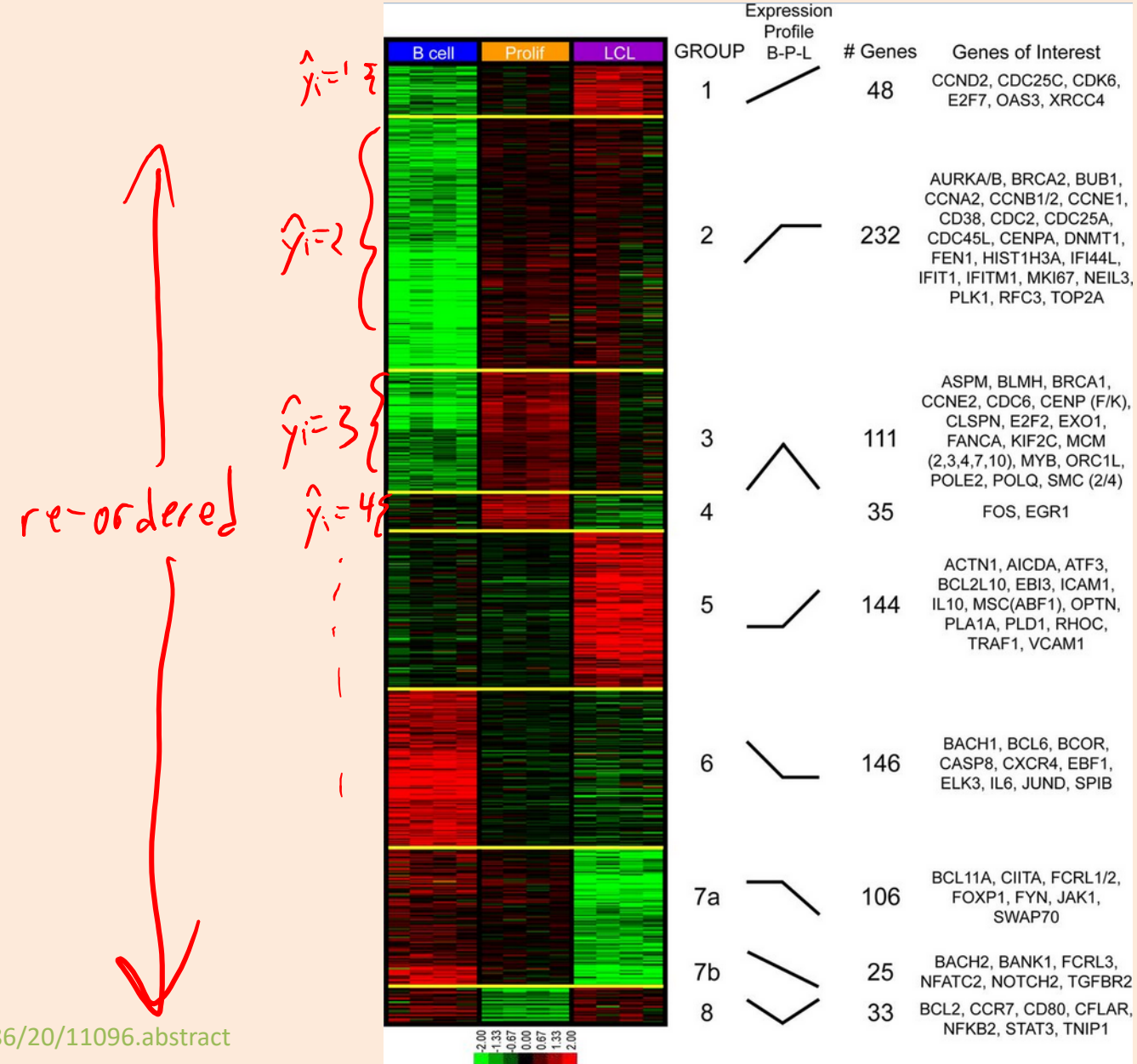
CPSC 340:
Machine Learning and Data Mining

K-Means Clustering

Bonus slides

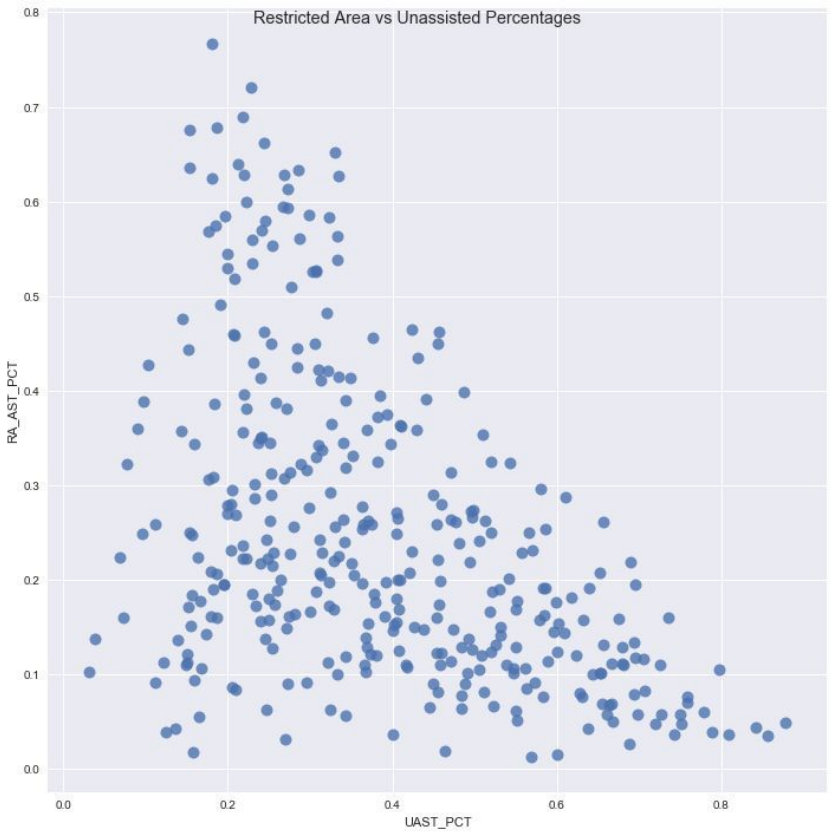
Clustering of Epstein-Barr Virus



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

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/

(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/

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:

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

- Bayesian model averaging treats model 'w_j' as a random variable:

Assume $w_j \perp x_i$

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

- 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 $w_j \perp X$

- 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' $p(y \mid w_j, X)$ 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 are are 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 are are 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:

$$f(w_{cj}) = \sum_{i \in c'} \sum_{j'=1}^{d} (w_{c'j'} - x_{ij'})^2 + (\text{constant})$$

$\hookrightarrow$ set of examples with $\hat{y}_i = c'$

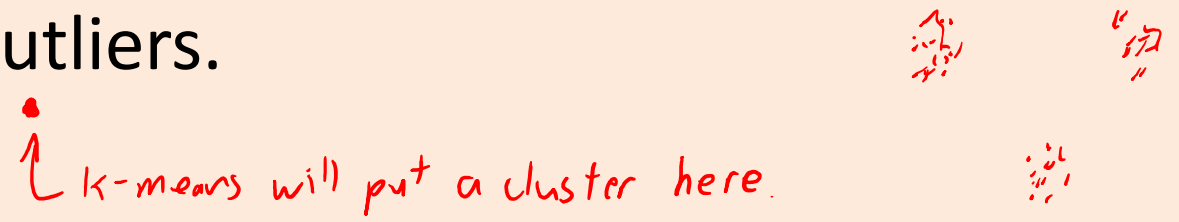
- Derivative is given by: $f'(w_{cj}) = 2 \sum_{i \in c} (w_{cj} - x_{ij})$
- Setting equal to 0 and solving for w_{cj} gives: $\sum_{i \in c} w_{cj} = \sum_{i \in c} x_{ij}$ or $w_{cj} * n_c = \sum_{i \in c} x_{ij}$

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: $\sum_{i=1}^{n} \|w_{y_i} - x_i\|_1$

- 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.



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 $(d_i)^2$.

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

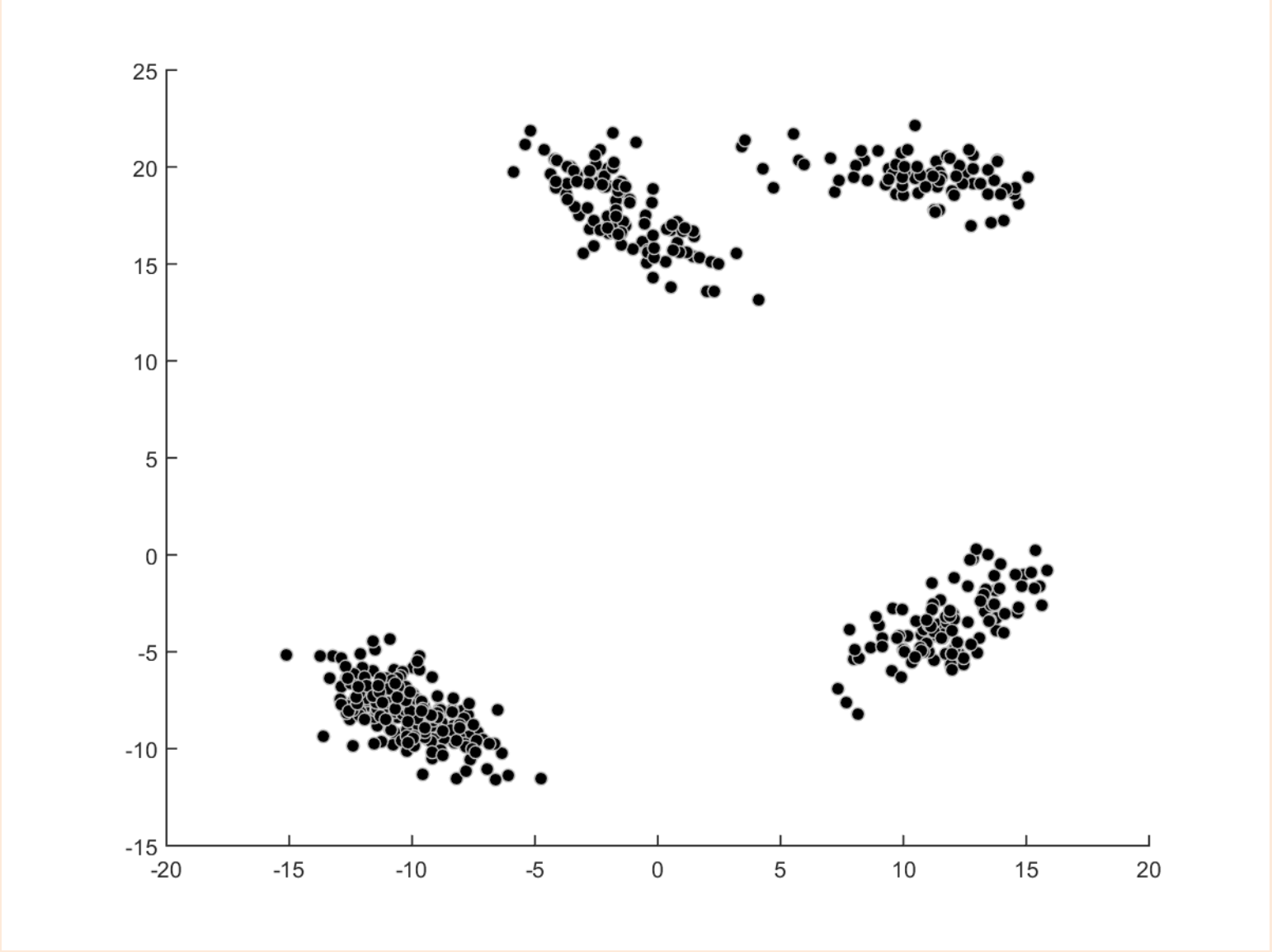
Can be done in $O(n)$

"probability that we choose x_i as next mean"

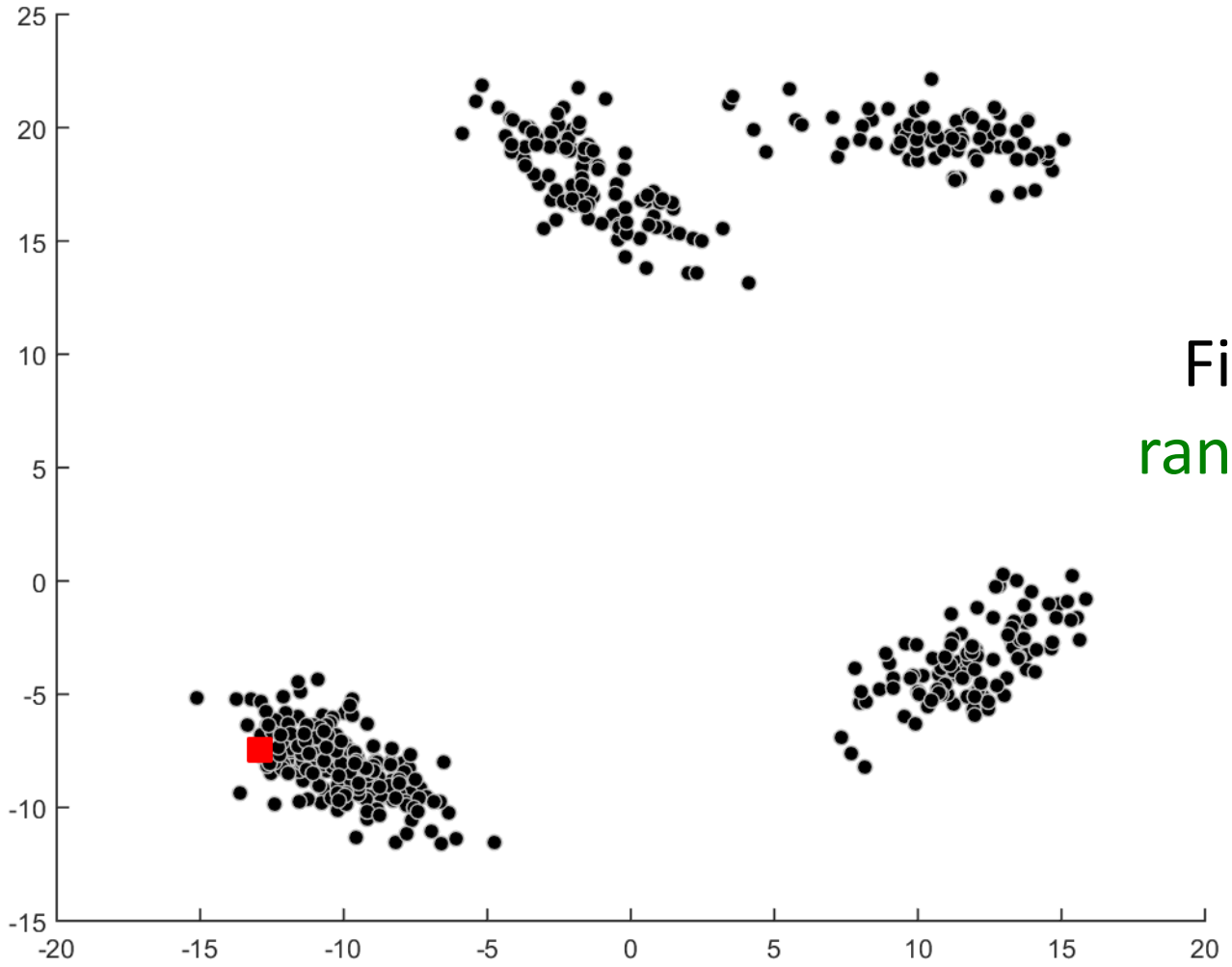
  5. Keep returning to step 2 until we have k-means.

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

K-Means++

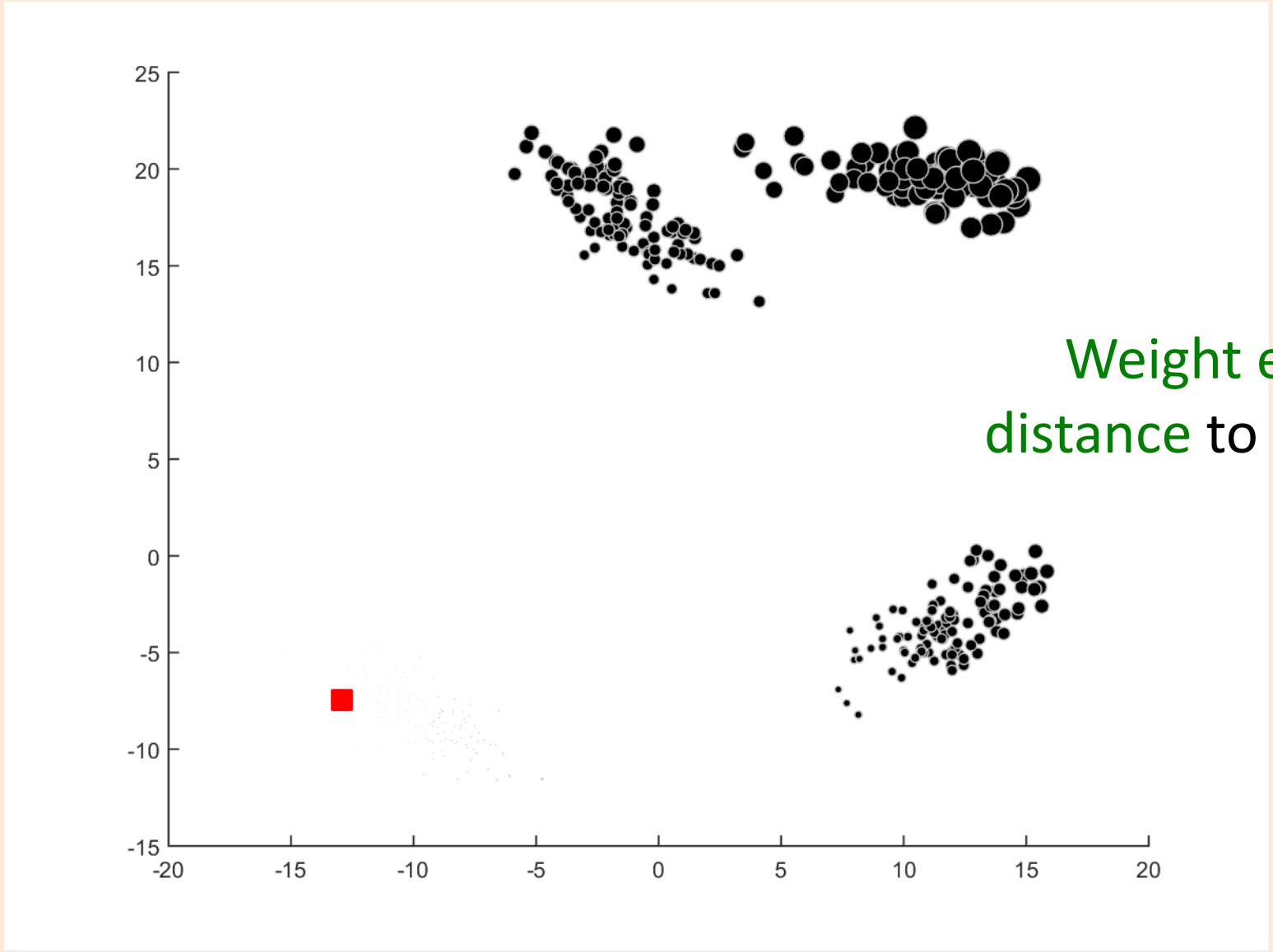


K-Means++



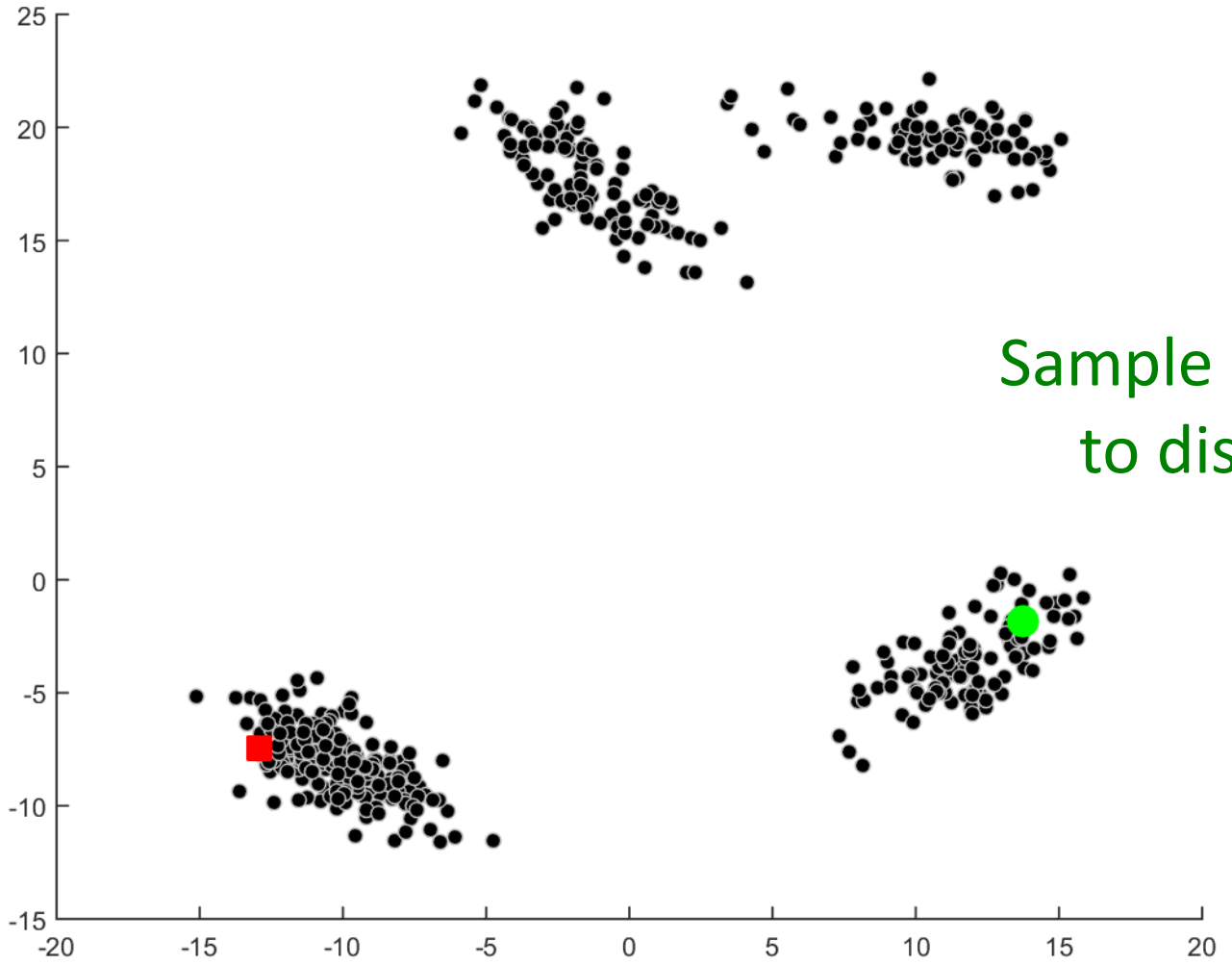
First mean is a random example.

K-Means++



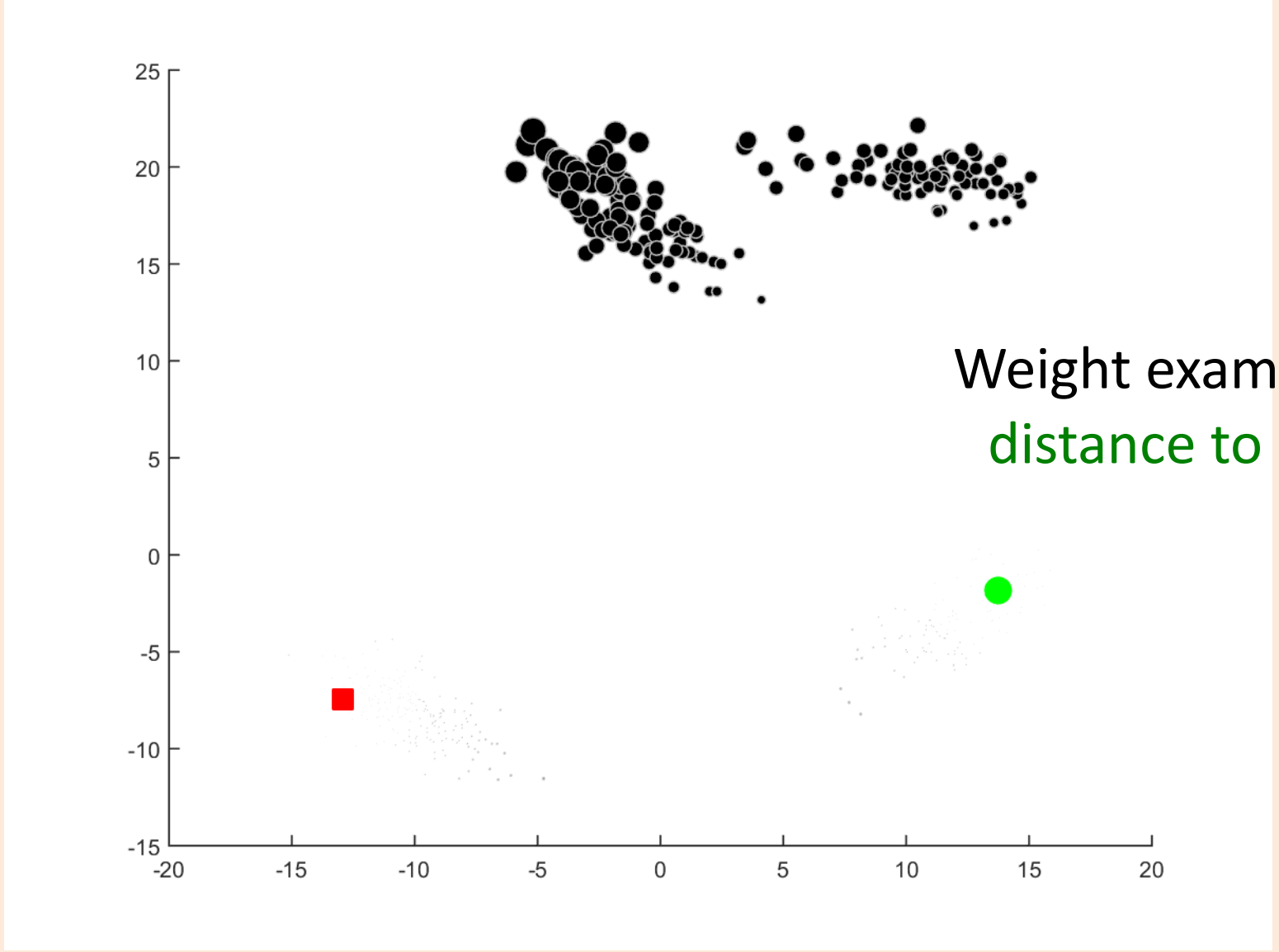
Weight examples by distance to mean squared.

K-Means++



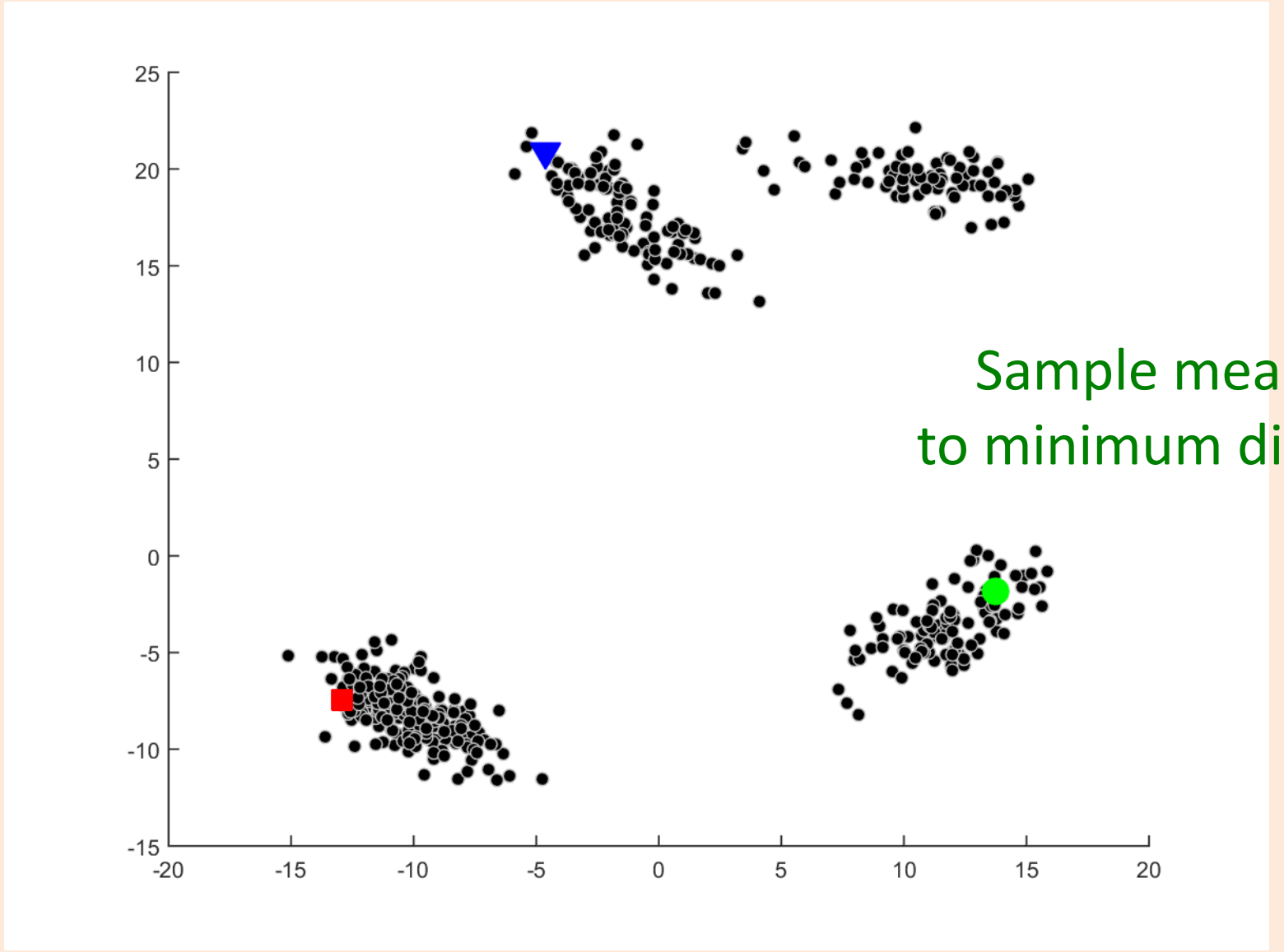
Sample mean proportional to distances squared.

K-Means++



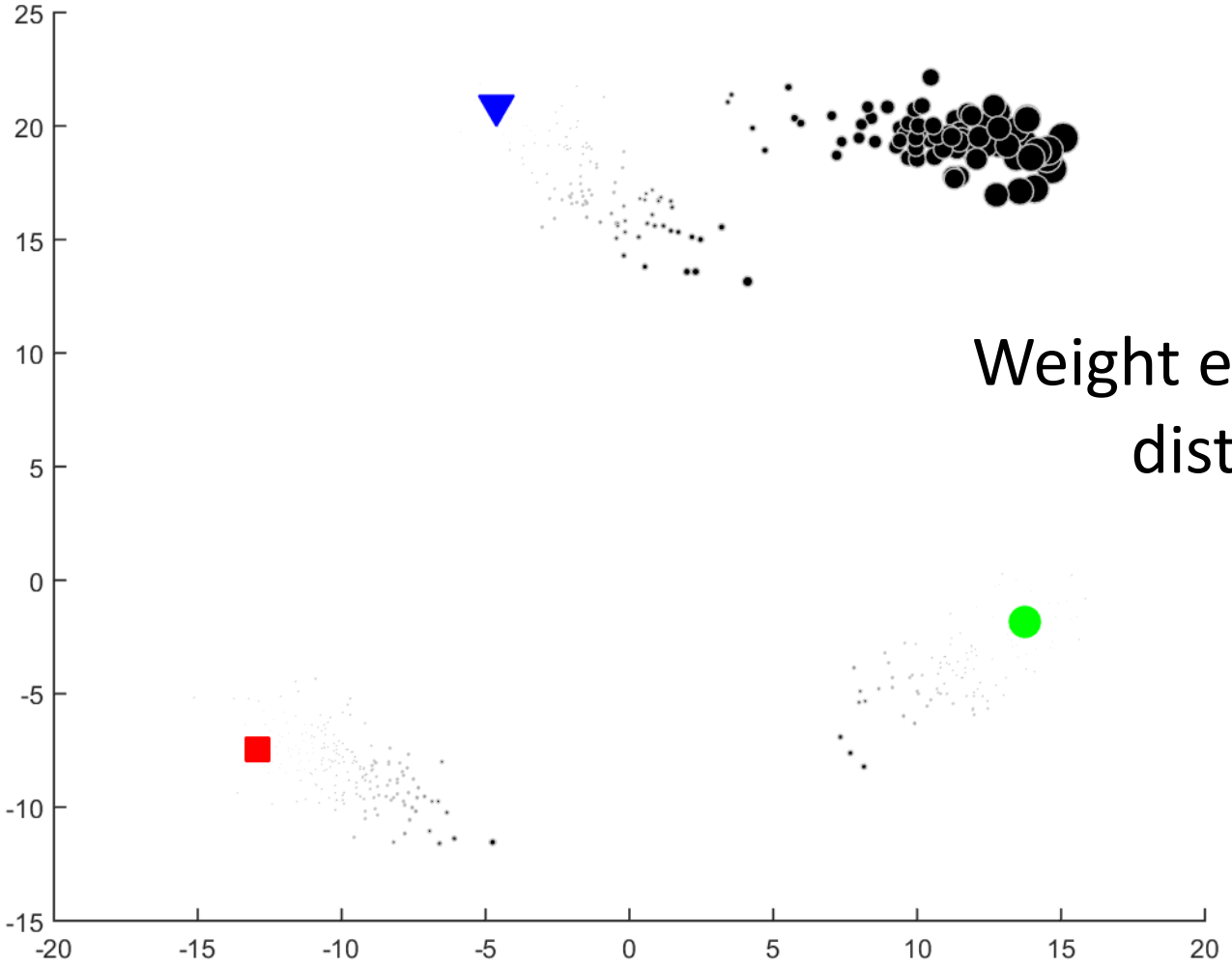
Weight examples by squared distance to nearest mean.

K-Means++



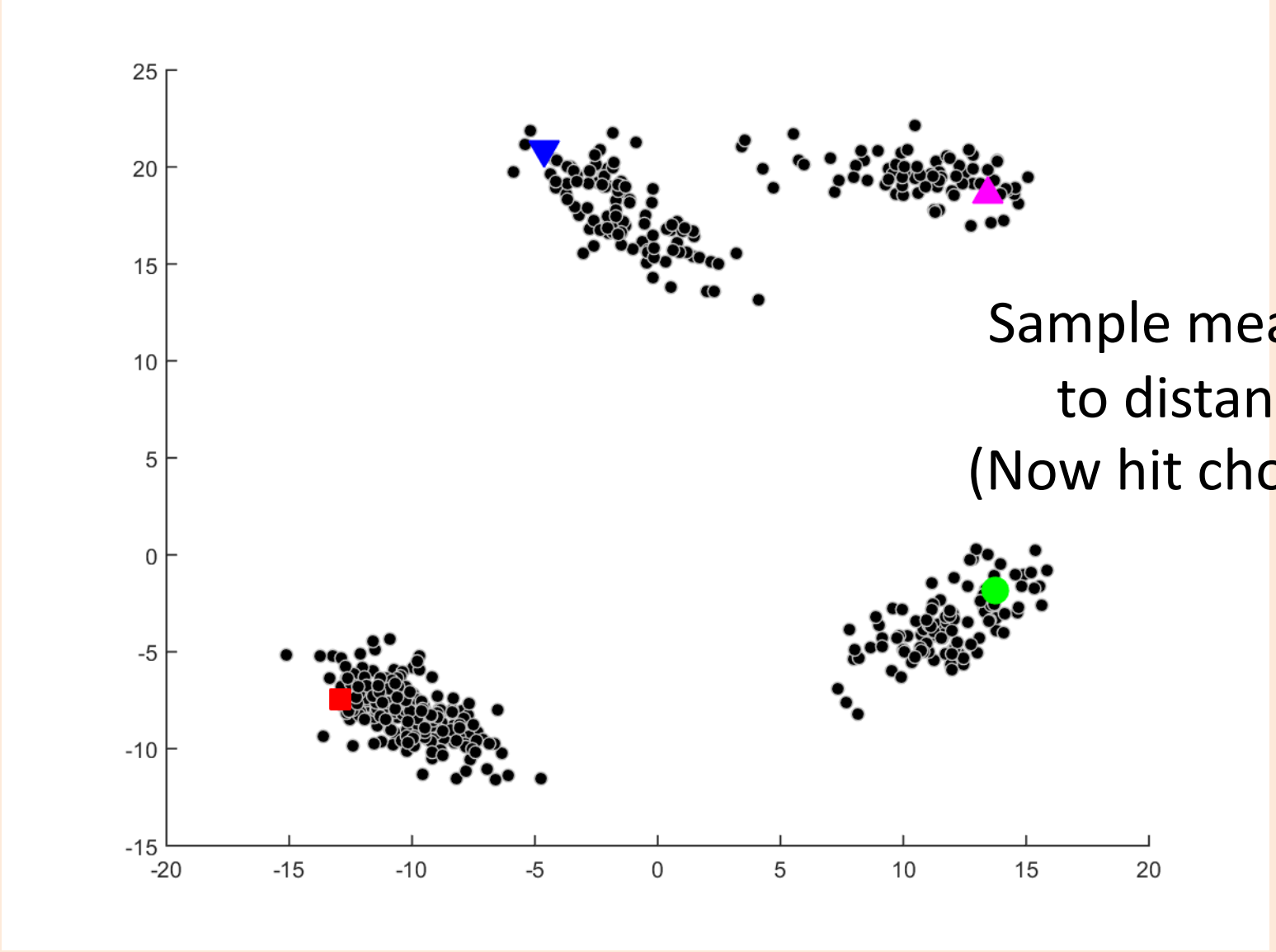
Sample mean proportional
to minimum distances squared.

K-Means++



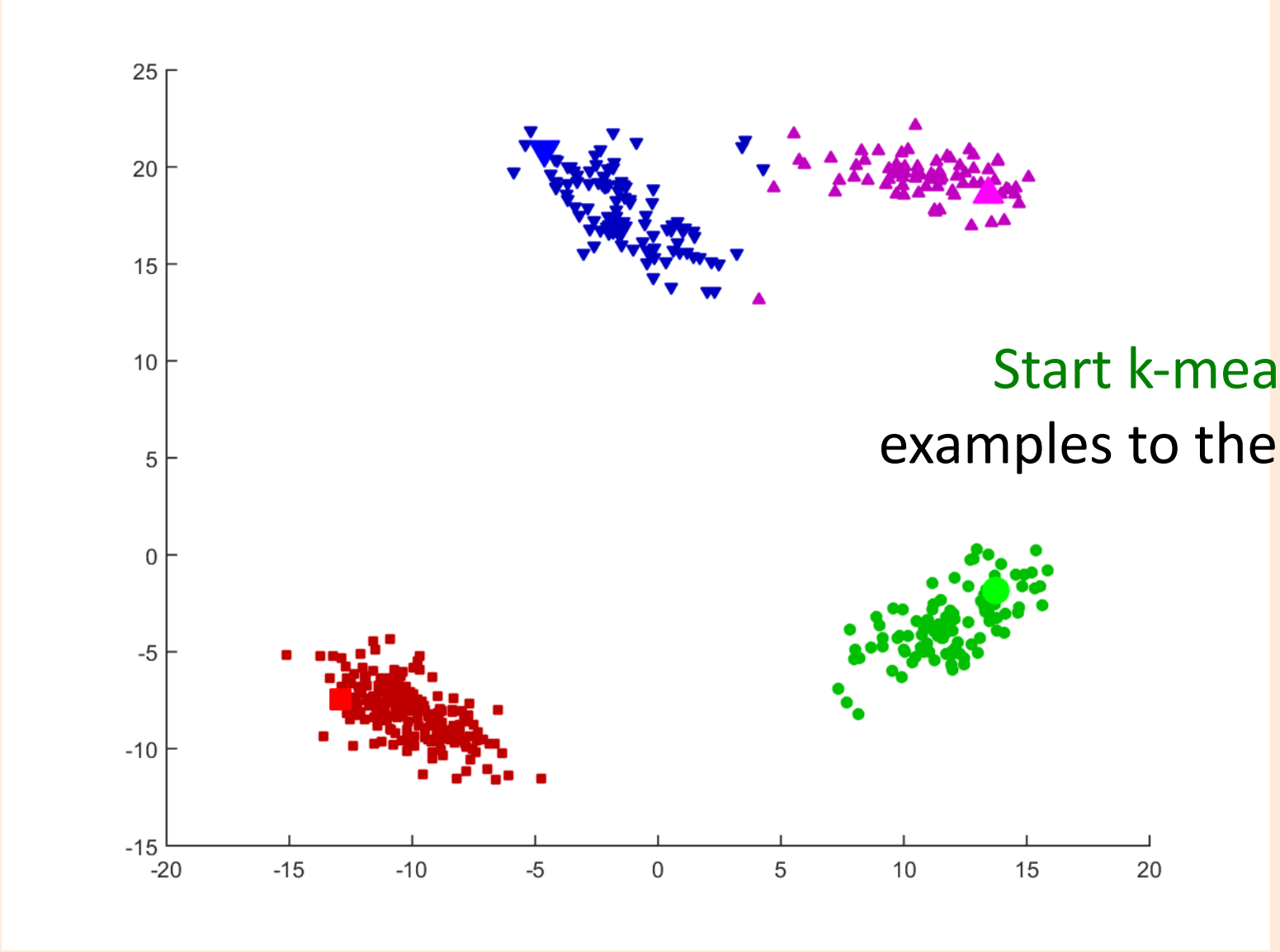
Weight examples by squared distance to mean.

K-Means++



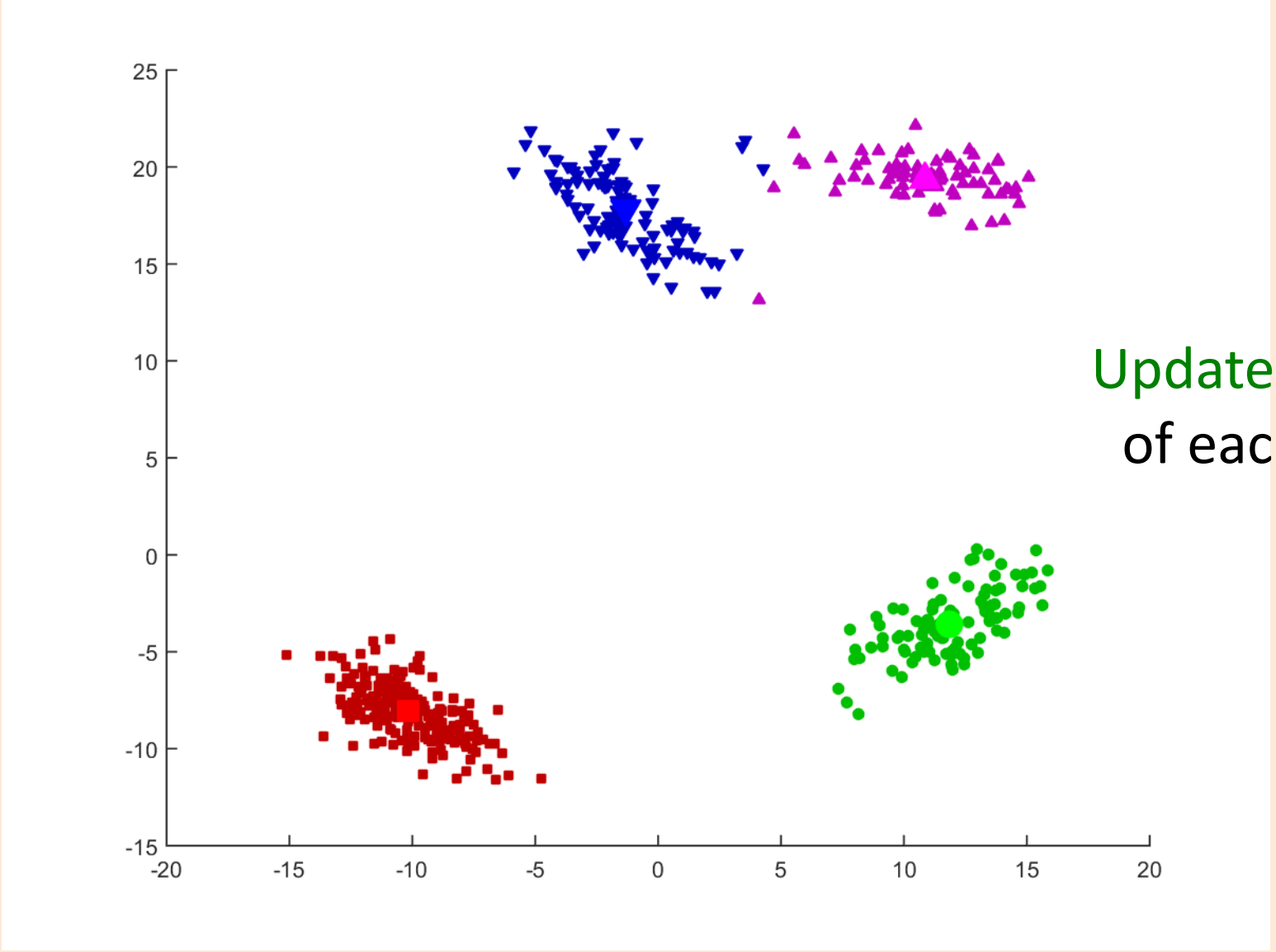
Sample mean proportional
to distances squared.
(Now hit chosen target k=4.)

K-Means++



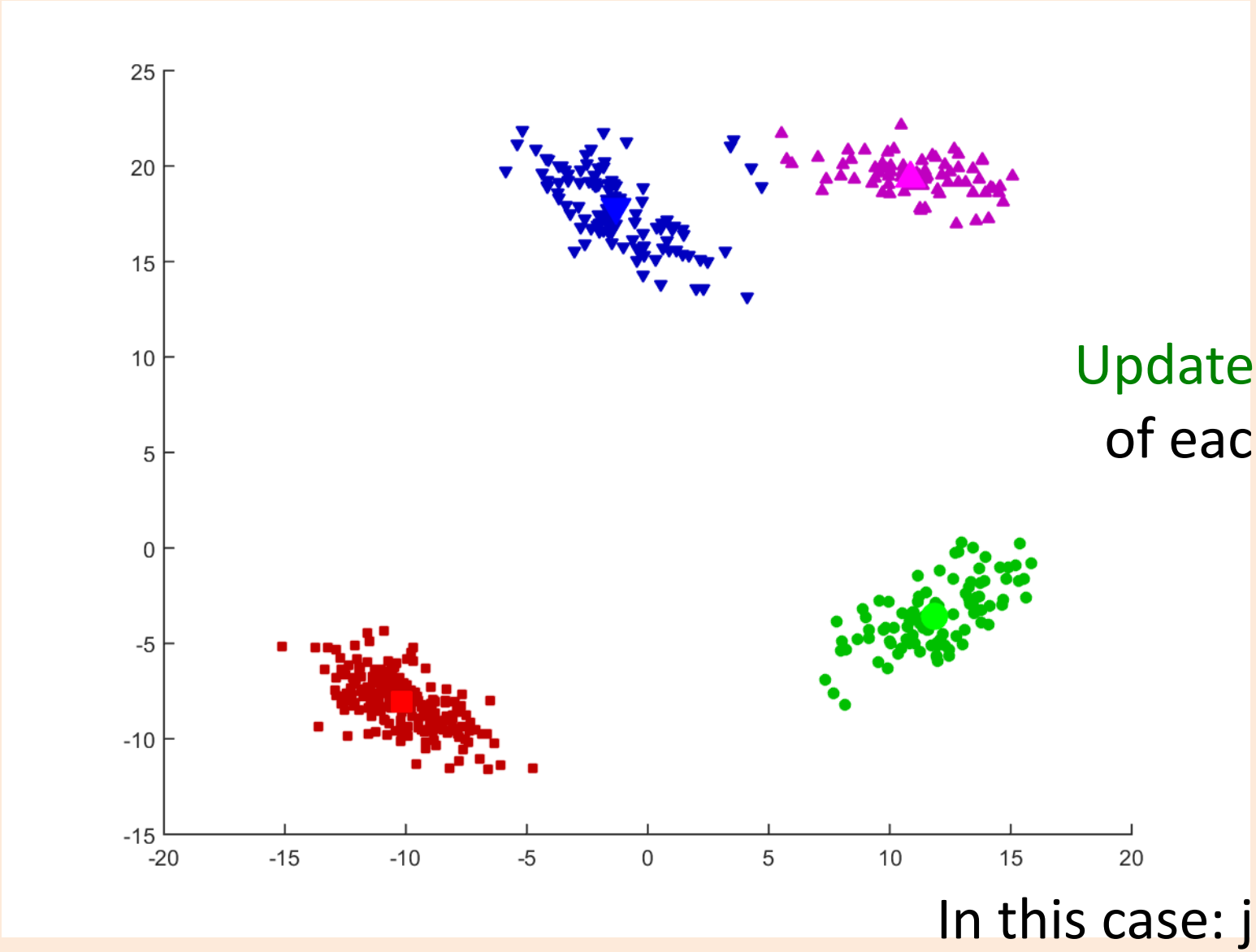
Start k-means: assign examples to the closest mean.

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:

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

all means; all assignments

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

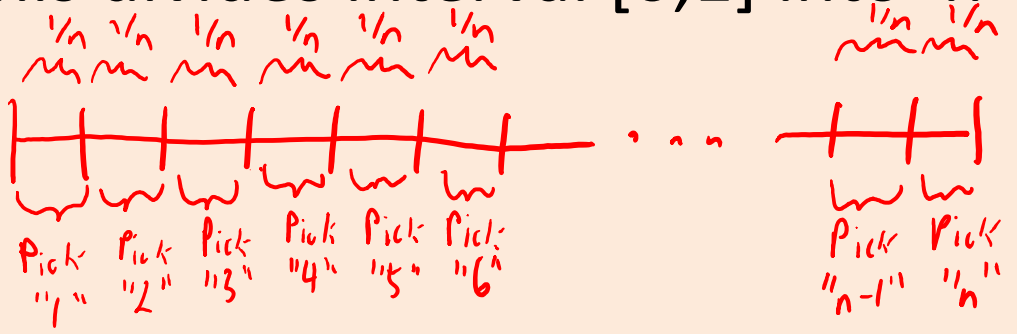
$$\frac{E[f(W, c)]}{f(W^*, c^*)} \leq O(\log(k))$$

expectation over random samples; "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.

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 ≤ i/n.
- Conceptually, this divides interval [0,1] into ‘n’ equal-size pieces:

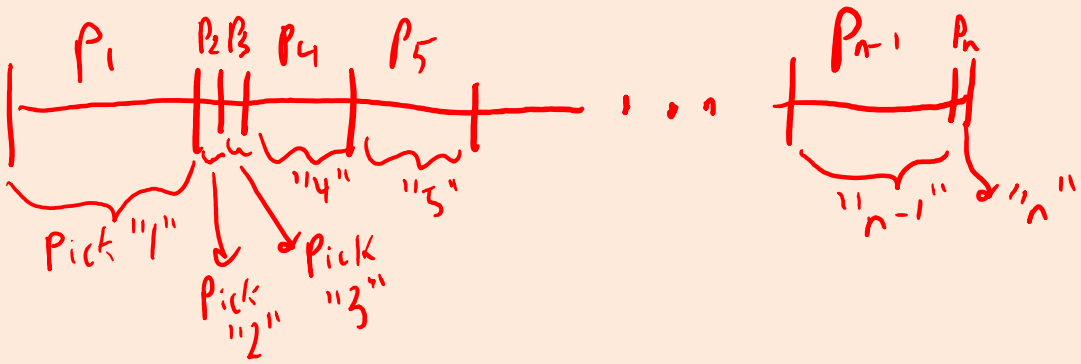


- This assumes $p_i = 1/n$ for all ‘i’.

↑ 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 $u \leq \sum_{j=1}^{i} p_i$
- 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.

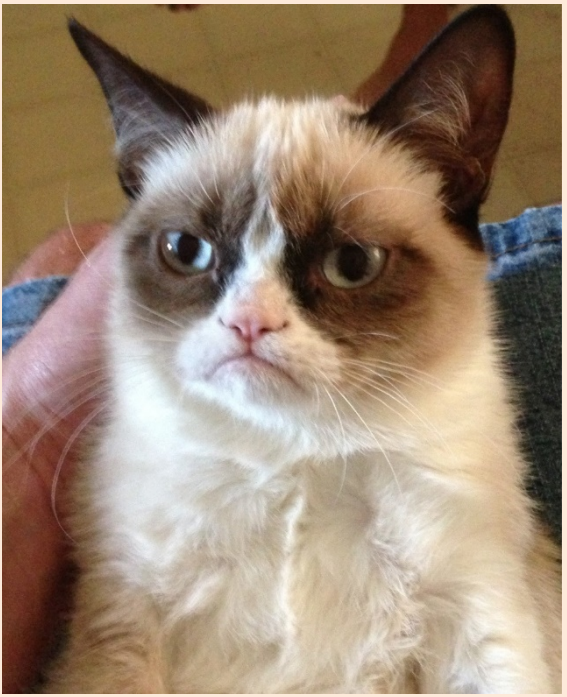
How many iterations does k-means take?

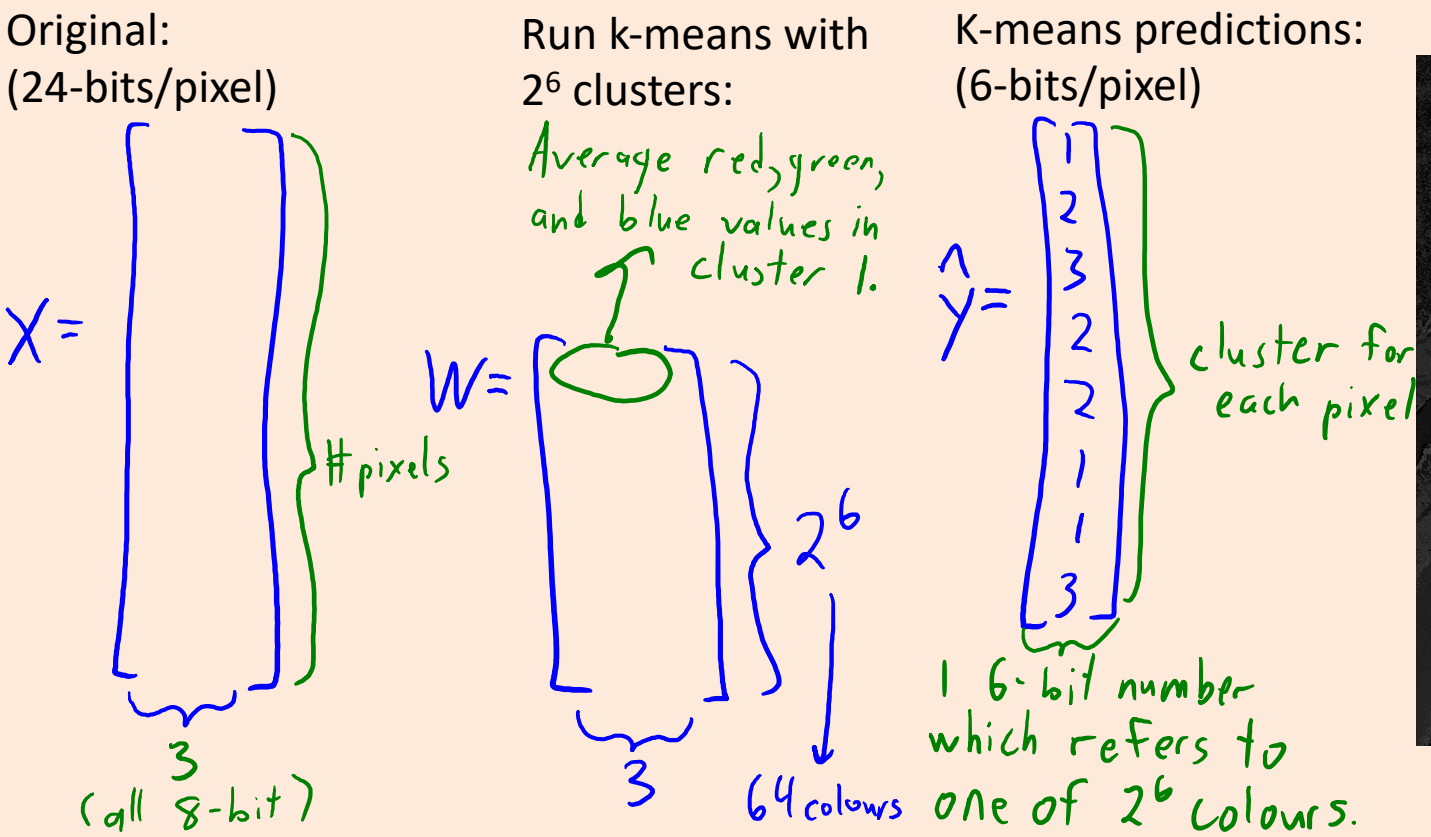
- Each update of the '$\hat{y}_i$' or 'w_c' does not increase the objective 'f'.
- And there are k^n possible assignments of the $\hat{y}_i$ to 'k' clusters.
- So within k^n iterations you cannot improve the objective by changing $\hat{y}_i$, and the algorithm stops.

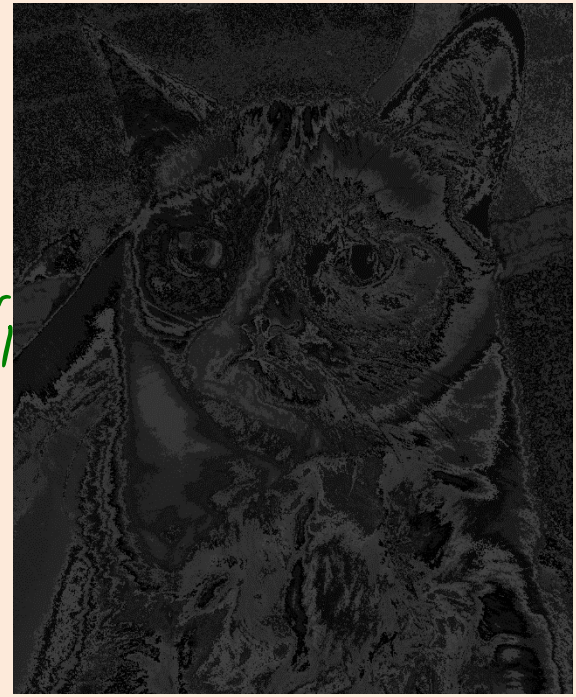
- Tighter-but-more-complicated "smoothed" analysis:
  - https://arxiv.org/pdf/0904.1113.pdf

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.

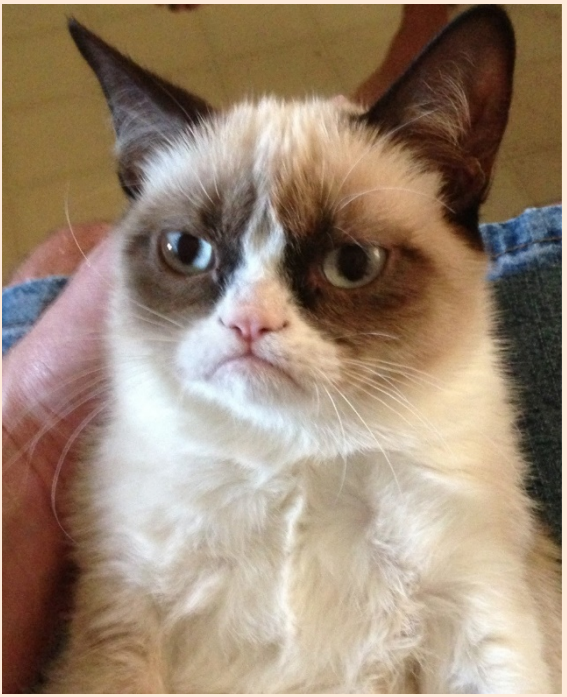




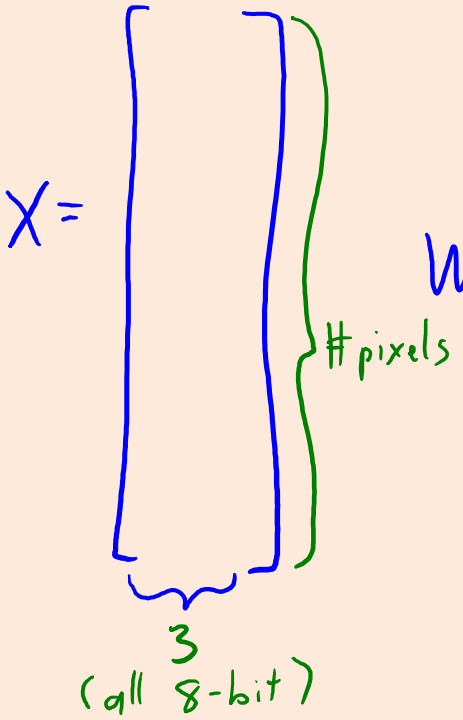


Vector Quantization: Image Colors

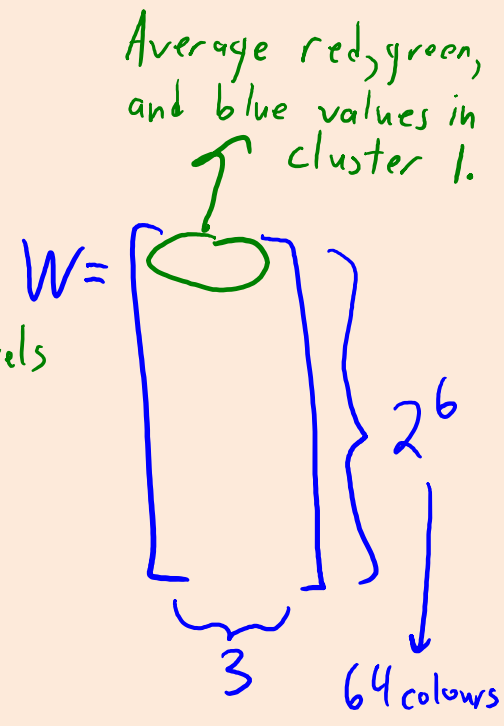
- Usual RGB representation of a pixel's color: three 8-bit numbers.
  - For example, [241 13 50] = .
  - Can apply k-means to find set of prototype colours.



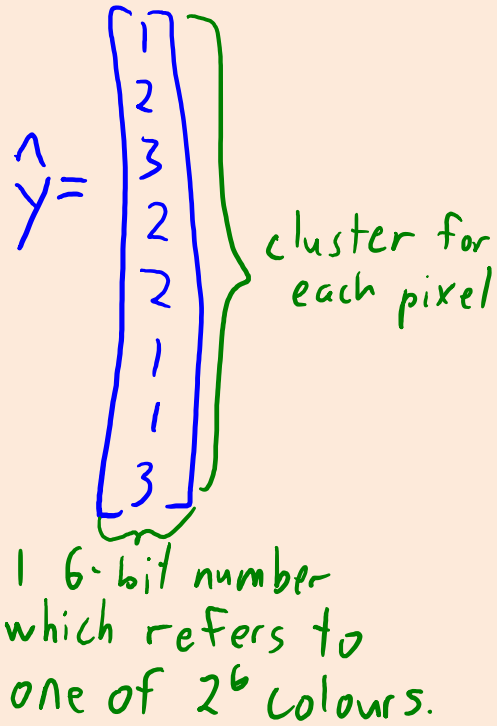
Original:
(24-bits/pixel)



Run k-means with
2^6 clusters:



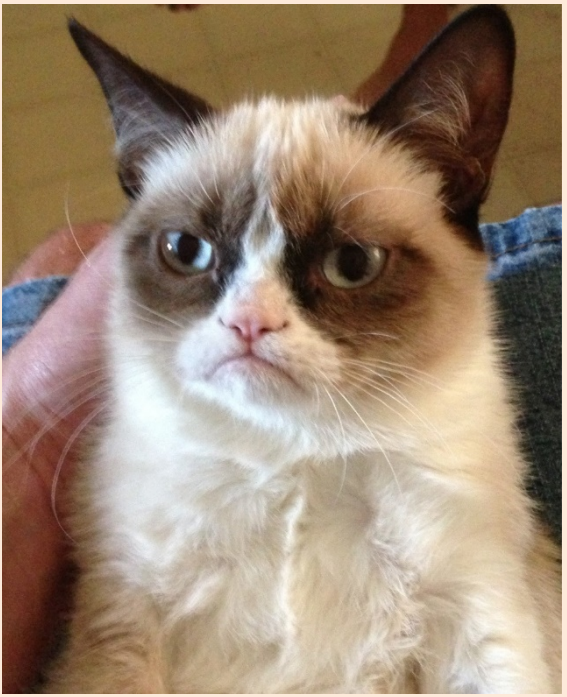
K-means predictions:
(6-bits/pixel)



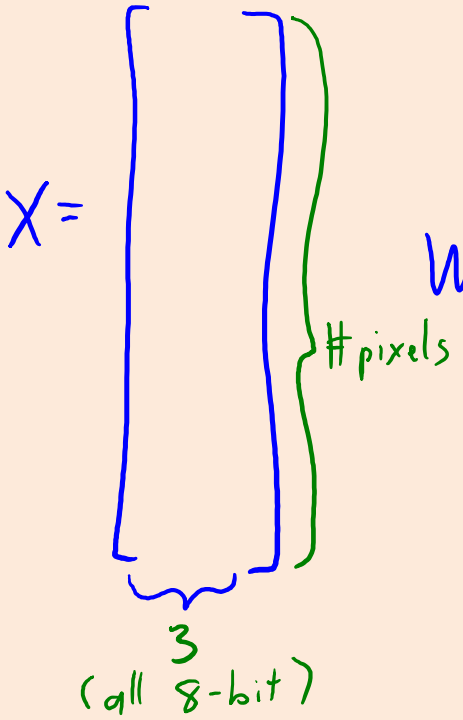
Replace cluster with mean:

Vector Quantization: Image Colors

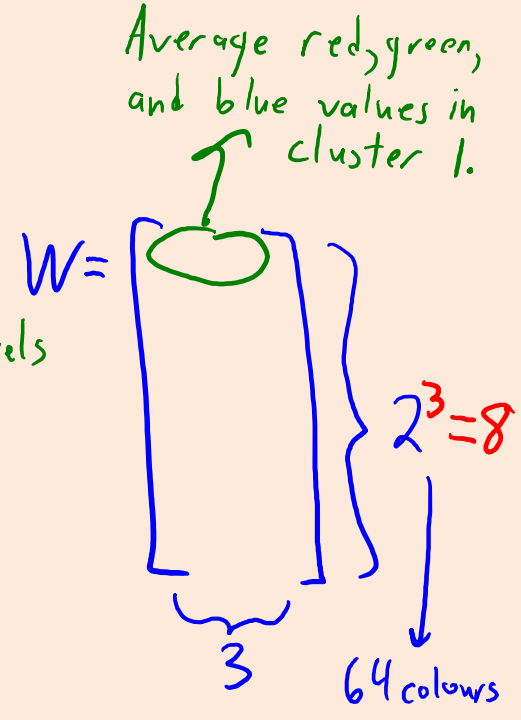
- Usual RGB representation of a pixel's color: three 8-bit numbers.
  - For example, [241 13 50] = .
  - Can apply k-means to find set of prototype colours.



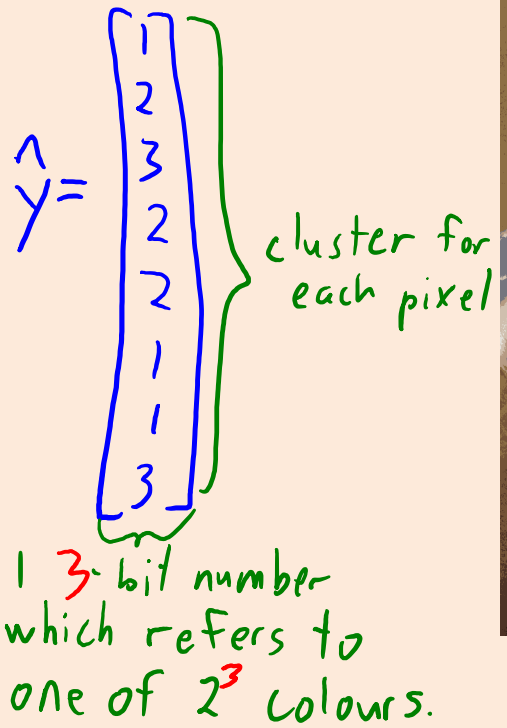
Original:
(24-bits/pixel)



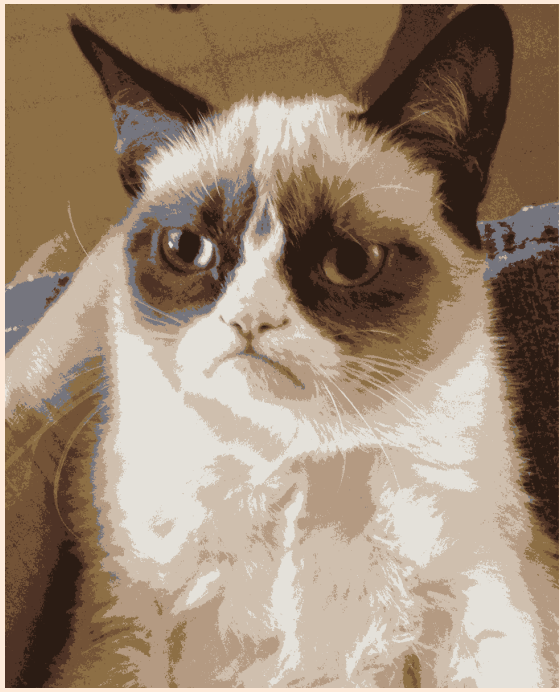
Run k-means with
2^6 clusters:



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

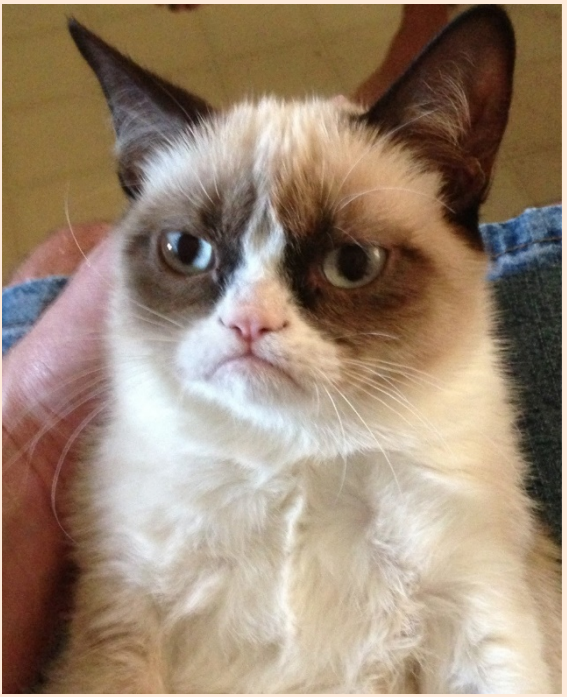


Replace cluster with mean:

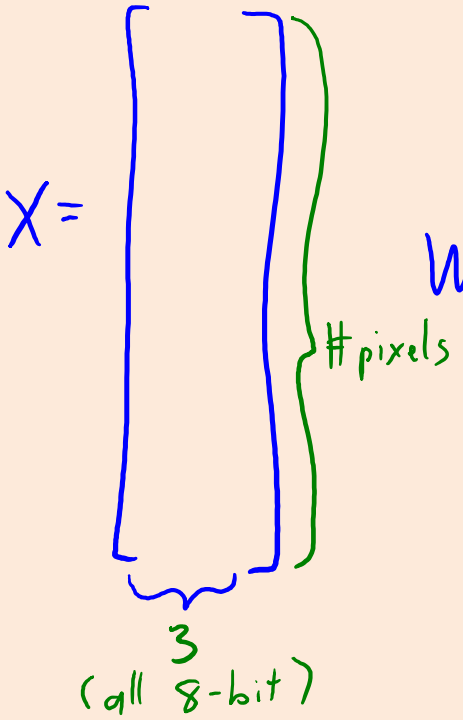


Vector Quantization: Image Colors

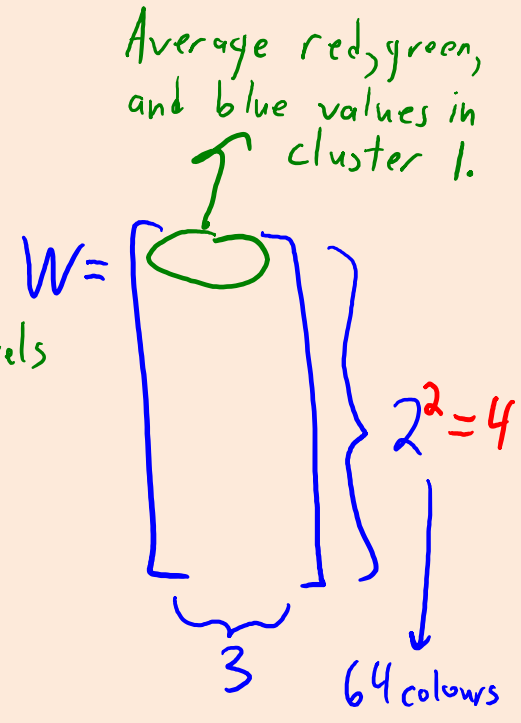
- Usual RGB representation of a pixel's color: three 8-bit numbers.
  - For example, [241 13 50] = .
  - Can apply k-means to find set of prototype colours.



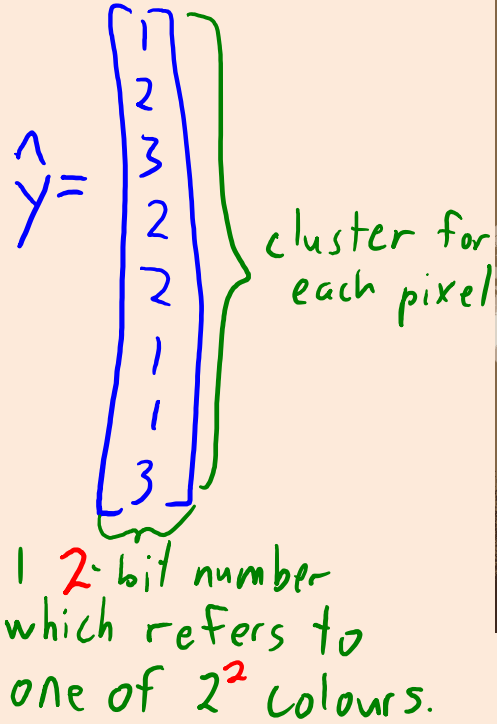
Original:
(24-bits/pixel)



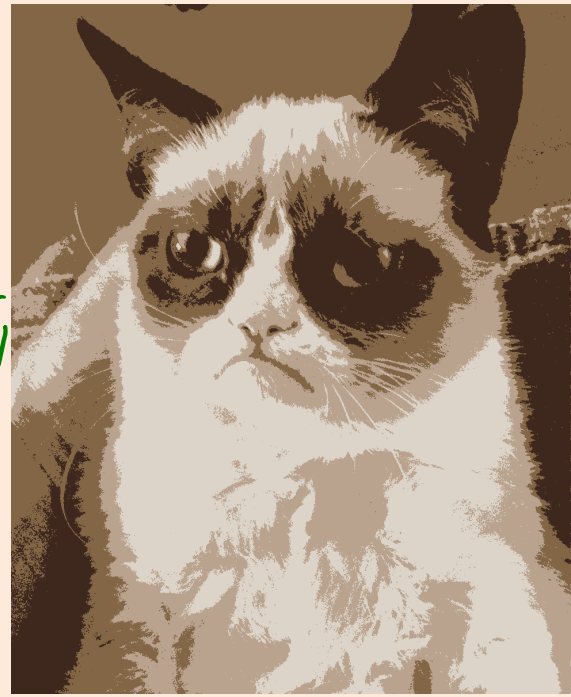
Run k-means with
2^6 clusters:



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

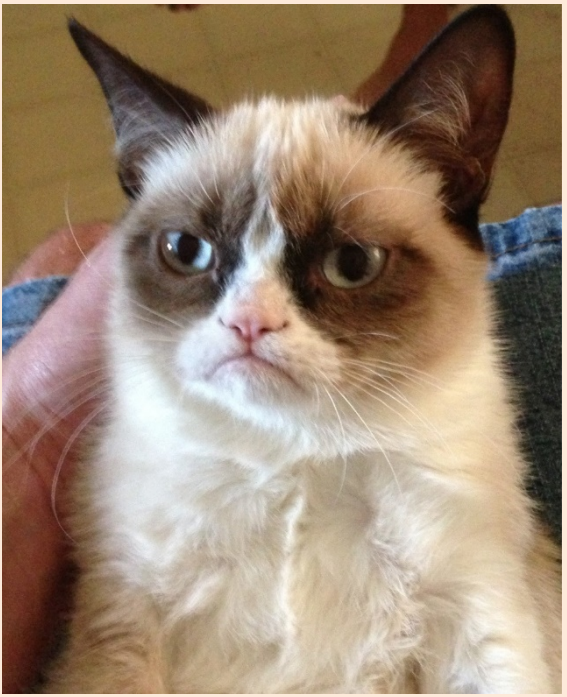


Replace cluster with mean:

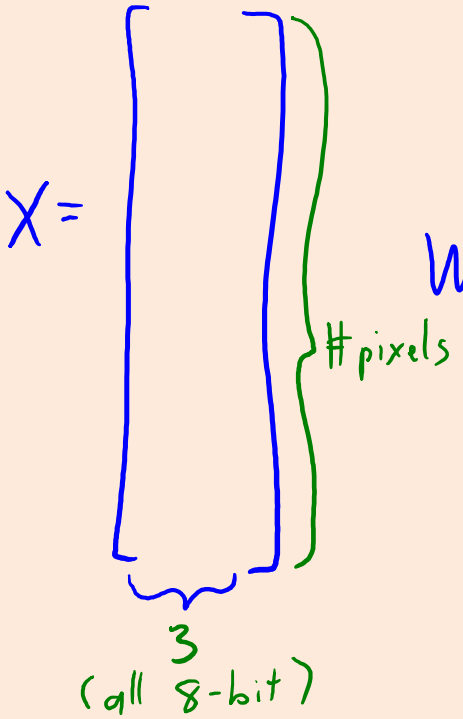


Vector Quantization: Image Colors

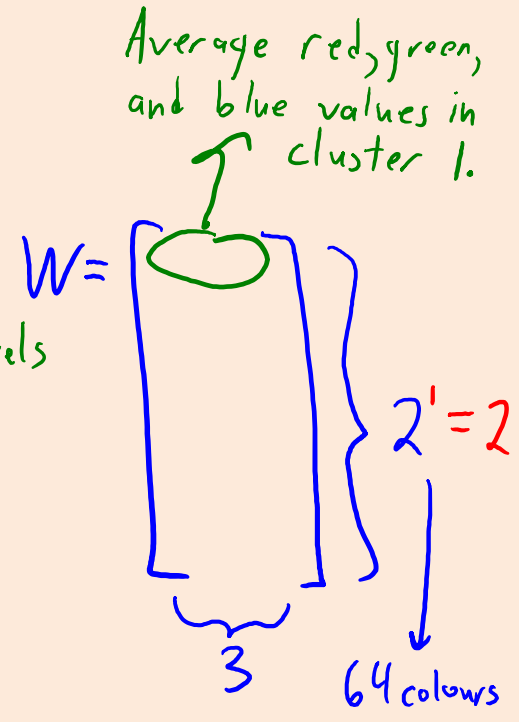
- Usual RGB representation of a pixel's color: three 8-bit numbers.
  - For example, [241 13 50] = .
  - Can apply k-means to find set of prototype colours.



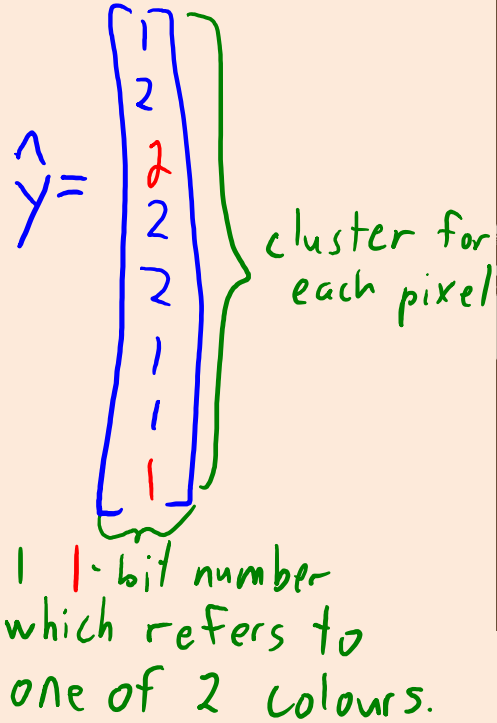
Original:
(24-bits/pixel)



Run k-means with
2^6 clusters:



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



Replace cluster with mean: