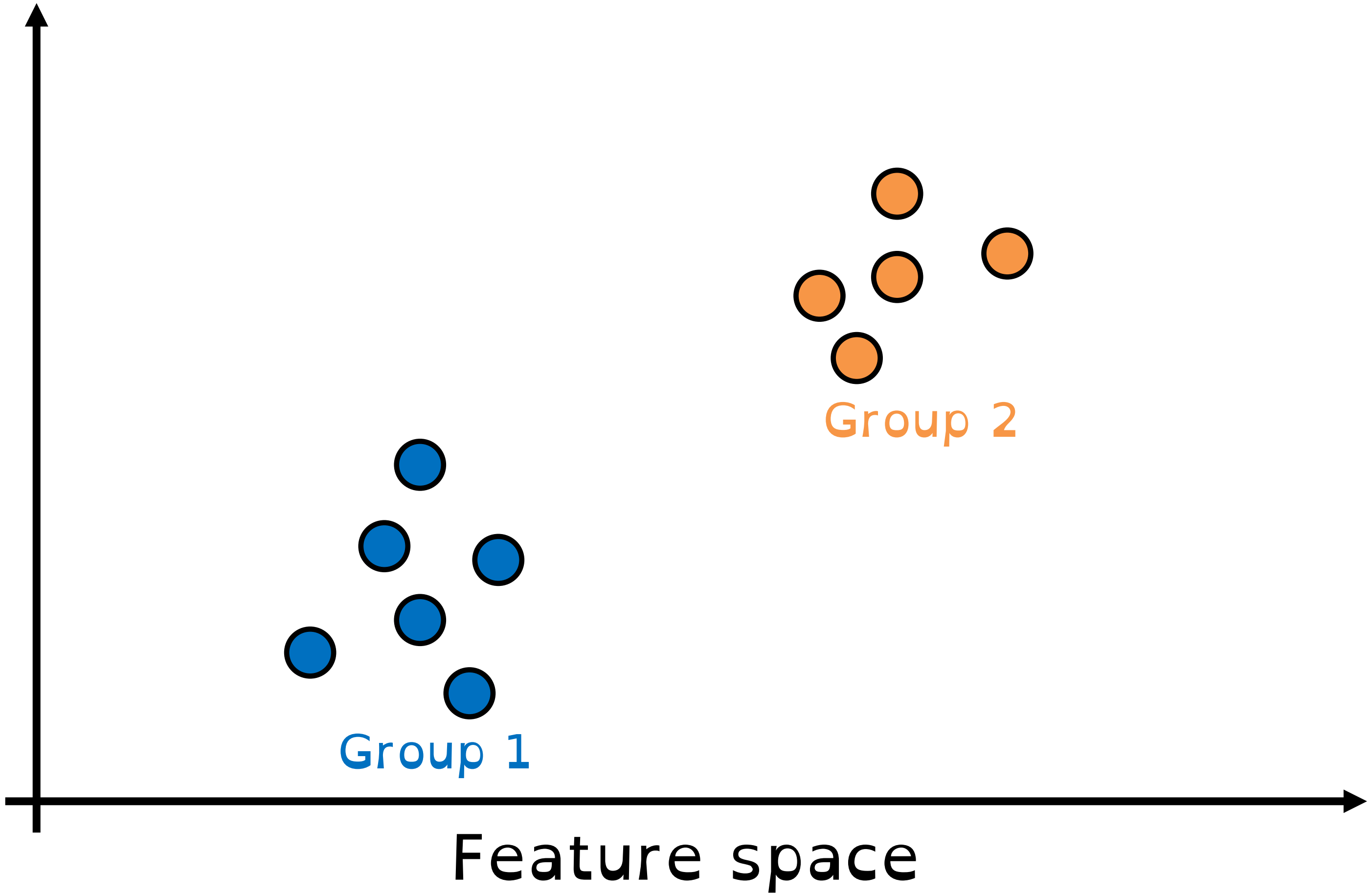
Clustering

- Assumption: different-looking examples belong to different groups



Clustering

- Assumption: different-looking examples belong to different groups



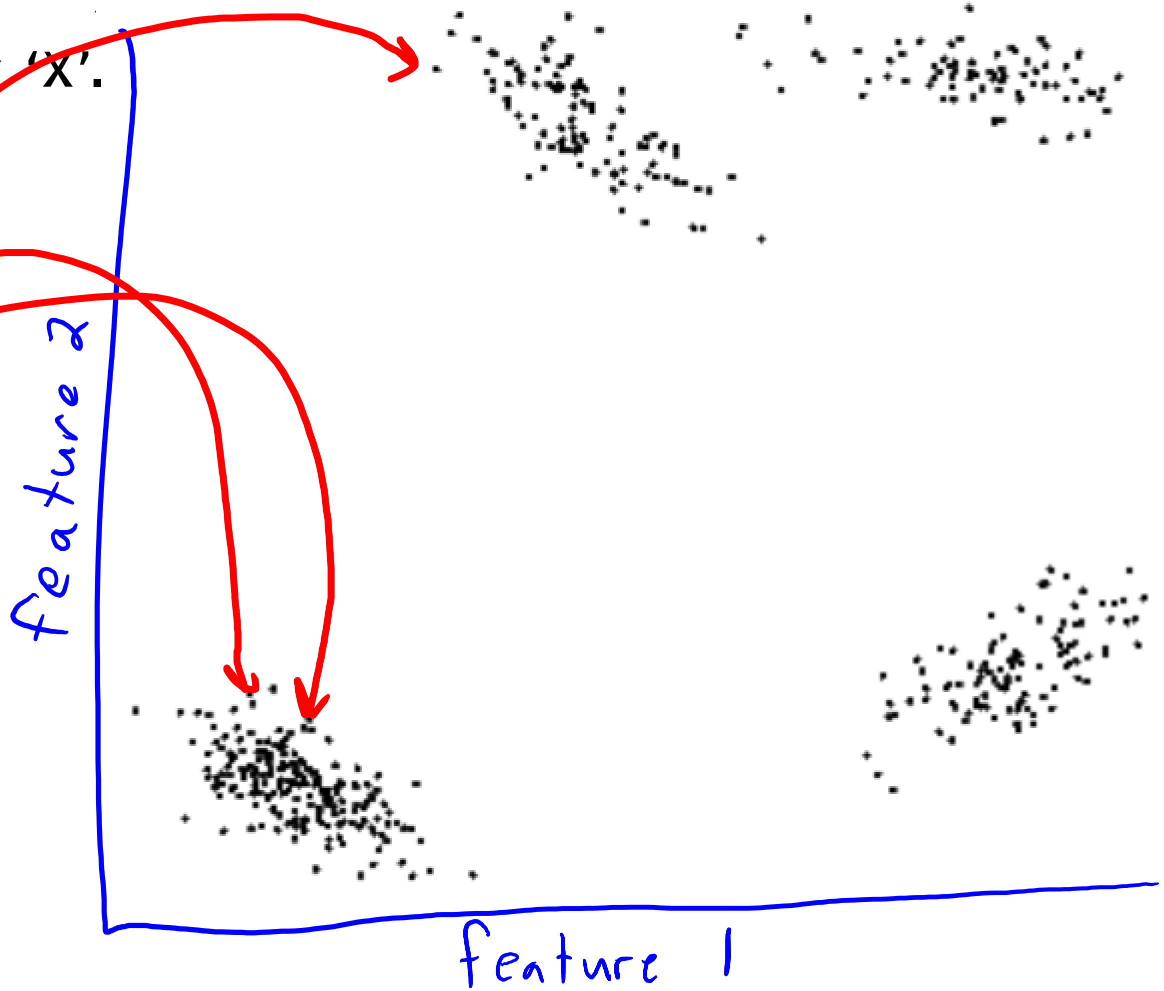
Clustering

- 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 email:
  - 'Spam' group.
  - 'Not spam' group.
- With in 'Spam' group:
  - 'Weight loss' group
  - 'Fake special reward' group
  - 'Free money' group

Clustering Example

Input: feature matrix 'X'.

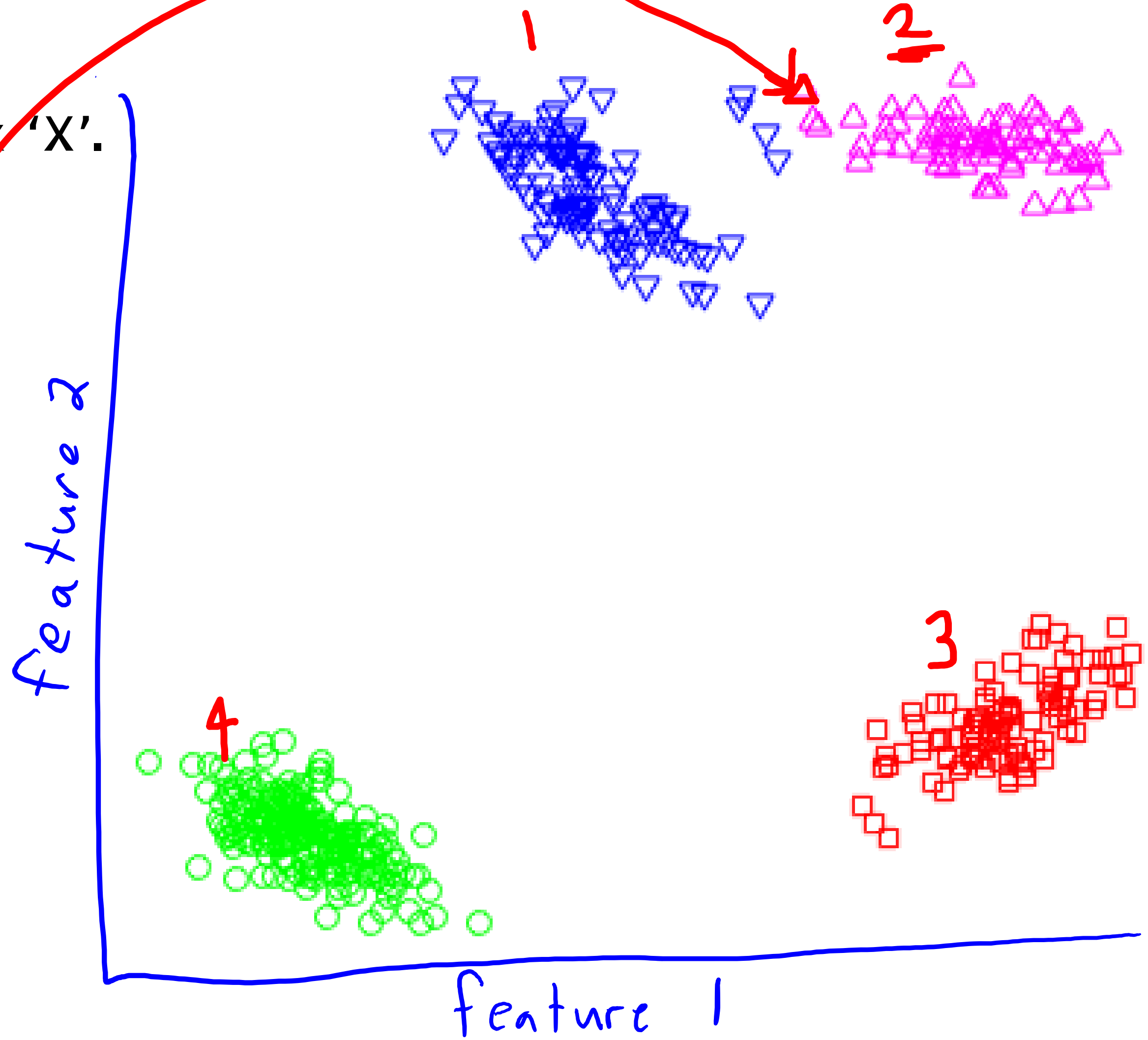
$$X = \underset{2}{\begin{bmatrix} -9.0 & -7.3 \\ -10.9 & -9.0 \\ 13.7 & 19.3 \\ 13.8 & 20.4 \\ 12.8 & 20.6 \\ \vdots & \vdots \end{bmatrix}}$$



Clustering Example

Input: feature 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 $\hat{y}$.

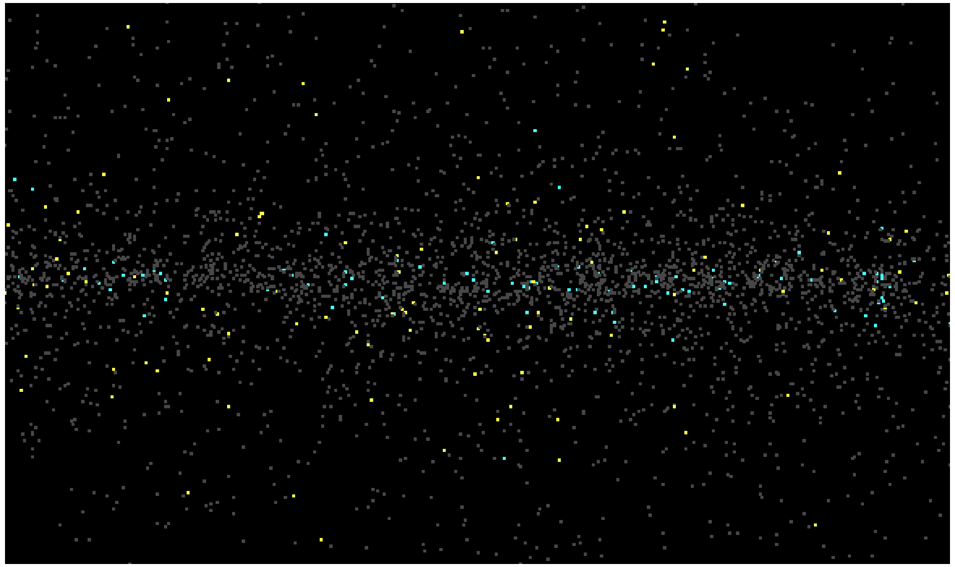
$$\hat{y} = \begin{bmatrix} 2 \\ 2 \\ 3 \\ 3 \\ 1 \\ \vdots \end{bmatrix}$$

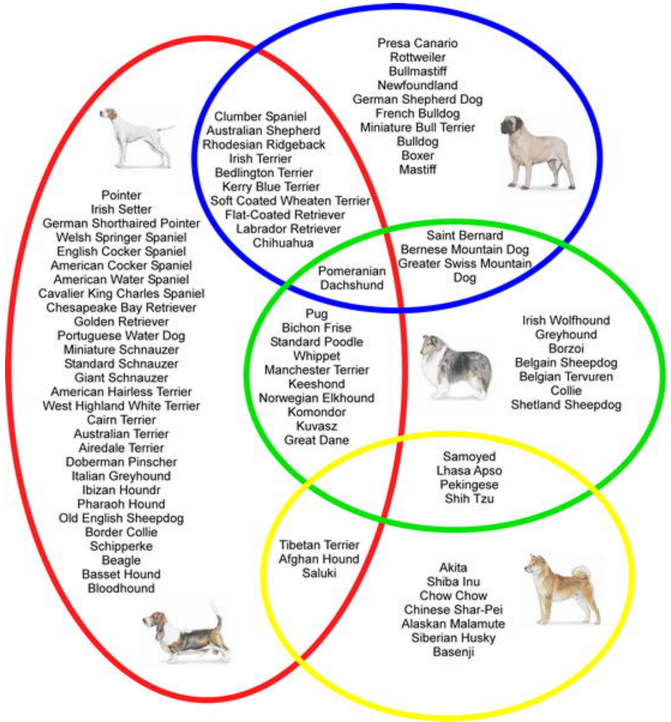
Data Clustering

- 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 ‘canonical’ example for each group.
    - E.g. what does the most typical breakfast look like?

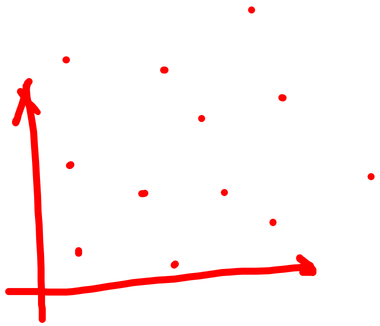
Other Clustering Applications

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





http://www.eecs.wsu.edu/~cook/dm/lectures/l9/index.html
http://www.biology-online.org/articles/canine_genomics_genetics_running/figures.html



Coming Up Next

K-MEANS CLUSTERING

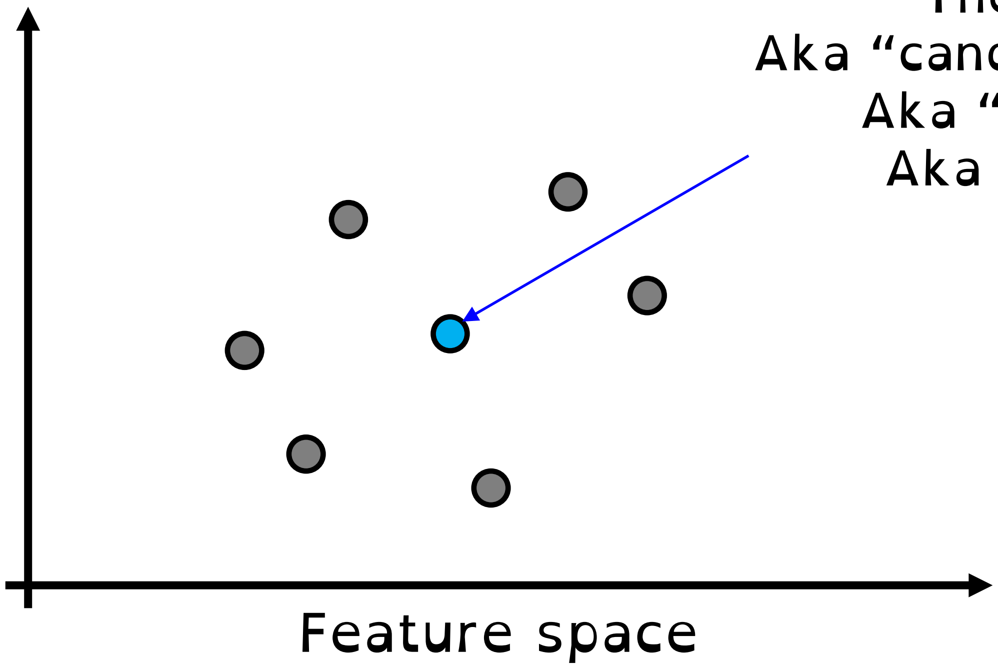
K-Means

- 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.
- A 2-step iterative algorithm:
  1. Assign each x_i to label of closest mean. $\hat{y}_i$
  2. Update the means based on the assignment.

  Repeat until convergence.

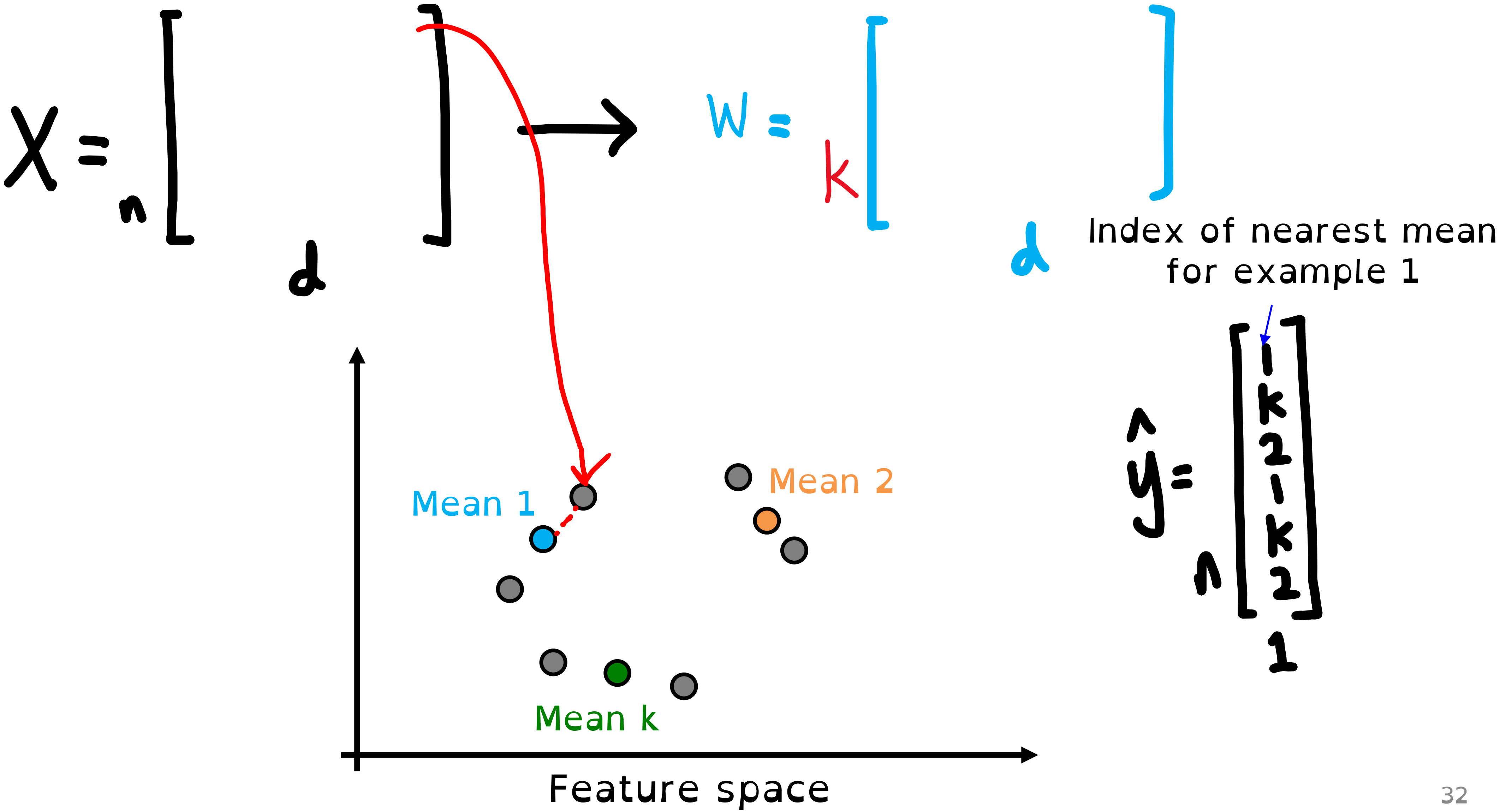
“Mean”

$$X = {}_{n}\underset{d}{\Big[\qquad\qquad\Big]} \longrightarrow W = {}_{1}\underset{d}{\Big[\qquad\qquad\Big]}$$



The “mean”
Aka “canonical example”
Aka “prototype”
Aka “centroid”

K “Means”

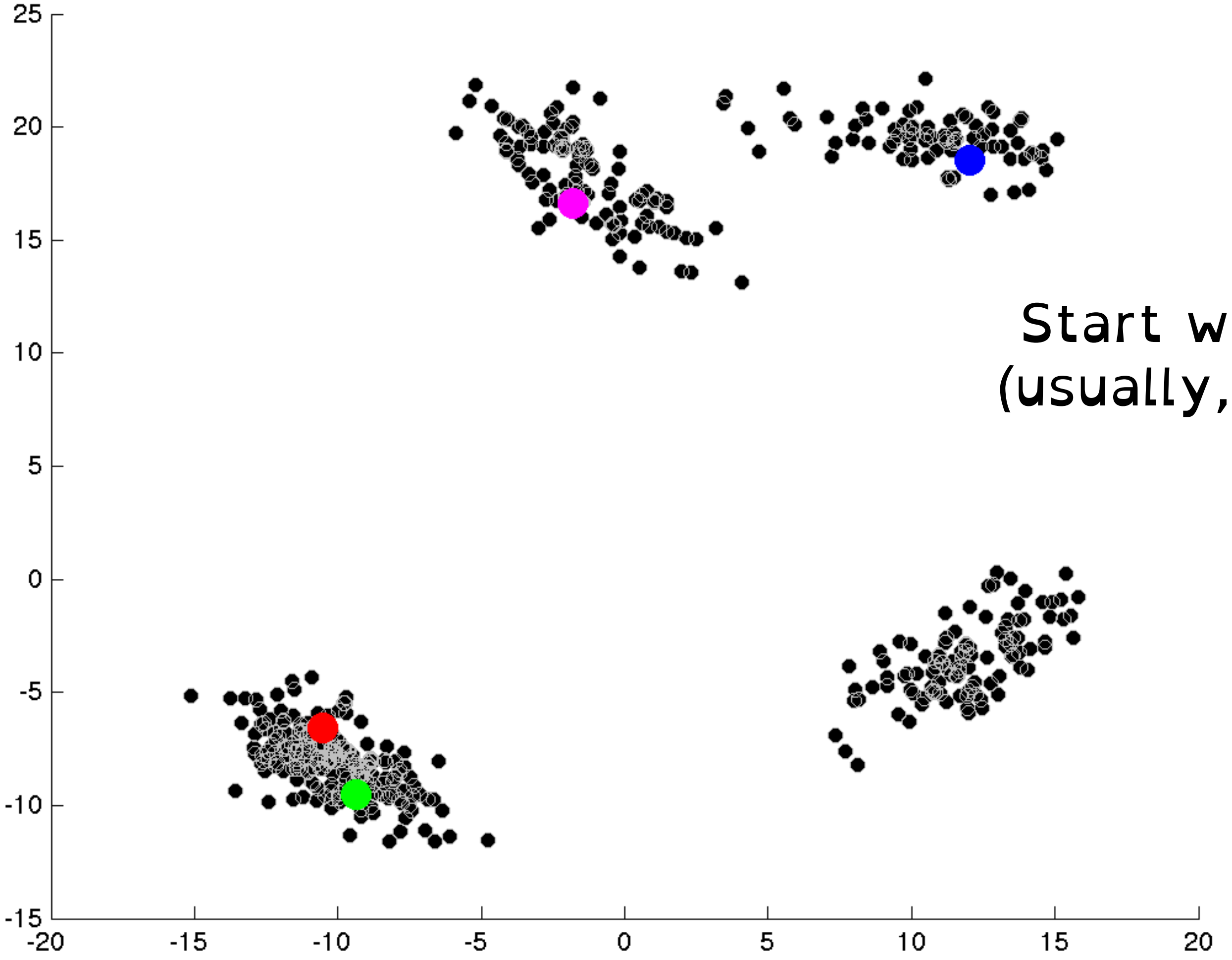


Index of nearest mean for example 1

K-Means Example

Input: feature matrix 'X'.

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

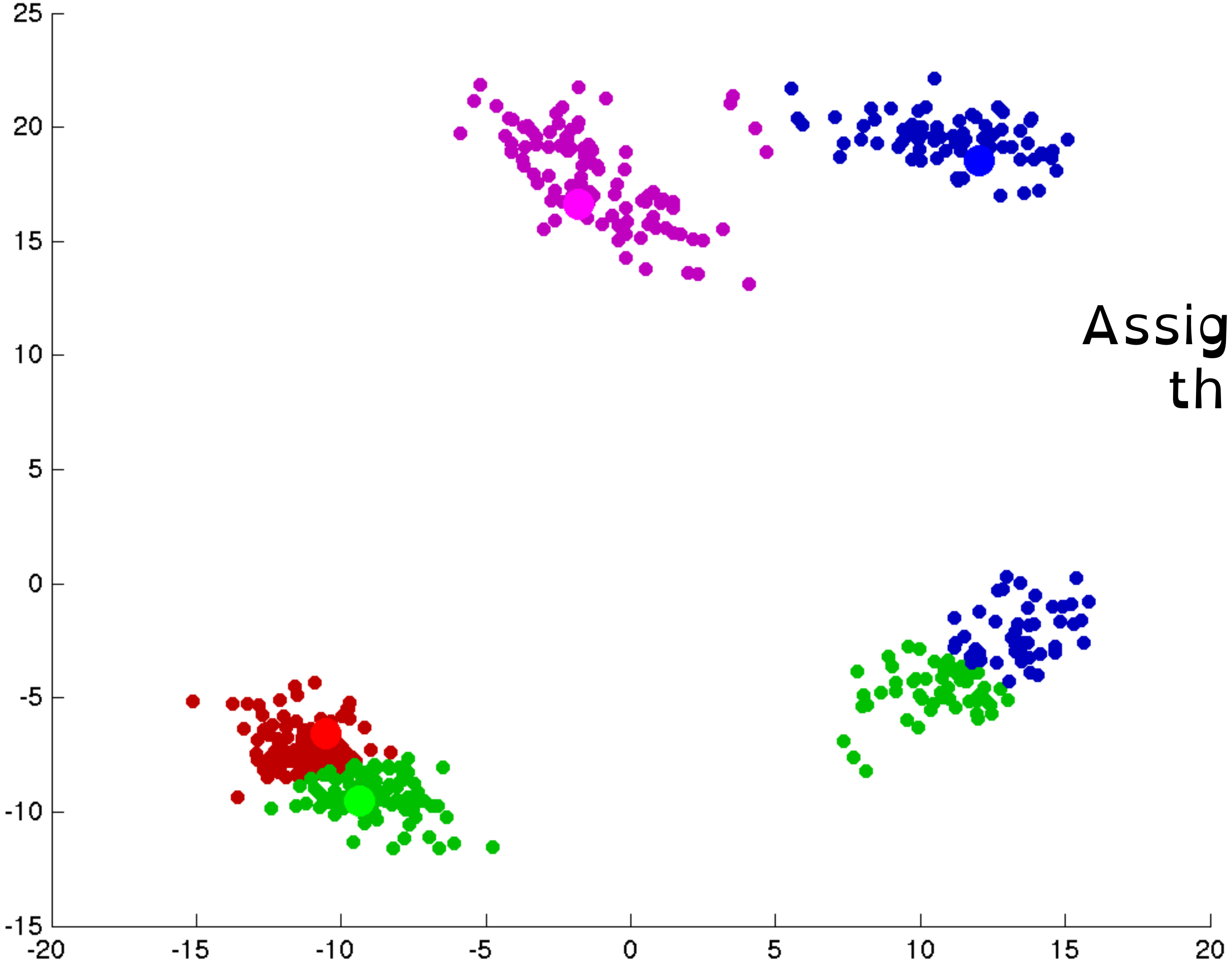


Start with 'k' initial 'means'
(usually, random data points)

K-Means Example

Input: feature matrix ‘X’.

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

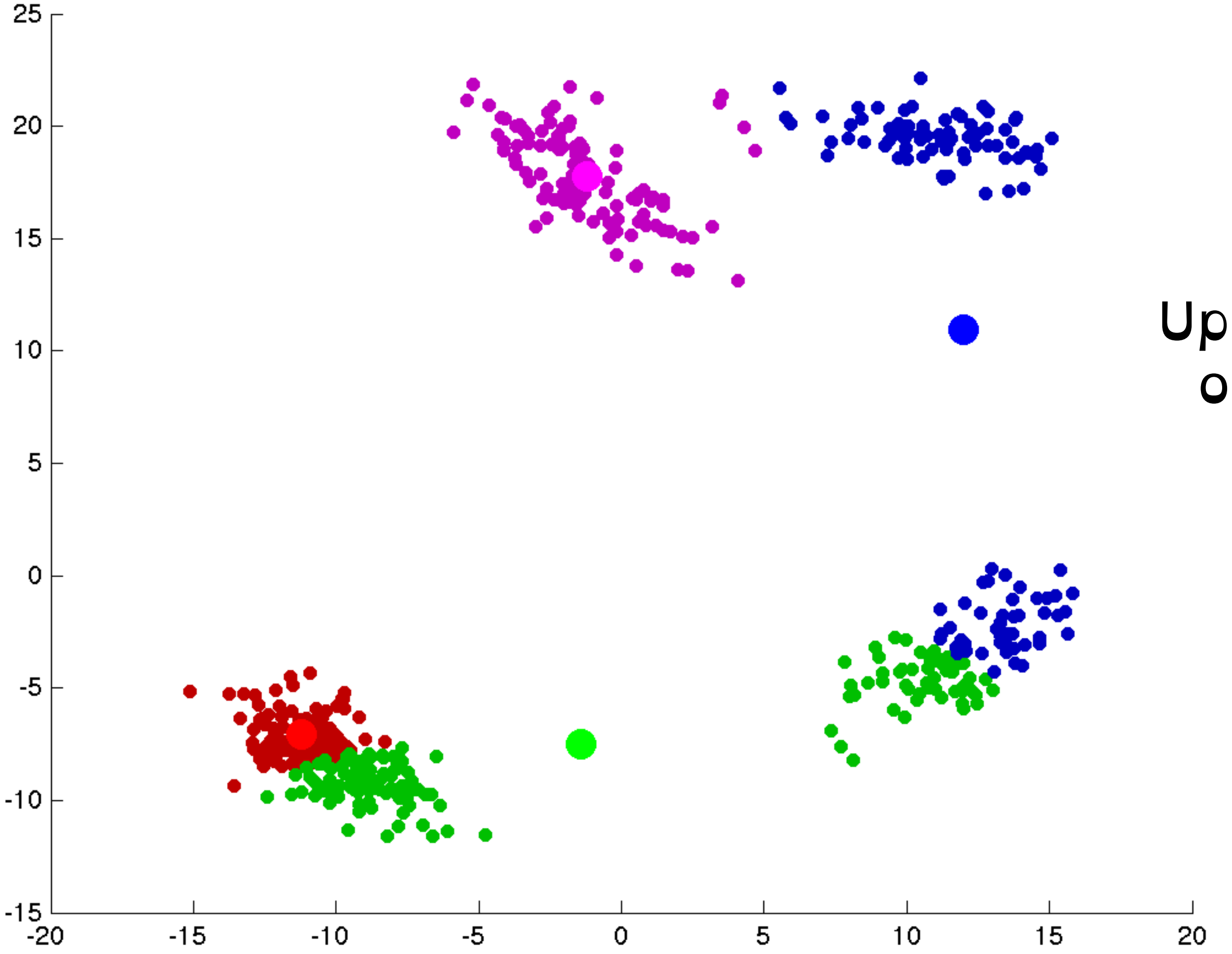


Assign each example to the closest mean.

K-Means Example

Input: feature matrix ‘X’.

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

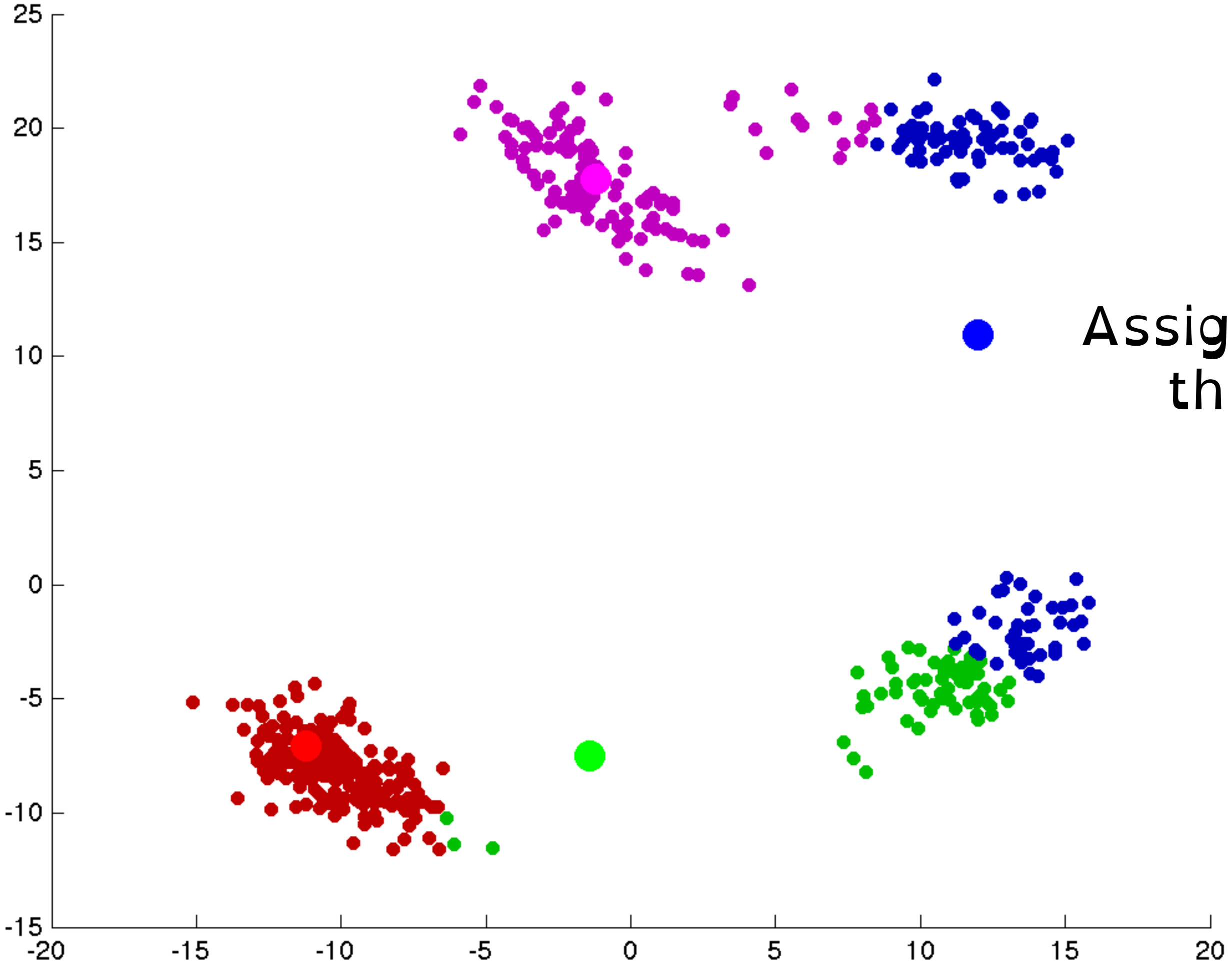


Update the mean of each group.

K-Means Example

Input: feature matrix ‘X’.

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

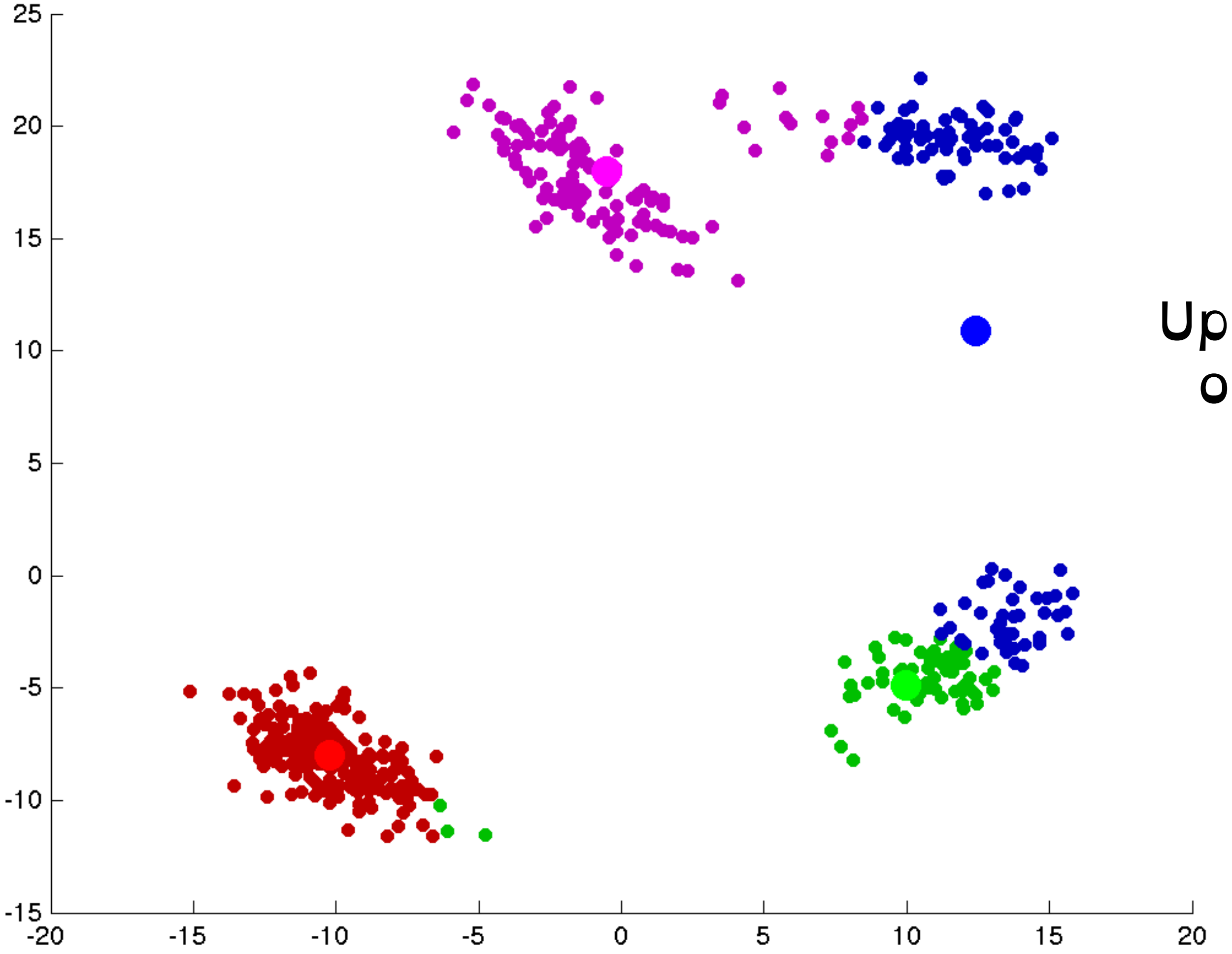


Assign each example to the closest mean.

K-Means Example

Input: feature matrix ‘X’.

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

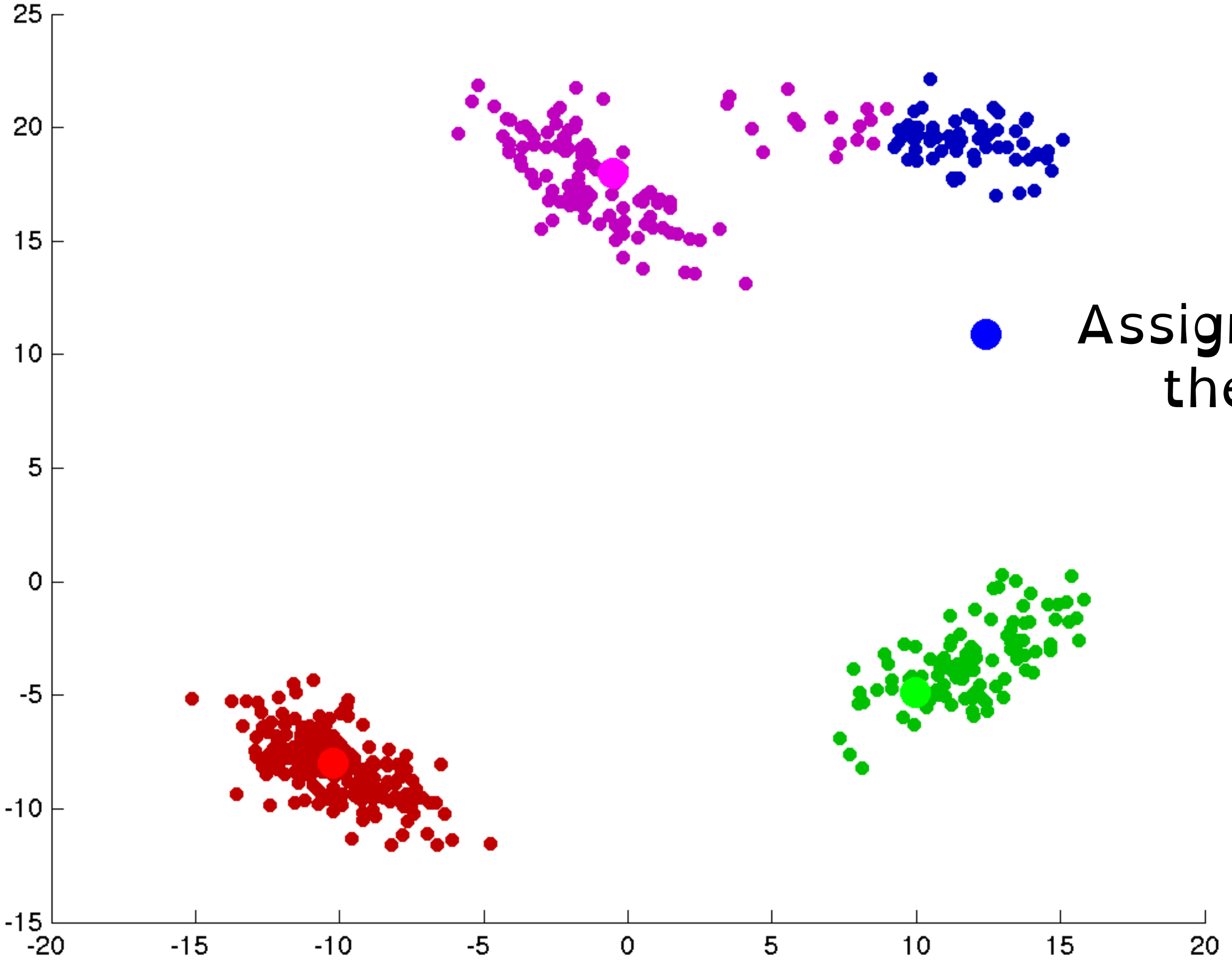


Update the mean of each group.

K-Means Example

Input: feature matrix ‘X’.

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

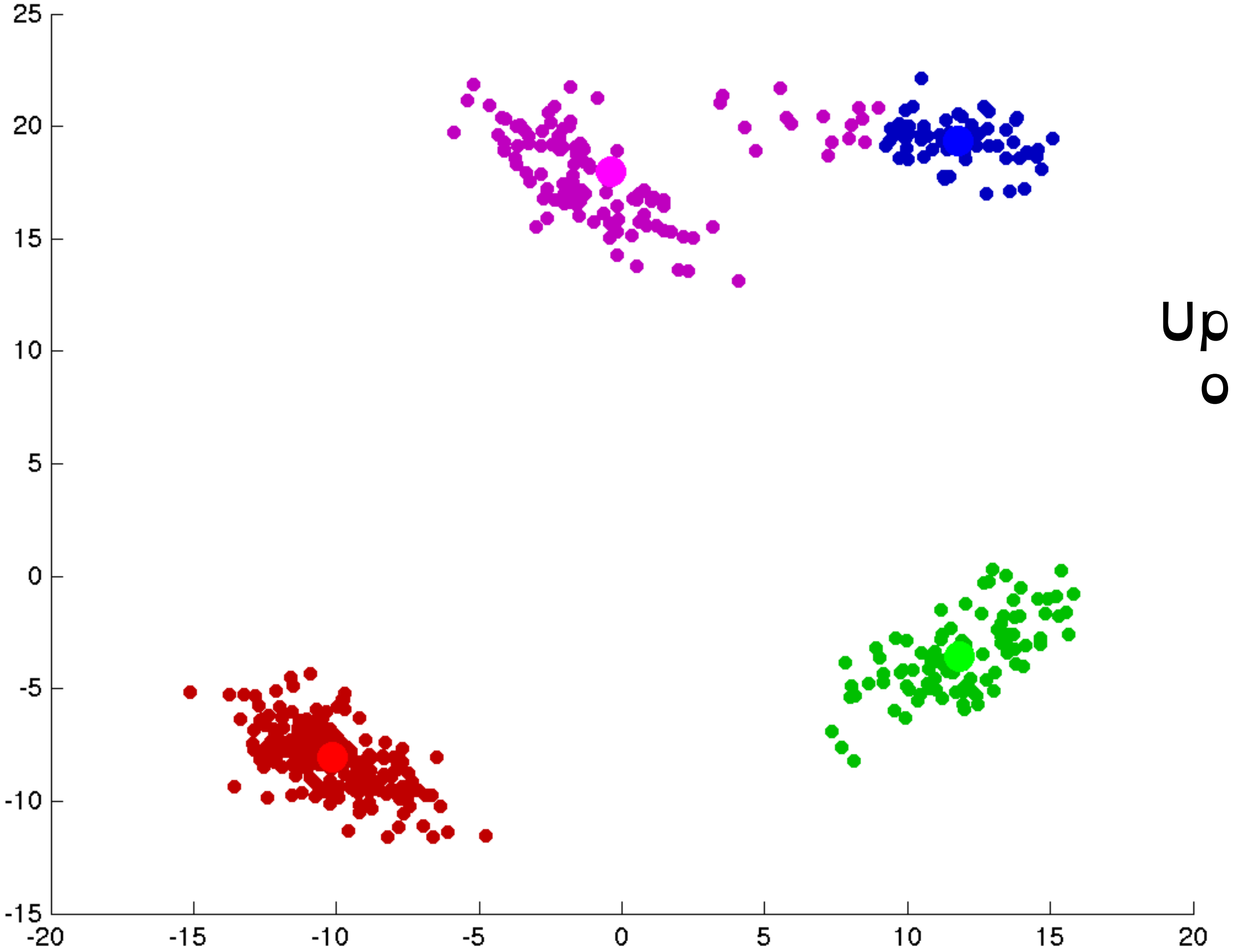


Assign each example to the closest mean.

K-Means Example

Input: feature matrix ‘X’.

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

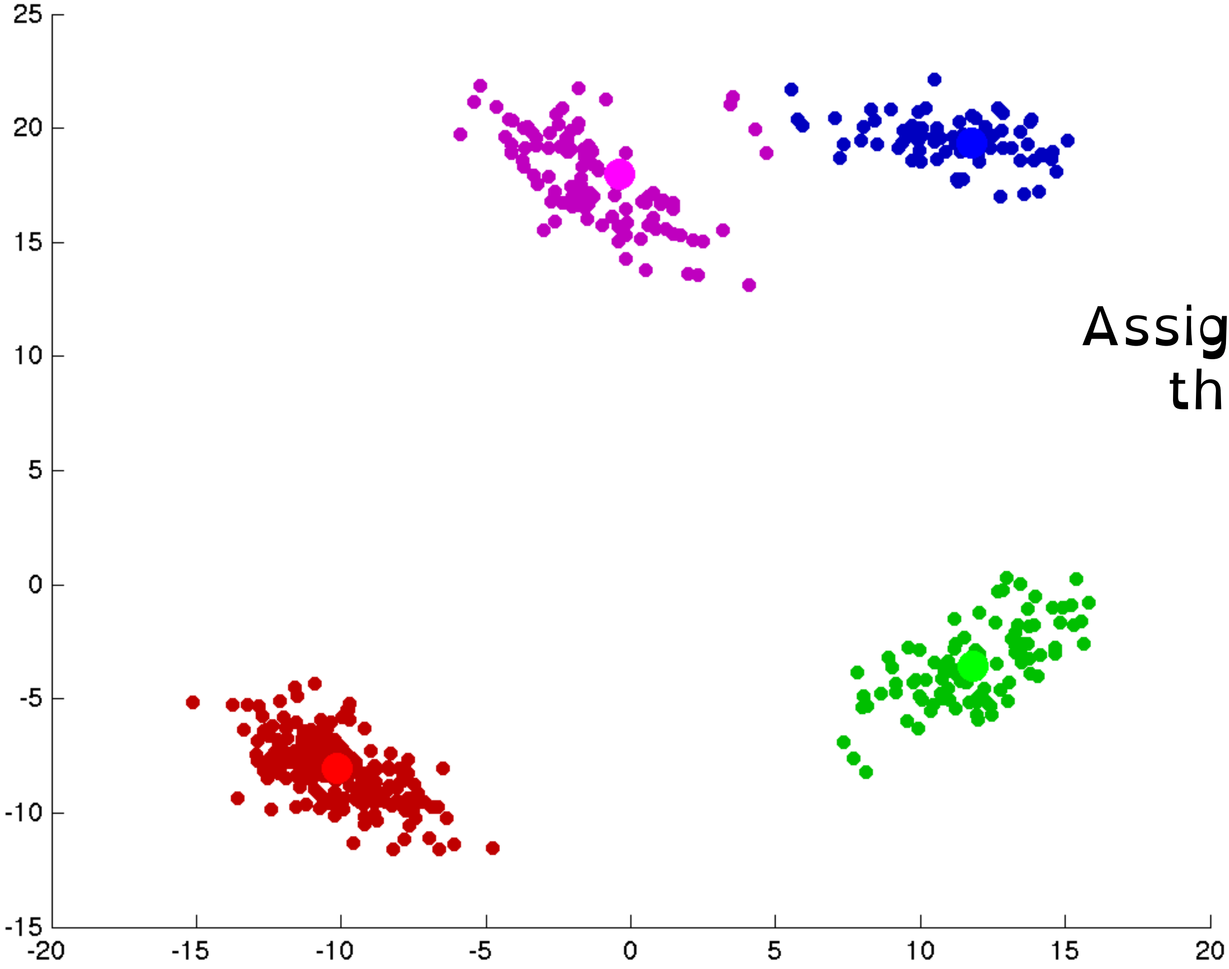


Update the mean of each group.

K-Means Example

Input: feature matrix 'X'.

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

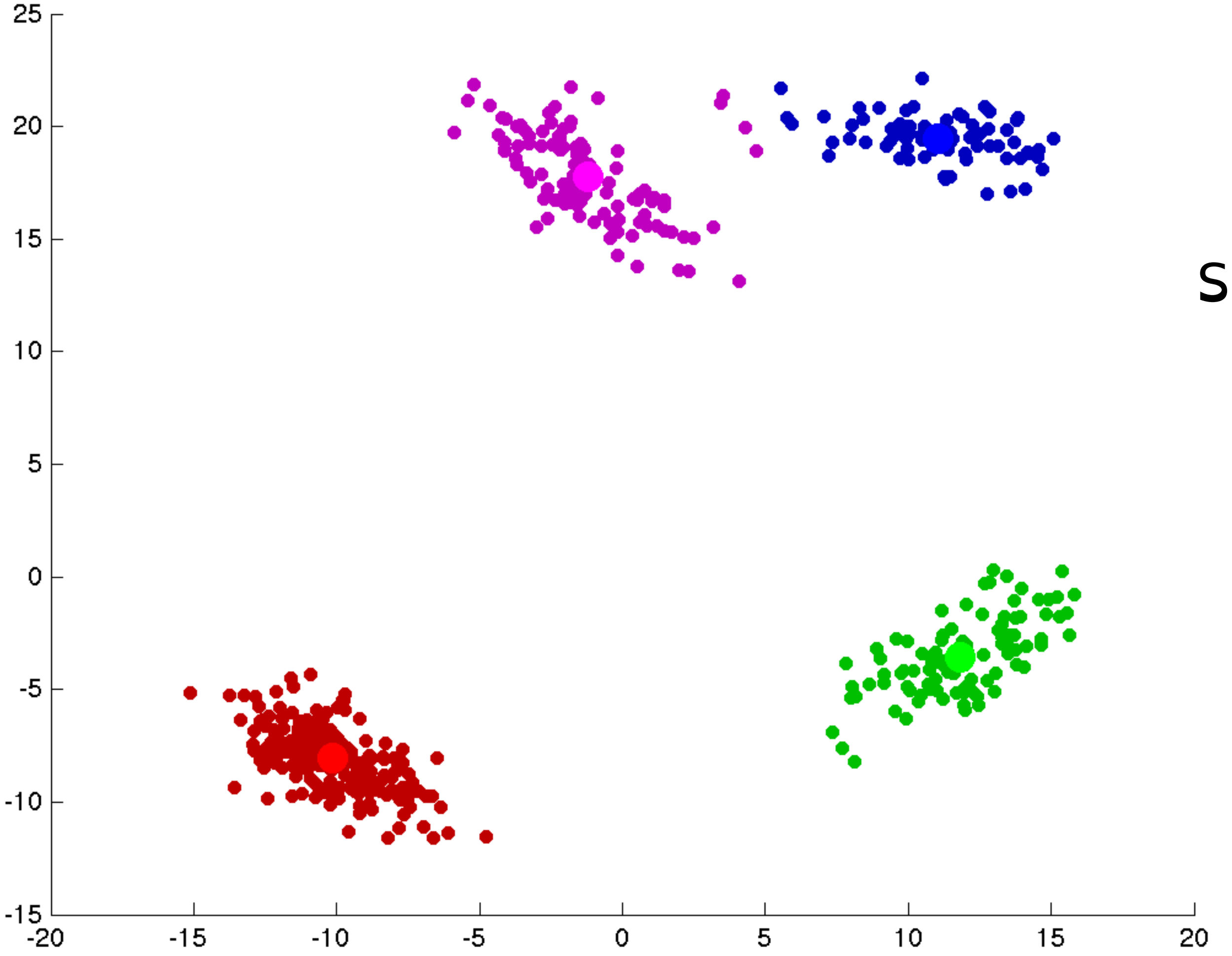


Assign each example to the closest mean.

K-Means Example

Input: feature matrix ‘X’.

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

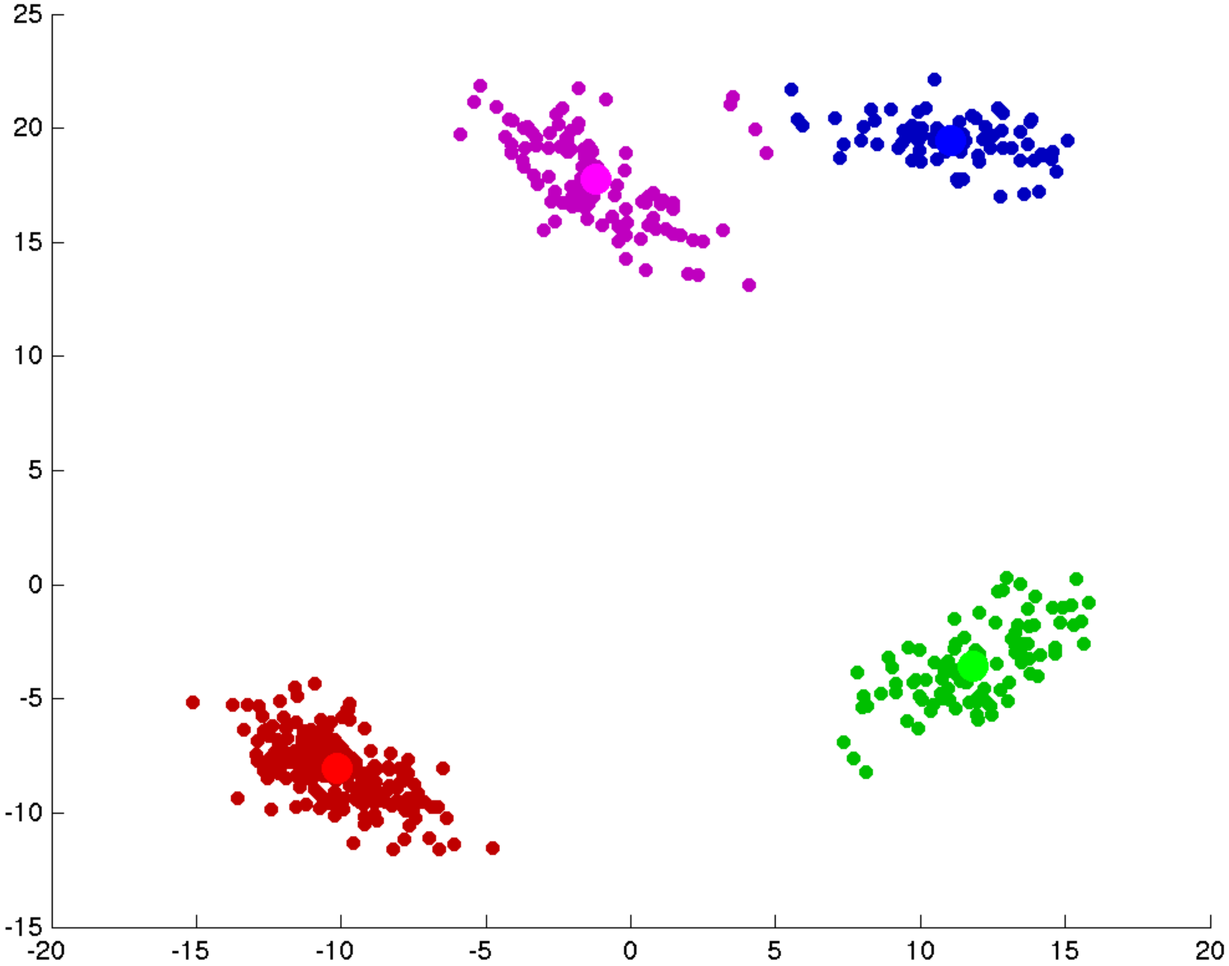


Stop if no examples change groups. (Convergence!)

K-Means Example

Input: feature 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 '$\hat{y}$'.
- Means 'W'.

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

$$W = \underbrace{\left.\begin{bmatrix} -1.2 & 17.8 \\ -10.2 & -8.0 \\ 11.0 & 19.5 \\ 11.8 & -3.6 \end{bmatrix}\right\} k}_{d}$$

Interactive demo:
https://www.naftaliharris.com/blog/visualizing-k-means-clustering

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

- Each example is assigned to one (and only one) cluster: (hard assignment)
  - No possibility for overlapping clusters or leaving examples unassigned.

Q: Does K-means always converge to the "optimal" clustering?

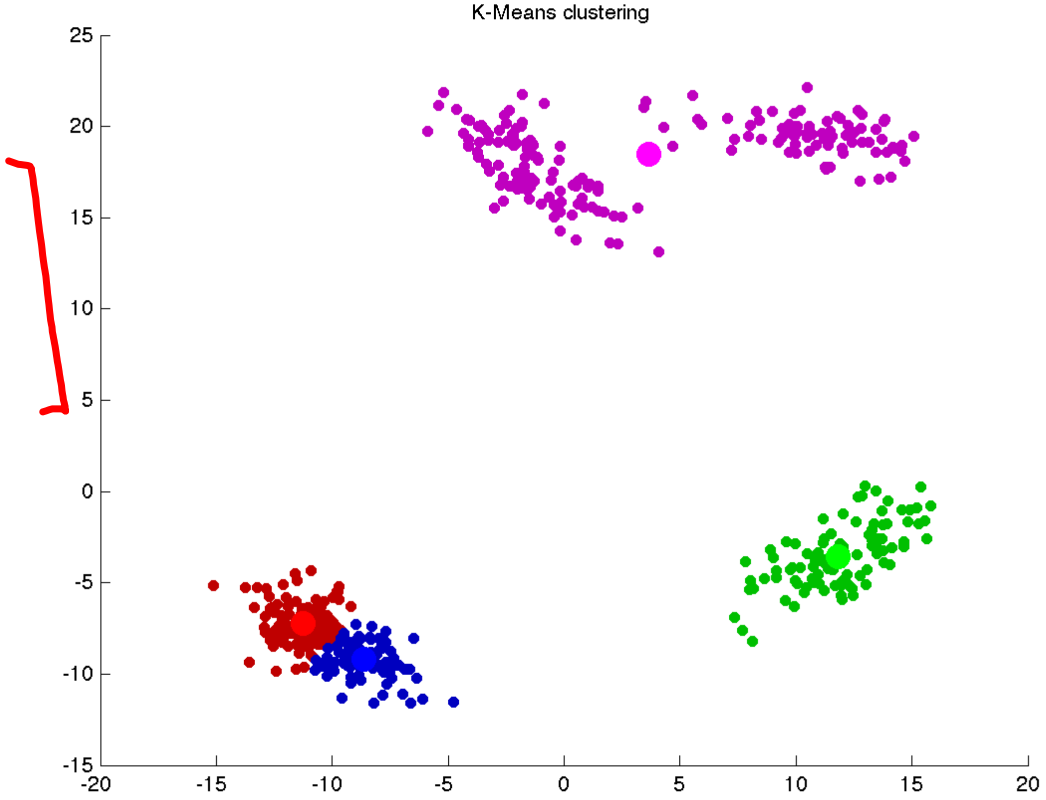
intial means

NO, Sensitive to initialization

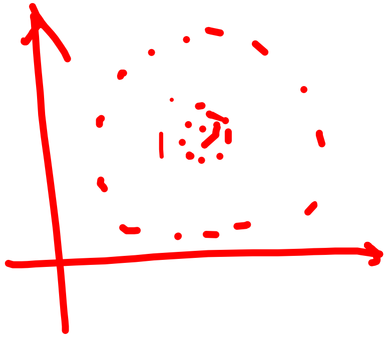
K-Means Clustering with Different Initialization

W = $\left[\; \right]$ k d

k=8

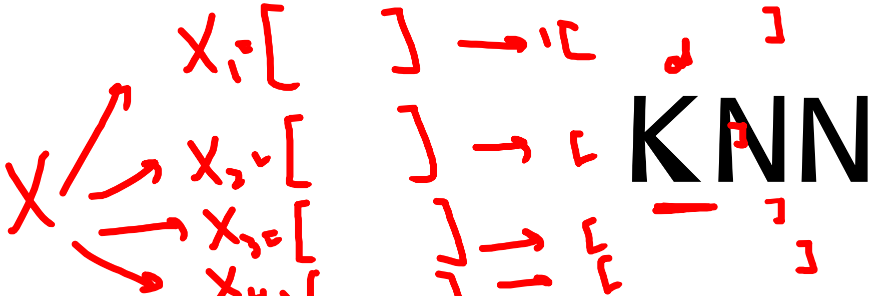


(non-convexity)



- 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


X
$x_1 = [\quad] \to [\quad d \quad]$
$x_2 = [\quad] \to [\quad]$
$x_3 = [\quad] \to [\quad]$
$x_4 = [\quad] \to [\quad]$

- 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.

$$W = \underbrace{\begin{bmatrix} \text{---} & w_1 & \text{---} \\ \text{---} & w_2 & \text{---} \\ & \vdots & \\ \text{---} & w_k & \text{---} \end{bmatrix}}_{d} \left.\vphantom{\begin{bmatrix} a \\ a \\ a \\ a \end{bmatrix}}\right\} k$$

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).
- Use 'f' 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).

$$W = \underbrace{\begin{bmatrix} — & w_1 & — \\ — & w_2 & — \\ & \vdots & \\ — & w_k & — \end{bmatrix}}_{d} \left.\vphantom{\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}}\right\} k$$

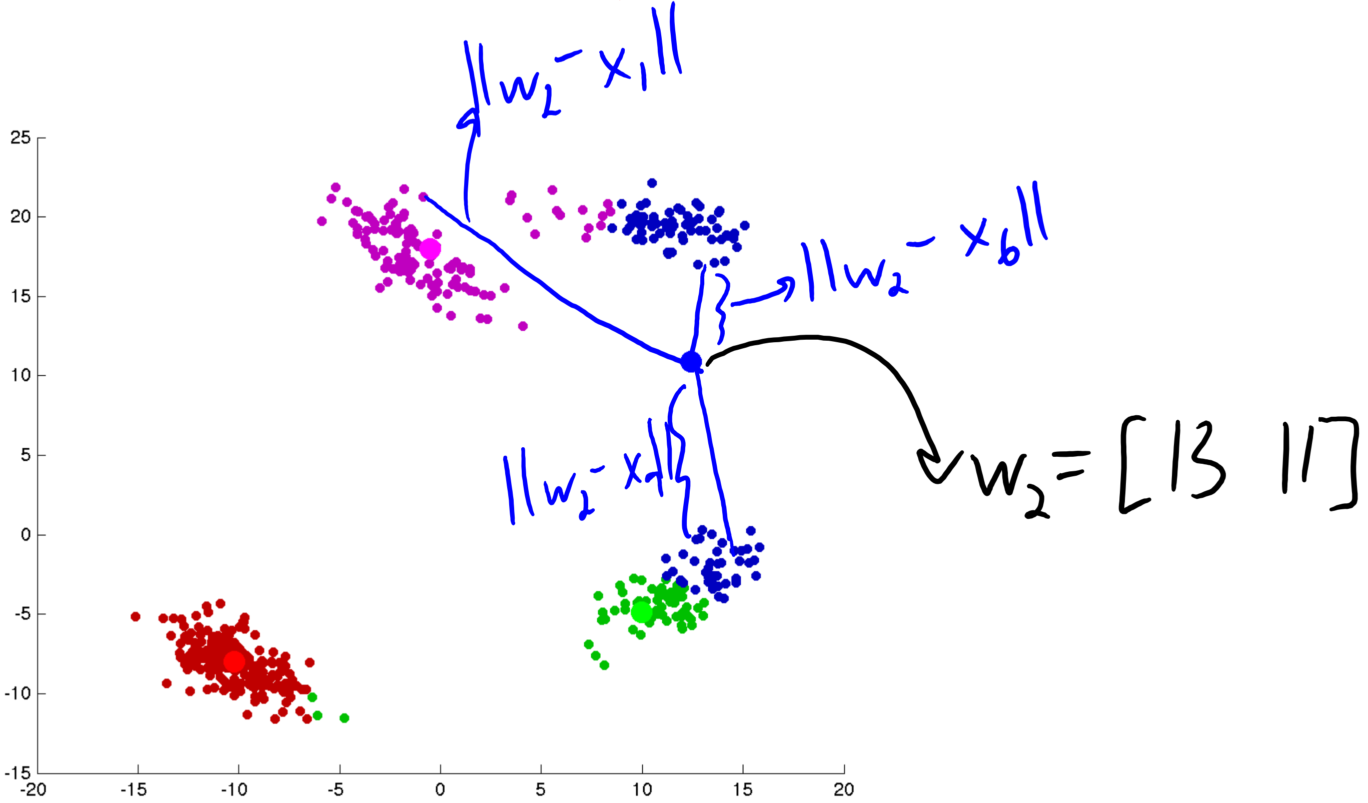
Cost of K-means

- Bottleneck is calculating distance from each $\underbrace{x_i}$ to each mean $\underbrace{w_c}$:

d-dimensional vector

O(d)

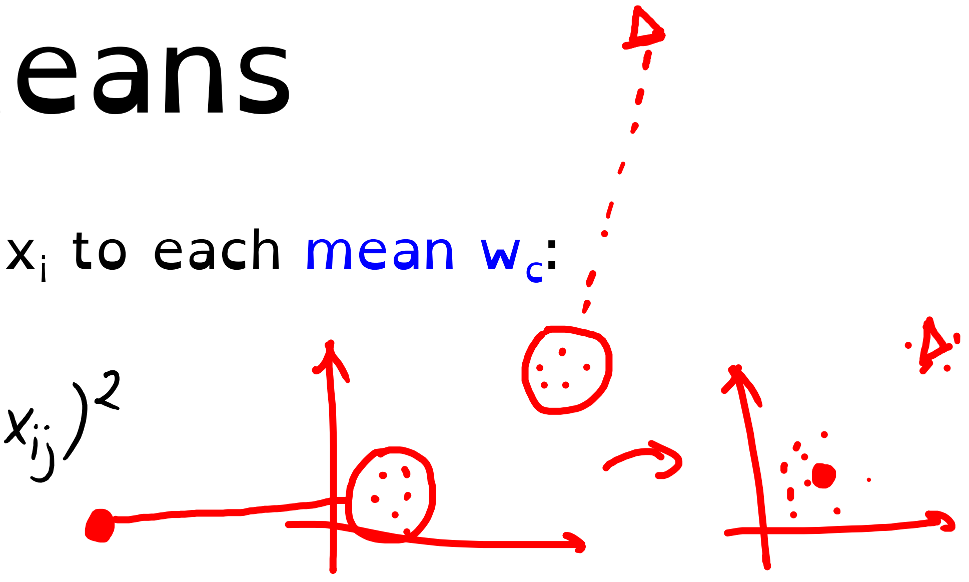
$$\|\underset{d\times1}{w_c} - \underset{d\times1}{x_i}\|^2 = \sum_{j=1}^{d} (\underline{w_{cj}} - \underline{x_{ij}})^2$$



Cost of K-means

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

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



  - 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).

- For each cluster 'c', compute $w_c = \frac{1}{n_c} \sum_{i \in C} x_i$

Object in cluster.

Loop over objects in cluster.

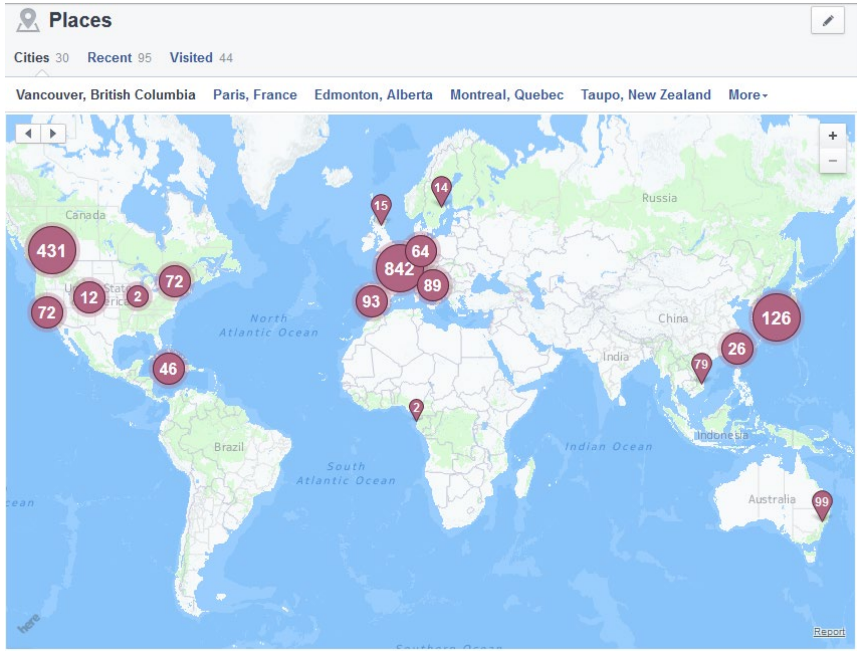
Number of objects in cluster 'c'

Vector Quantization

- K-means originally comes from signal processing.
- Designed for vector quantization:
  - Replace examples with the mean of their cluster ("prototype")
  - An instance of learned compression algorithm

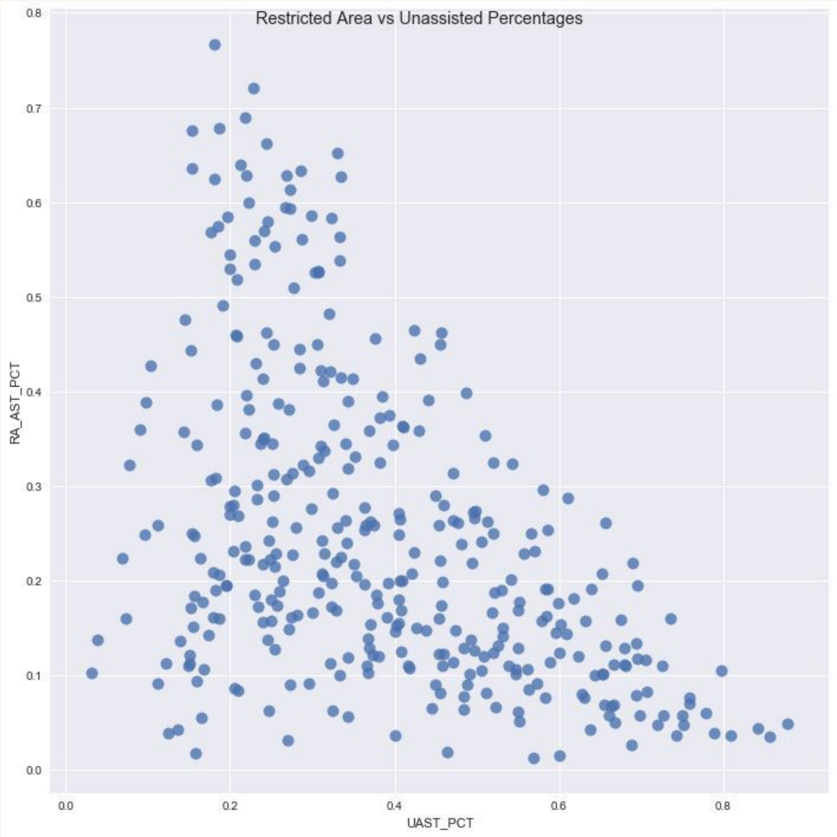
- Example:
  - Facebook places: 1 location summarizes many.
  - What sizes of clothing should I make?

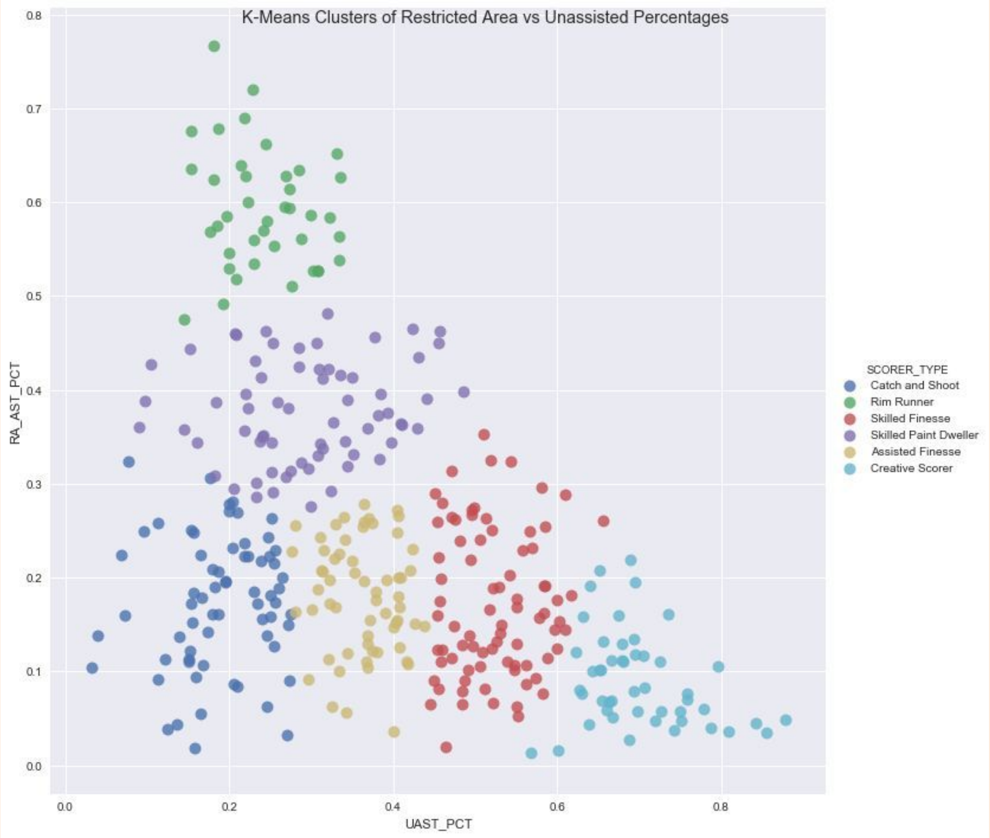
http://wannabite.com/wp-content/uploads/2014/10/ragu-pasta-sauce-printable-coupon.jpg



Vector Quantization for Basketball Players

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

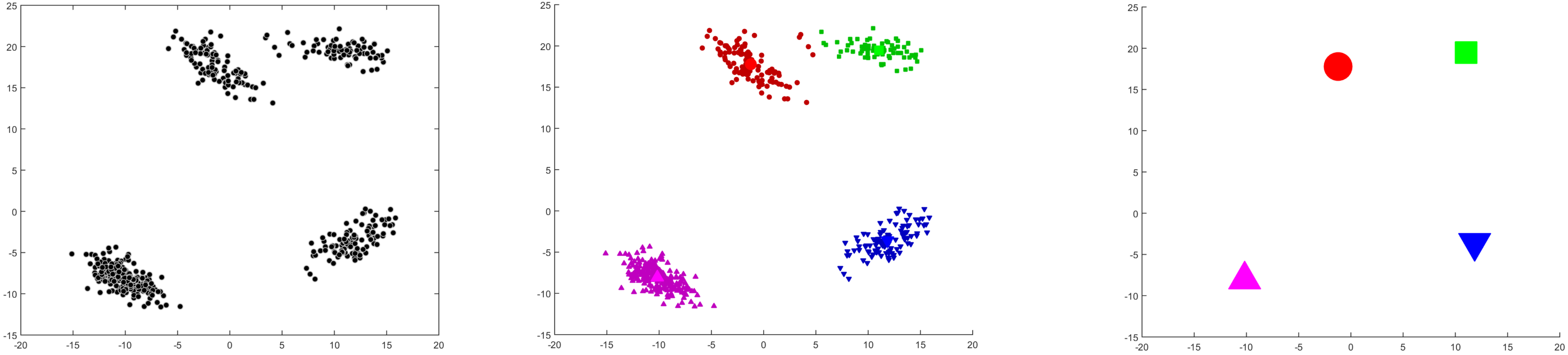




- The “prototypes” (means) give offensive styles (like “catch and shoot”).

https://fansided.com/2018/08/23/nylon-calculus-shooting-volume-versus-efficiency/

Vector Quantization Example



$$X = \overset{n \times d}{\begin{bmatrix} -9.0 & -7.3 \\ -10.9 & -9.0 \\ 13.7 & 19.3 \\ 13.8 & 20.4 \\ 12.8 & 20.6 \\ \vdots & \vdots \end{bmatrix}} \xrightarrow{\text{Run k-means}} W = \underbrace{\overset{k \times d}{\begin{bmatrix} -1.2 & 17.8 \\ -10.2 & -8.0 \\ 11.0 & 19.5 \\ 11.8 & -3.6 \end{bmatrix}}}_{\text{these mean values are the prototypes.}} \quad \hat{y} = \overset{n \times 1}{\begin{bmatrix} 2 \\ 2 \\ 3 \\ 3 \\ 1 \\ \vdots \end{bmatrix}} \xrightarrow[\text{with means.}]{\text{Approximate objects}} \hat{X} = \begin{bmatrix} -10.2 & -8.0 \\ -10.2 & -8.0 \\ 11.0 & 19.5 \\ 11.0 & 19.5 \\ -1.2 & 17.8 \\ \vdots & \vdots \end{bmatrix}$$

Assume each example can be approximated by prototype

(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.

https://globalnews.ca/news/5191123/federal-election-seat-projection-trudeau-liberals-minority/

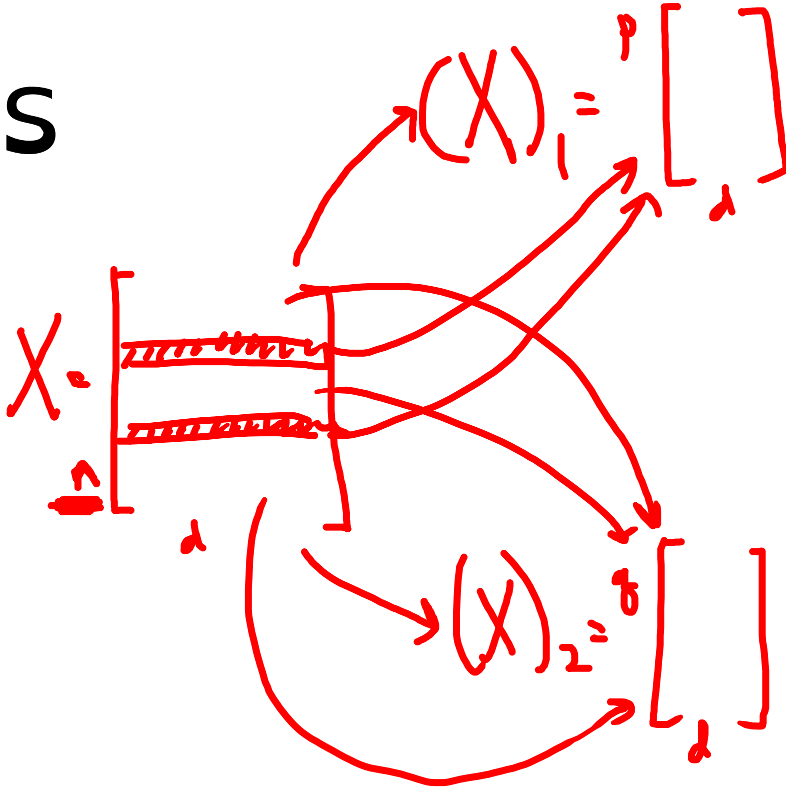
Summary

- Random forests: bagging of deep randomized decision trees.
  - One of the best “out of the box” classifiers.
- Type of ensemble methods:
  - “Boosting” reduces E_{train} and “averaging” reduces E_{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:
  - John Snow and non-parametric clustering.

Review Questions

- Q1: How do bootstrap samples look different from X and y? How do they look like X and y?
- Q2: What makes unsupervised learning useful in the wild?
- Q3: What does it mean K-means clustering does not converge to optimal clustering every time? How do we address this?
- Q4: How does vector quantization compress the data?



$O(pd)\quad \underline{\text{colmean}}((X)_1) = w_1$

$O(qd)\quad \underline{\text{colmean}}((X)_2) = w_2$

$O(pd+qd) = O((\underbrace{p+q}_{n})d) = O(nd)$

Clustering of Epstein-Barr Virus

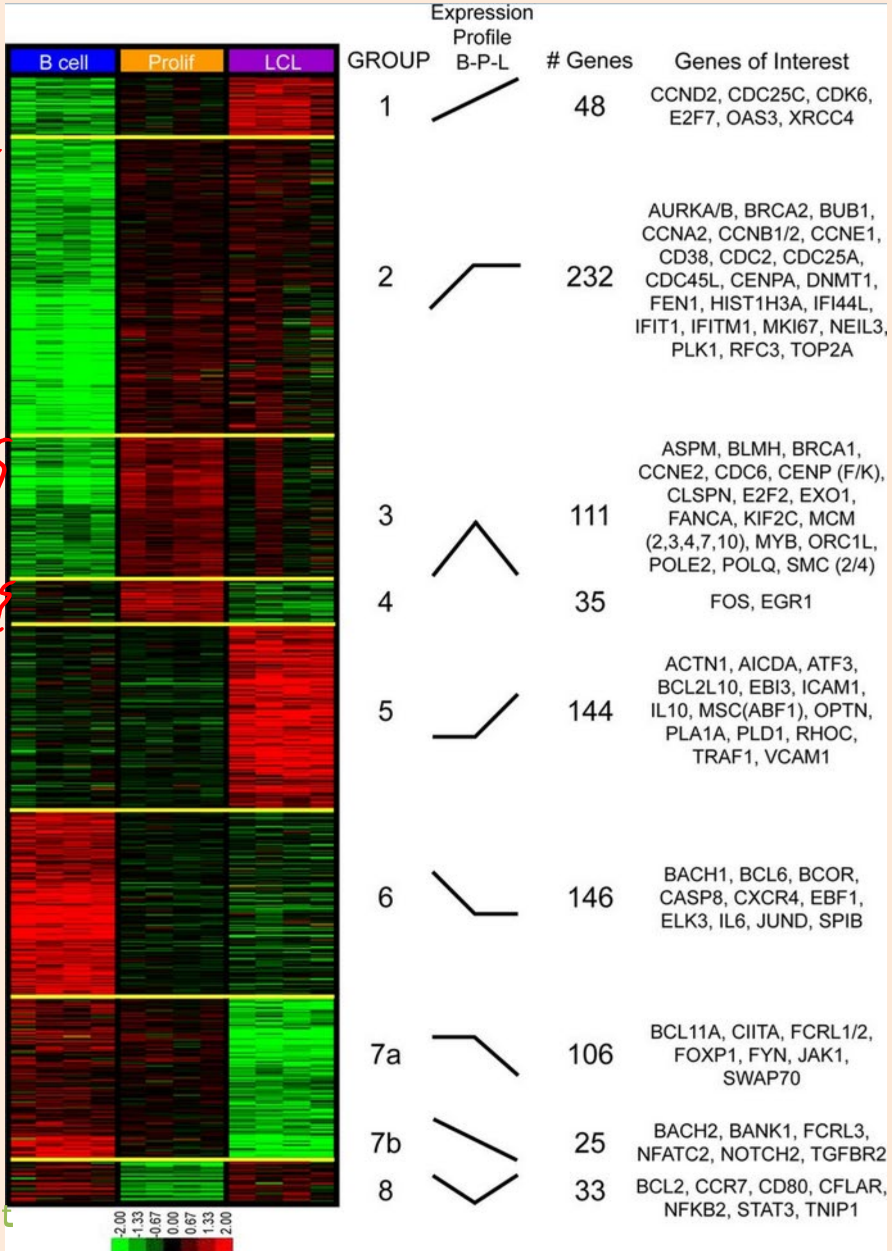
$\hat{y}_i=1$

$\hat{y}_i=2$

$\hat{y}_i=3$

$\hat{y}_i=4$

re-ordered



GROUP	Expression Profile B-P-L	# Genes	Genes of Interest
1		48	CCND2, CDC25C, CDK6, E2F7, OAS3, XRCC4
2		232	AURKA/B, BRCA2, BUB1, CCNA2, CCNB1/2, CCNE1, CD38, CDC2, CDC25A, CDC45L, CENPA, DNMT1, FEN1, HIST1H3A, IFI44L, IFIT1, IFITM1, MKI67, NEIL3, PLK1, RFC3, TOP2A
3		111	ASPM, BLMH, BRCA1, CCNE2, CDC6, CENP (F/K), CLSPN, E2F2, EXO1, FANCA, KIF2C, MCM (2,3,4,7,10), MYB, ORC1L, POLE2, POLQ, SMC (2/4)
4		35	FOS, EGR1
5		144	ACTN1, AICDA, ATF3, BCL2L10, EBI3, ICAM1, IL10, MSC(ABF1), OPTN, PLA1A, PLD1, RHOC, TRAF1, VCAM1
6		146	BACH1, BCL6, BCOR, CASP8, CXCR4, EBF1, ELK3, IL6, JUND, SPIB
7a		106	BCL11A, CIITA, FCRL1/2, FOXP1, FYN, JAK1, SWAP70
7b		25	BACH2, BANK1, FCRL3, NFATC2, NOTCH2, TGFBR2
8		33	BCL2, CCR7, CD80, CFLAR, NFKB2, STAT3, TNIP1

http://jvi.asm.org/content/86/20/11096.abstract

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.

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.

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:

Assume $w_j \perp x_i$

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

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

Bayesian Model Averaging

Again, assuming $w_j \mid X$

- Can get better weights by conditioning on training set: ↗

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

- 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.

What is K-Means Doing?

- How is a k-means step 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.

What is K-Means Doing?

- How is a k-means step 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:

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

- 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 $w_{cj} * n_c = \sum_{i \in c} x_{ij}$

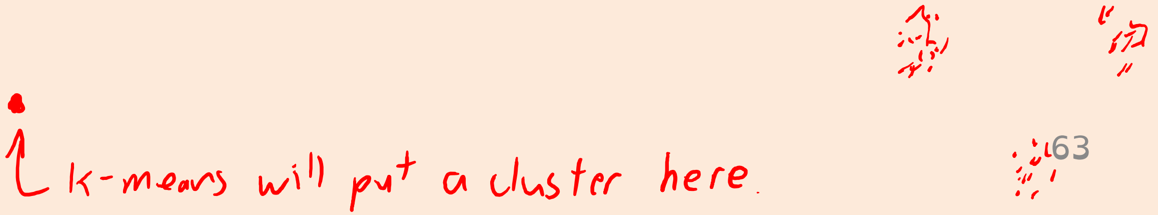
  or $w_{cj} = \frac{1}{n_c} \sum_{i \in c} x_{ij}$

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:

$$\sum_{i=1}^{n} \|w_{\hat{y}_i} - x_i\|_1$$

- 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.

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 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).

K-Medoids Clustering

- A disadvantage of k-means in some applications:
  - The means might not be valid data points.
  - May be important for vector quantiziation.
- 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.

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.

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 $(d_i)^2$.

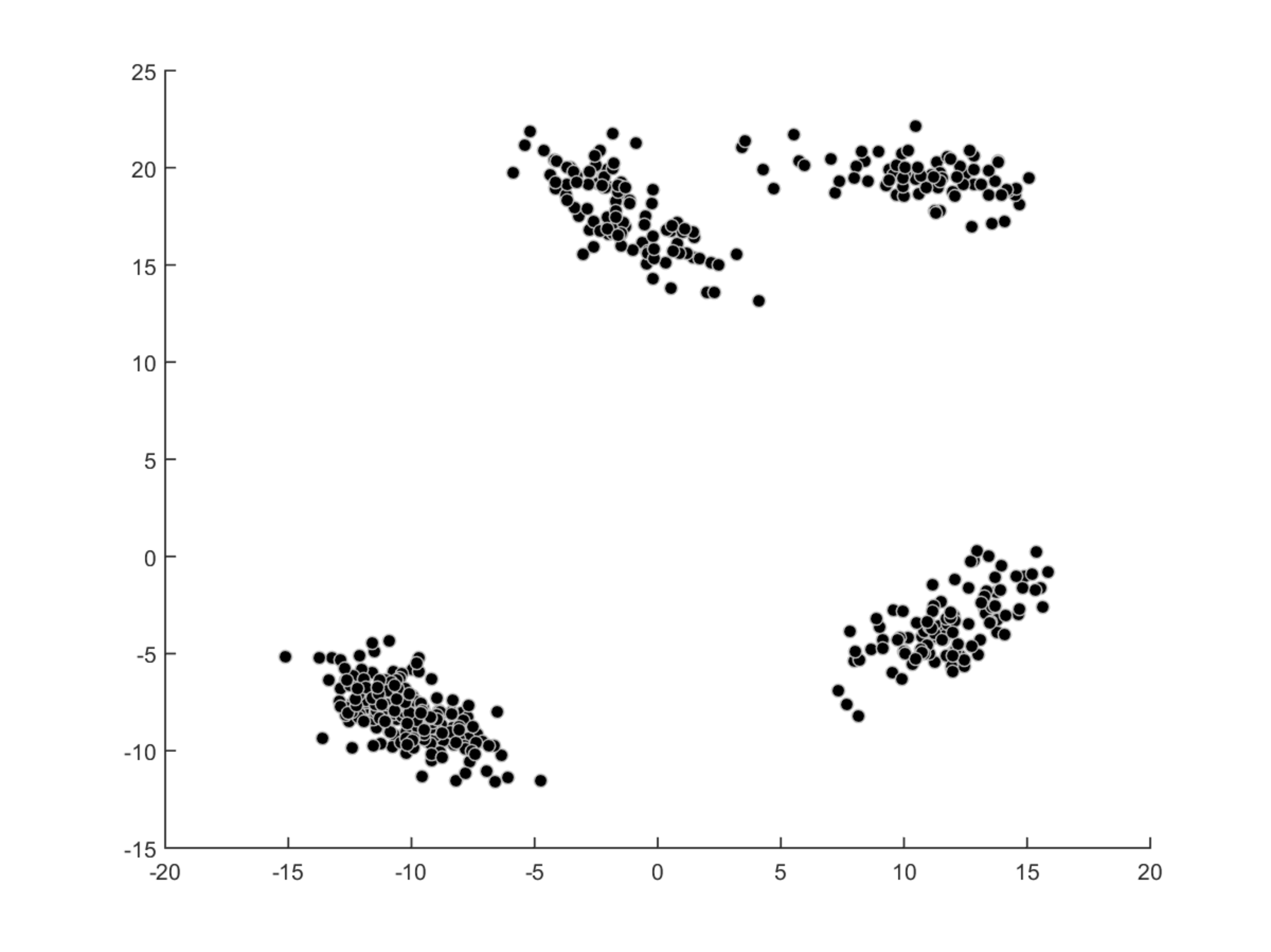
$$p_i \propto d_i^2 \Rightarrow p_i = \frac{d_i^2}{\sum_{j=1}^{n} d_j^2}$$

Can be done in $O(n)$.

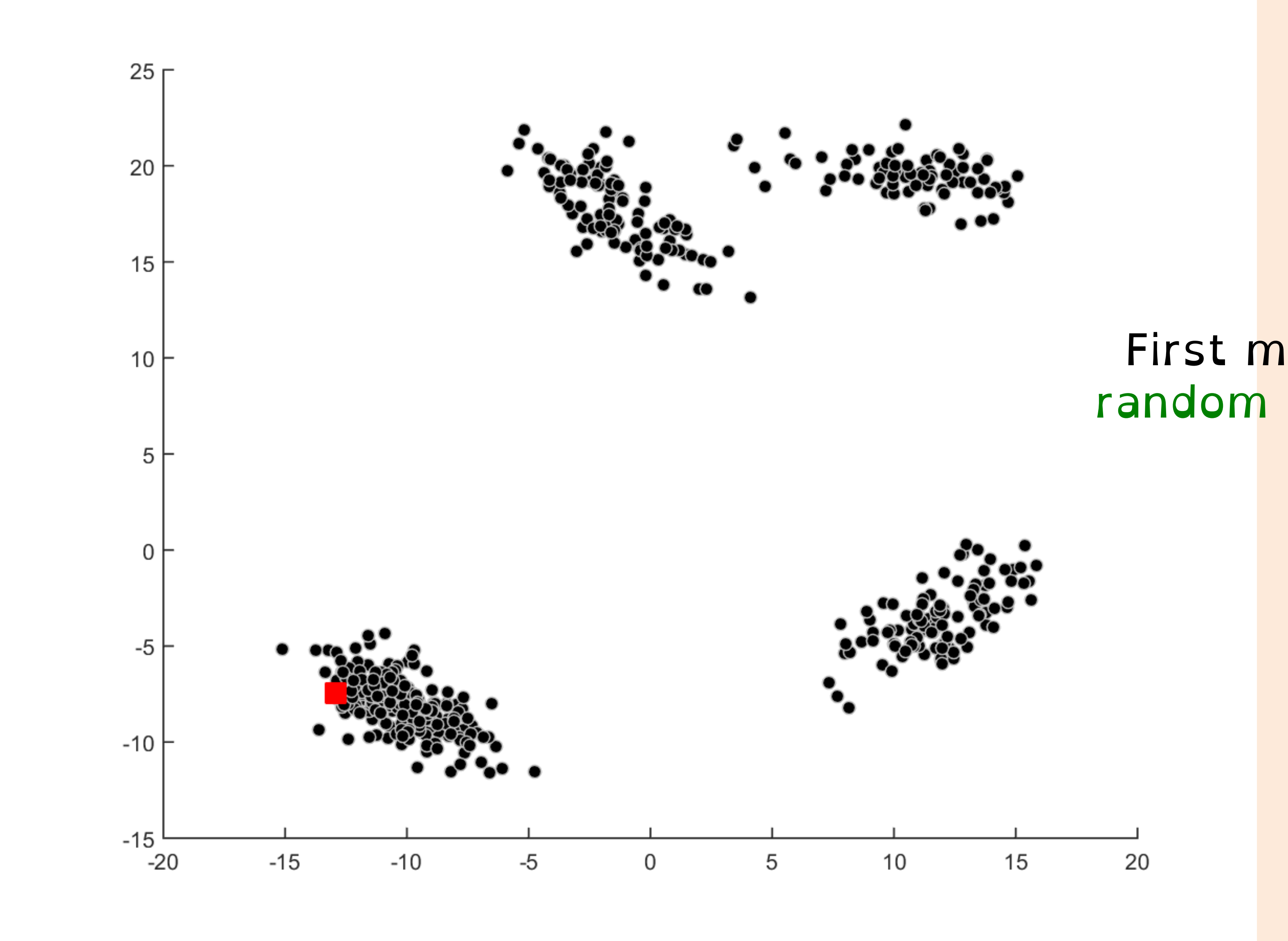
"probability that we choose x_i as next mean"

  5. Keep returning to step 2 until we have k-means.
- Expected approximation ratio is O(log(k)).

K-Means++

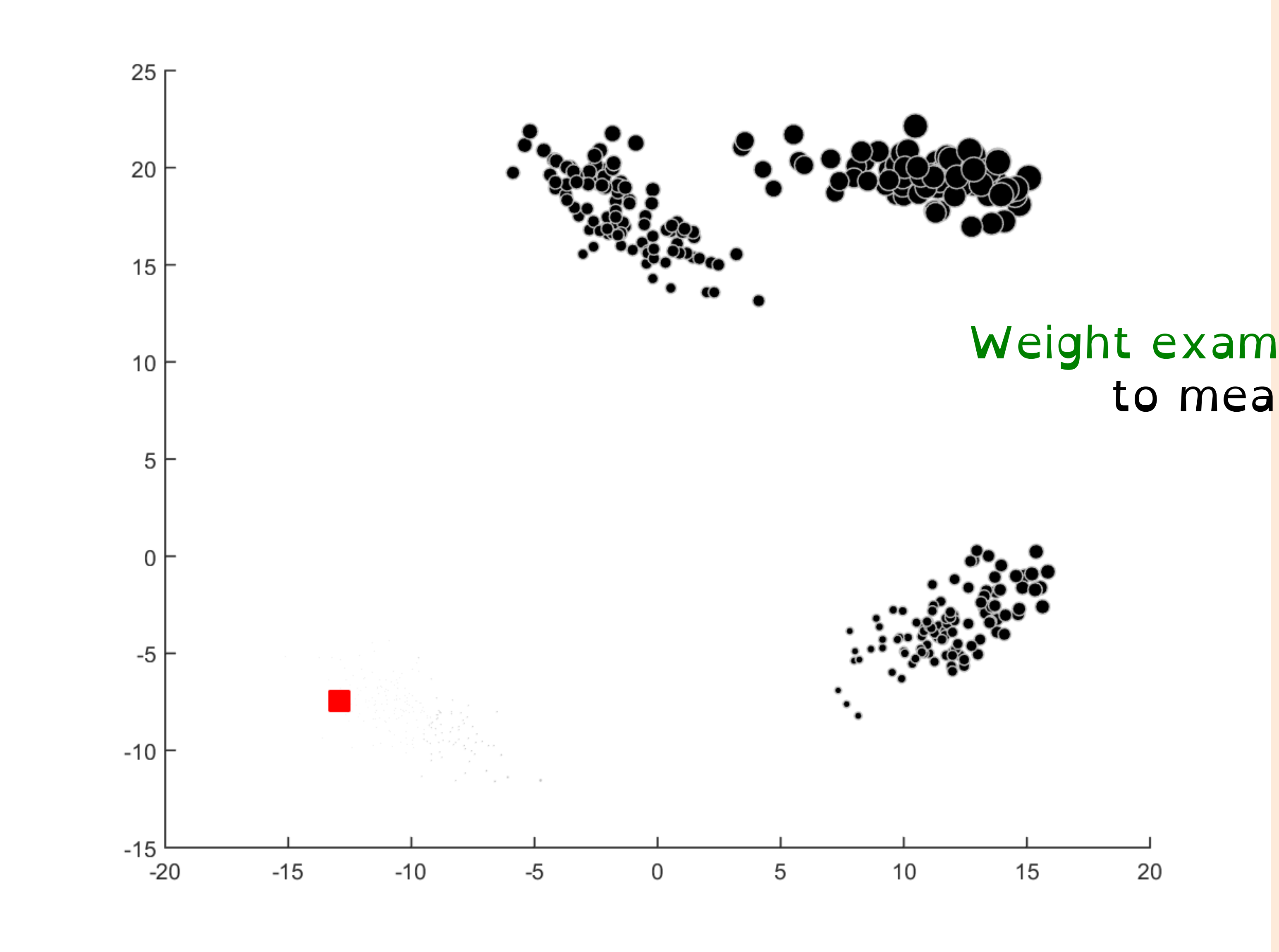


K-Means++



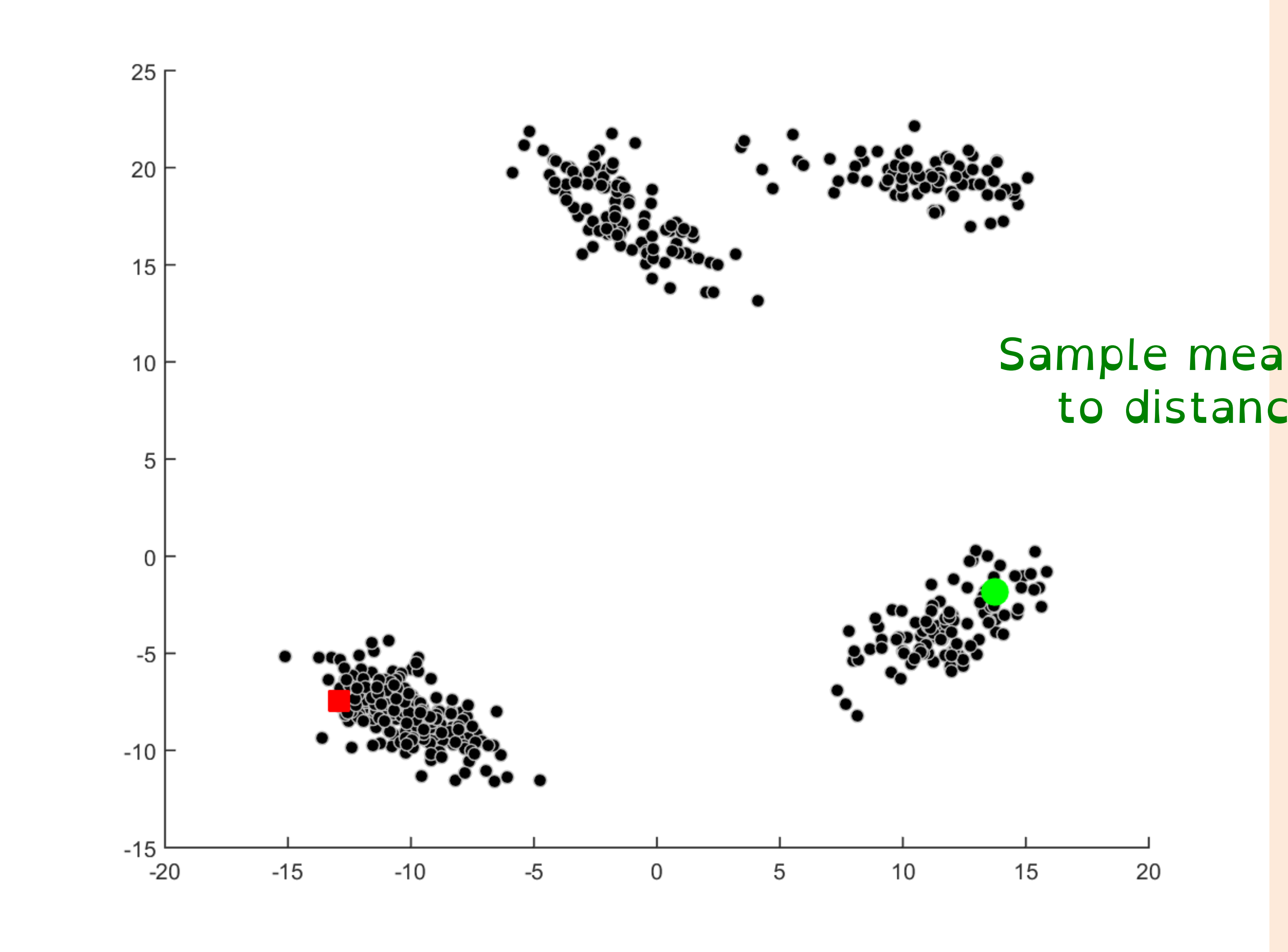
First mean is a random example.

K-Means++



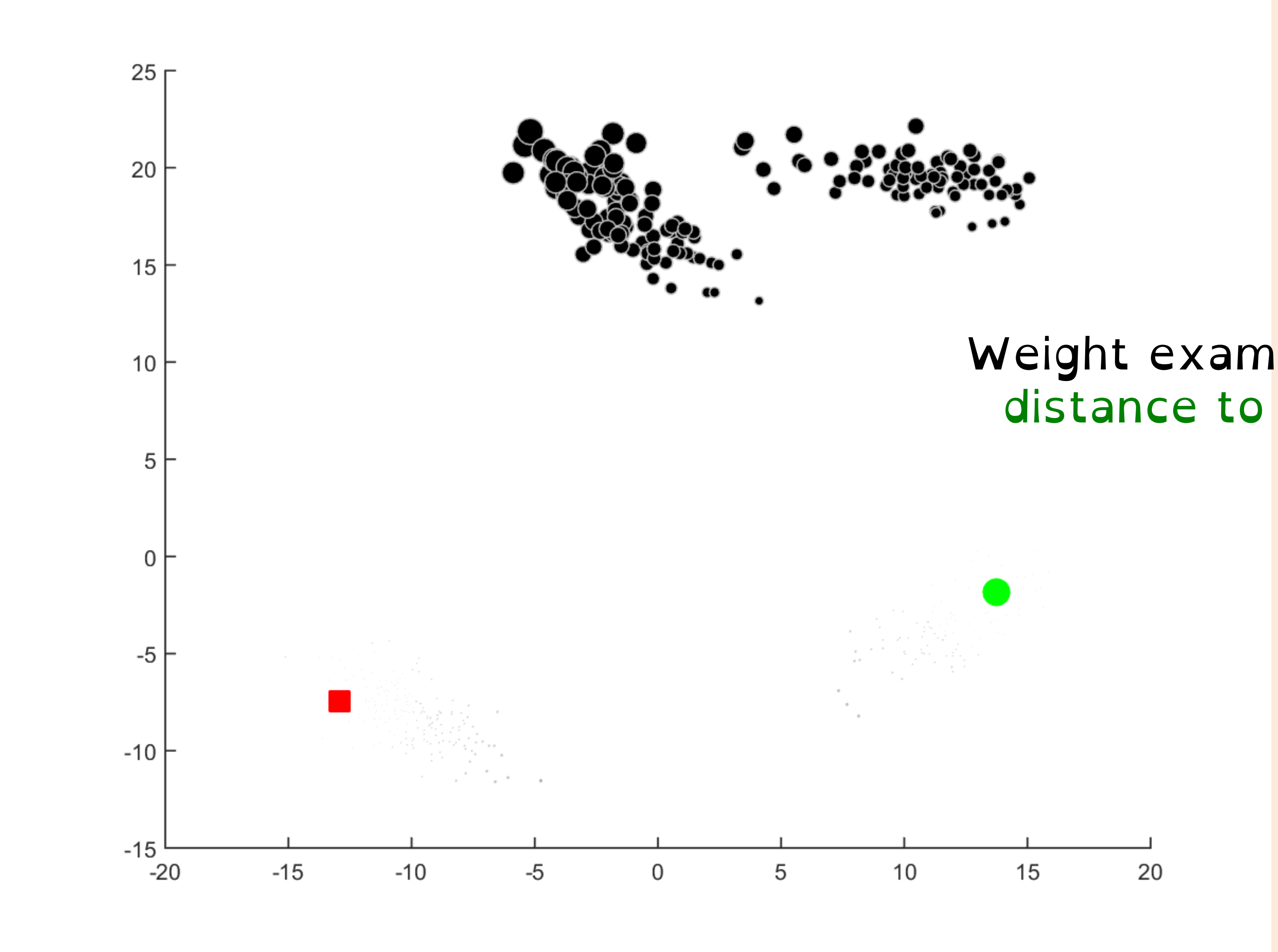
Weight examples by distance to mean squared.

K-Means++



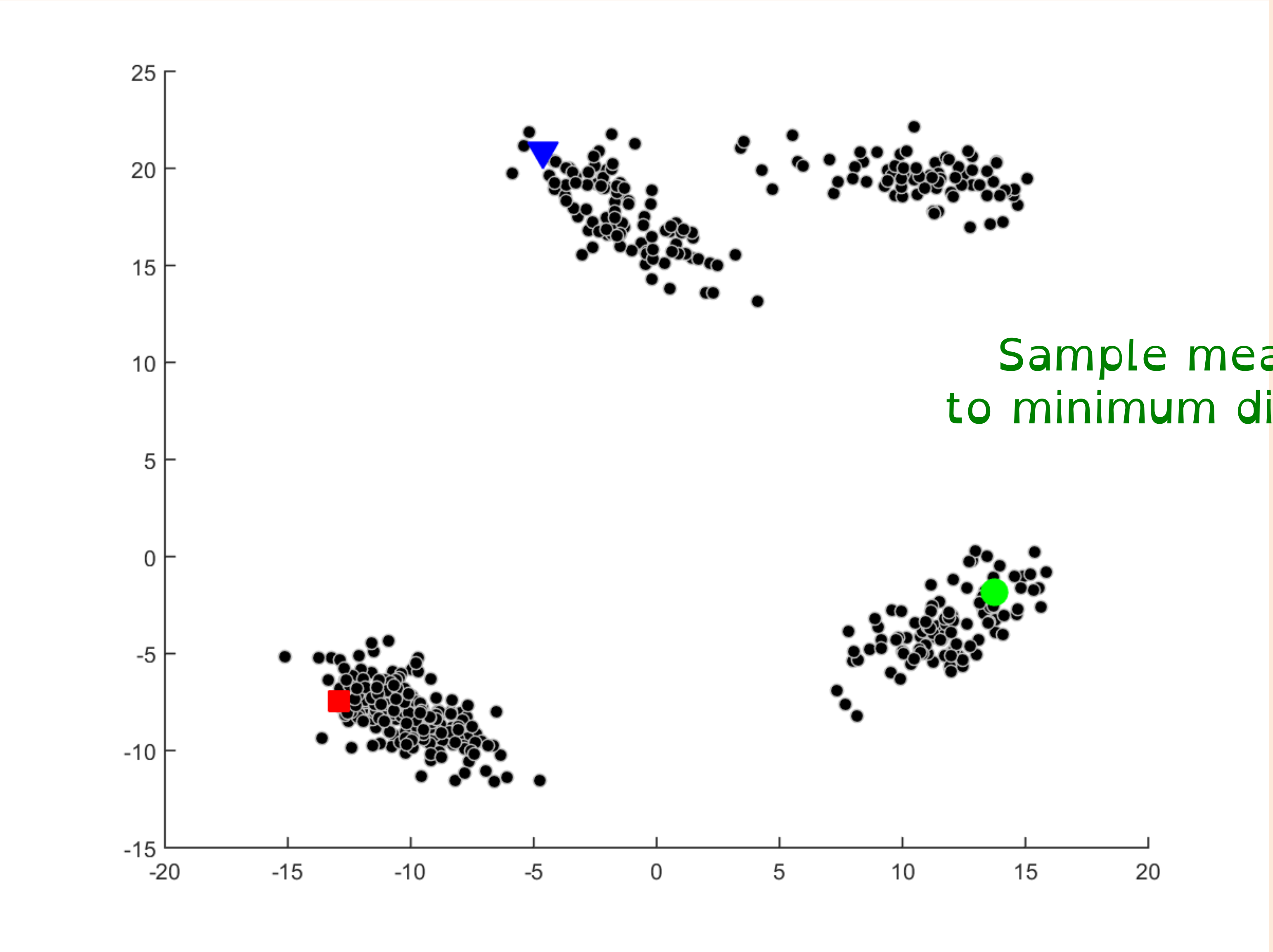
Sample mean proportional to distances squared.

K-Means++



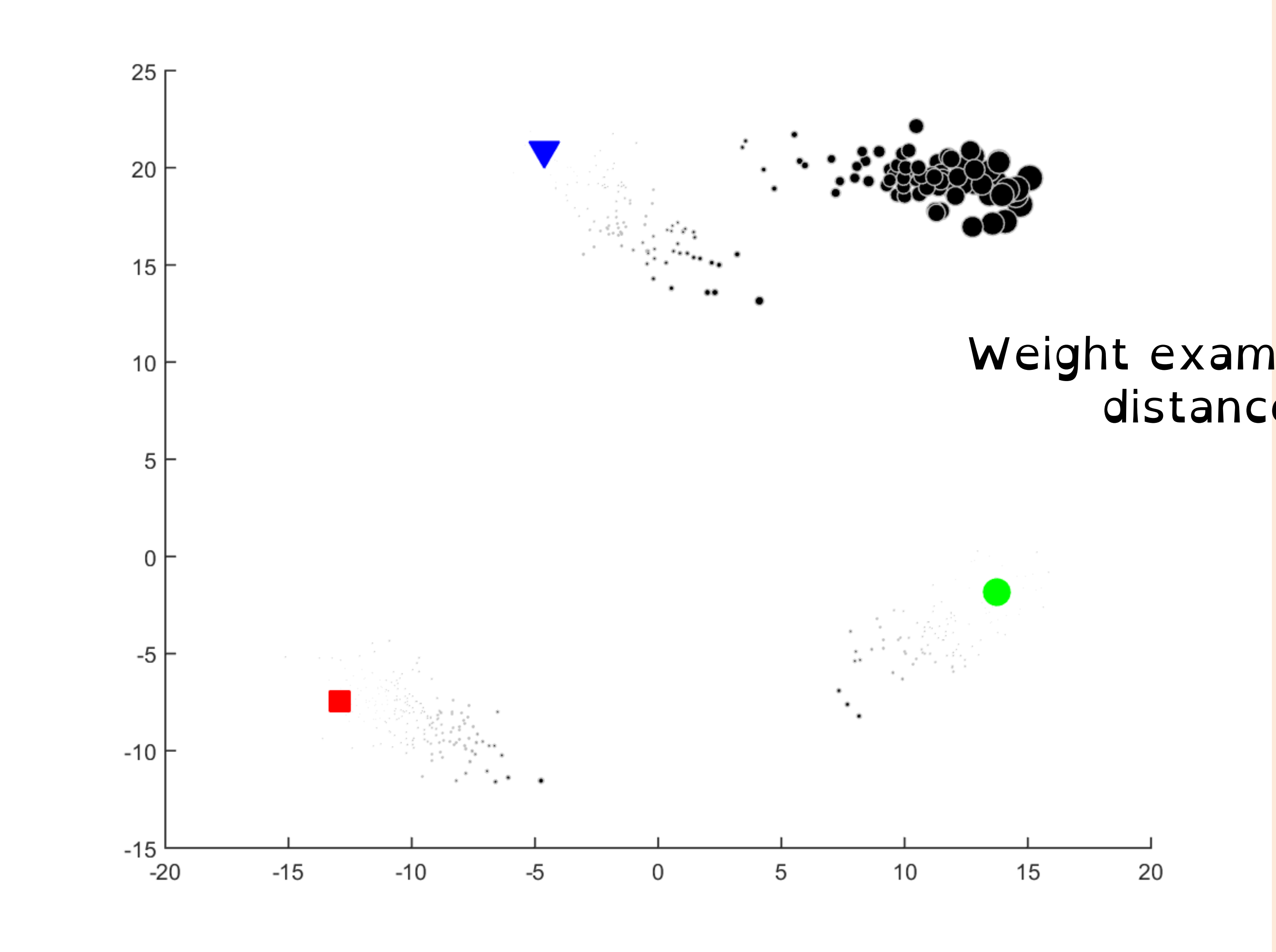
Weight examples by squared distance to nearest mean.

K-Means++



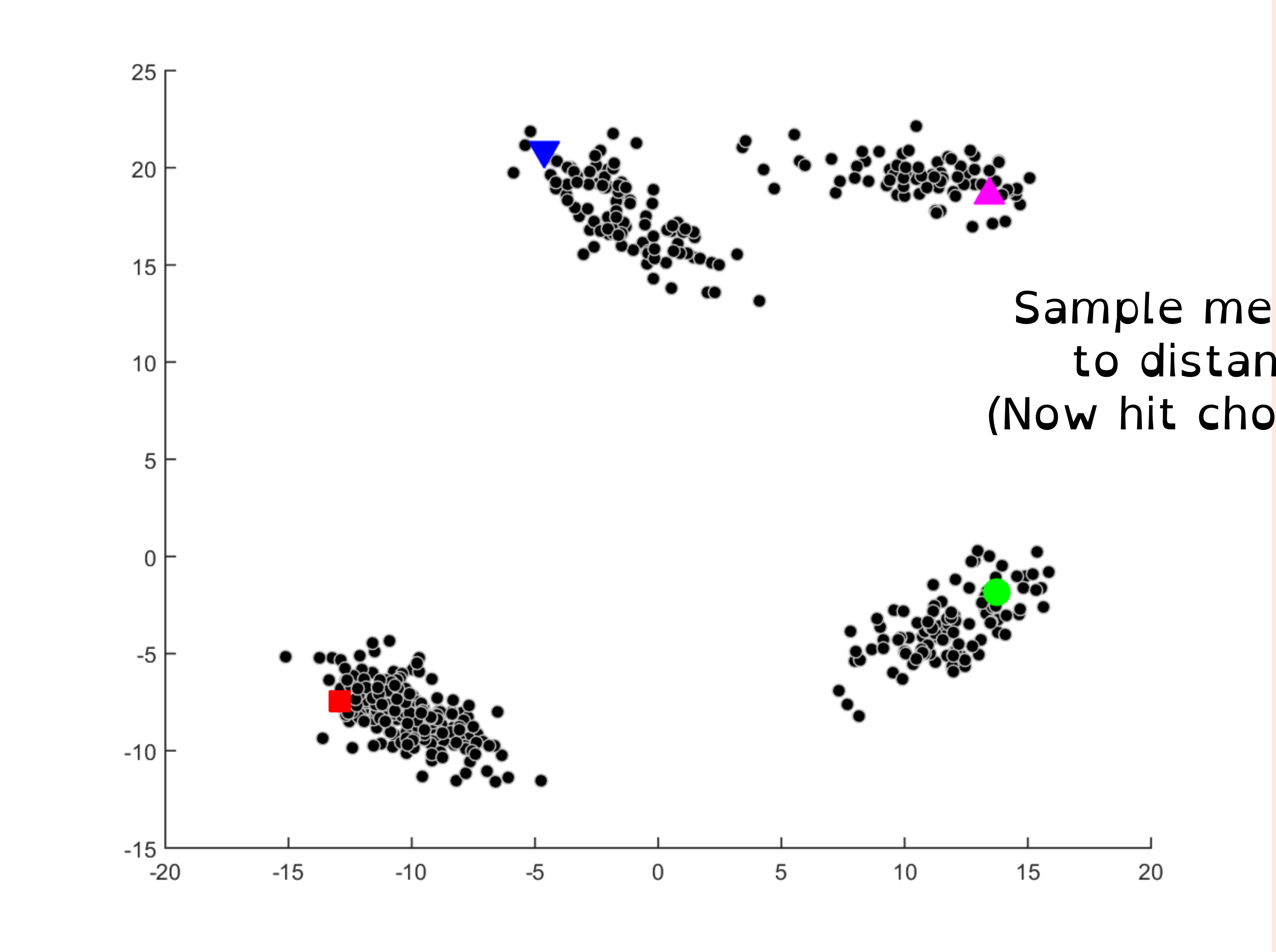
Sample mean proportional to minimum distances squared.

K-Means++



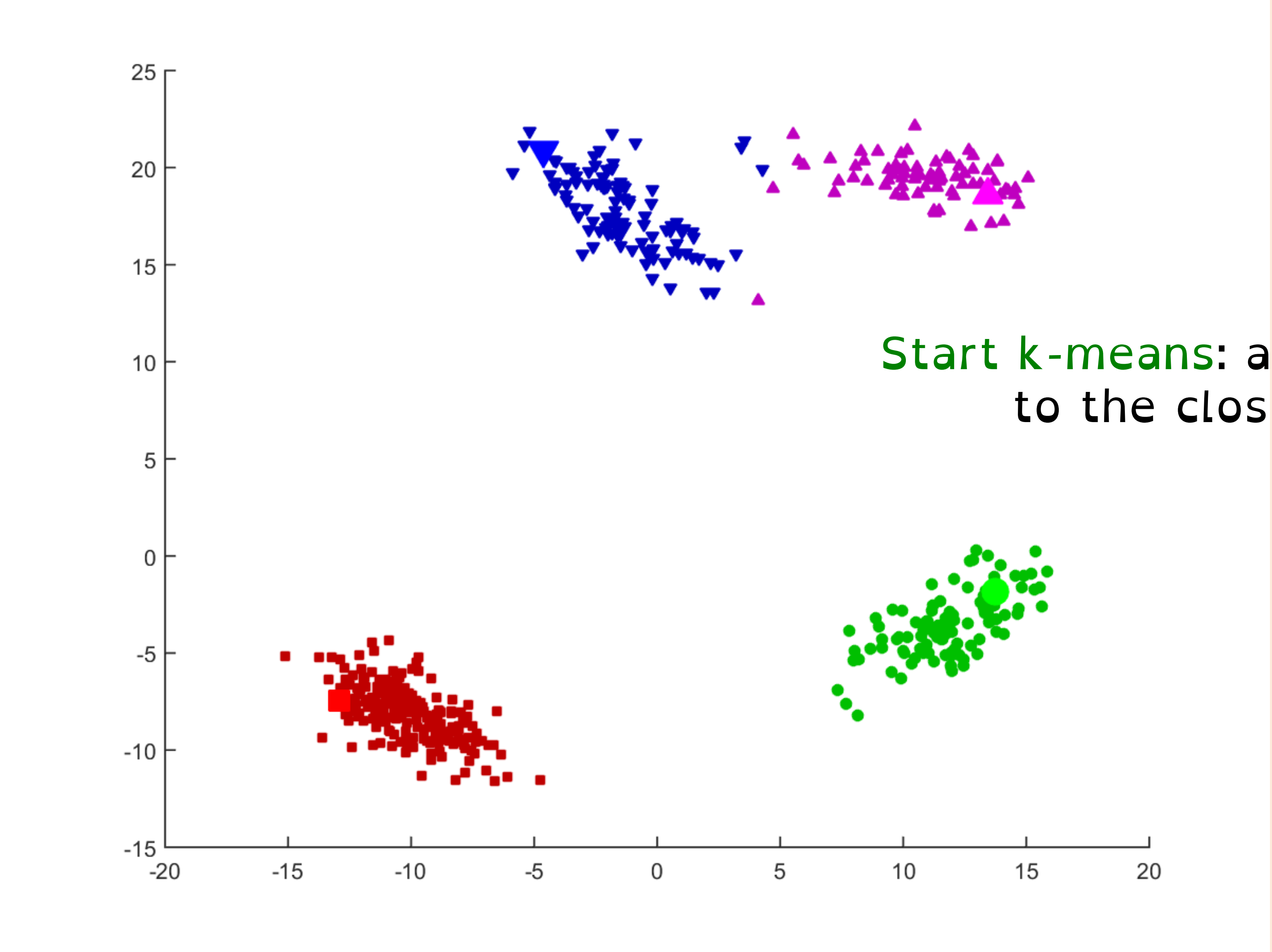
Weight examples by squared distance to mean.

K-Means++



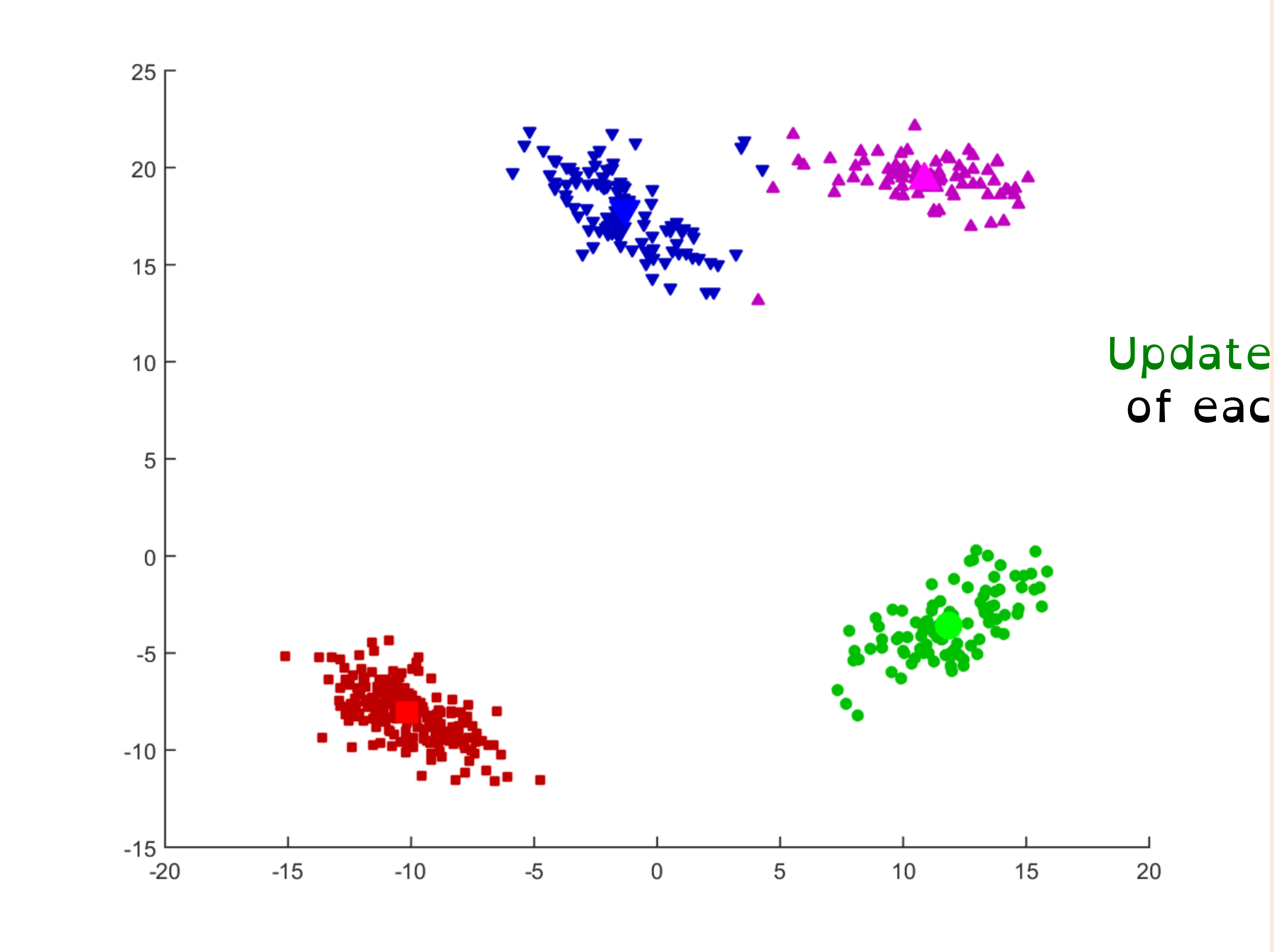
Sample mean proportional
to distances squared.
(Now hit chosen target k=4.)

K-Means++



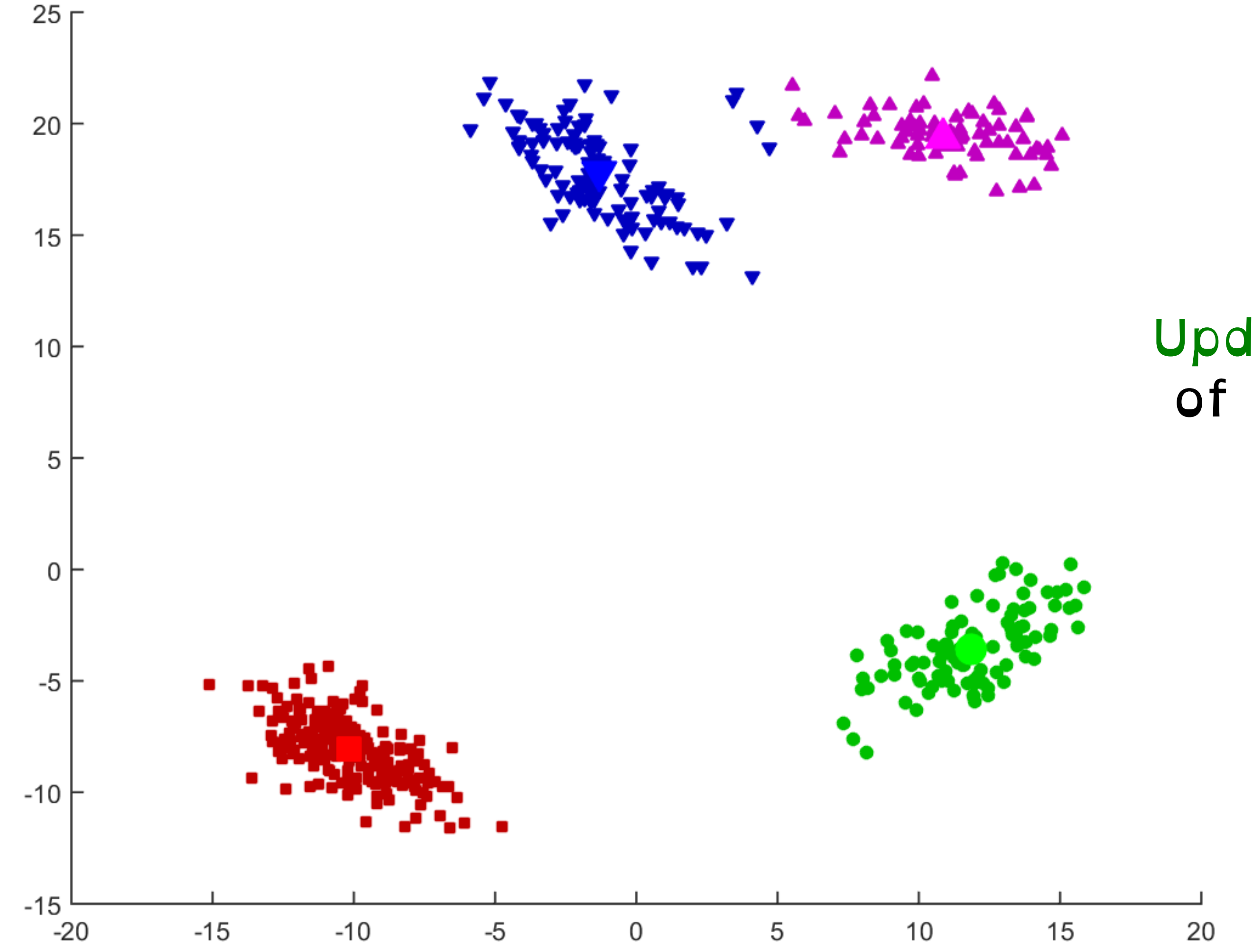
Start k-means: assign examples to the closest mean.

K-Means++



Update the mean of each cluster.

K-Means++



Update the mean of each cluster.

In this case: just 2 iterations!

Discussion of K-Means++

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

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

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

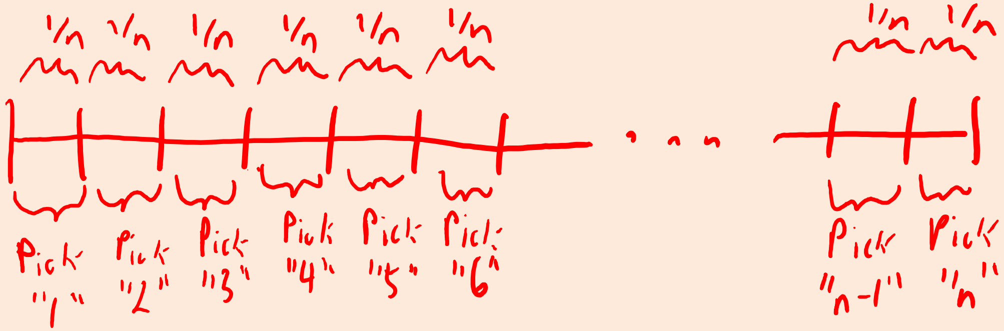
$$\frac{\underset{\text{expectation over random samples}}{E}\left[f(w, c) \right]}{f(\underbrace{w^*, c^*}_{\text{``Best'' mean and clustering according to objective.}})} \le O(\log(k))$$

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

Uniform Sampling

- Standard approach to generating a random number from {1,2,…,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:

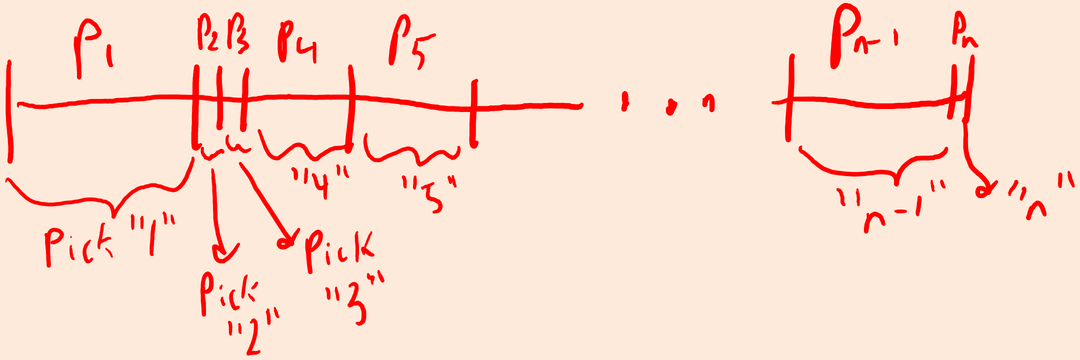


- This assumes $p_i = 1/n$ for all ‘i’.

↑ probability of picking number 'i'.

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 ≤ $\sum_{j=1}^{i} p_i$
- 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.

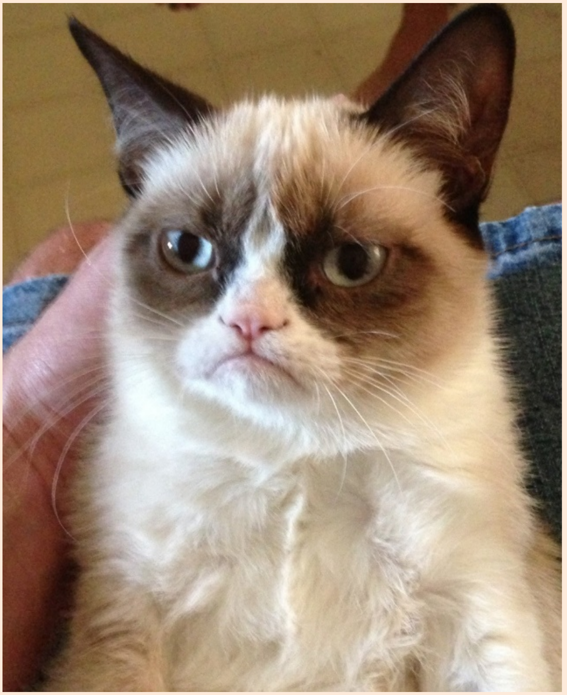
How many iterations does k-means take?

- 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

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)

$$X = \left.\underbrace{\begin{bmatrix} \quad & \quad & \quad \\ \\ \\ \\ \end{bmatrix}}_{\substack{3 \\ \text{(all 8-bit)}}}\right\} \text{\#pixels}$$

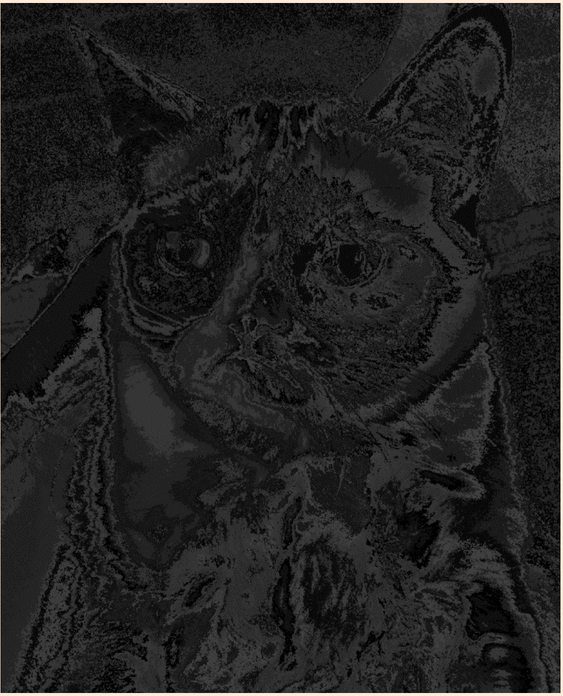
Run k-means with 2^6 clusters:

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

$$W = \left.\underbrace{\begin{bmatrix} \bigcirc & \quad & \quad \\ \\ \\ \end{bmatrix}}_{3}\right\} 2^6 \downarrow \text{64 colours}$$

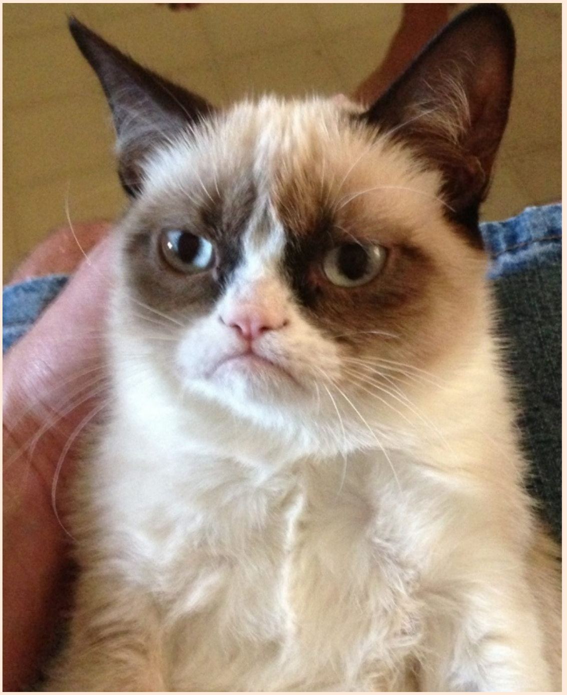
K-means predictions:
(6-bits/pixel)

$$\hat{y} = \left.\underbrace{\begin{bmatrix} 1 \\ 2 \\ 3 \\ 2 \\ 2 \\ 1 \\ 1 \\ 3 \end{bmatrix}}_{\substack{1 \text{ 6-bit number} \\ \text{which refers to} \\ \text{one of } 2^6 \text{ colours.}}}\right\} \text{cluster for each pixel}$$



Vector Quantization: Image Colors

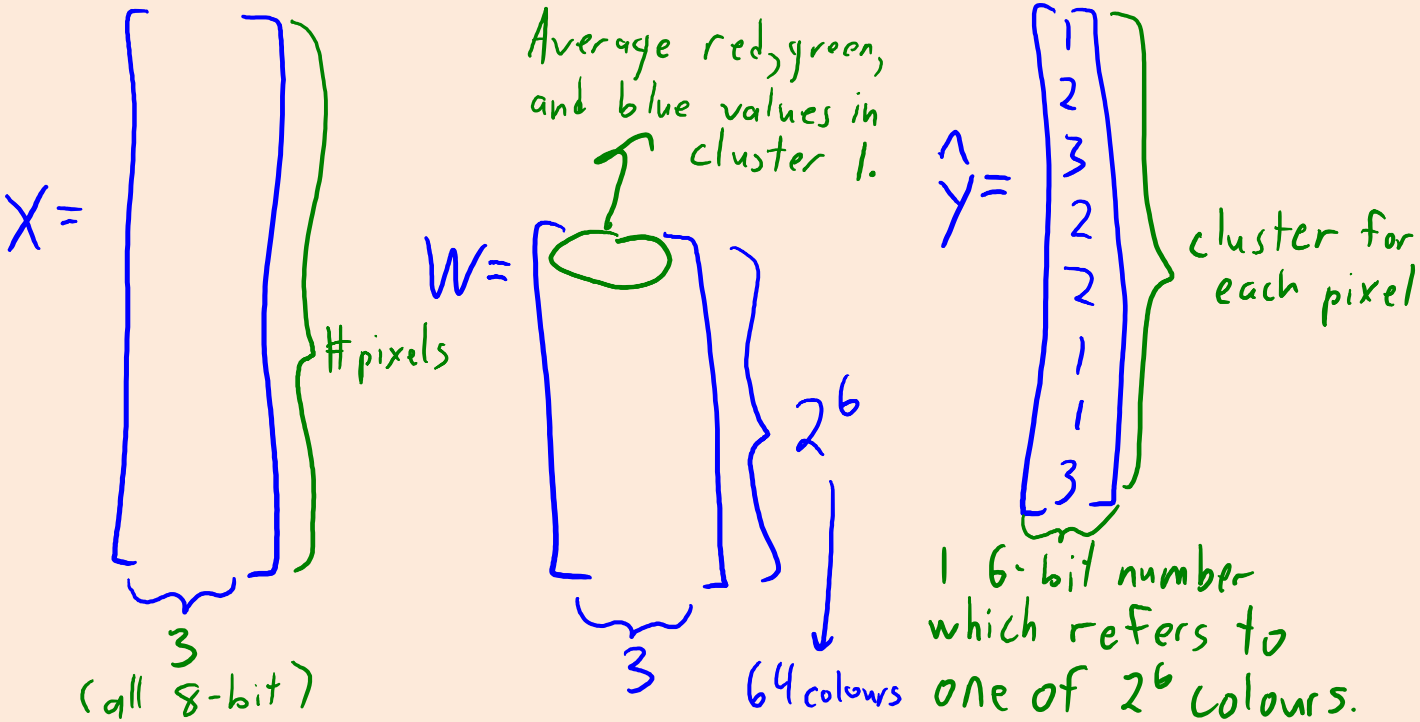
- Usual RGB representation of a pixel's color: three 8-bit numbers.
  - For example, [241 13 50] = 🟥 .
  - Can apply k-means to find set of prototype colours.



Original:
(24-bits/pixel)

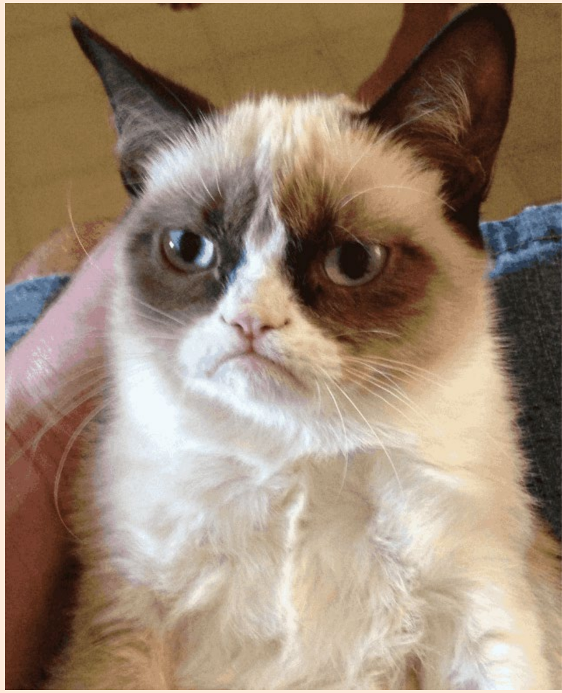
Run k-means with 2^6 clusters:

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



X=
#pixels
3
(all 8-bit)
Average red, green, and blue values in cluster 1.
W=
2^6
3
64 colours
ŷ=
1
2
3
2
2
1
1
3
cluster for each pixel
1 6-bit number which refers to one of 2^6 colours.

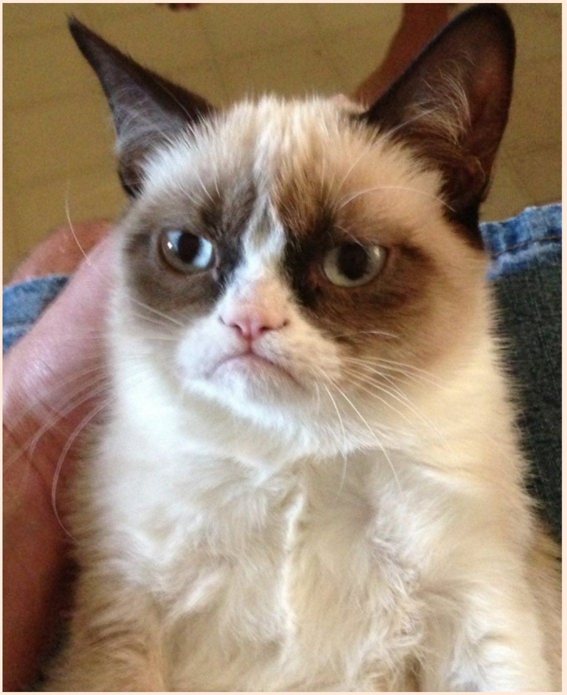
Replace cluster with mean:



Vector Quantization: Image Colors

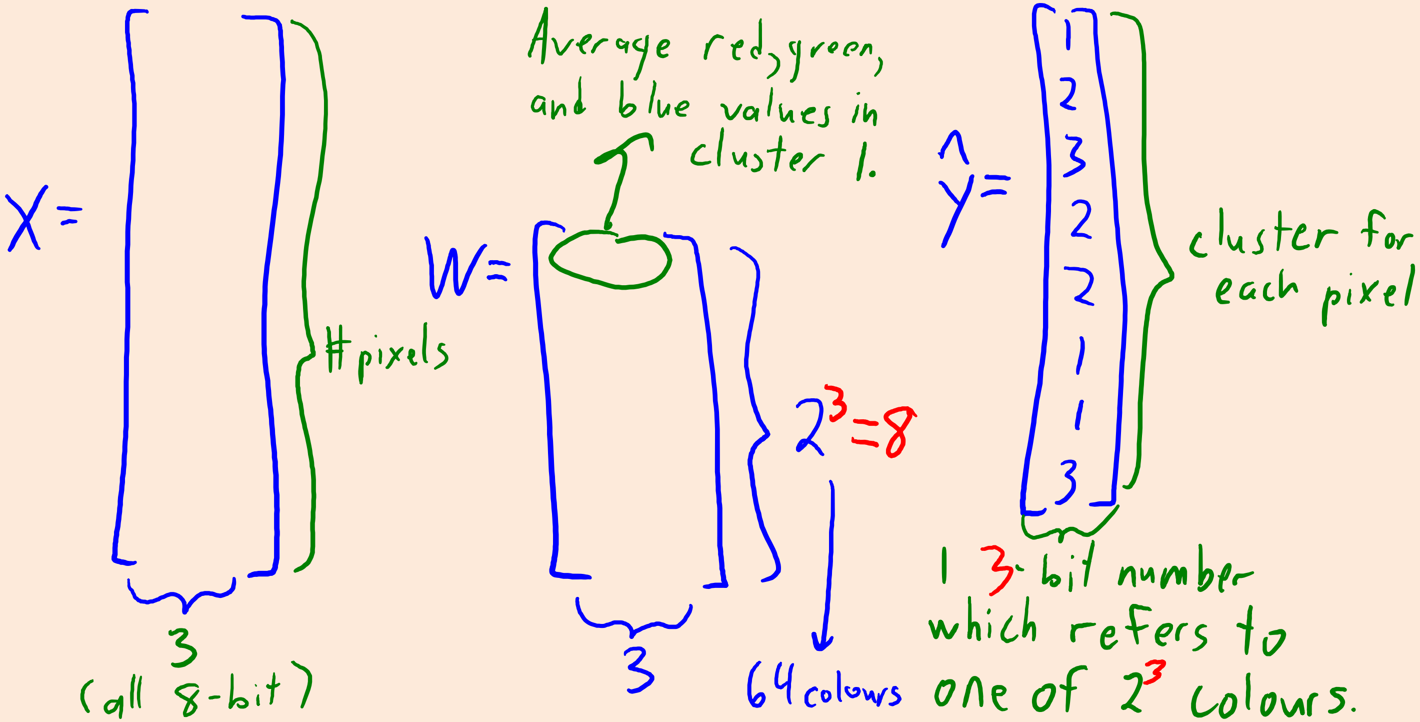
- Usual RGB representation of a pixel's color: three 8-bit numbers.
  - For example, [241 13 50] = 🟥 .
  - Can apply k-means to find set of prototype colours.

Original:
(24-bits/pixel)



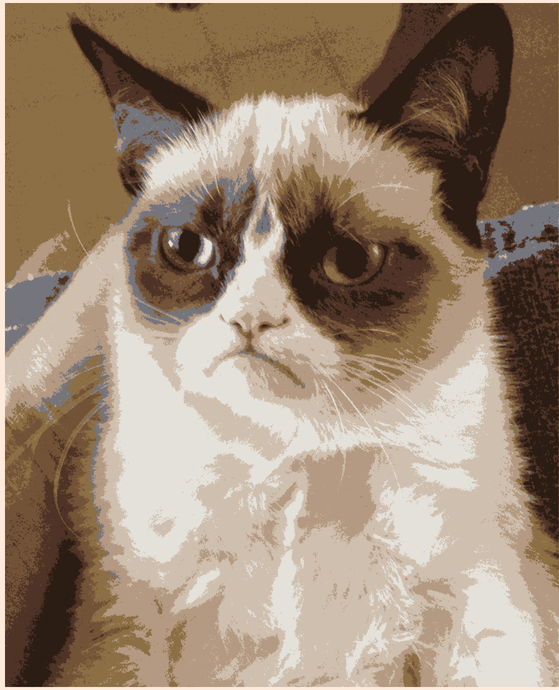
Run k-means with 2^6 clusters:

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



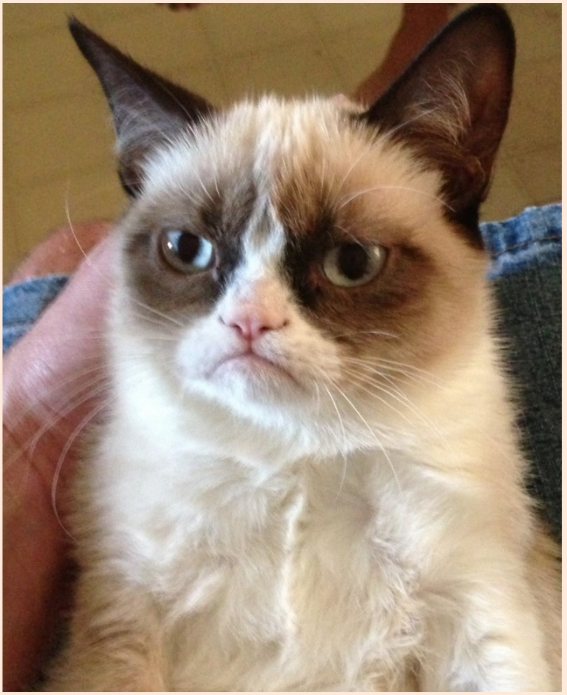
X =
#pixels
3
(all 8-bit)
W =
Average red, green, and blue values in cluster 1.
$2^3=8$
64 colours
3
$\hat{y}$ =
1
2
3
2
2
1
1
3
cluster for each pixel
1 3-bit number which refers to one of 2^3 colours.

Replace cluster with mean:



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)

$$X = \underbrace{\begin{bmatrix} \quad \\ \\ \\ \\ \end{bmatrix}}_{\substack{3 \\ (\text{all 8-bit})}} \left.\vphantom{\begin{bmatrix} \\ \\ \\ \\ \end{bmatrix}}\right\} \#\text{pixels}$$

Run k-means with
2^6 clusters:

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

$$W = \underbrace{\begin{bmatrix} \bigcirc \\ \\ \\ \end{bmatrix}}_{3} \left.\vphantom{\begin{bmatrix} \\ \\ \\ \end{bmatrix}}\right\} 2^2 = 4$$

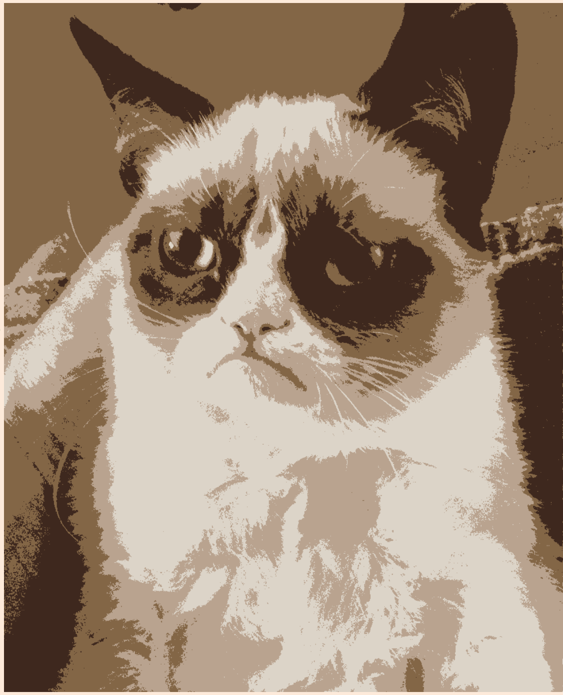
$\downarrow$ 64 colours

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

$$\hat{y} = \underbrace{\begin{bmatrix} 1 \\ 2 \\ 3 \\ 2 \\ 2 \\ 1 \\ 1 \\ 3 \end{bmatrix}}_{} \left.\vphantom{\begin{bmatrix} 1 \\ 2 \\ 3 \\ 2 \\ 2 \\ 1 \\ 1 \\ 3 \end{bmatrix}}\right\} \text{cluster for each pixel}$$

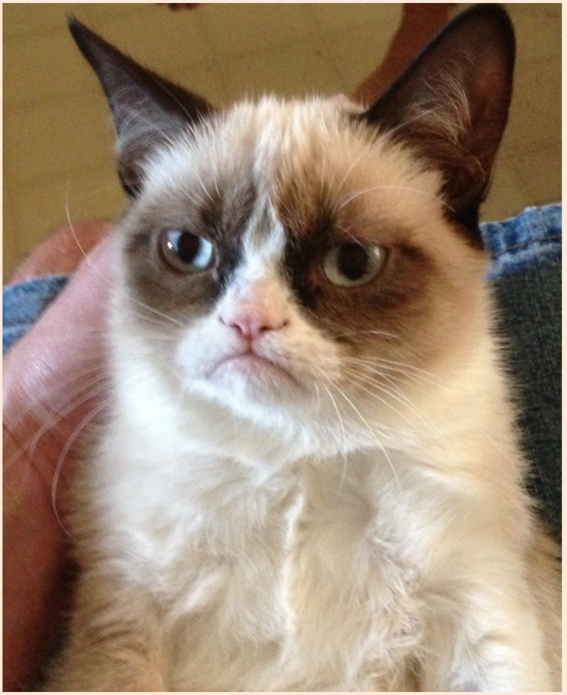
1 2-bit number which refers to one of 2^2 colours.

Replace cluster with mean:



Vector Quantization: Image Colors

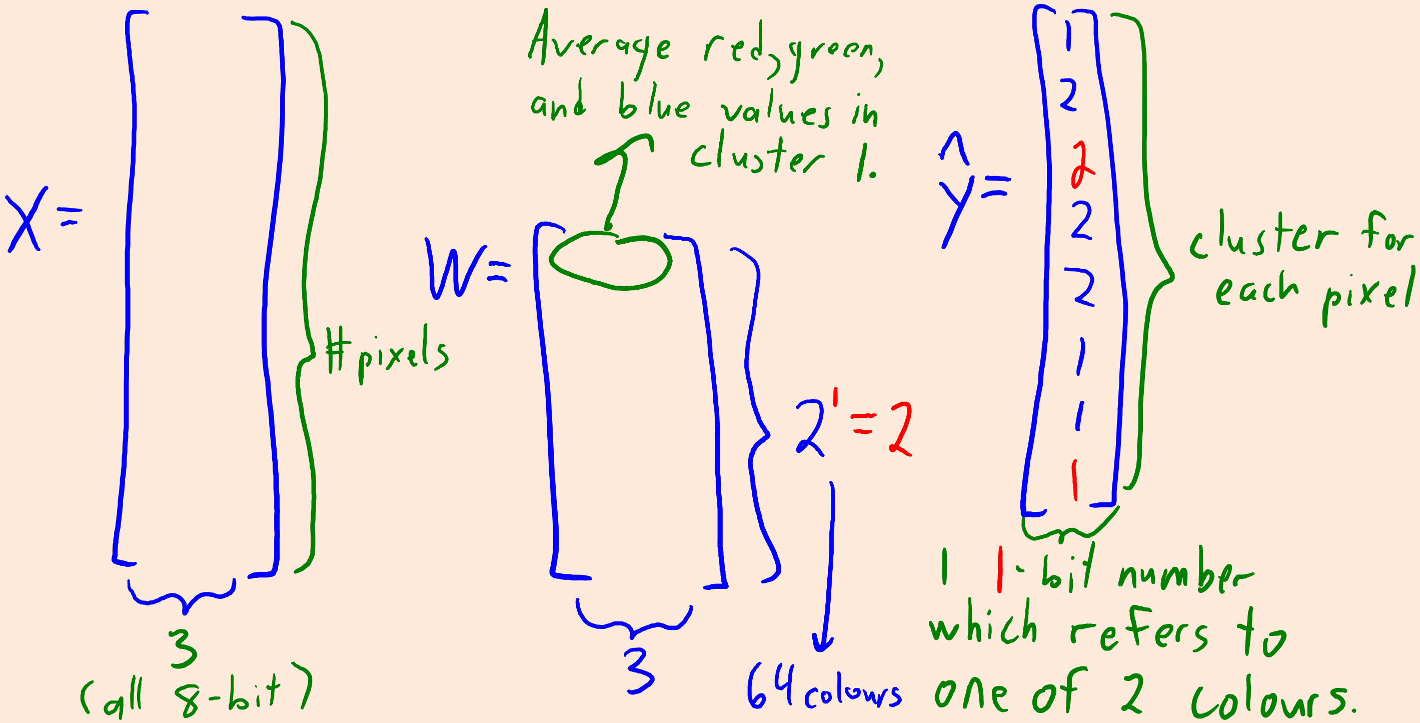
- Usual RGB representation of a pixel’s color: three 8-bit numbers.
  - For example, [241 13 50] = 🟥 .
  - Can apply k-means to find set of prototype colours.



Original:
(24-bits/pixel)

Run k-means with
2^6 clusters:

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


X=
#pixels
3
(all 8-bit)
Average red, green, and blue values in cluster 1.
W=
$2^1 = 2$
3
64 colours
$\hat{y}$=
1
2
2
2
2
1
1
1
1
cluster for each pixel
1 1-bit number which refers to one of 2 colours.

Replace cluster with mean: