Machine Learning Department
School of Computer Science
Carnegie Mellon University

Ensemble Methods: Bagging

+

K-Means

Matt Gormley & Henry Chai
Lecture 24
Nov. 20, 2024

Reminders

- **Homework 8: Deep RL**
  - **Out: Sun, Nov. 17**
  - **Due: Mon, Nov. 25 at 11:59pm**
- **Homework 9: Learning Paradigms**
  - **Out: Mon, Nov. 25**
  - **Due: Thu, Dec. 5 at 11:59pm (only two grace/late days permitted)**

ADABOOST

Comparison

Weighted Majority Algorithm

- an example of an ensemble method
- assumes the classifiers are learned ahead of time
- only learns (majority vote) weight for each classifiers

AdaBoost

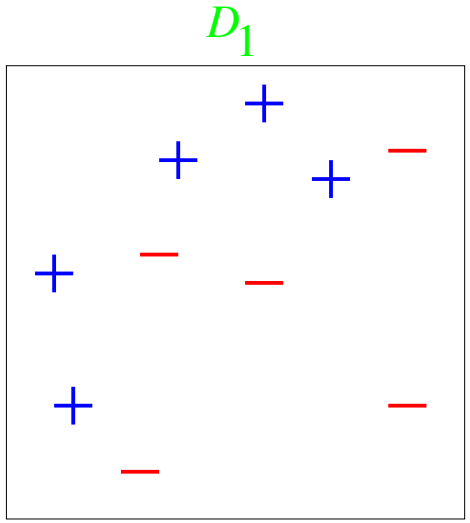
- an example of a boosting method
- simultaneously learns:
  - the classifiers themselves
  - (majority vote) weight for each classifiers

AdaBoost

- Definitions
  - Def: a **weak learner** is one that returns a hypothesis that is not much better than random guessing
  - Def: a **strong learner** is one that returns a hypothesis of arbitrarily low error

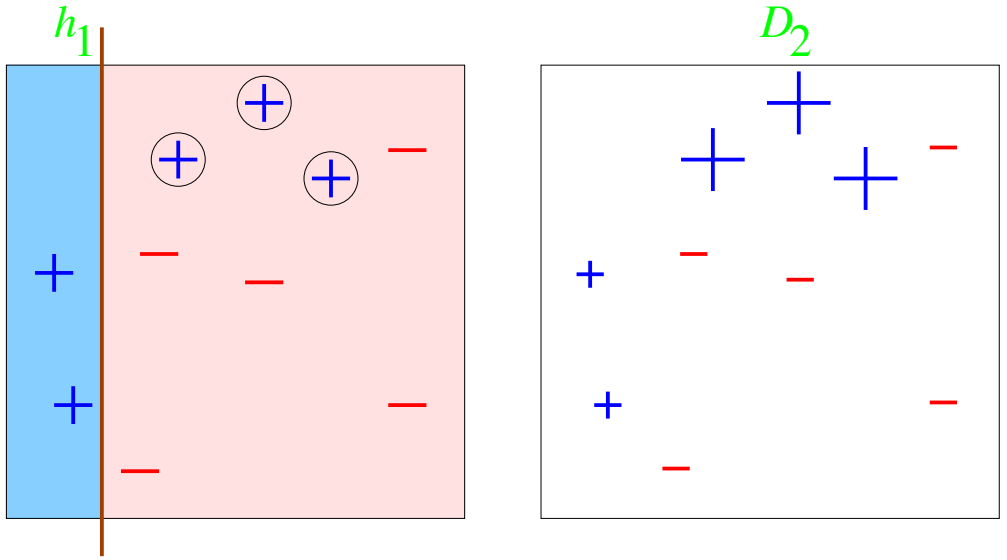
- AdaBoost answers the following question:
  - *Does that exist an efficient learning algorithm that can combine weak learners to obtain a strong learner?*

AdaBoost: Toy Example



weak classifiers = vertical or horizontal half-planes

AdaBoost: Toy Example

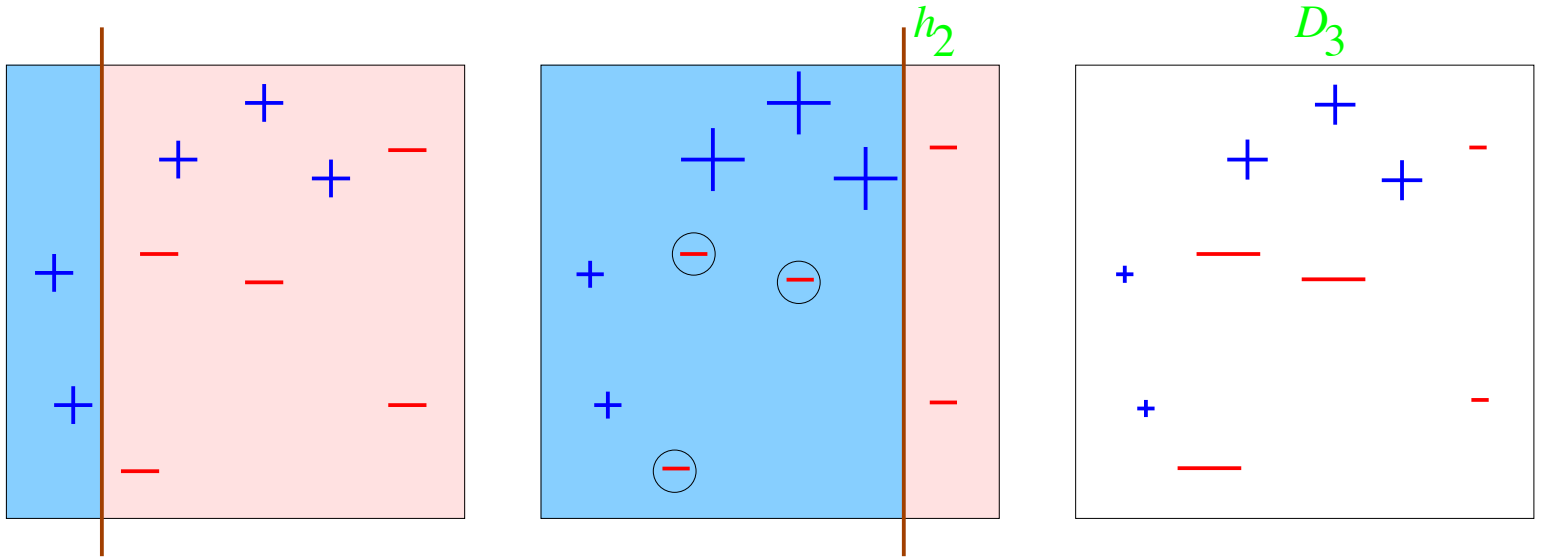


$\varepsilon_1 = 0.30$

$\alpha_1 = 0.42$

Slide from Schapire NeurIPS Tutorial

AdaBoost: Toy Example

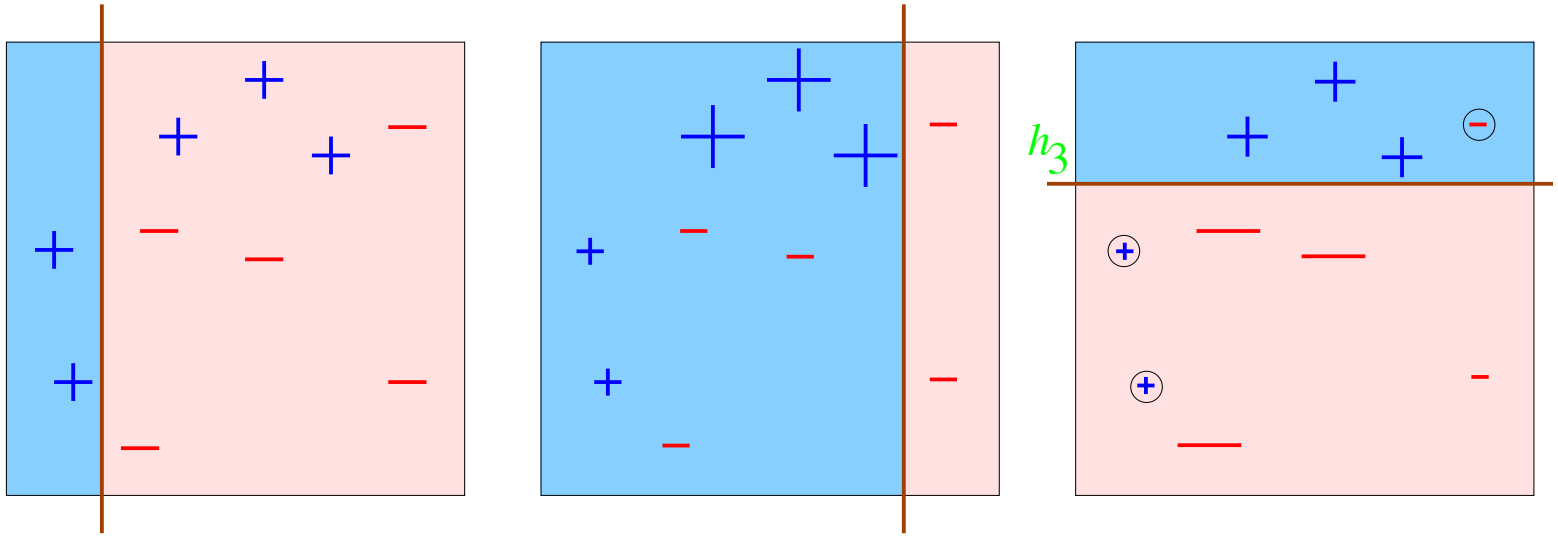


$\varepsilon_2 = 0.21$

$\alpha_2 = 0.65$

Slide from Schapire NeurIPS Tutorial

AdaBoost: Toy Example

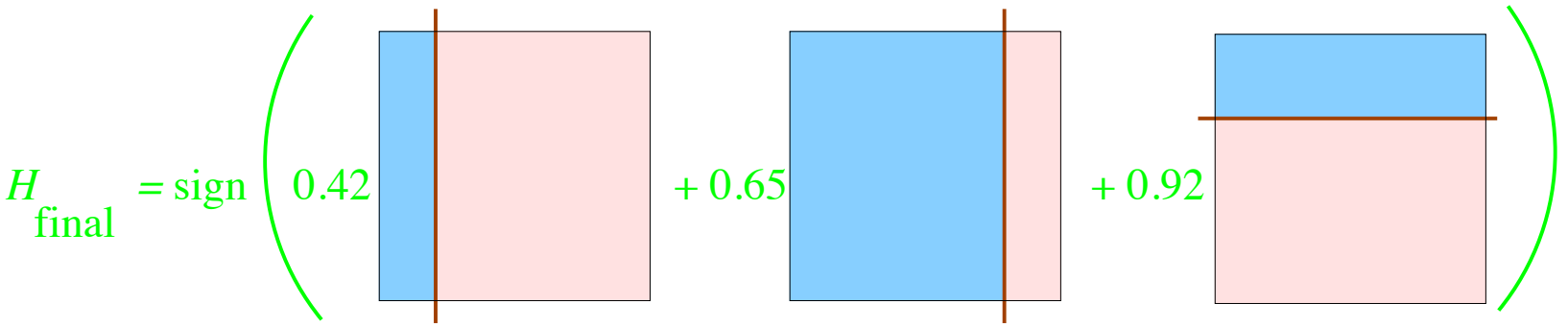


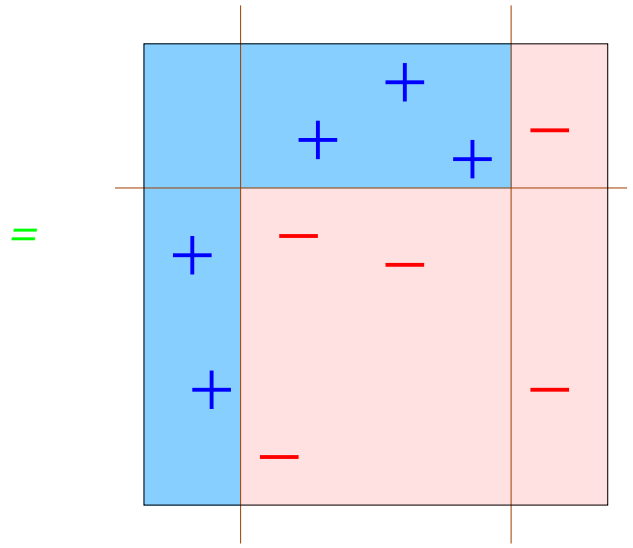
$\varepsilon_3 = 0.14$

$\alpha_3 = 0.92$

Slide from Schapire NeurIPS Tutorial

AdaBoost: Toy Example





Slide from Schapire NeurIPS Tutorial

AdaBoost

Algorithm 1 AdaBoost Algorithm

1: Given: $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$ where $\mathbf{x}_i \in \mathbb{R}^M, y_i \in \{-1, +1\}$

2: Initialize $D_1(i) = \frac{1}{N}$

3: **for** $t = 1, \ldots, T$ **do**

4: Train weak learner using distribution D_t.

5: Get weak hypothesis $h_t : \mathbb{R}^M \to \{-1, +1\}$ with error

$$\epsilon_t = P_{i \sim D_t}[h_t(\mathbf{x}_i) \neq y_i]$$

6: Choose $\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$.

for high error, we get _____ α_t

for low error, we get _____ α_t

7: **for** $i = 1, \ldots, N$ **do**

8: Update:

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } h_t(\mathbf{x}_i) = y_i \\ e^{\alpha_t} & \text{if } h_t(\mathbf{x}_i) \neq y_i \end{cases}$$

$$= \frac{D_t(i) \exp(-\alpha_t y_i h_t(\mathbf{x}_i))}{Z_t}$$

if correct, _____

if incorrect, _____

↳ for high error, _____

↳ for low error, _____

where normalization const. Z_t chosen s.t. D_{t+1} is a distribution.

9: Output the final hypothesis: $H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}) \right)$.

Algorithm adapted from (Freund & Schapire, 1999)

AdaBoost: Theory

(Training) Mistake Bound

The most basic theoretical property of AdaBoost concerns its ability to reduce the training error. Let us write the error ϵ_t of h_t as $\frac{1}{2} - \gamma_t$. Since a hypothesis that guesses each instance's class at random has an error rate of $1/2$ (on binary problems), γ_t thus measures how much better than random are h_t's predictions. Freund and Schapire [23] prove that the training error (the fraction of mistakes on the training set) of the final hypothesis H is at most

$$\prod_t \left[2\sqrt{\epsilon_t(1-\epsilon_t)}\right] = \prod_t \sqrt{1-4\gamma_t^2} \leq \exp\left(-2\sum_t \gamma_t^2\right). \quad (1)$$

Thus, if each weak hypothesis is slightly better than random so that $\gamma_t \geq \gamma$ for some $\gamma > 0$, then the training error drops exponentially fast.

Figure from (Freund & Schapire, 1999)

AdaBoost: Theory

Generalization Error

Freund and Schapire [23] showed how to bound the generalization error of the final hypothesis in terms of its training error, the sample size N, the VC-dimension d of the weak hypothesis space and the number of boosting rounds T. (The VC-dimension is a standard measure of the "complexity" of a space of hypotheses. See, for instance, Blumer et al. [5].) Specifically, they used techniques from Baum and Haussler [4] to show that the generalization error, with high probability, is at most

$$\hat{\Pr}\left[H(x) \neq y\right] + \tilde{O}\left(\sqrt{\frac{Td}{N}}\right)$$

where $\hat{\Pr}\left[\cdot\right]$ denotes empirical probability on the training sample. This bound suggests that boosting will overfit if run for too many rounds, i.e., as T becomes large. In fact, this sometimes does happen. However, in early experiments, several authors [9, 15, 36] observed empirically that boosting often does *not* overfit, even when run for thousands of rounds. Moreover, it was observed that AdaBoost would sometimes continue to drive down the generalization error long after the training error had reached zero, clearly contradicting the spirit of the bound above. For instance, the left

Figure from (Freund & Schapire, 1999)

AdaBoost

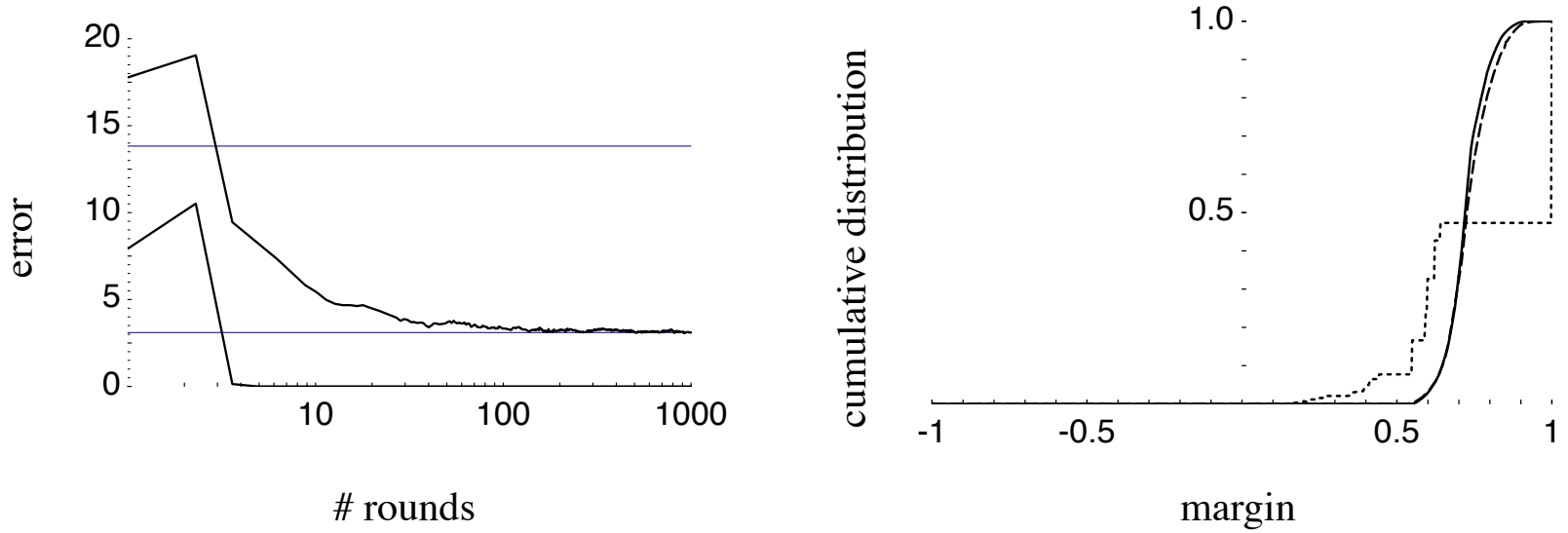


Figure 2: Error curves and the margin distribution graph for boosting C4.5 on the letter dataset as reported by Schapire et al. [41]. *Left*: the training and test error curves (lower and upper curves, respectively) of the combined classifier as a function of the number of rounds of boosting. The horizontal lines indicate the test error rate of the base classifier as well as the test error of the final combined classifier. *Right*: The cumulative distribution of margins of the training examples after 5, 100 and 1000 iterations, indicated by short-dashed, long-dashed (mostly hidden) and solid curves, respectively.

Figure from (Freund & Schapire, 1999)

Learning Objectives

Ensemble Methods: Boosting

You should be able to…

1. Explain how a weighted majority vote over linear classifiers can lead to a non-linear decision boundary
2. Implement AdaBoost
3. Describe a surprisingly common empirical result regarding Adaboost train/test curves

Ensemble Methods

Ensemble methods learn a collection of models (i.e. the **ensemble**) and combine their predictions on a test instance.

We consider two types:

- **Bagging**: learns models in parallel by taking many subsets of the training data
- **Boosting**: learns models serially by reweighting the training data

BAGGING

Bagging

"BAGGing" is also called Boostrap AGGregretion

Bagging answers the question:
How can I obtain many classifiers/regressors to ensemble together?

We'll consider three possible answers:

1. **(sample) bagging**
2. **feature bagging** (aka. random subspace method)
3. **random forests** (which combine sample bagging and feature bagging to train a "forest" of decision trees)

(Sample) Bagging

Key idea: Repeatedly sample with replacement a collection of training examples and train a model on that sample.

Return an ensemble of the trained models; combine predictions by majority vote for classification and by averaging for regression.

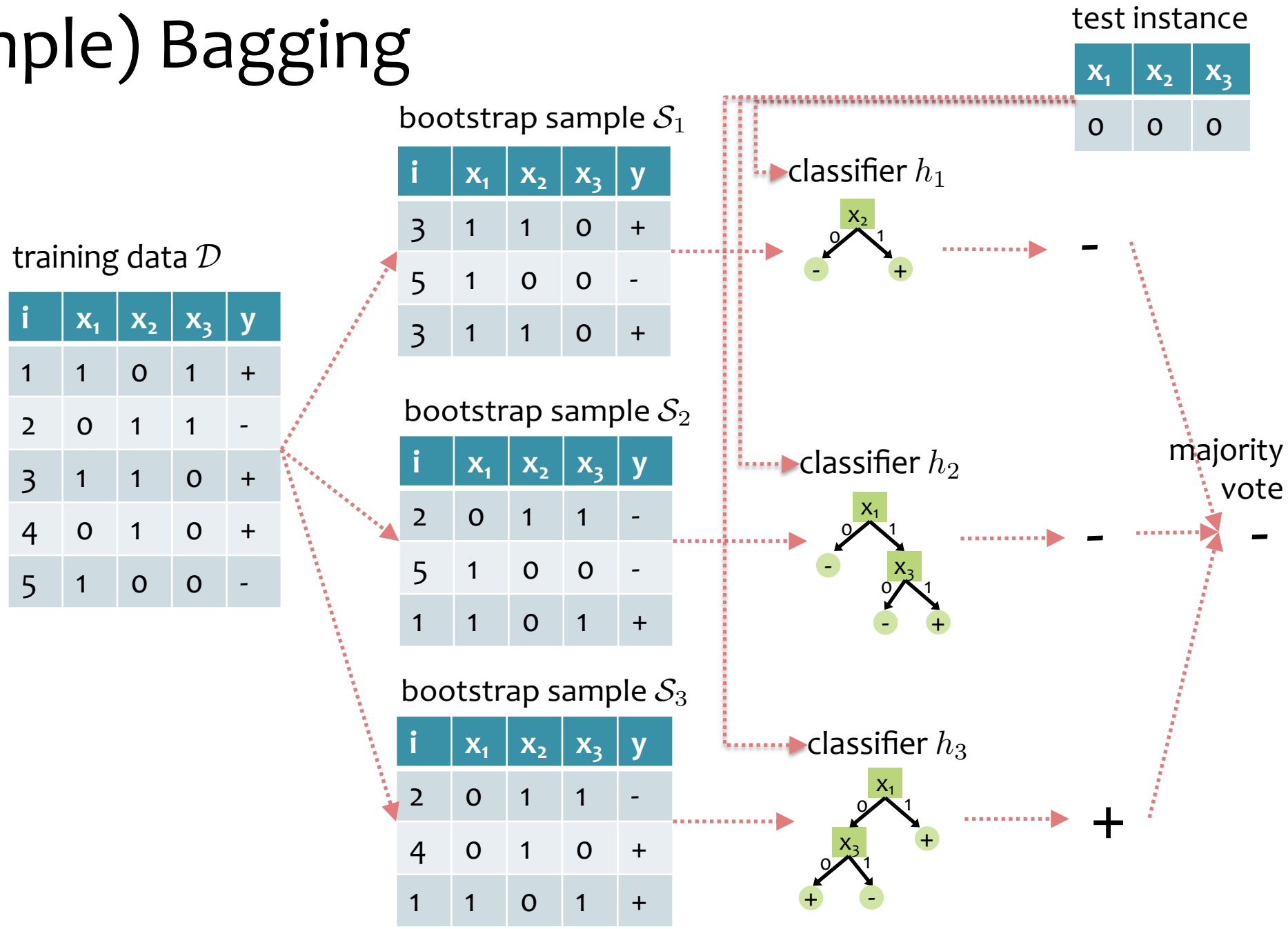
Algorithm 1 (Sample) Bagging

procedure SAMPLEBAGGING($\mathcal{D}$, T, S)
  for $t = 1, \ldots, T$ **do**
    for $s = 1, \ldots, S$ **do**
      $i_s \sim \text{Uniform}(1, \ldots, N)$
    $\mathcal{S}_t = \{(\mathbf{x}^{(i_s)}, y^{(i_s)})\}_{s=1}^{S}$ ▷ Bootstrap sample
    $h_t = \text{train}(\mathcal{S}_t)$ ▷ Classifier
  return $\hat{h}(\mathbf{x}) = \text{aggregate}(h_1, \ldots, h_T)$ ▷ Ensemble

for classification: $\hat{h}(\mathbf{x}) = \text{argmax}_{y \in \mathcal{Y}} \sum_{t=1}^{T} \mathbb{I}[y = h_t(\mathbf{x})]$ ▷ Majority vote

for regression: $\hat{h}(\mathbf{x}) = \frac{1}{T} \sum_{t=1}^{T} h_t(\mathbf{x})$ ▷ Average

(Sample) Bagging



Feature Bagging

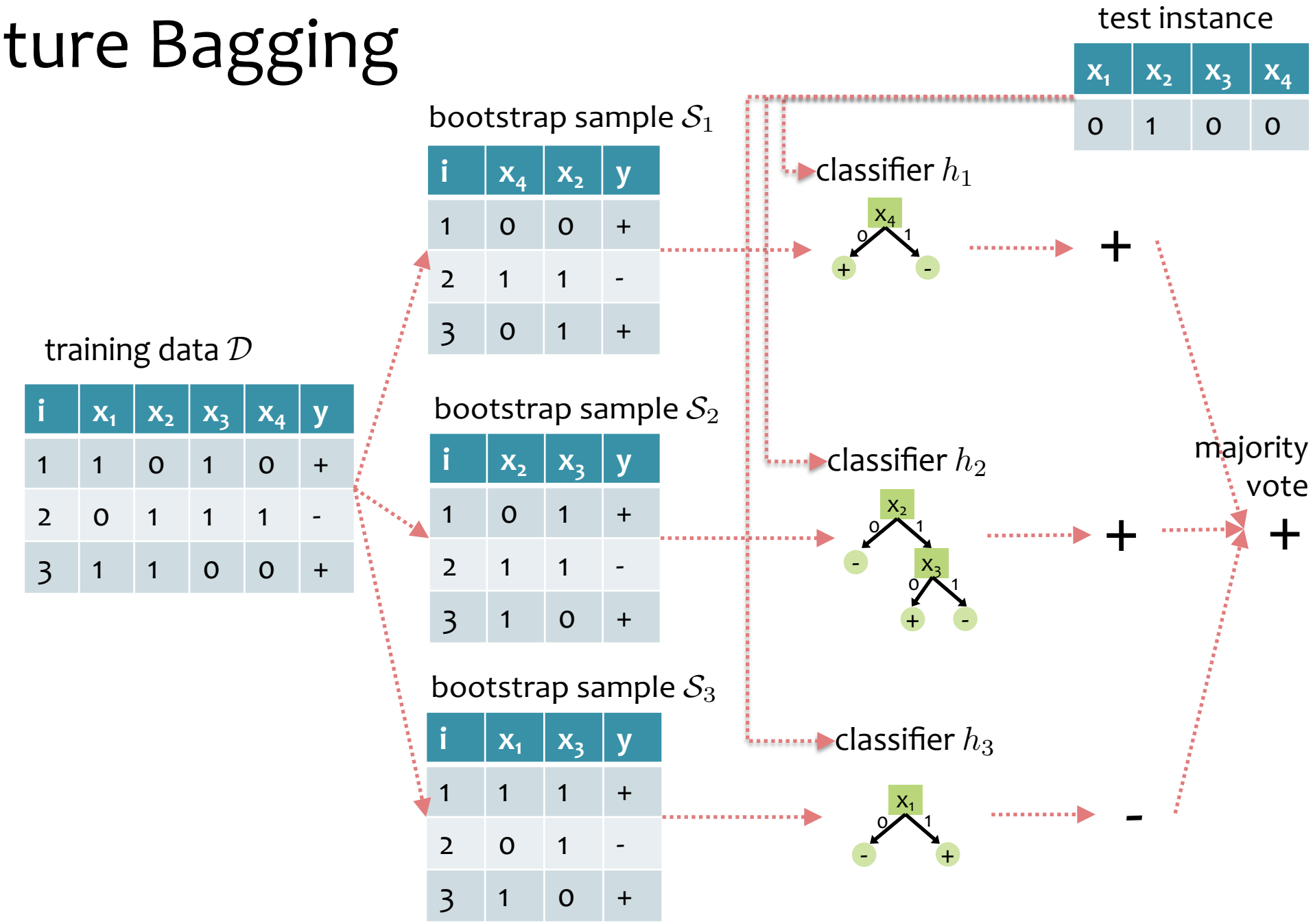
Key idea: Repeatedly sample with replacement a subset of the features, create a copy of the training data with only those features, and train a model on the copy.

Return an ensemble of the trained models; combine predictions by majority vote for classification and by averaging for regression.

Algorithm 2 Feature Bagging

procedure SAMPLEBAGGING($\mathcal{D}$, T, S)
  for $t = 1, \ldots, T$ **do**
    for $s = 1, \ldots, S$ **do**
      $m_s \sim \text{Uniform}(1, \ldots, M)$
    for $i = 1, \ldots, N$ **do**
      $\tilde{\mathbf{x}}^{(i)} = [x_{m_1}^{(i)}, x_{m_2}^{(i)}, \ldots, x_{m_S}^{(i)}]^T$
    $\mathcal{D}_t = \{(\tilde{\mathbf{x}}^{(i)}, y^{(i)}\}_{i=1}^N$ ▷ Random subspace
    $h_t = \text{train}(\mathcal{D}_t)$ ▷ Classifier
  return $\hat{h}(\mathbf{x}) = \text{aggregate}(h_1, \ldots, h_T)$ ▷ Ensemble

Feature Bagging



RANDOM FORESTS

Random Forests

Key idea: Combine (sample) bagging and a specific variant of feature bagging to train decision trees.

Repeat the following to train many decision trees:

- draw a sample with replacement from the training examples,
- recursively learn the decision tree
- but at each node when choosing a feature on which to split, first randomly sample a subset of the features, then pick the best feature from among that subset.

Return an ensemble of the trained decision trees.

Random Forests

- Combines the prediction of many diverse decision trees to reduce their variability
- If B independent random variables $x^{(1)}, x^{(2)}, \ldots, x^{(B)}$ all have variance σ^2, then the variance of $\frac{1}{B}\sum_{b=1}^{B} x^{(b)}$ is $\frac{\sigma^2}{B}$
- Random forests = sample bagging + feature bagging

  = **b**ootstrap **agg**regat**ing** + split-feature randomization

Split-feature Randomization

- Issue: decision trees trained on bootstrapped samples still behave similarly
- Idea: in addition to sampling the data points (i.e., the rows), also sample the features (i.e., the columns)
- Each time a split is being considered, limit the possible features to a randomly sampled subset

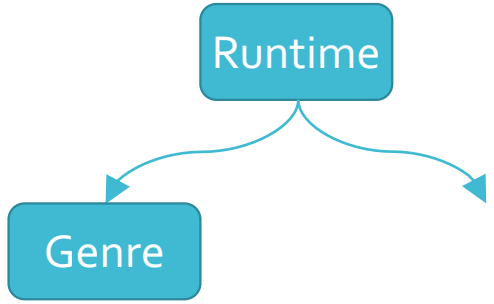
Runtime	Genre	Budget	Year	IMDB	Rating

Split-feature Randomization

- Issue: decision trees trained on bootstrapped samples still behave similarly
- Idea: in addition to sampling the data points (i.e., the rows), also sample the features (i.e., the columns)
- Each time a split is being considered, limit the possible features to a randomly sampled subset

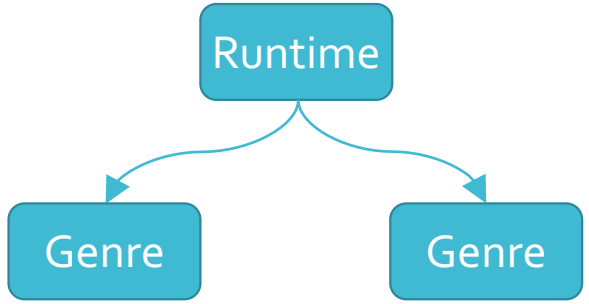
Split-feature Randomization

- Issue: decision trees trained on bootstrapped samples still behave similarly
- Idea: in addition to sampling the data points (i.e., the rows), also sample the features (i.e., the columns)
- Each time a split is being considered, limit the possible features to a randomly sampled subset



Split-feature Randomization

- Issue: decision trees trained on bootstrapped samples still behave similarly
- Idea: in addition to sampling the data points (i.e., the rows), also sample the features (i.e., the columns)
- Each time a split is being considered, limit the possible features to a randomly sampled subset



Random Forests

- Input: $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(n)}, y^{(n)}\right)\right\}_{n=1}^{N}, B, \rho$
- For $b = 1, 2, \ldots, B$
  - Create a dataset, $\mathcal{D}_b$, by sampling N points from the original training data $\mathcal{D}$ ***with replacement***
  - Learn a decision tree, t_b, using $\mathcal{D}_b$ and the ID3 algorithm ***with split-feature randomization***, sampling ρ features for each split
- Output: $\bar{t} = f(t_1, \ldots, t_B)$, the aggregated hypothesis

How can we set B and ρ?

- Input: $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(n)}, y^{(n)}\right)\right\}_{n=1}^{N}, B, \rho$
- For $b = 1, 2, \ldots, B$
  - Create a dataset, $\mathcal{D}_b$, by sampling N points from the original training data $\mathcal{D}$ ***with replacement***
  - Learn a decision tree, t_b, using $\mathcal{D}_b$ and the ID3 algorithm ***with split-feature randomization***, sampling ρ features for each split
- Output: $\bar{t} = f(t_1, \ldots, t_B)$, the aggregated hypothesis

Recall: Validation Sets

- Suppose we want to compare multiple hyperparameter settings $\theta_1, \ldots, \theta_K$
- For $k = 1, 2, \ldots, K$
  - Train a model on D_{train} using θ_k
- Evaluate each model on D_{val} and find the best hyperparameter setting, θ_{k^*}
- Compute the error of a model trained with θ_{k^*} on D_{test}

Out-of-bag Error

- For each training point, $\boldsymbol{x}^{(n)}$, there are some decision trees which $\boldsymbol{x}^{(n)}$ was not used to train (roughly B/e trees or 37%)
  - Let these be $t^{(-n)} = \left\{t_1^{(-n)}, t_2^{(-n)}, \ldots, t_{N-n}^{(-n)}\right\}$
- Compute an aggregated prediction for each $\boldsymbol{x}^{(n)}$ using the trees in $t^{(-n)}$, $\bar{t}^{(-n)}\left(\boldsymbol{x}^{(n)}\right)$
- Compute the out-of-bag (OOB) error, e.g., for regression

$$E_{OOB} = \frac{1}{N}\sum_{n=1}^{N}\left(\bar{t}^{(-n)}\left(\boldsymbol{x}^{(n)}\right) - y^{(n)}\right)^2$$

Out-of-bag Error

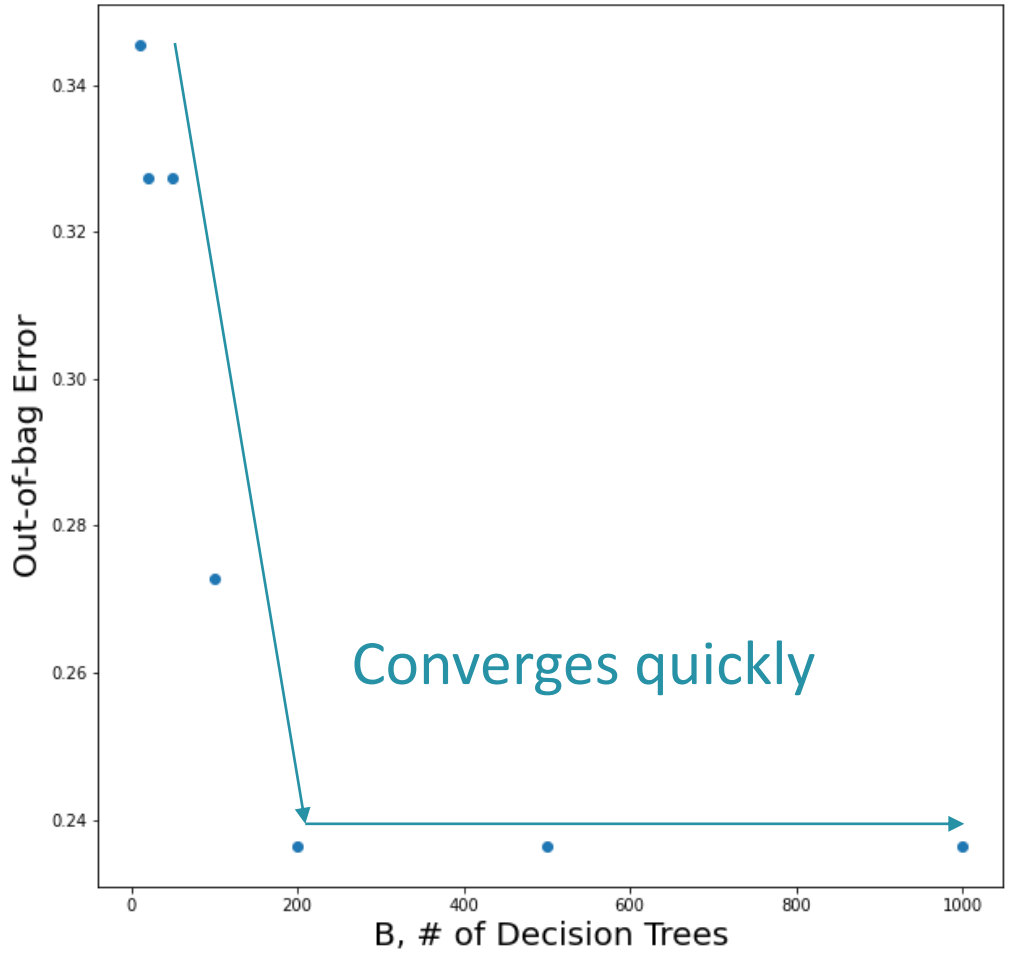
- For each training point, $\boldsymbol{x}^{(n)}$, there are some decision trees which $\boldsymbol{x}^{(n)}$ was not used to train (roughly B/e trees or 37%)
  - Let these be $t^{(-n)} = \left\{t_1^{(-n)}, t_2^{(-n)}, \ldots, t_{N-n}^{(-n)}\right\}$
- Compute an aggregated prediction for each $\boldsymbol{x}^{(n)}$ using the trees in $t^{(-n)}$, $\bar{t}^{(-n)}\left(\boldsymbol{x}^{(n)}\right)$
- Compute the out-of-bag (OOB) error, e.g., for classification

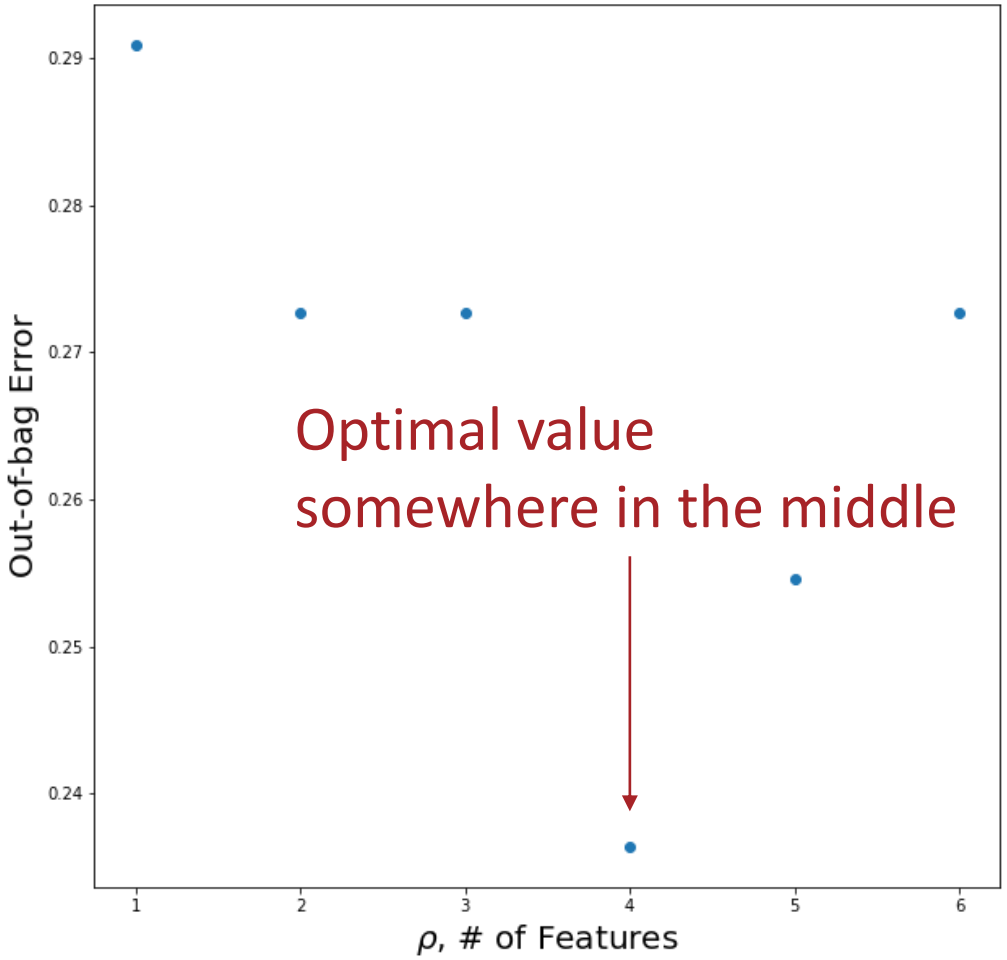
$$E_{OOB} = \frac{1}{N}\sum_{n=1}^{N} \mathbb{1}\left(\bar{t}^{(-n)}\left(\boldsymbol{x}^{(n)}\right) \neq y^{(n)}\right)$$

- E_{OOB} can be used for hyperparameter optimization!

Out-of-bag Error

- Suppose we want to compare different numbers of trees in our random forest $B_1, \ldots, B_K$
- For $k = 1, 2, \ldots, K$
  - Train a random forest on D_{train} with B_k trees
- Compute E_{OOB} for each random forest and find the best number of trees, B_{k^*}
- Evaluate the random forest with B_{k^*} trees on D_{test}



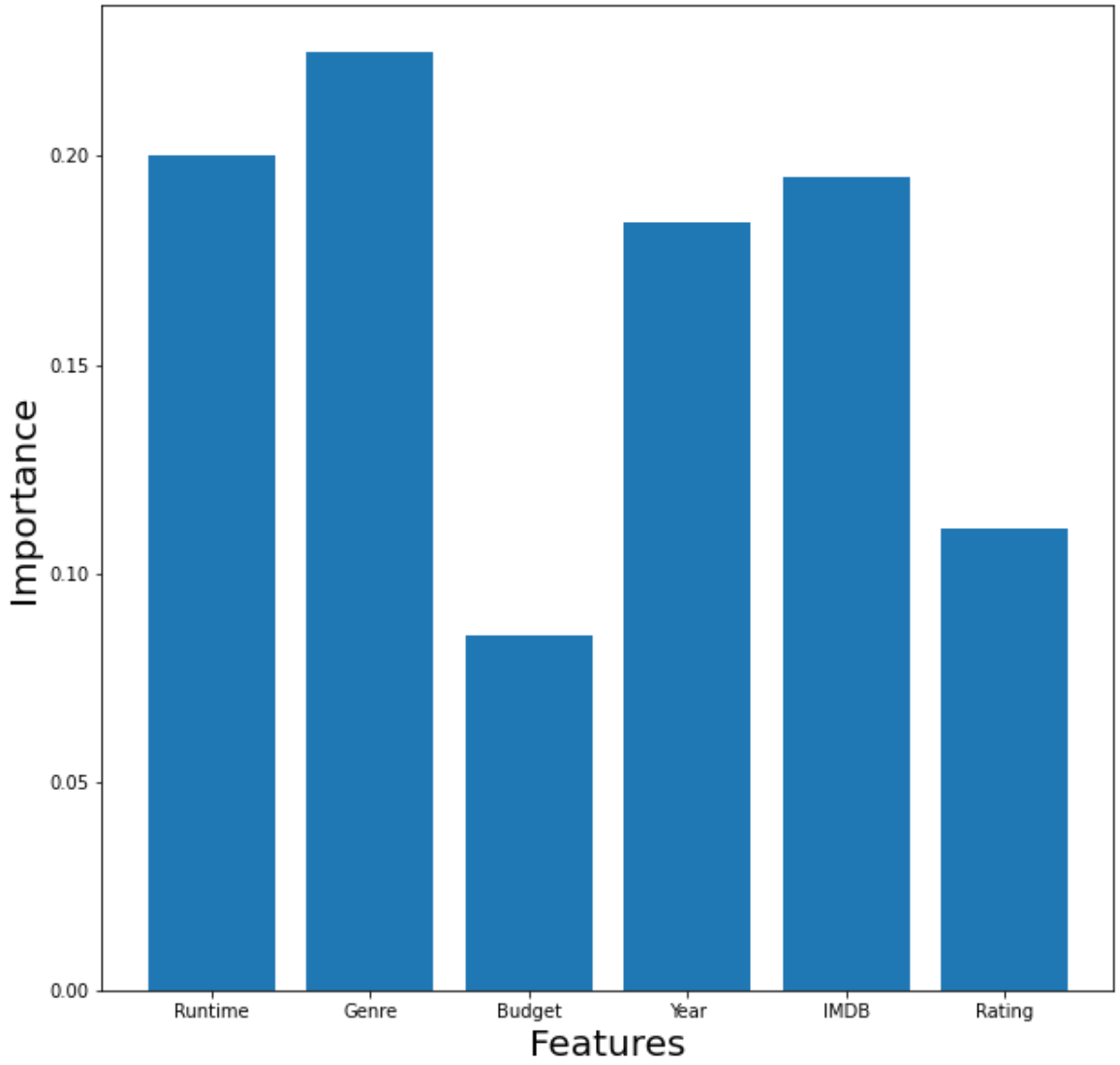


Setting Hyperparameters

Feature Importance

- Some of the interpretability of decision trees gets lost when switching to random forests
- Random forests allow for the computation of “feature importance”, a way of ranking features based on how useful they are at predicting the target
- Initialize each feature’s importance to zero
- Each time a feature is chosen to be split on, add the reduction in entropy (weighted by the number of data points in the split) to its importance

Feature Importance



Key Takeaways

- Ensemble methods employ a “wisdom of crowds” philosophy
  - Can reduce the variance of high variance methods
- Random forests = bagging + split-feature randomization
  - Aggregate multiple decision trees together
  - Bootstrapping and split-feature randomization increase diversity in the decision trees
  - Use out-of-bag errors for hyperparameter optimization
  - Use feature importance to identify useful attributes

Learning Objectives

Ensemble Methods: Bagging

You should be able to…

1. Distinguish between (sample) bagging, the random subspace method, and random forests.
2. Implement (sample) bagging for an arbitrary base classifier/regressor.
3. Implement the random subspace method for an arbitrary base classifier/ regressor.
4. Implement random forests.
5. Contrast out-of-bag error with cross-validation error.
6. Differentiate boosting from bagging.
7. Compare and contrast weighted and unweighted majority vote of a collection of classifiers.
8. Discuss the relation in bagging between the sample size and variance of the base classifier/regressor.
9. Bound the generalization error of a random forest classifier.

CLUSTERING

Clustering, Informal Goals

Goal: Automatically partition unlabeled data into groups of similar data points.

Question: When and why would we want to do this?

Useful for:

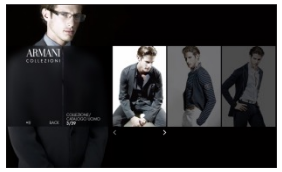
- Automatically organizing data.
- Understanding hidden structure in data.
- Preprocessing for further analysis.
  - Representing high-dimensional data in a low-dimensional space (e.g., for visualization purposes).

Slide courtesy of Nina Balcan

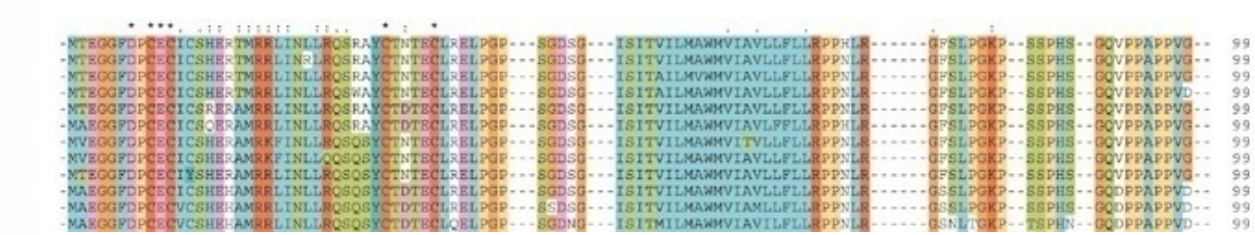
Applications (Clustering comes up everywhere…)

- Cluster news articles or web pages or search results by topic.

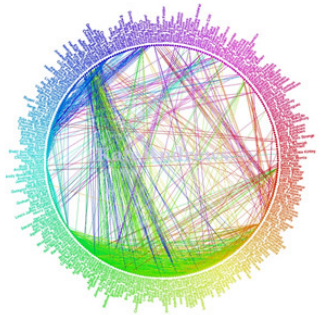




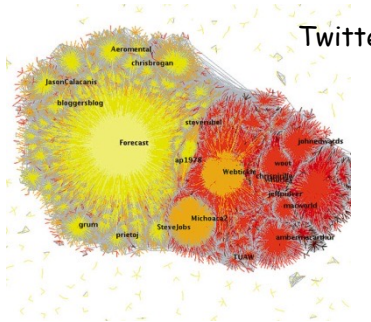
- Cluster protein sequences by function or genes according to expression profile.

- Cluster users of social networks by interest (community detection).



Facebook network

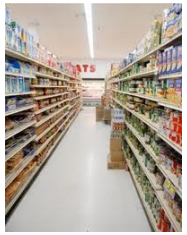


Twitter Network

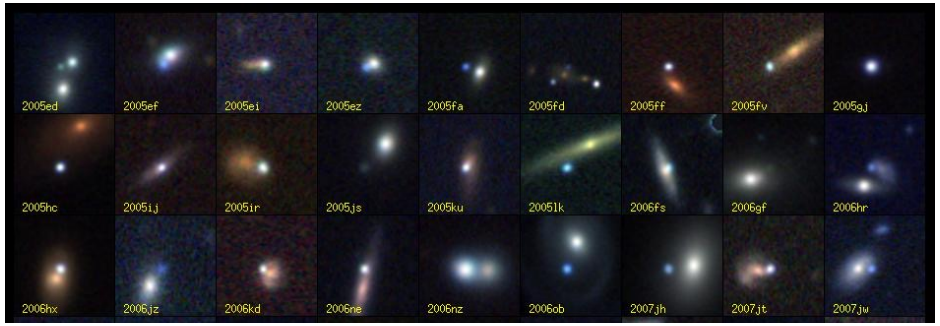
Slide courtesy of Nina Balcan

Applications (Clustering comes up everywhere…)

- Cluster customers according to purchase history.



- Cluster galaxies or nearby stars (e.g. Sloan Digital Sky Survey)

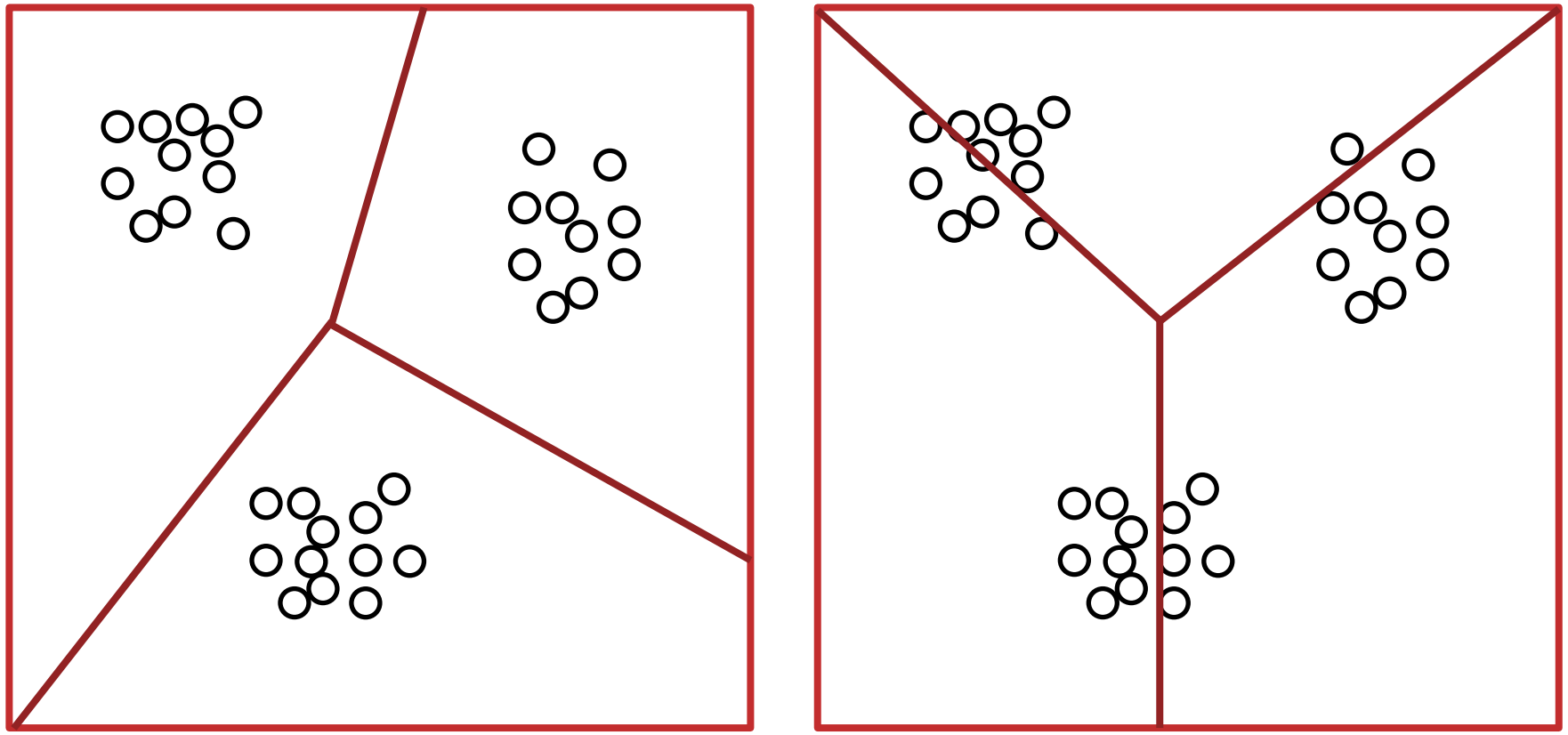


- And many many more applications….

Slide courtesy of Nina Balcan

Clustering

Question: Which of these partitions is “better”?



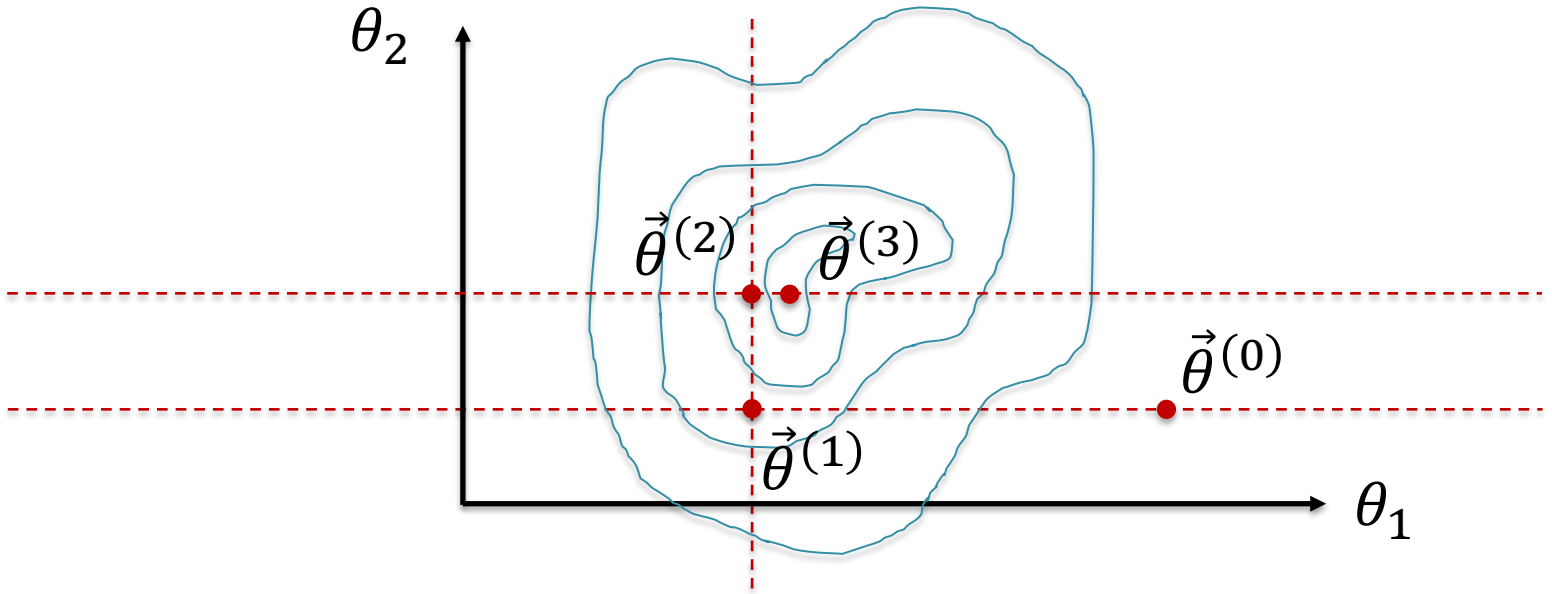
OPTIMIZATION BACKGROUND

Coordinate Descent

- Goal: minimize some objective

$$\vec{\theta}^* = \underset{\vec{\theta}}{\operatorname{argmin}} J(\vec{\theta})$$

- Idea: iteratively pick one variable and minimize the objective w.r.t. just that one variable, *keeping all the others fixed*.



Block Coordinate Descent

- Goal: minimize some objective (with 2 blocks)

$$\vec{\alpha}^*, \vec{\beta}^* = \underset{\vec{\alpha},\vec{\beta}}{\operatorname{argmin}} J(\vec{\alpha}, \vec{\beta})$$

- Idea: iteratively pick one *block* of variables ($\vec{\alpha}$ or $\vec{\beta}$) and minimize the objective w.r.t. that block, keeping the other(s) fixed.

while not converged:

$$\vec{\alpha} = \underset{\vec{\alpha}}{\operatorname{argmin}} J(\vec{\alpha}, \vec{\beta})$$

$$\vec{\beta} = \underset{\vec{\beta}}{\operatorname{argmin}} J(\vec{\alpha}, \vec{\beta})$$

K-MEANS

K-Means Algorithm (Derivation)

Recipe for K-Means Derivation:

1) Define a Model.
2) Choose an objective function.
3) Optimize it!

K-Means Algorithm (Derivation)

- Input: unlabeled data $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^{N},\ \mathbf{x}^{(i)} \in \mathbb{R}^M$
- Goal: Find an assignment of points to clusters
- Model Paramters:
  - cluster centers: $\mathbf{C} = [\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_K],\ \mathbf{c}_j \in \mathbb{R}^M$
  - cluster assignments: $\mathbf{z} = [z^{(1)}, z^{(2)}, \ldots, z^{(N)}],\ z^{(i)} \in \{1, \ldots, K\}$
- Decision Rule: assign each point $\mathbf{x}^{(i)}$ to its nearest cluster center $\mathbf{c}_j$

K-Means Algorithm (Derivation)

- Input: unlabeled data $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^{N},\ \mathbf{x}^{(i)} \in \mathbb{R}^M$
- Goal: Find an assignment of points to clusters
- Model Paramters:
  - cluster centers: $\mathbf{C} = [\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_K],\ \mathbf{c}_j \in \mathbb{R}^M$
  - cluster assignments: $\mathbf{z} = [z^{(1)}, z^{(2)}, \ldots, z^{(N)}],\ z^{(i)} \in \{1, \ldots, K\}$
- Decision Rule: assign each point $\mathbf{x}^{(i)}$ to its nearest cluster center $\mathbf{c}_j$
- Objective:

$$\hat{\mathbf{C}} = \underset{\mathbf{C}}{\operatorname{argmin}} \sum_{i=1}^{N} \min_{j} ||\mathbf{x}^{(i)} - \mathbf{c}_j||_2^2$$

Question: In English, what is this quantity?

Answer:

K-Means Algorithm (Derivation)

- Input: unlabeled data $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^{N},\ \mathbf{x}^{(i)} \in \mathbb{R}^M$
- Goal: Find an assignment of points to clusters
- Model Paramters:
  - cluster centers: $\mathbf{C} = [\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_K],\ \mathbf{c}_j \in \mathbb{R}^M$
  - cluster assignments: $\mathbf{z} = [z^{(1)}, z^{(2)}, \ldots, z^{(N)}],\ z^{(i)} \in \{1, \ldots, K\}$
- Decision Rule: assign each point $\mathbf{x}^{(i)}$ to its nearest cluster center $\mathbf{c}_j$
- Objective:

$$\hat{\mathbf{C}} = \underset{\mathbf{C}}{\operatorname{argmin}} \sum_{i=1}^{N} \min_{j} ||\mathbf{x}^{(i)} - \mathbf{c}_j||_2^2$$

$$= \underset{\mathbf{C}}{\operatorname{argmin}} \sum_{i=1}^{N} \min_{z^{(i)}} ||\mathbf{x}^{(i)} - \mathbf{c}_{z^{(i)}}||_2^2$$

K-Means Algorithm (Derivation)

- Input: unlabeled data $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^{N},\ \mathbf{x}^{(i)} \in \mathbb{R}^M$
- Goal: Find an assignment of points to clusters
- Model Paramters:
  - cluster centers: $\mathbf{C} = [\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_K],\ \mathbf{c}_j \in \mathbb{R}^M$
  - cluster assignments: $\mathbf{z} = [z^{(1)}, z^{(2)}, \ldots, z^{(N)}],\ z^{(i)} \in \{1, \ldots, K\}$
- Decision Rule: assign each point $\mathbf{x}^{(i)}$ to its nearest cluster center $\mathbf{c}_j$
- Objective:

$$\hat{\mathbf{C}} = \underset{\mathbf{C}}{\operatorname{argmin}} \sum_{i=1}^{N} \min_{j} ||\mathbf{x}^{(i)} - \mathbf{c}_j||_2^2$$

$$= \underset{\mathbf{C}}{\operatorname{argmin}} \sum_{i=1}^{N} \min_{z^{(i)}} ||\mathbf{x}^{(i)} - \mathbf{c}_{z^{(i)}}||_2^2$$

$$\hat{\mathbf{C}}, \hat{\mathbf{z}} = \underset{\mathbf{C},\mathbf{z}}{\operatorname{argmin}} \sum_{i=1}^{N} ||\mathbf{x}^{(i)} - \mathbf{c}_{z^{(i)}}||_2^2$$

K-Means Algorithm (Derivation)

- Input: unlabeled data $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^{N},\ \mathbf{x}^{(i)} \in \mathbb{R}^M$
- Goal: Find an assignment of points to clusters
- Model Paramters:
  - cluster centers: $\mathbf{C} = [\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_K],\ \mathbf{c}_j \in \mathbb{R}^M$
  - cluster assignments: $\mathbf{z} = [z^{(1)}, z^{(2)}, \ldots, z^{(N)}],\ z^{(i)} \in \{1, \ldots, K\}$
- Decision Rule: assign each point $\mathbf{x}^{(i)}$ to its nearest cluster center $\mathbf{c}_j$
- Objective:

$$\hat{\mathbf{C}} = \underset{\mathbf{C}}{\operatorname{argmin}} \sum_{i=1}^{N} \min_{j} ||\mathbf{x}^{(i)} - \mathbf{c}_j||_2^2$$

$$= \underset{\mathbf{C}}{\operatorname{argmin}} \sum_{i=1}^{N} \min_{z^{(i)}} ||\mathbf{x}^{(i)} - \mathbf{c}_{z^{(i)}}||_2^2$$

$$\hat{\mathbf{C}}, \hat{\mathbf{z}} = \underset{\mathbf{C},\mathbf{z}}{\operatorname{argmin}} \sum_{i=1}^{N} ||\mathbf{x}^{(i)} - \mathbf{c}_{z^{(i)}}||_2^2$$

$$= \boxed{\underset{\mathbf{C},\mathbf{z}}{\operatorname{argmin}}\ J(\mathbf{C}, \mathbf{z})}$$

Now apply
Block Coordinate Descent!

K-Means Algorithm

1) **Given** unlabeled feature vectors
   $D = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(N)}\}$
2) **Initialize** cluster centers $c = \{\mathbf{c}_1, \ldots, \mathbf{c}_K\}$
3) **Repeat** until convergence:
   a) $\mathbf{z} \leftarrow \text{argmin}_{\mathbf{z}}\ J(\mathbf{C}, \mathbf{z})$
      (pick each *cluster assignment* to minimize distance)
   b) $\mathbf{C} \leftarrow \text{argmin}_{\mathbf{C}}\ J(\mathbf{C}, \mathbf{z})$
      (pick each *cluster center* to minimize distance)

This is an application of
Block Coordinate Descent!
The only remaining step is to figure out
what the argmins boil down to…

K-Means Algorithm

1) **Given** unlabeled feature vectors
   $D = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(N)}\}$
2) **Initialize** cluster centers $c = \{\mathbf{c}_1, \ldots, \mathbf{c}_K\}$
3) **Repeat** until convergence:

   a) for i in {1,…, N}
      $z^{(i)} \leftarrow \textbf{argmin}_j \left(\| \mathbf{x}^{(i)} - \mathbf{c}_j \|_2\right)^2$

   b) for j in {1,…,K}
      $\mathbf{c}_j \leftarrow \textbf{argmin} \sum_{i:z^{(i)}=j} \left(\| \mathbf{x}^{(i)} - \mathbf{c}_j \|_2\right)^2$

The minimization over cluster assignments decomposes, so that we can find each $z^{(i)}$ independently of the others

Likewise, the minimization over cluster centers decomposes, so we can find each $\mathbf{c}_j$ independently

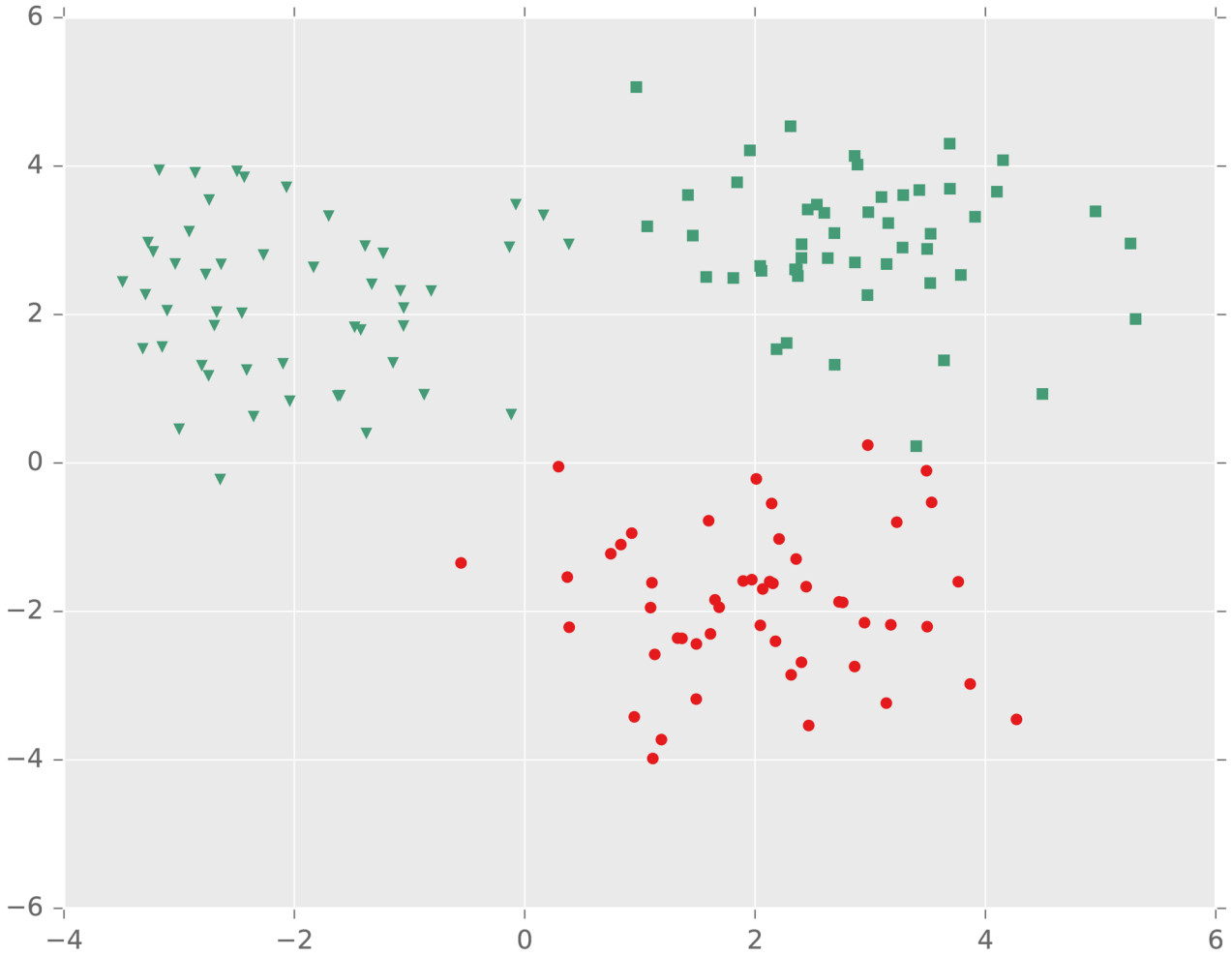
K-Means Algorithm

1) **Given** unlabeled feature vectors
   $D = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(N)}\}$
2) **Initialize** cluster centers $c = \{\mathbf{c}_1, \ldots, \mathbf{c}_K\}$
3) **Repeat** until convergence:
   a) for i in {1,…, N}
      $z^{(i)} \leftarrow$ **index** j of cluster center **nearest** to $\mathbf{x}^{(i)}$
   b) for j in {1,…,K}
      $\mathbf{c}_j \leftarrow$ **mean** of **all** points assigned to cluster j

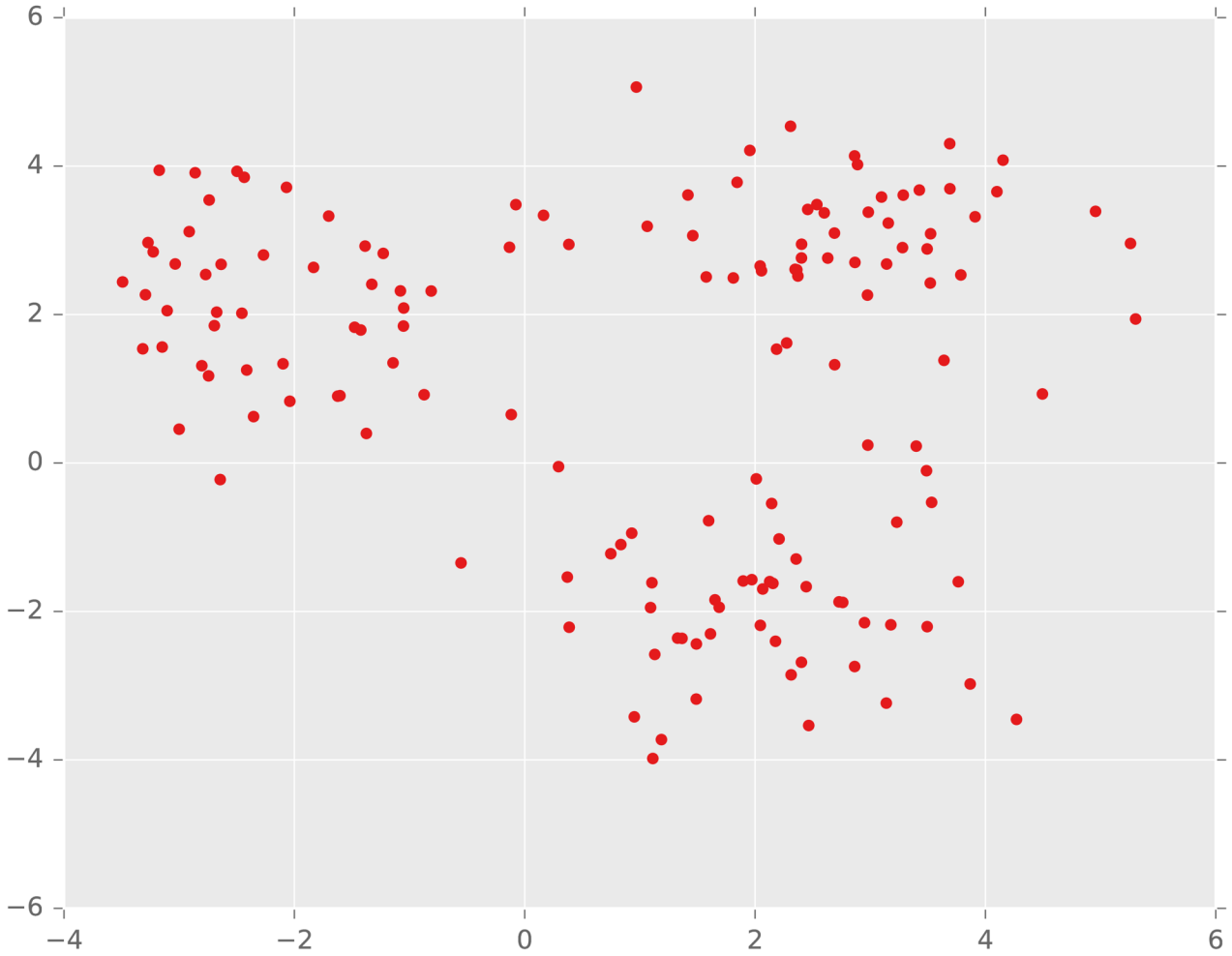
K=3 cluster centers

K-MEANS EXAMPLE

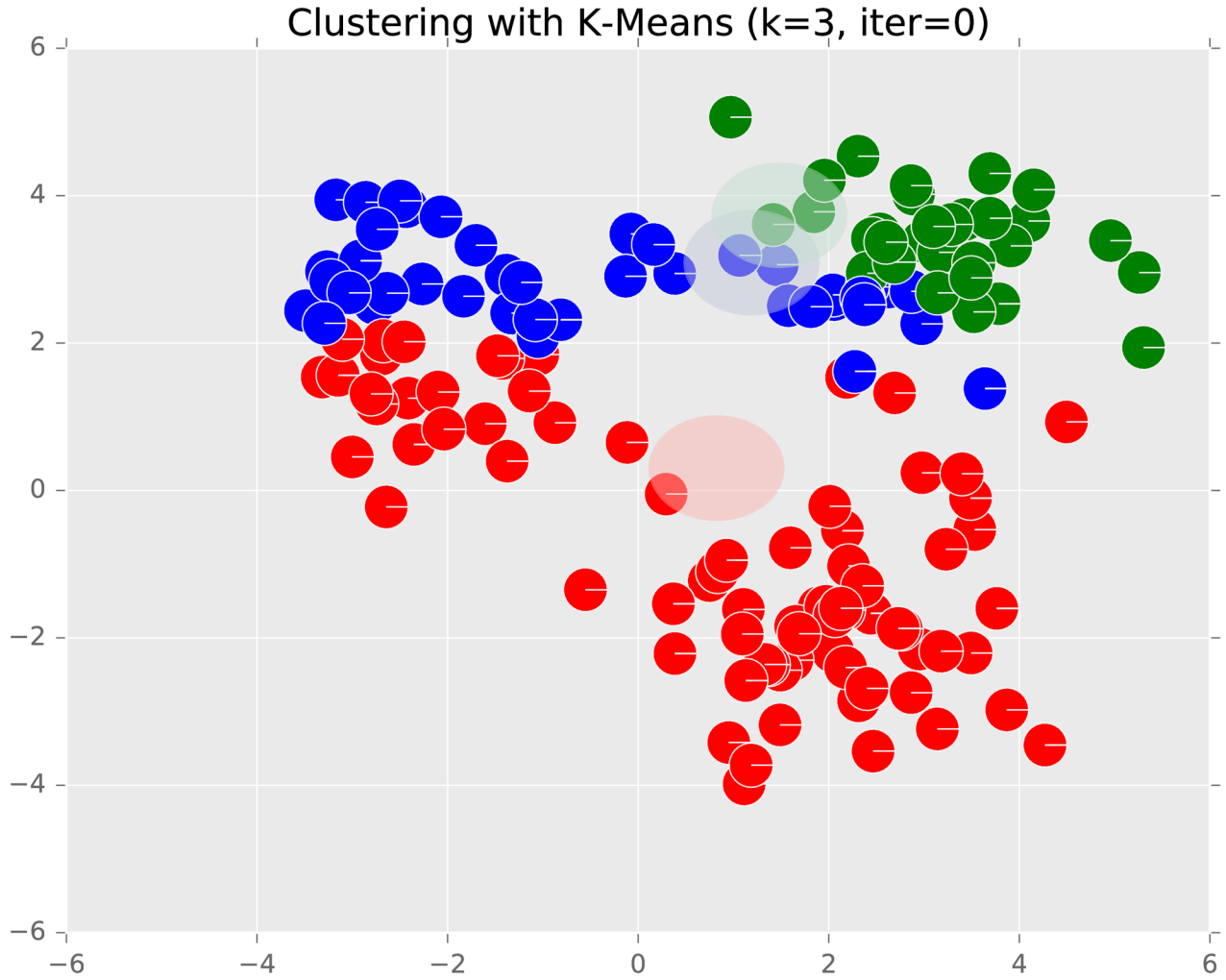
Example: K-Means



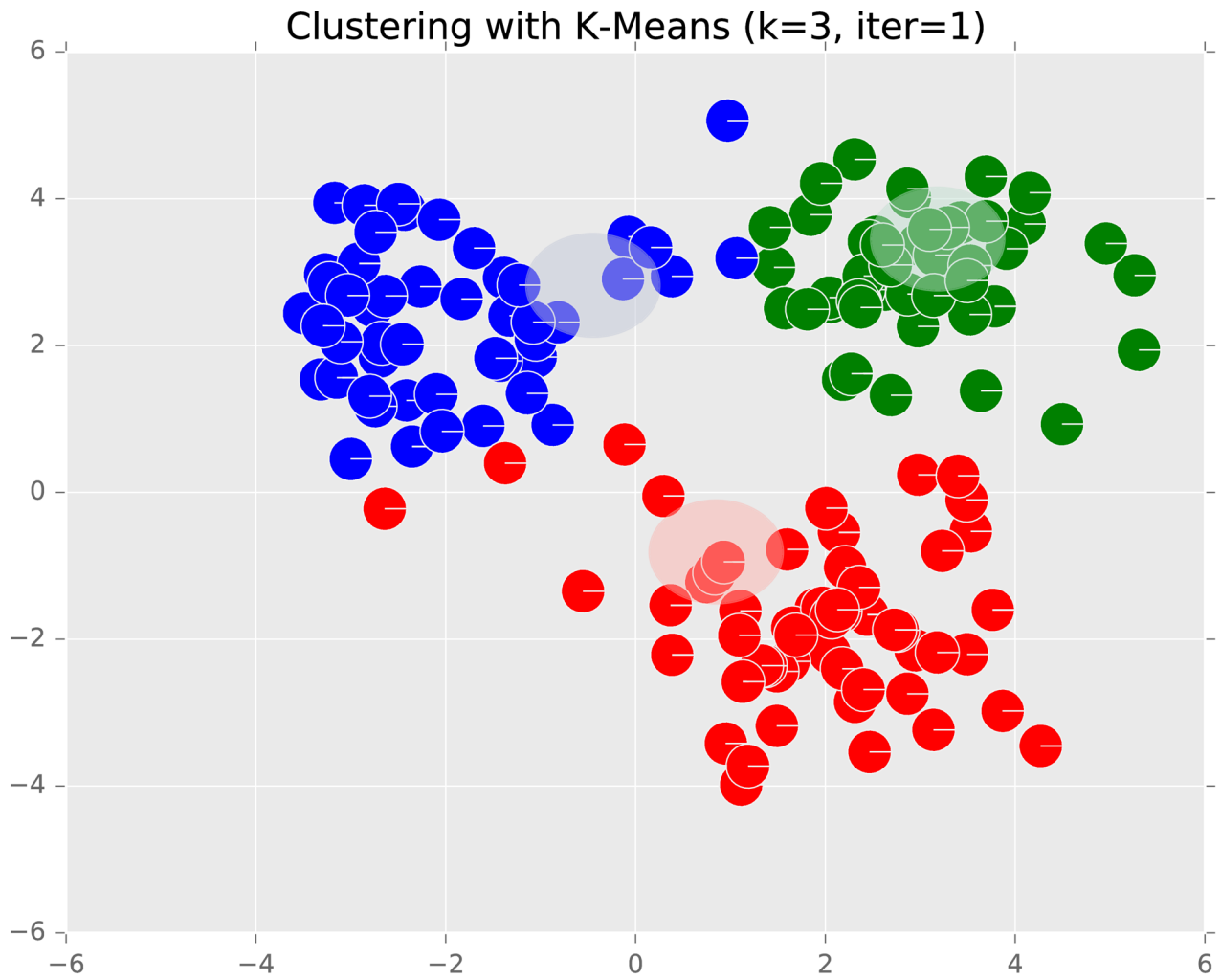
Example: K-Means



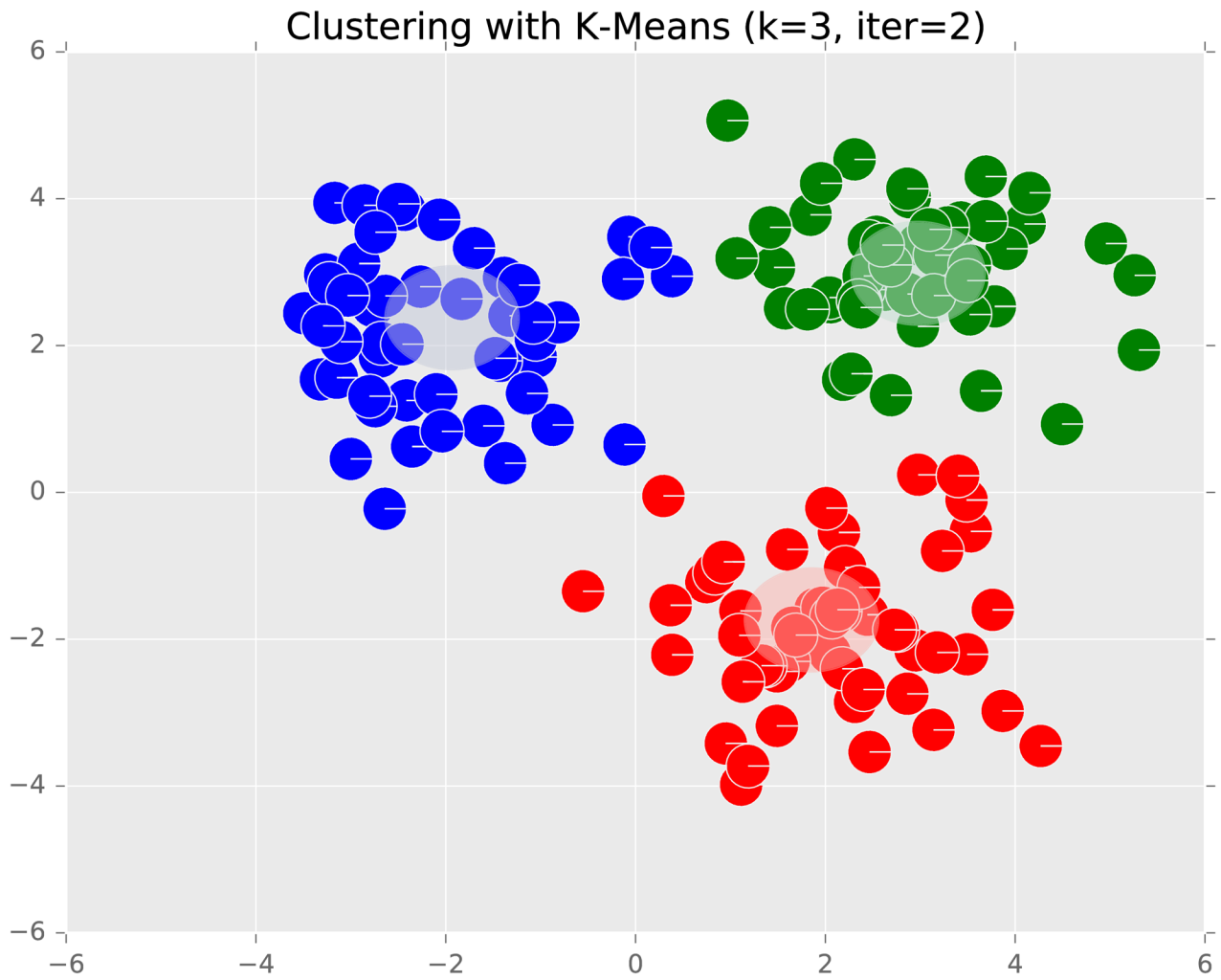
Example: K-Means



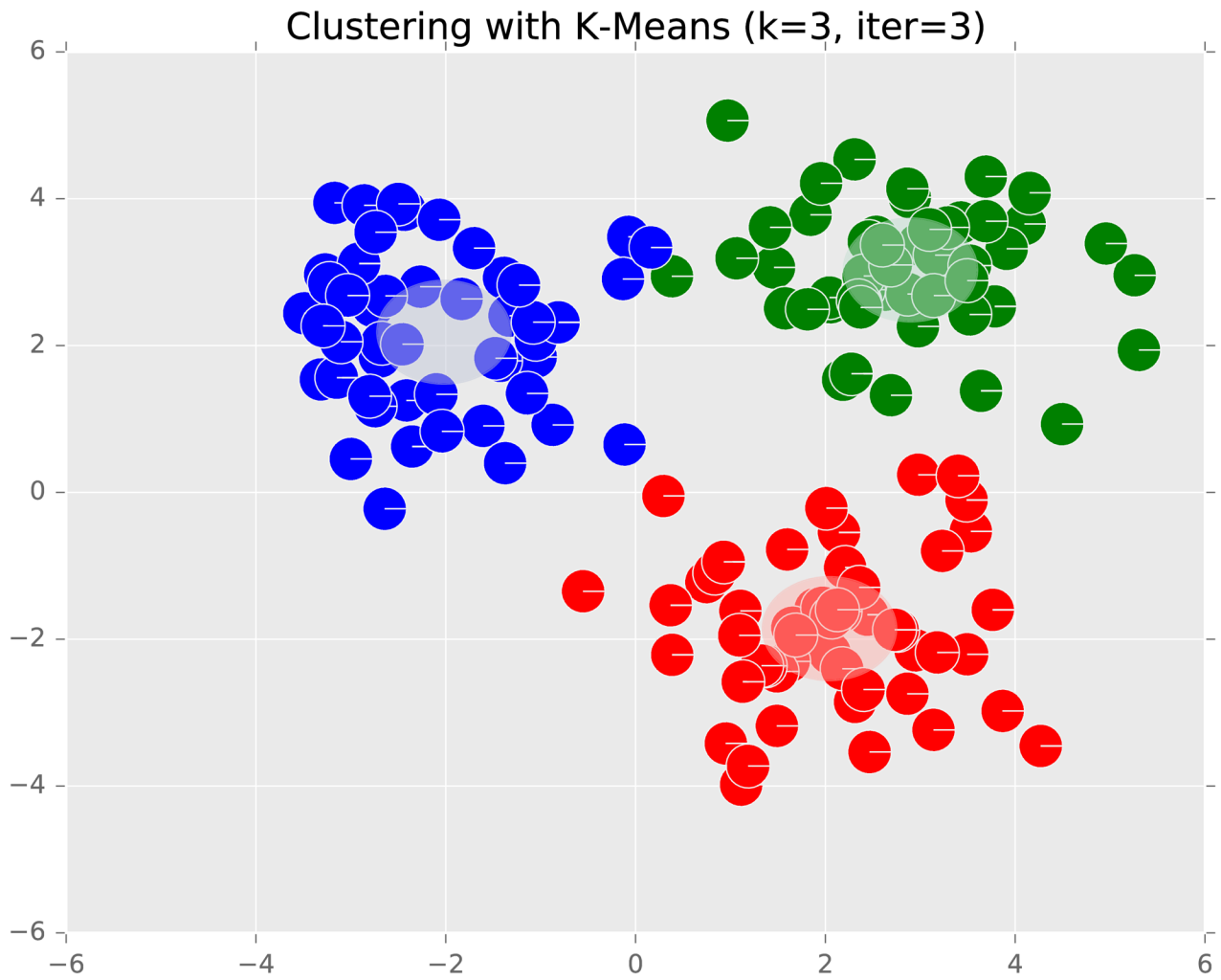
Example: K-Means



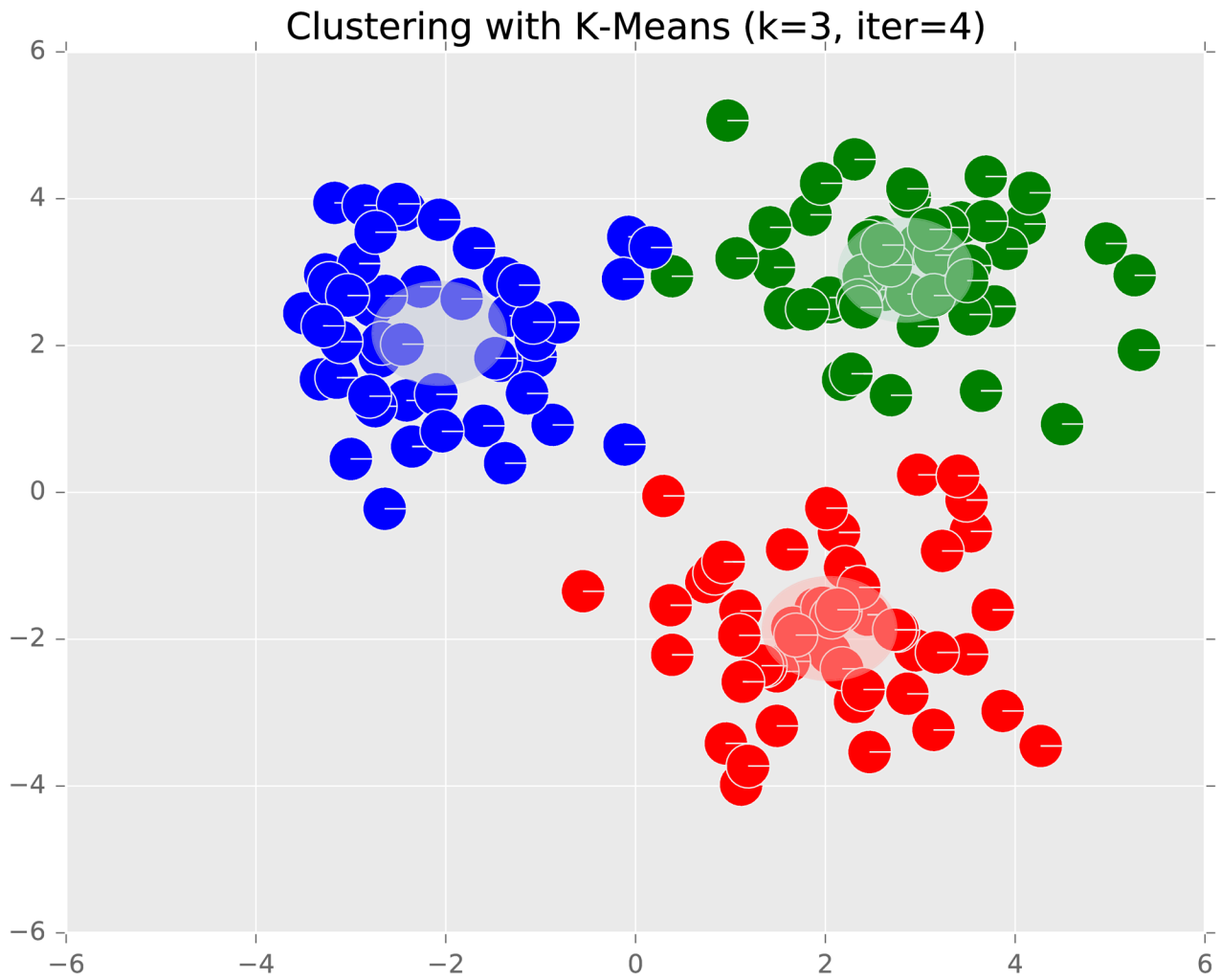
Example: K-Means



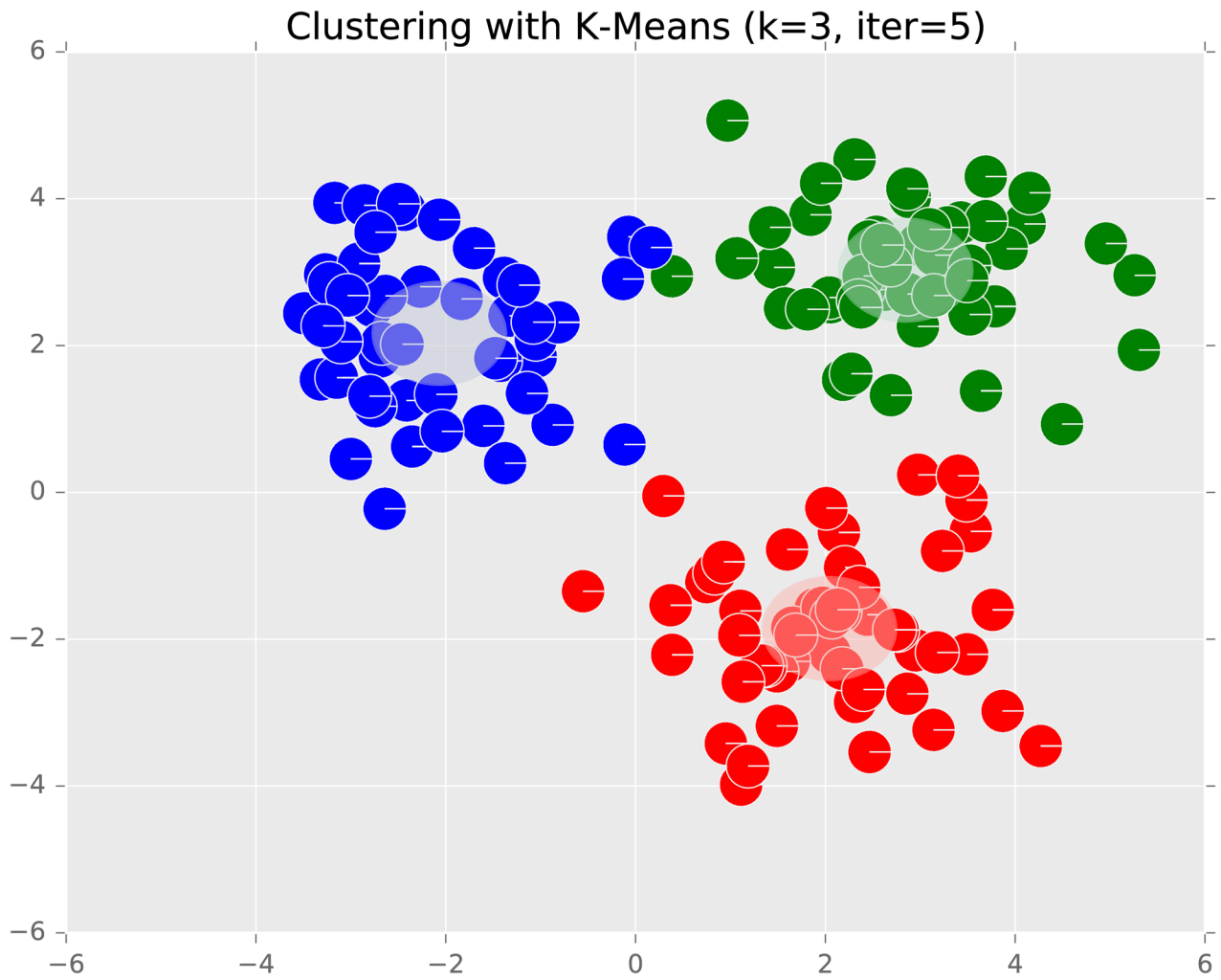
Example: K-Means



Example: K-Means



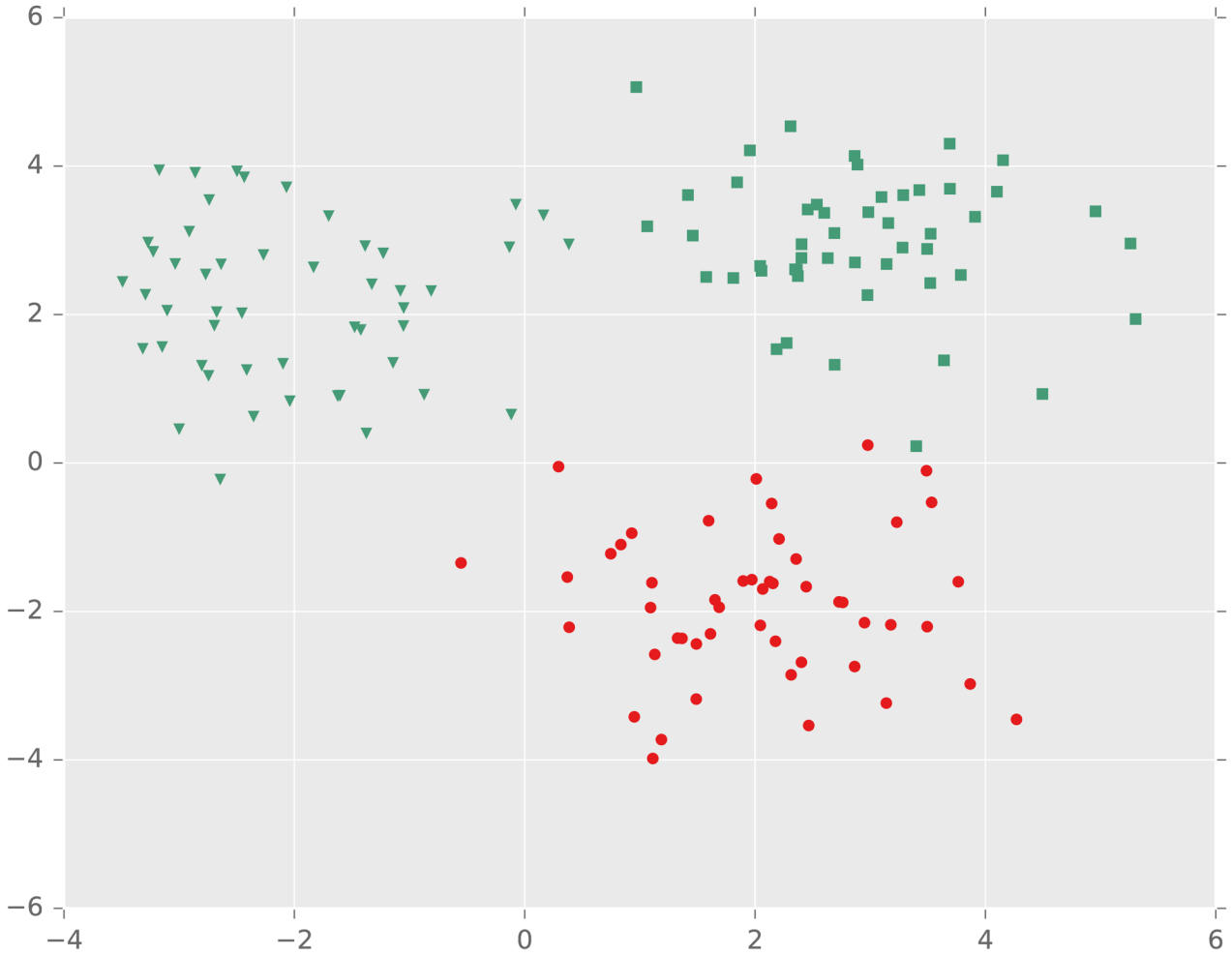
Example: K-Means



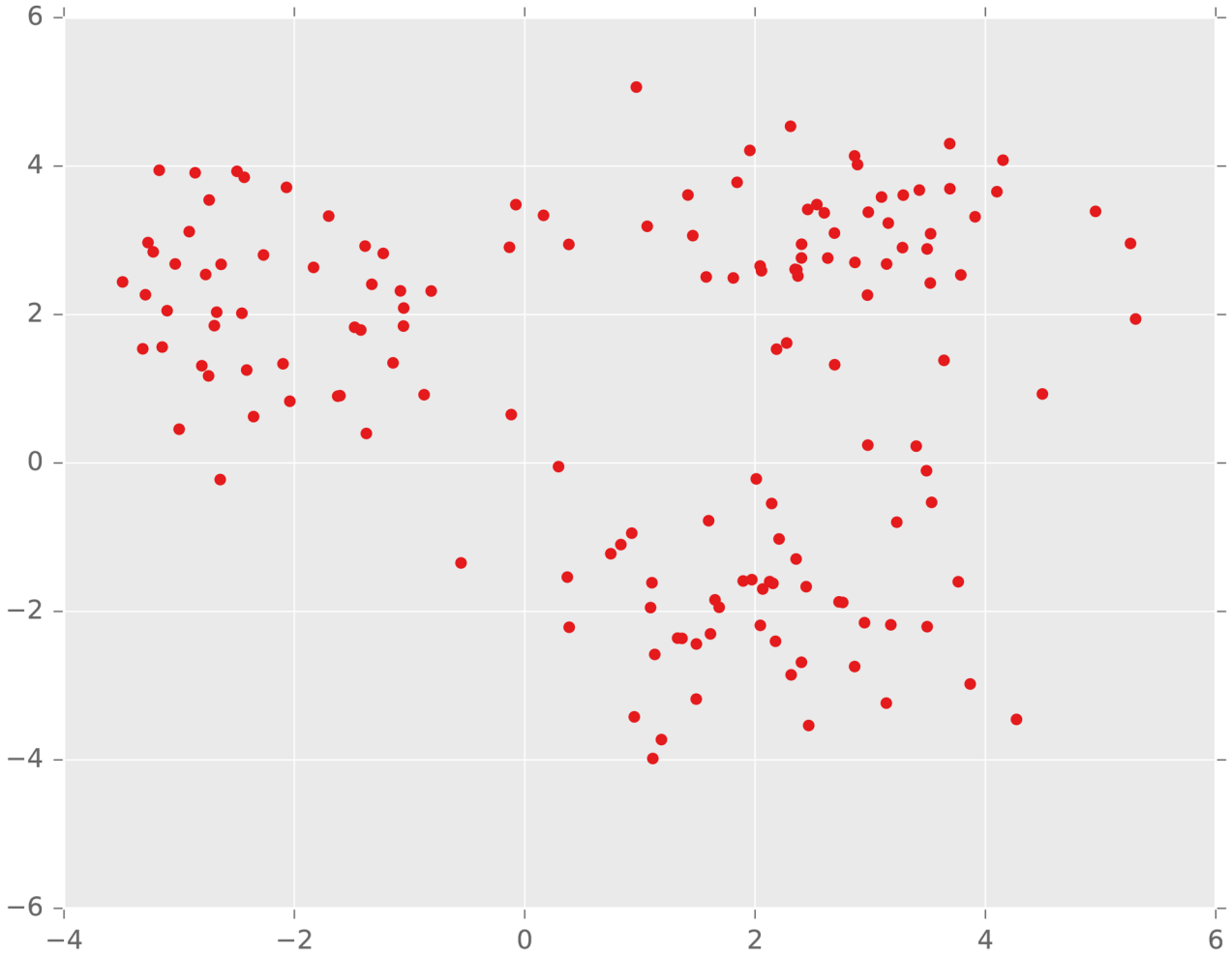
K=2 cluster centers

K-MEANS EXAMPLE

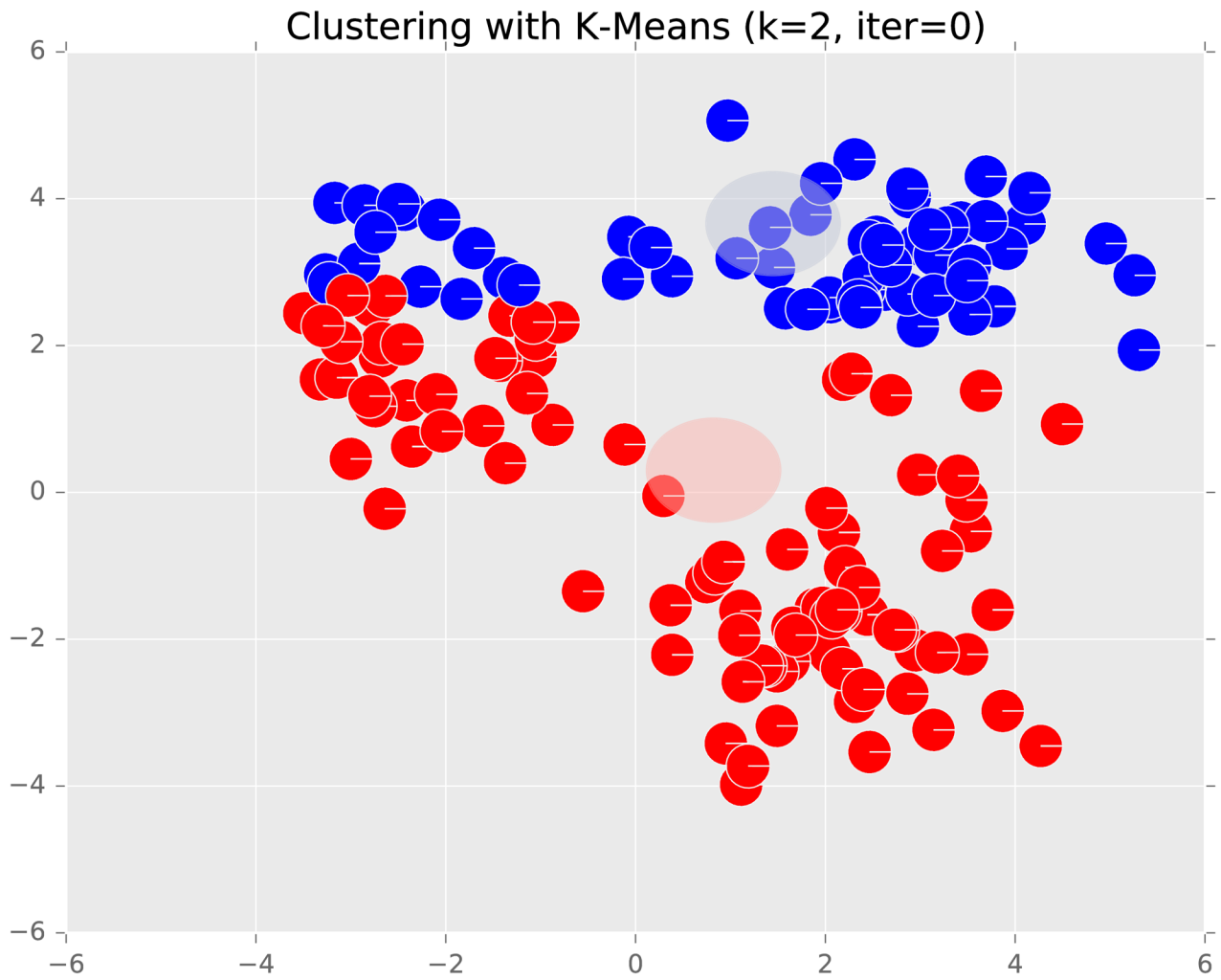
Example: K-Means



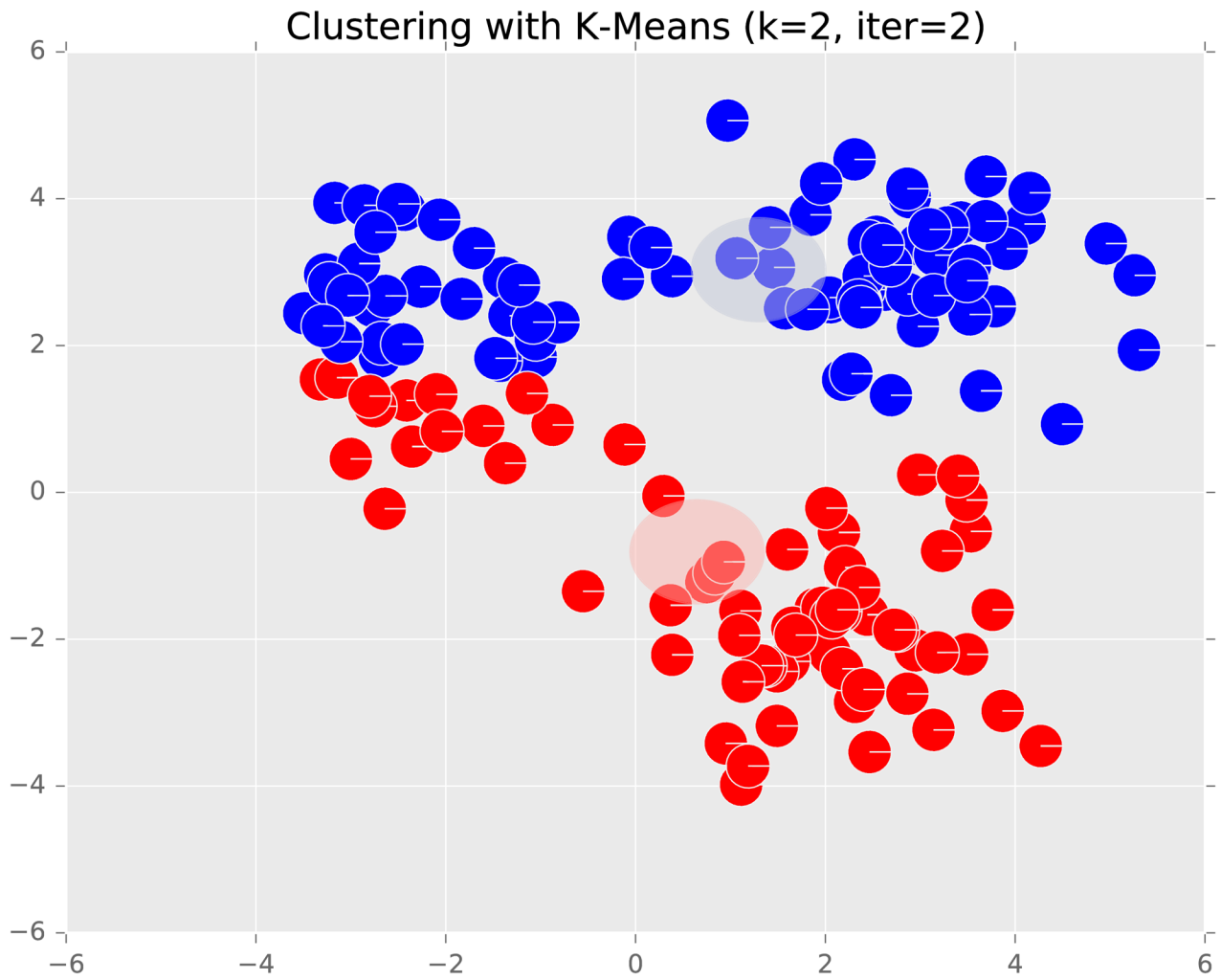
Example: K-Means



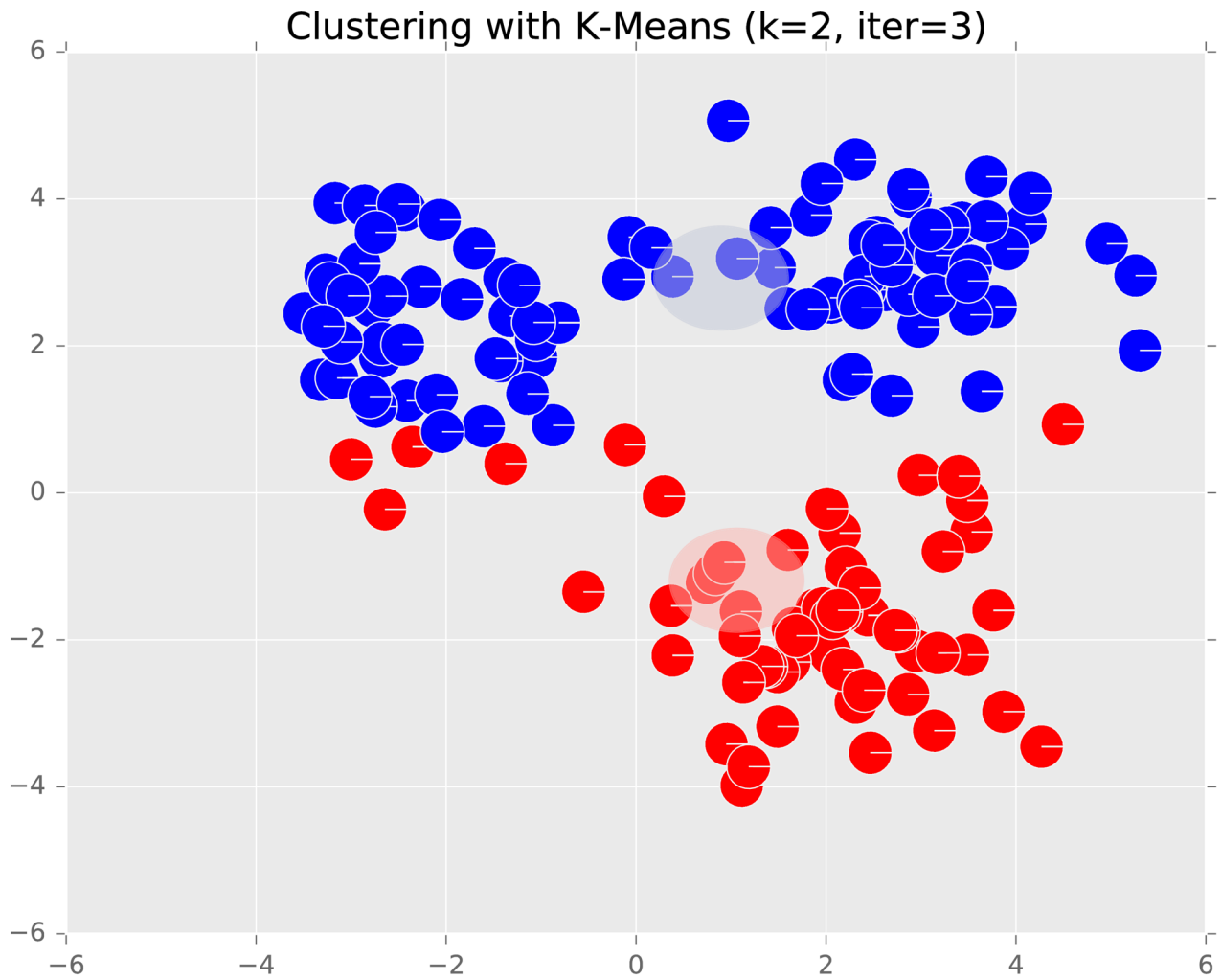
Example: K-Means



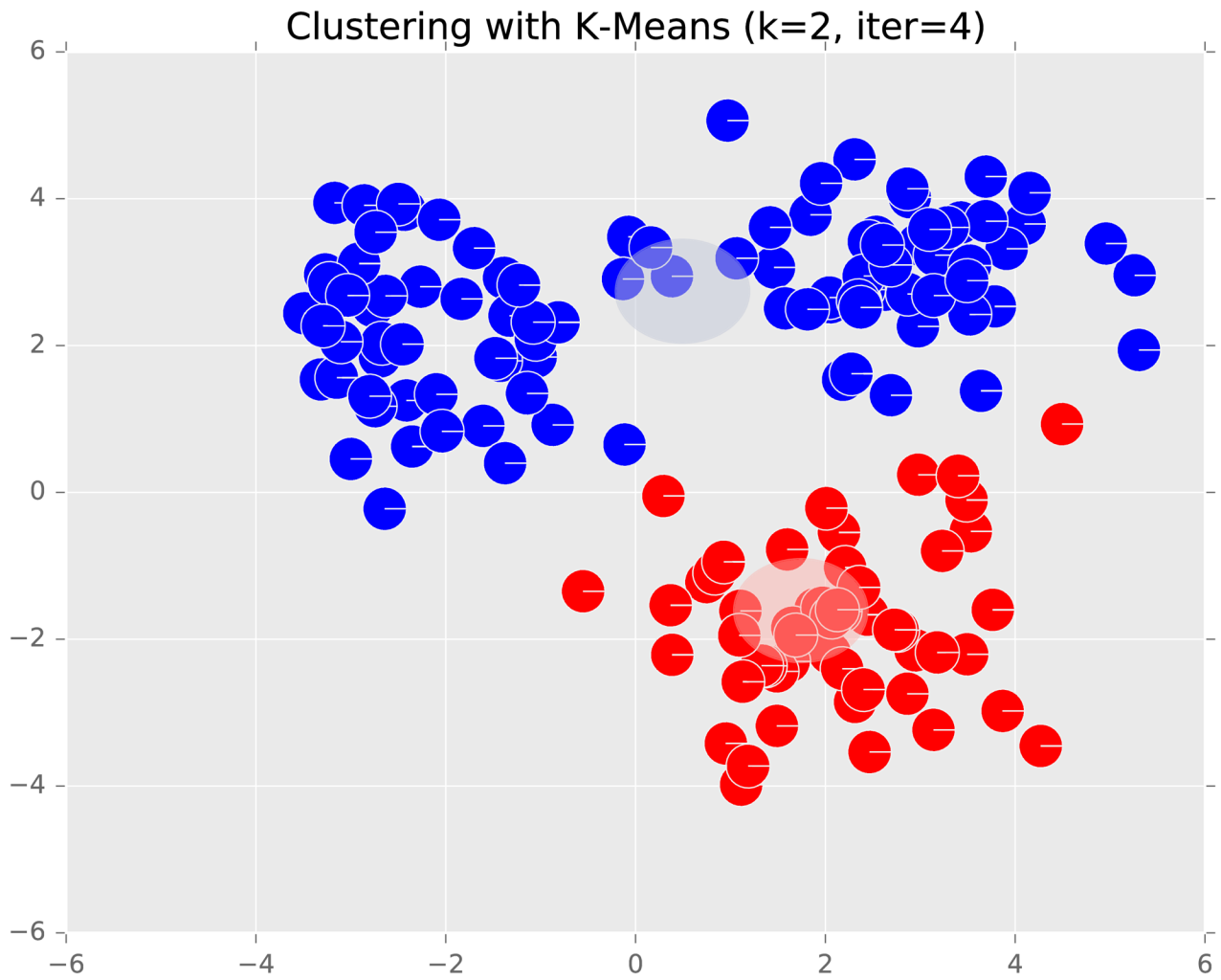
Example: K-Means



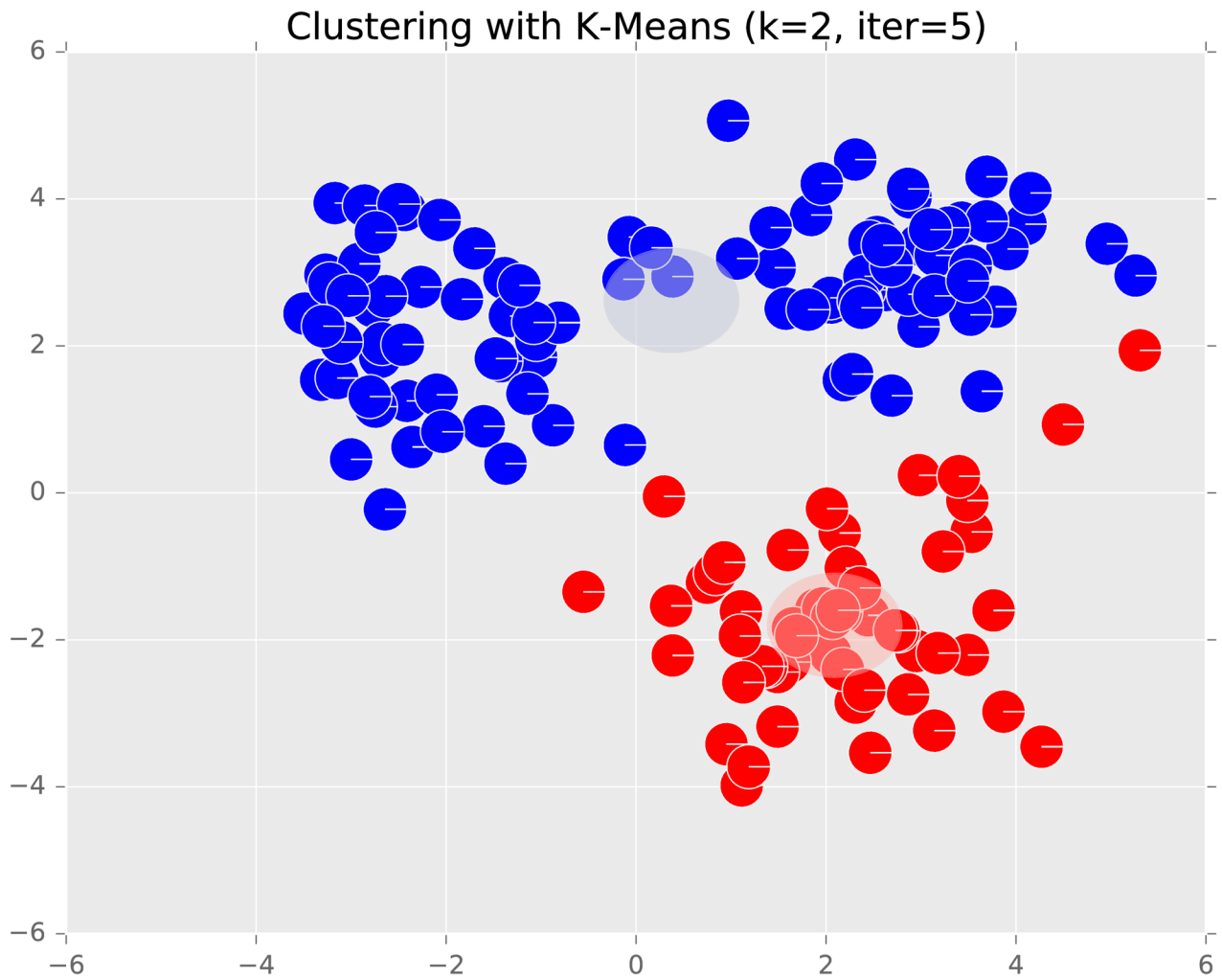
Example: K-Means



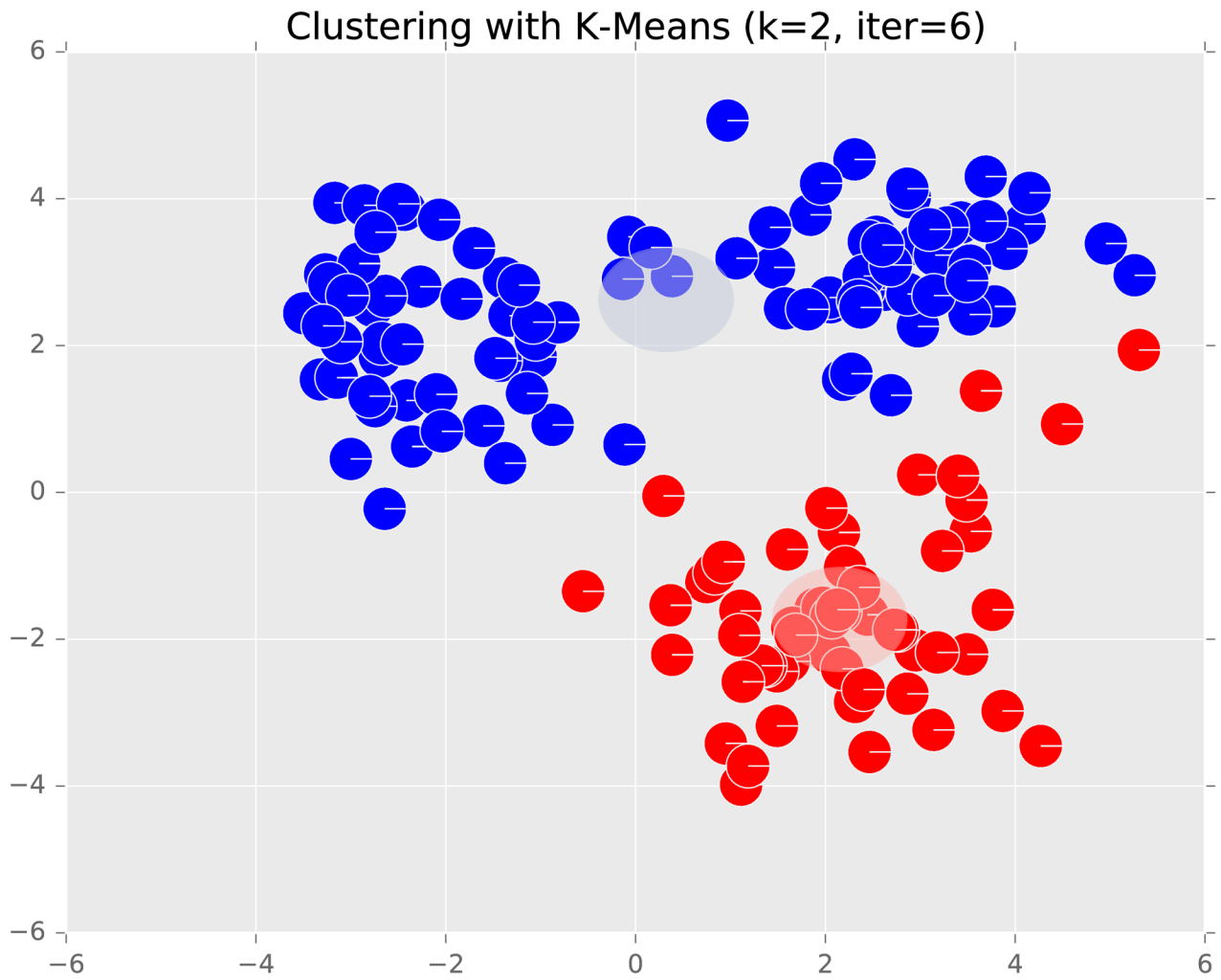
Example: K-Means



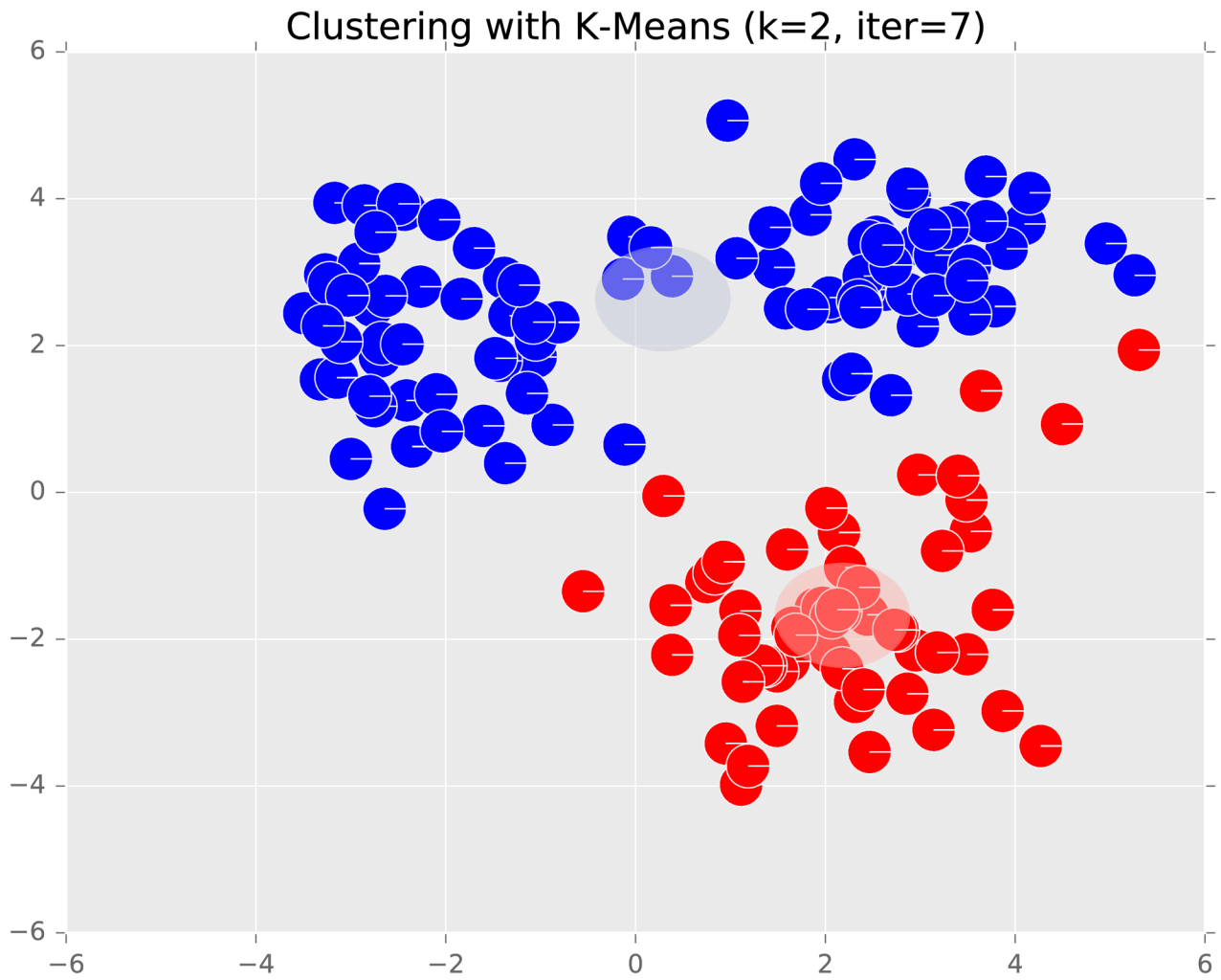
Example: K-Means



Example: K-Means



Example: K-Means



INITIALIZING K-MEANS

Initialization of K-Means

K-Means Algorithm

1) **Given** unlabeled feature vectors
   $D = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(N)}\}$
2) **Initialize** cluster centers $c = \{\mathbf{c}_1, \ldots, \mathbf{c}_K\}$
3) **Repeat** until conv
   a) for i in {1,…, N}
      $z^{(i)} \leftarrow$ inde
   b) for j in {1,…,K}
      $\mathbf{c}_j \leftarrow$ mean

Remaining Question:
How should we initialize the cluster centers?

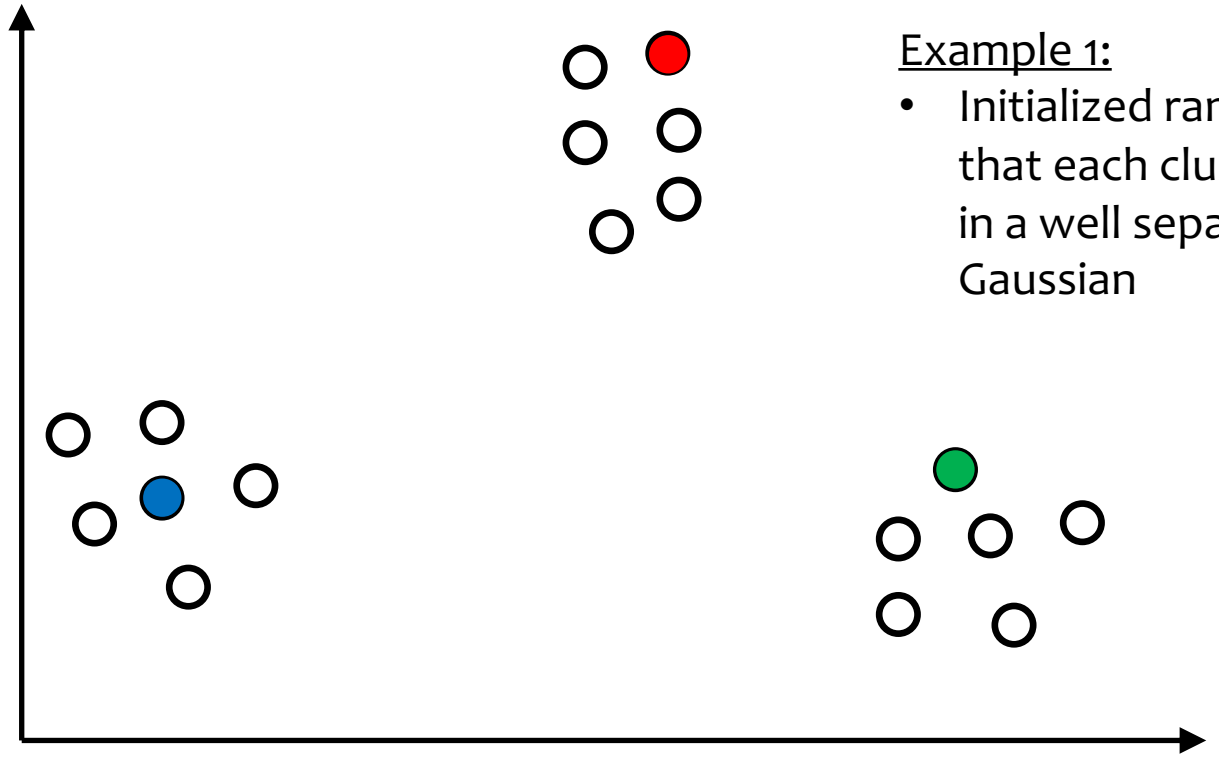
Three Solutions:

1. Random centers (picked from the data points)
2. Furthest point heuristic
3. K-Means++

Initialization for K-Means

Algorithm #1: Random Initialization
Select each cluster center uniformly at random from the data points in the training data

Observations:
Even when data comes from well-separated Gaussians…
- …sometimes works great!
- …sometimes get stuck in poor local optima.

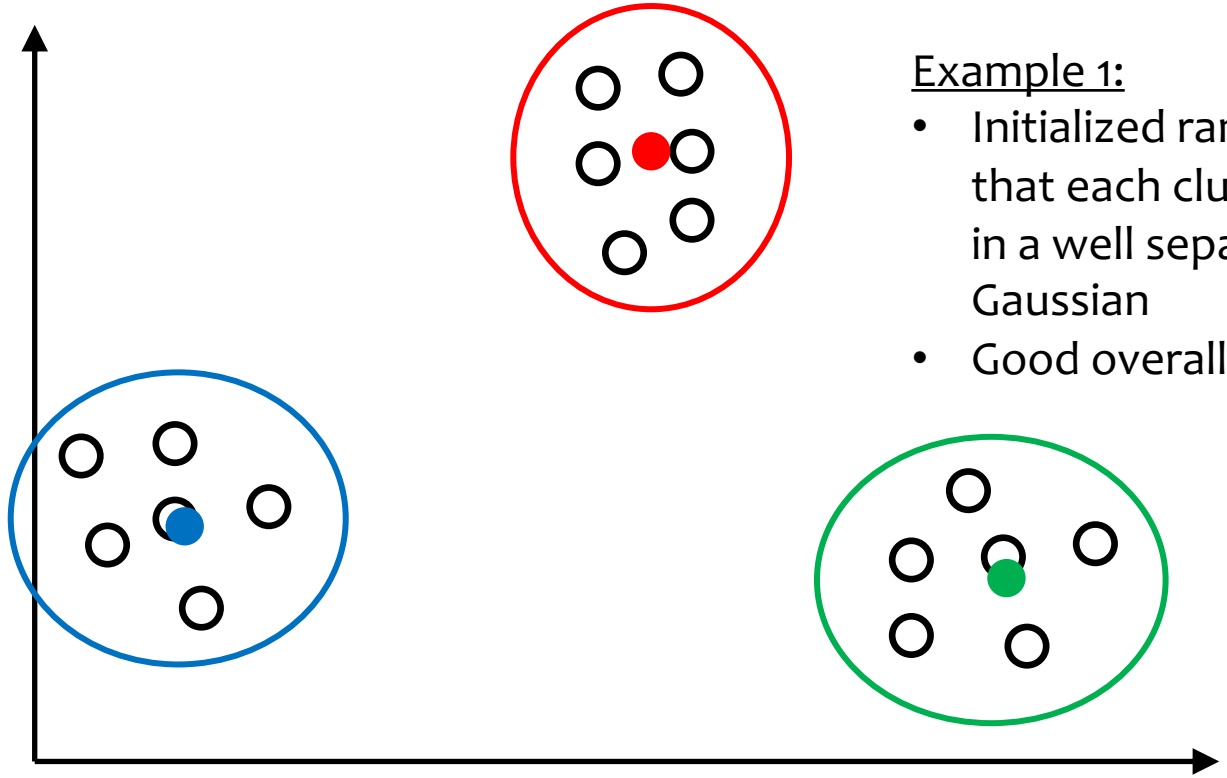


Example 1:
- Initialized randomly such that each cluster center is in a well separated Gaussian

Initialization for K-Means

Algorithm #1: Random Initialization
Select each cluster center uniformly at random from the data points in the training data

Observations:
Even when data comes from well-separated Gaussians…
- …sometimes works great!
- …sometimes get stuck in poor local optima.

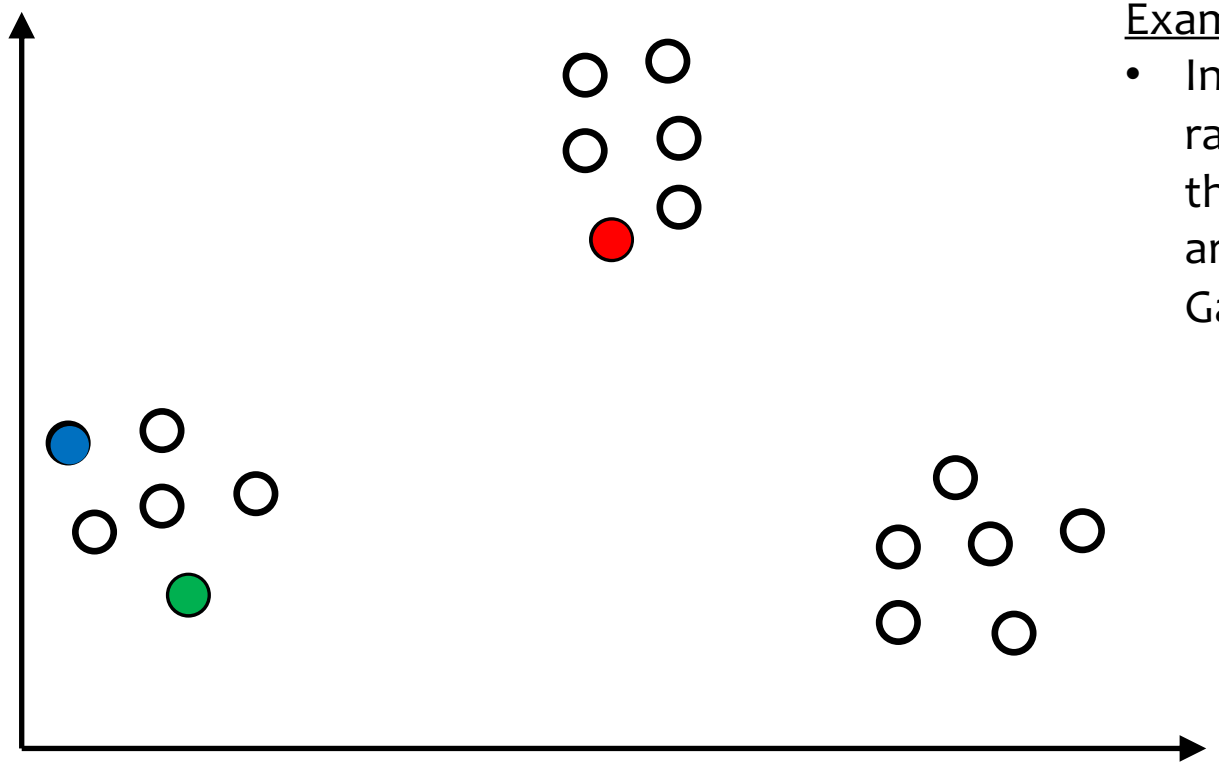


Example 1:
- Initialized randomly such that each cluster center is in a well separated Gaussian
- Good overall performance

Initialization for K-Means

Algorithm #1: Random Initialization
Select each cluster center uniformly at random from the data points in the training data

Observations:
Even when data comes from well-separated Gaussians…
- …sometimes works great!
- …sometimes get stuck in poor local optima.

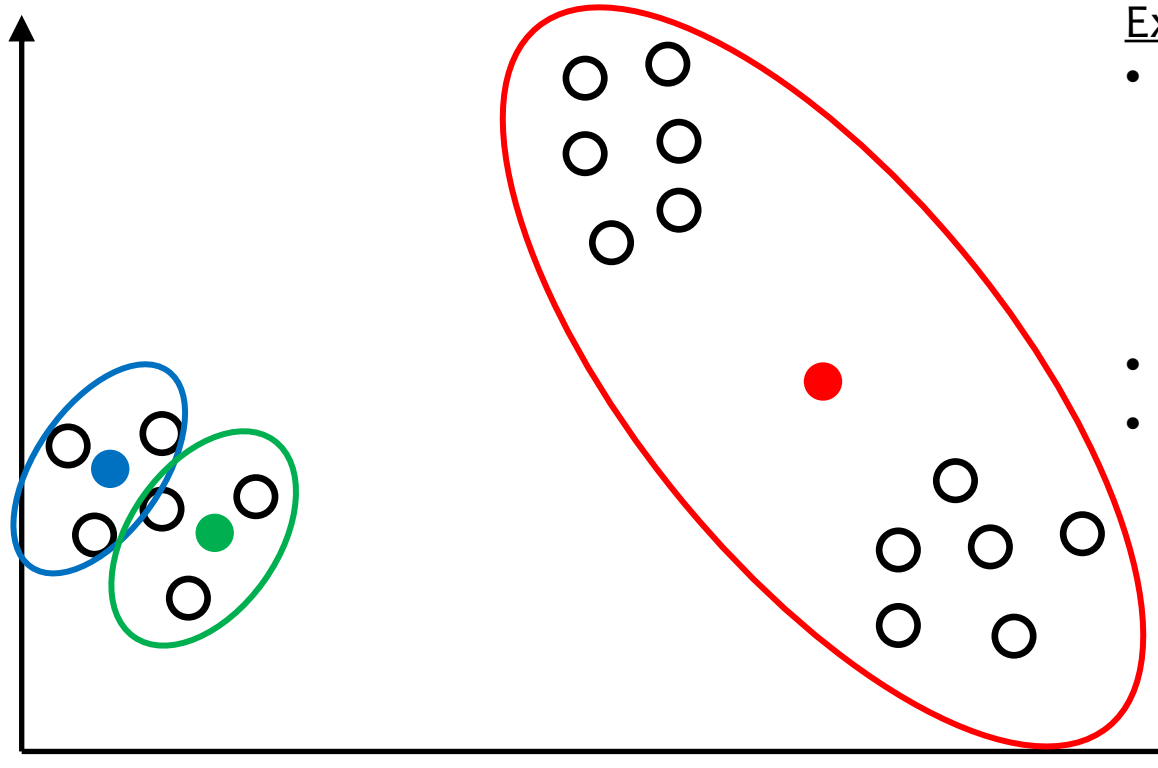


Example 2:
- Initialized randomly such that two centers are in the same Gaussian cluster

Initialization for K-Means

Algorithm #1: Random Initialization
Select each cluster center uniformly at random from the data points in the training data

Observations:
Even when data comes from well-separated Gaussians…
- …sometimes works great!
- …sometimes get stuck in poor local optima.



Example 2:
- Initialized randomly such that two centers are in the same Gaussian cluster
- Poor performance
- Can be **arbitrarily bad** (imagine the final red cluster points moving arbitrarily far away!)

Initialization for K-Means

K-Mean Performance (with Random Initialization)

If we do **random initialization**, as k increases, it becomes more likely we won't have perfectly picked one center per Gaussian in our initialization (so K-Means will output a bad solution).

- For k equal-sized Gaussians,

$$\Pr[\text{each initial center is in a different Gaussian}] \approx \frac{k!}{k^k} \approx \frac{1}{e^k}$$

- Becomes unlikely as k gets large.

Slide courtesy of Nina Balcan

Initialization for K-Means

Algorithm #2: Furthest Point Heuristic

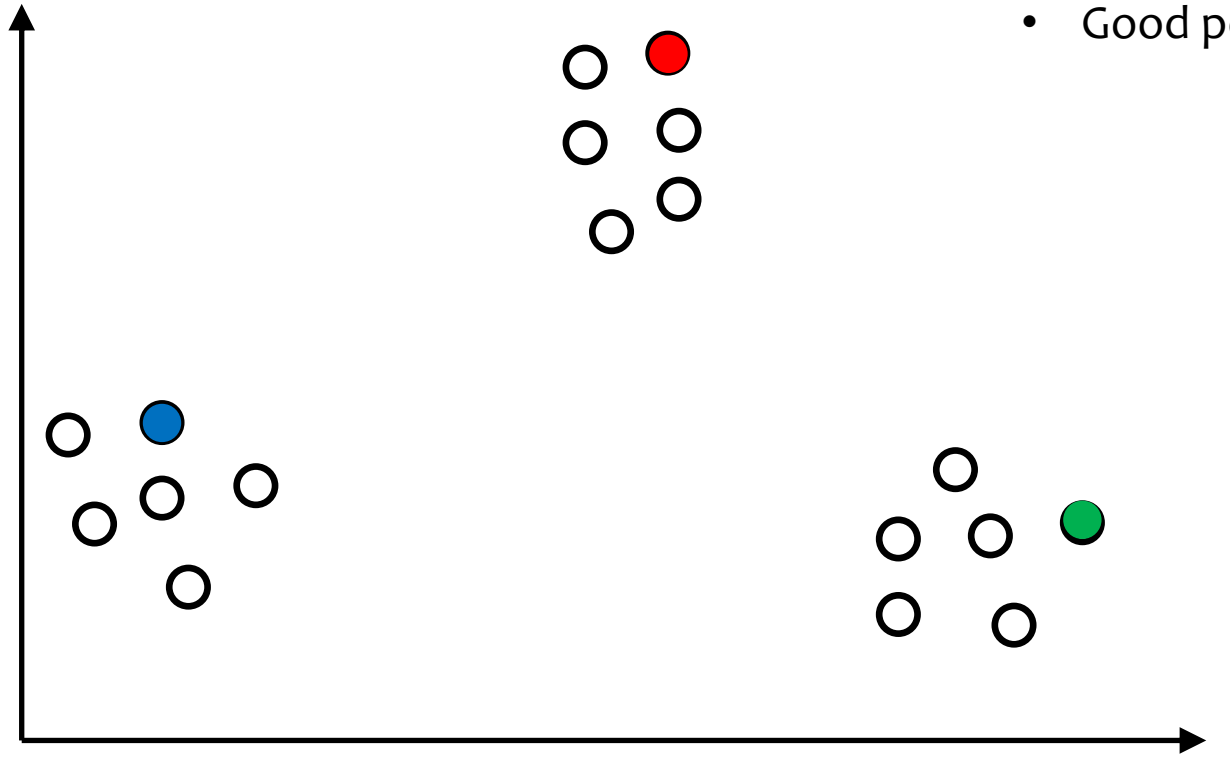
1. Pick the first cluster center $\mathbf{c}_1$ **randomly**
2. Pick each subsequent center $\mathbf{c}_j$ so that it is **as far as possible** from the previously chosen centers $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_{j-1}$

Observations:

- Solves the problem with Gaussian data
- But outliers pose a new problem!

Example 1:

- No outliers
- Good performance



Initialization for K-Means

Algorithm #2: Furthest Point Heuristic

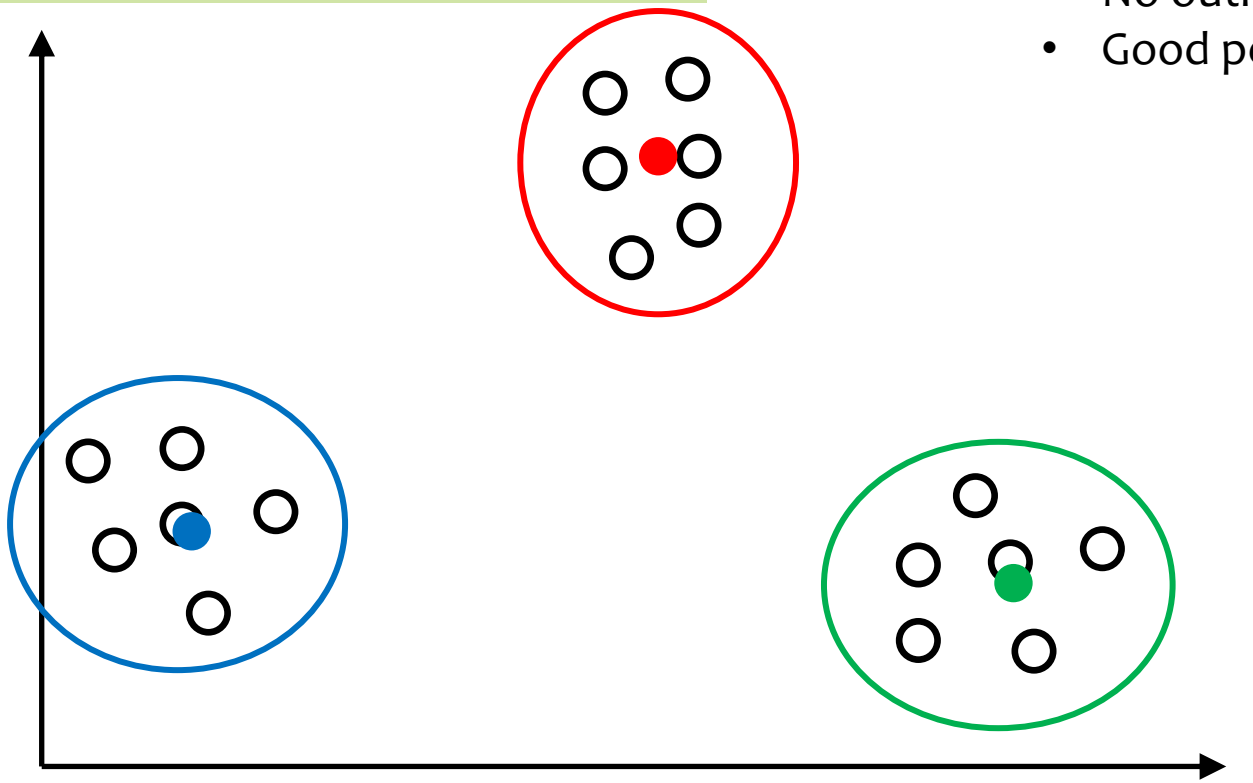
1. Pick the first cluster center $\mathbf{c}_1$ **randomly**
2. Pick each subsequent center $\mathbf{c}_j$ so that it is **as far as possible** from the previously chosen centers $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_{j-1}$

Observations:

- Solves the problem with Gaussian data
- But outliers pose a new problem!

Example 1:

- No outliers
- Good performance



Initialization for K-Means

Algorithm #2: Furthest Point Heuristic

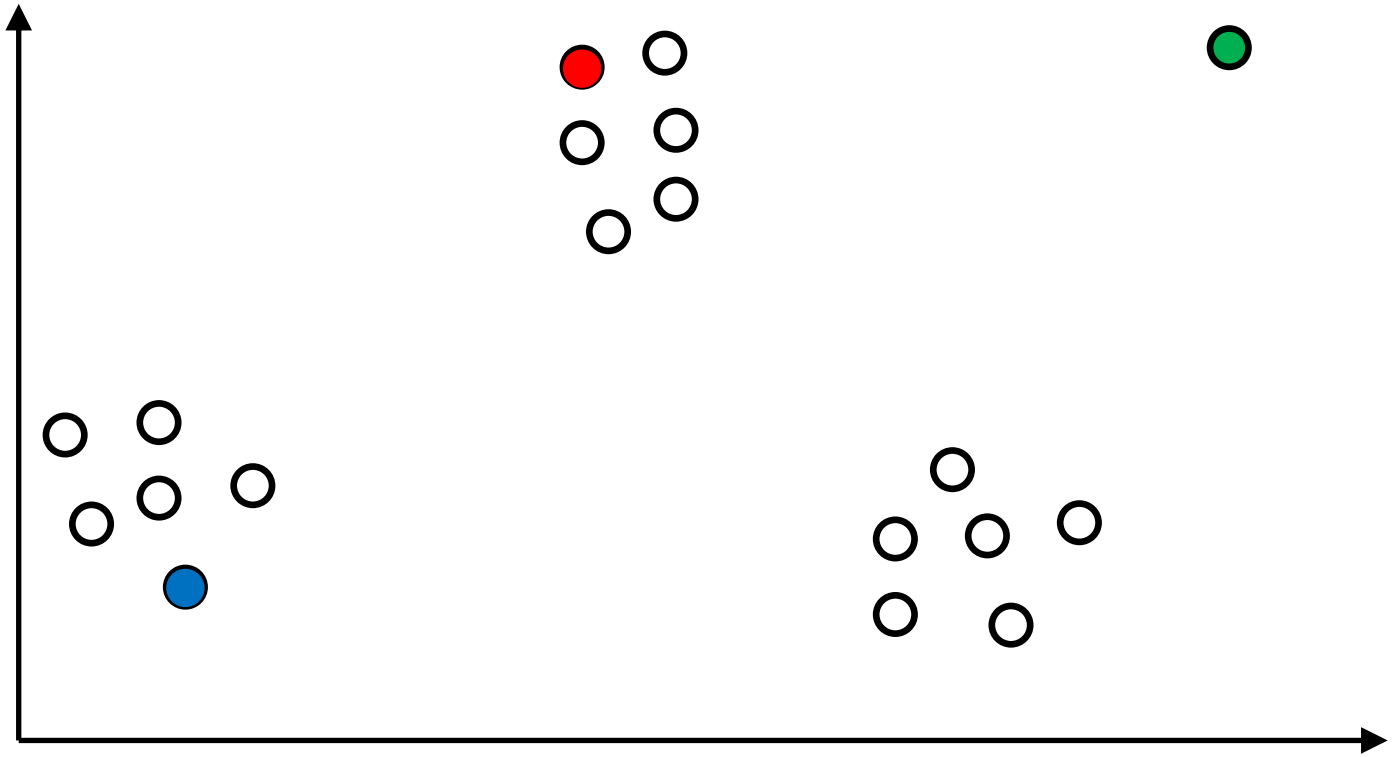
1. Pick the first cluster center $\mathbf{c}_1$ **randomly**
2. Pick each subsequent center $\mathbf{c}_j$ so that it is **as far as possible** from the previously chosen centers $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_{j-1}$

Observations:

- Solves the problem with Gaussian data
- But outliers pose a new problem!

Example 2:

- One outlier throws off the algorithm
- Poor performance



Initialization for K-Means

Algorithm #2: Furthest Point Heuristic

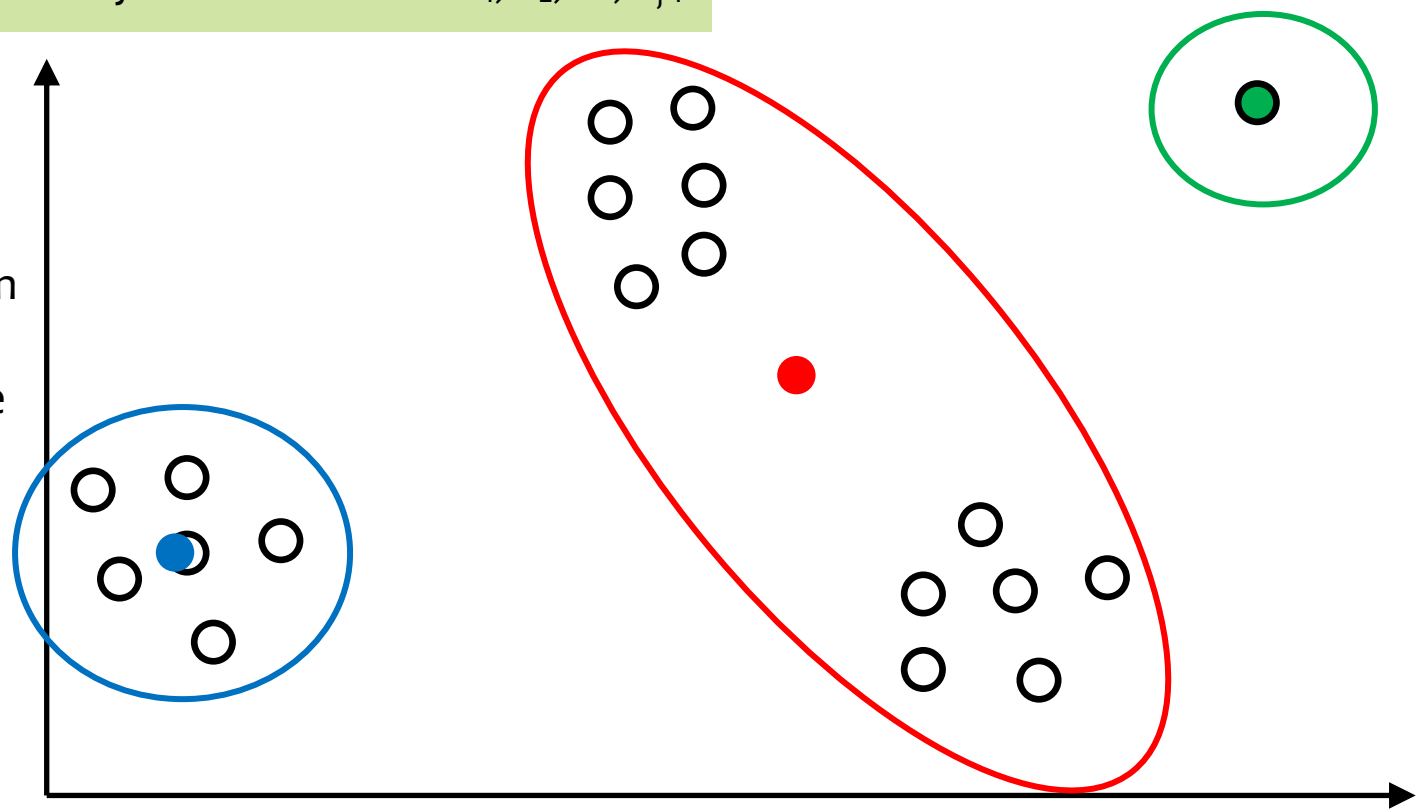
1. Pick the first cluster center $\mathbf{c}_1$ **randomly**
2. Pick each subsequent center $\mathbf{c}_j$ so that it is **as far as possible** from the previously chosen centers $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_{j-1}$

Observations:

- Solves the problem with Gaussian data
- But outliers pose a new problem!

Example 2:

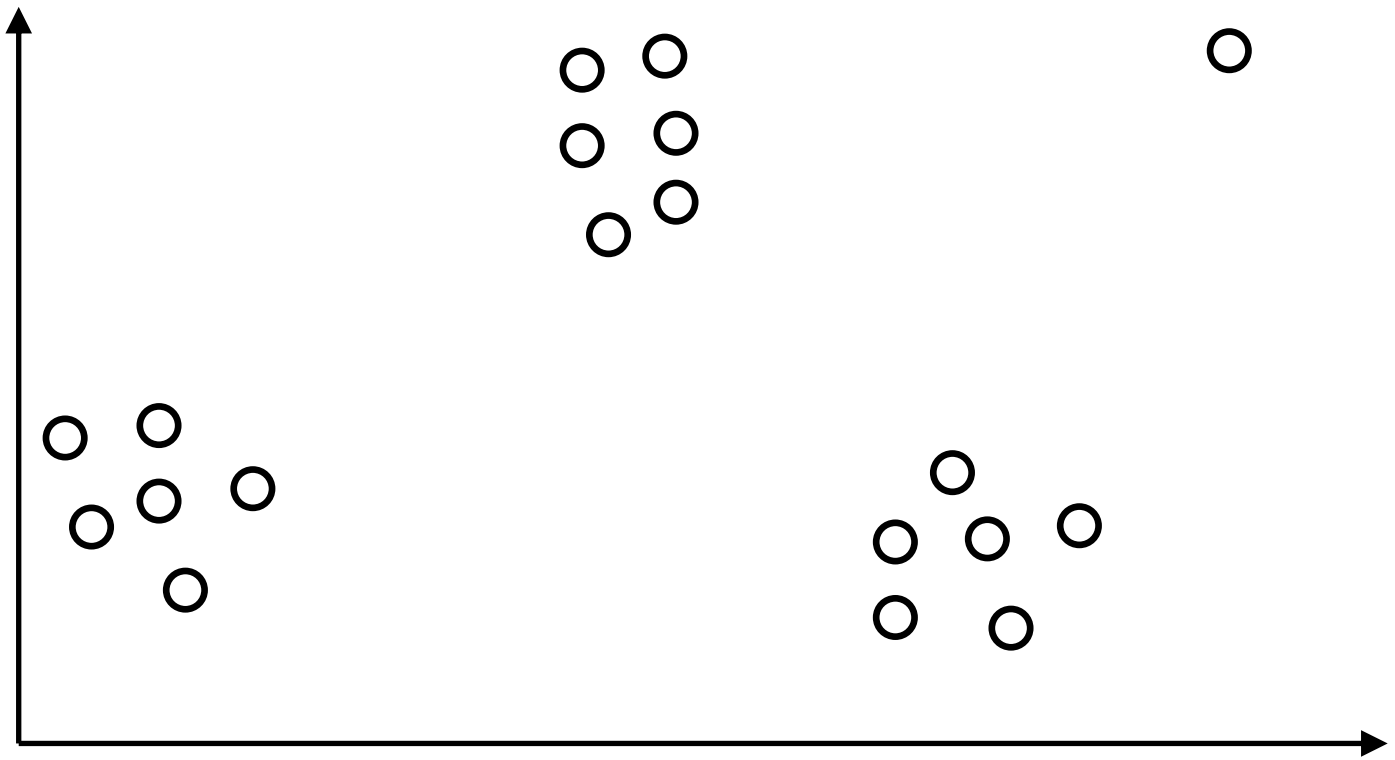
- One outlier throws off the algorithm
- Poor performance



Initialization for K-Means

Algorithm #3: K-Means++

- Let $D(\mathbf{x})$ be the distance between a point x and its nearest center. Chose the next center proportional to $D^2(\mathbf{x})$.



Initialization for K-M

i	D(x)	$D^2(x)$	$P(c_2 = x^{(i)})$
1	3	9	9/137
2	2	4	4/137
…			
7	4	16	16/137
…			
N	3	9	9/137
	Sum:	137	1.0

Algorithm #3: K-Means++

- Let $D(\mathbf{x})$ be the distance between a point x and its nearest center. Chose the next center proportional to $D^2(\mathbf{x})$.

- Choose $\mathbf{c_1}$ at random.
- For $j = 2, \ldots, K$
  - Pick $\mathbf{c_j}$ among $\mathbf{x^{(1)}}, \mathbf{x^{(2)}}, \ldots, \mathbf{x^{(n)}}$ according to the distribution

$$\mathbf{P(c_j = x^{(i)})} \propto \mathbf{min_{j'<j}} \left|\left|\mathbf{x^{(i)} - c_{j'}}\right|\right|^2 \; D^2(\mathbf{x^i})$$

Theorem: K-Means++ always attains an $O(\log k)$ approximation to optimal K-Means solution in expectation.

Slide adapted from Nina Balcan

Initialization for K-M

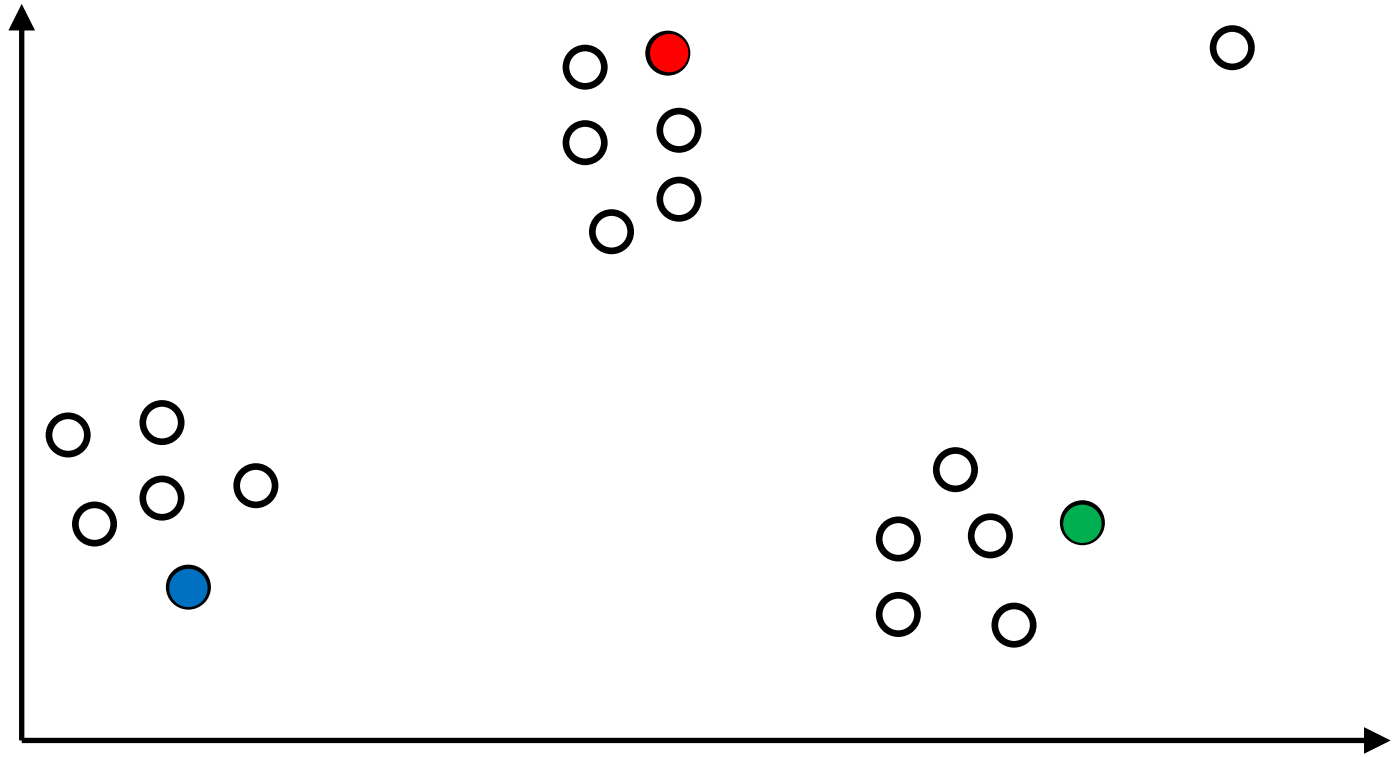
Algorithm #3: K-Means++

- Let $D(\mathbf{x})$ be the distance between a point x and its nearest center. Chose the next center proportional to $D^2(\mathbf{x})$.

i	D(x)	$D^2(x)$	$P(c_2 = x^{(i)})$
1	3	9	9/137
2	2	4	4/137
…			
7	4	16	16/137
…			
N	3	9	9/137
	Sum:	137	1.0

Example 1:

- One outlier
- Good performance



Initialization for K-M

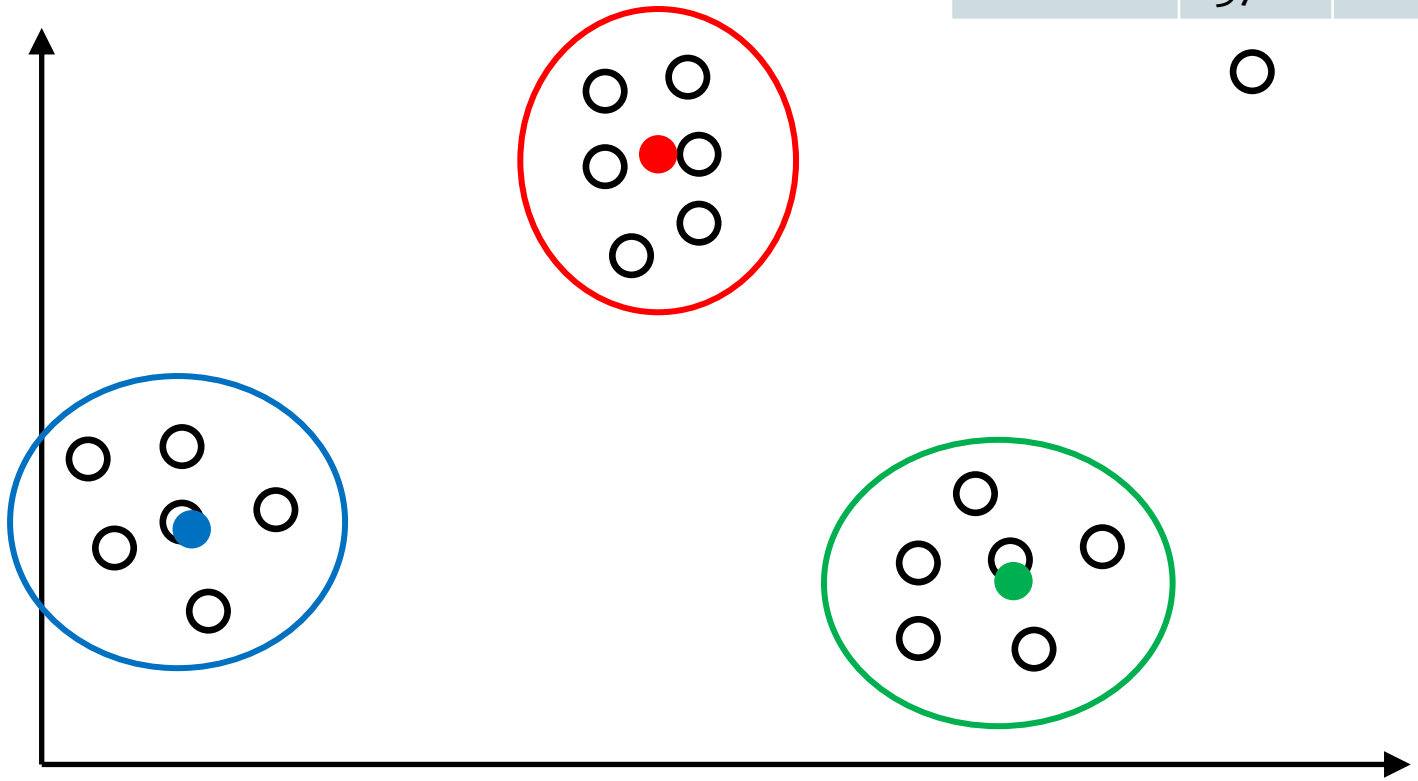
Algorithm #3: K-Means++

- Let $D(\mathbf{x})$ be the distance between a point x and its nearest center. Chose the next center proportional to $D^2(\mathbf{x})$.

i	D(x)	$D^2(x)$	$P(c_2 = x^{(i)})$
1	3	9	9/137
2	2	4	4/137
…			
7	4	16	16/137
…			
N	3	9	9/137
	Sum:	137	1.0

Example 1:

- One outlier
- Good performance



Initialization for K-Means

Algorithm #3: K-Means++

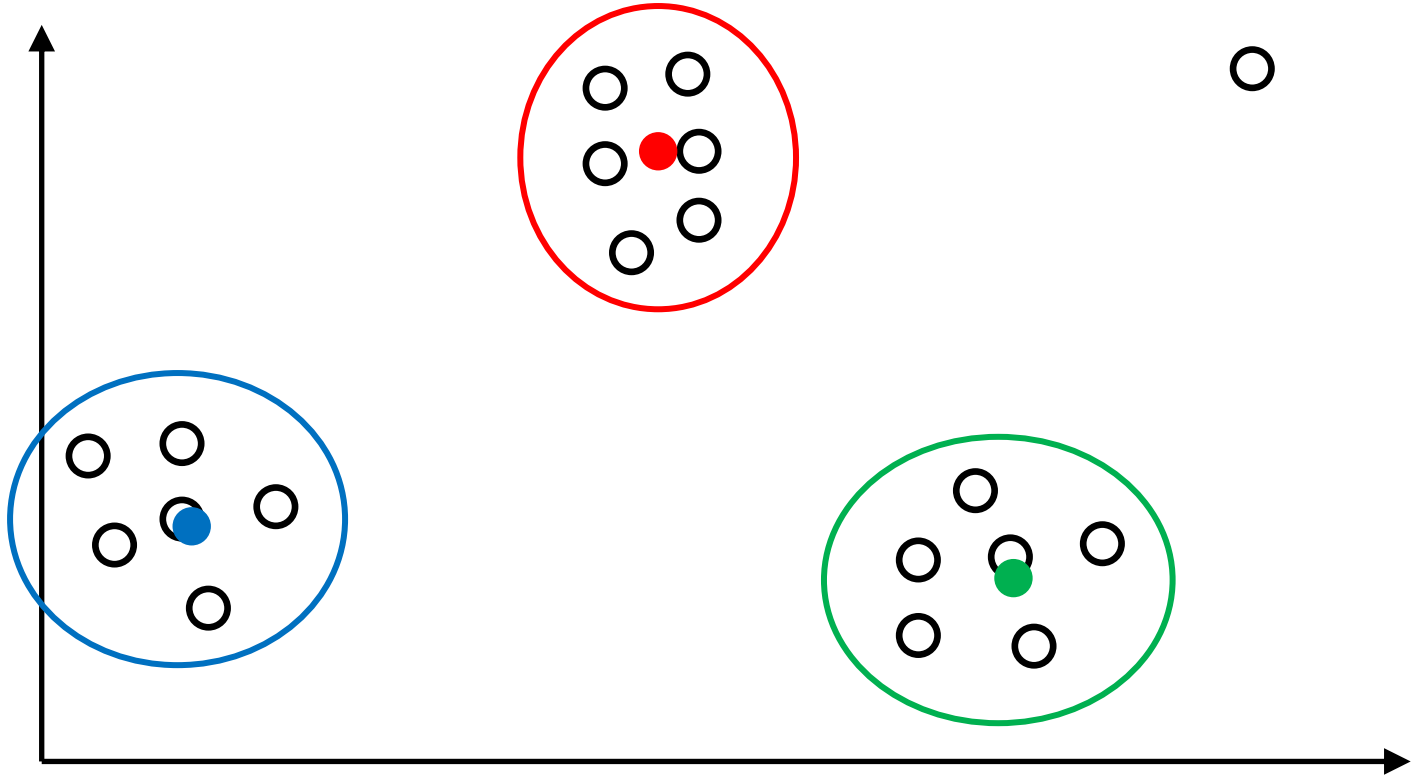
- Let $D(\mathbf{x})$ be the distance between a point x and its nearest center. Chose the next center proportional to $D^2(\mathbf{x})$.

Observations:

- Interpolates between random and farthest point initialization
- Solves the problem with Gaussian data
- **And** solves the outlier problem

Example 1:

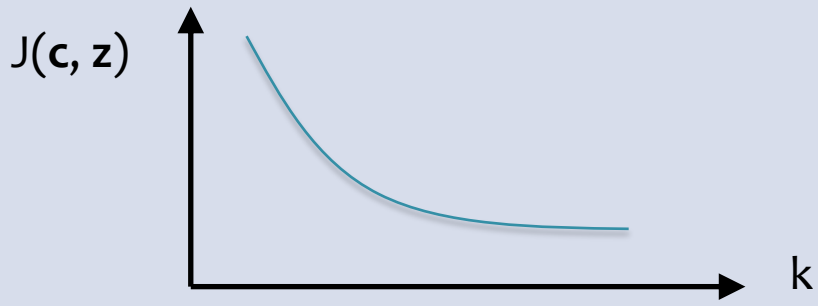
- One outlier
- Good performance



Q&A

Q: In k-Means, since we don't have a validation set, how do we pick k?

A: Look at the training objective function as a function of k and pick the value at the "elbo" of the curve.



Q: What if our random initialization for k-Means gives us poor performance?

A: Do **random restarts**: that is, run k-means from scratch, say, 10 times and pick the run that gives the lowest training objective function value.

The objective function is **nonconvex**, so we're just looking for the best local minimum.

Learning Objectives

K-Means

You should be able to…

1. Distinguish between coordinate descent and block coordinate descent
2. Define an objective function that gives rise to a "good" clustering
3. Apply block coordinate descent to an objective function preferring each point to be close to its nearest objective function to obtain the K-Means algorithm
4. Implement the K-Means algorithm
5. Connect the non-convexity of the K-Means objective function with the (possibly) poor performance of random initialization