



10-601 Introduction to Machine Learning

Machine Learning Department
School of Computer Science
Carnegie Mellon University

Deep RL + K-Means

Matt Gormley
Lecture 25
Apr. 15, 2020

Reminders

- **Homework 8: Reinforcement Learning**
 - Out: Fri, Apr 10
 - Due: Wed, Apr 22 at 11:59pm
- **Homework 9: Learning Paradigms**
 - Out: Wed, Apr. 22
 - Due: Wed, Apr. 29 at 11:59pm
 - Can only be submitted up to 3 days late, so we can return grades before final exam
- **Today's In-Class Poll**
 - <http://poll.mlcourse.org>

DEEP RL EXAMPLES

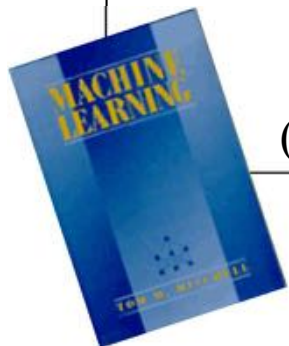
TD Gammon → Alpha Go

Learning to beat the masters at board games

THEN

“...the world’s top computer program for backgammon, TD-GAMMON (Tesauro, 1992, 1995), learned its strategy by playing over one million practice games against itself...”

(Mitchell, 1997)

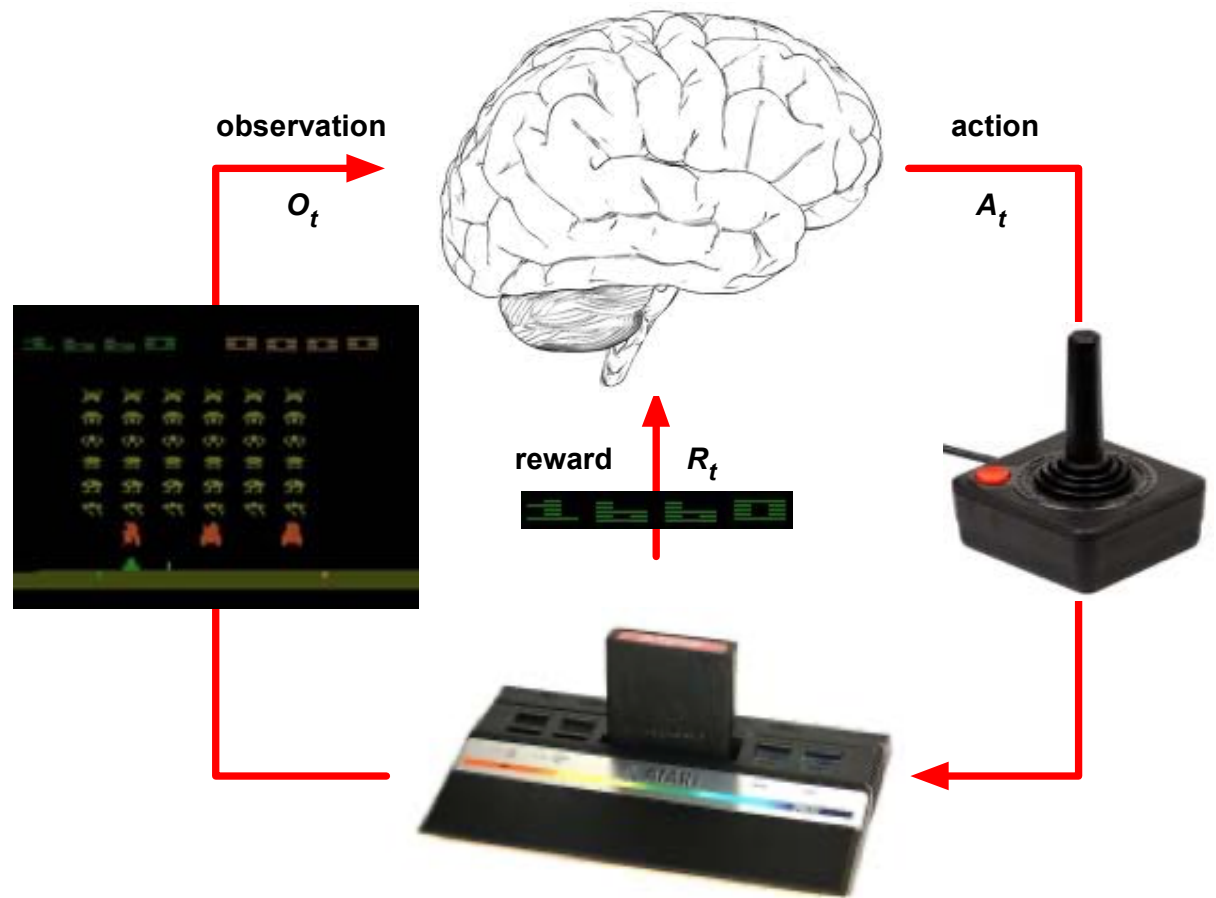


NOW



Playing Atari with Deep RL

- Setup: RL system observes the pixels on the screen
- It receives rewards as the game score
- Actions decide how to move the joystick / buttons



Playing Atari with Deep RL



Figure 1: Screen shots from five Atari 2600 Games: (Left-to-right) Pong, Breakout, Space Invaders, Seaquest, Beam Rider

Videos:

- Atari Breakout:

<https://www.youtube.com/watch?v=V1eYniJoRnk>

- Space Invaders:

<https://www.youtube.com/watch?v=ePvoFs9cGgU>

Playing Atari with Deep RL



Figure 1: Screen shots from five Atari 2600 Games: (Left-to-right) Pong, Breakout, Space Invaders, Seaquest, Beam Rider

	B. Rider	Breakout	Enduro	Pong	Q*bert	Seaquest	S. Invaders
Random	354	1.2	0	-20.4	157	110	179
Sarsa [3]	996	5.2	129	-19	614	665	271
Contingency [4]	1743	6	159	-17	960	723	268
DQN	4092	168	470	20	1952	1705	581
Human	7456	31	368	-3	18900	28010	3690
HNeat Best [8]	3616	52	106	19	1800	920	1720
HNeat Pixel [8]	1332	4	91	-16	1325	800	1145
DQN Best	5184	225	661	21	4500	1740	1075

Table 1: The upper table compares average total reward for various learning methods by running an ϵ -greedy policy with $\epsilon = 0.05$ for a fixed number of steps. The lower table reports results of the single best performing episode for HNeat and DQN. HNeat produces deterministic policies that always get the same score while DQN used an ϵ -greedy policy with $\epsilon = 0.05$.

Deep Q-Learning

Question: *What if our state space S is too large to represent with a table?*

Examples:

- s_t = pixels of a video game
- s_t = continuous values of a sensors in a manufacturing robot
- s_t = sensor output from a self-driving car

Answer: Use a parametric function to approximate the table entries

Key Idea:

1. Use a neural network $Q(s,a; \theta)$ to approximate $Q^*(s,a)$
2. Learn the parameters θ via SGD with training examples $\langle s_t, a_t, r_t, s_{t+1} \rangle$

Deep Q-Learning

Whiteboard

- Strawman loss function (i.e. what we cannot compute)
- Approximating the Q function with a neural network
- Approximating the Q function with a linear model
- Deep Q-Learning
- function approximators
($\langle \text{state}, \text{action}_i \rangle \rightarrow \text{q-value}$
vs.
state $\rightarrow$ all action q-values)

Experience Replay

- **Problems** with online updates for Deep Q-learning:
 - not i.i.d. as SGD would assume
 - quickly forget rare experiences that might later be useful to learn from
- **Uniform Experience Replay** (Lin, 1992):
 - Keep a *replay memory* $D = \{e_1, e_2, \dots, e_N\}$ of N most recent experiences $e_t = \langle s_t, a_t, r_t, s_{t+1} \rangle$
 - Alternate two steps:
 1. Repeat T times: randomly sample e_i from D and apply a Q-Learning update to e_i
 2. Agent selects an action using epsilon greedy policy to receive new experience that is added to D
- **Prioritized Experience Replay** (Schaul et al, 2016)
 - similar to Uniform ER, but sample so as to prioritize experiences with high error

Alpha Go

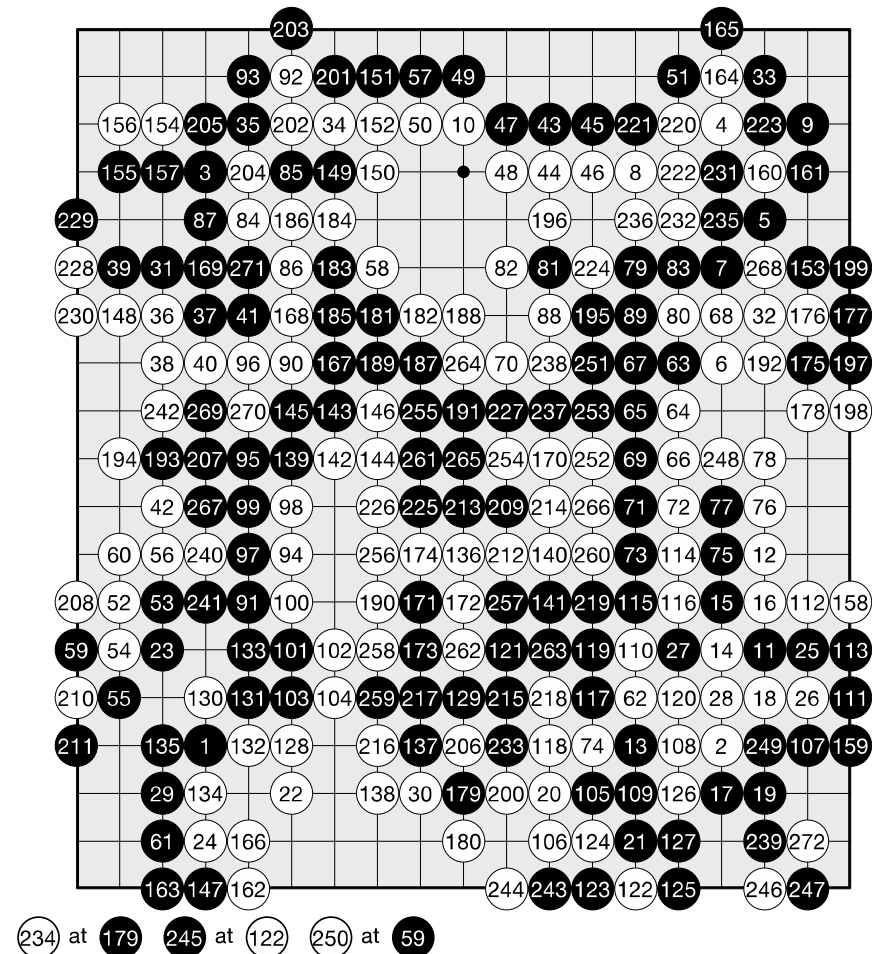
Game of Go (圍棋)

- 19x19 board
- Players alternately play black/white stones
- **Goal** is to fully encircle the largest region on the board
- **Simple** rules, but **extremely complex** game play

Game 1

Fan Hui (Black), AlphaGo (White)

AlphaGo wins by 2.5 points



Alpha Go

- State space is too large to represent explicitly since # of sequences of moves is $O(b^d)$
 - Go: $b=250$ and $d=150$
 - Chess: $b=35$ and $d=80$
- Key idea:
 - Define a neural network to approximate the value function
 - Train by policy gradient

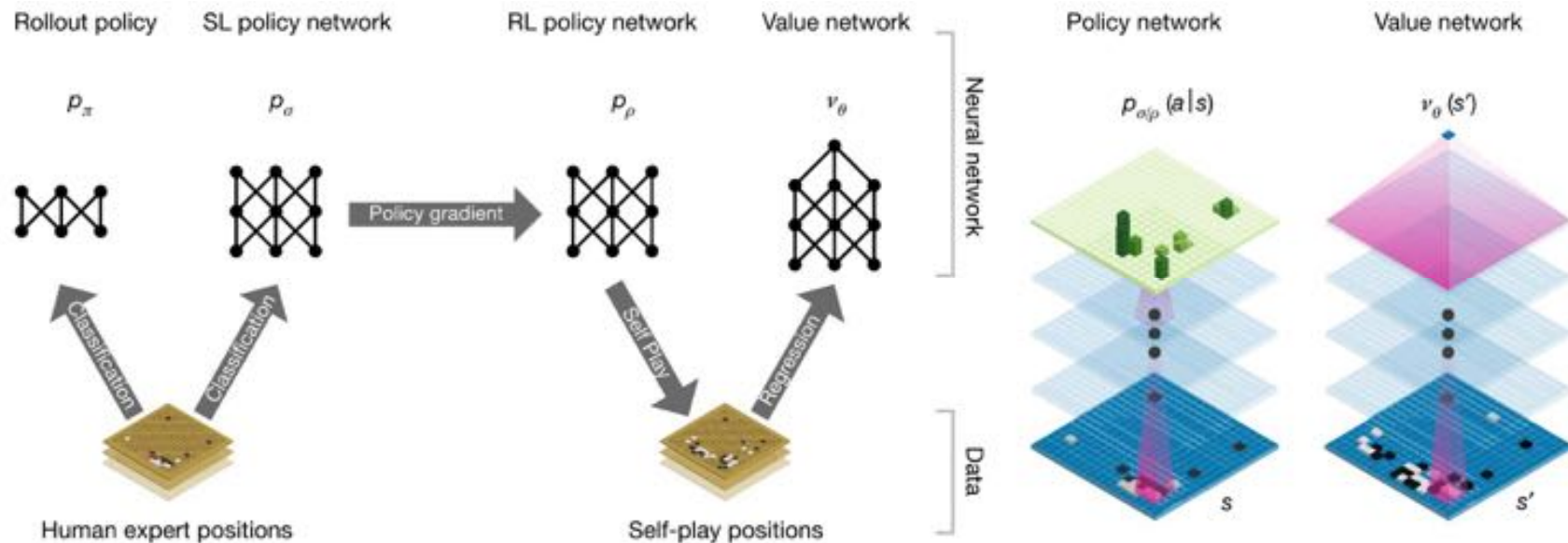
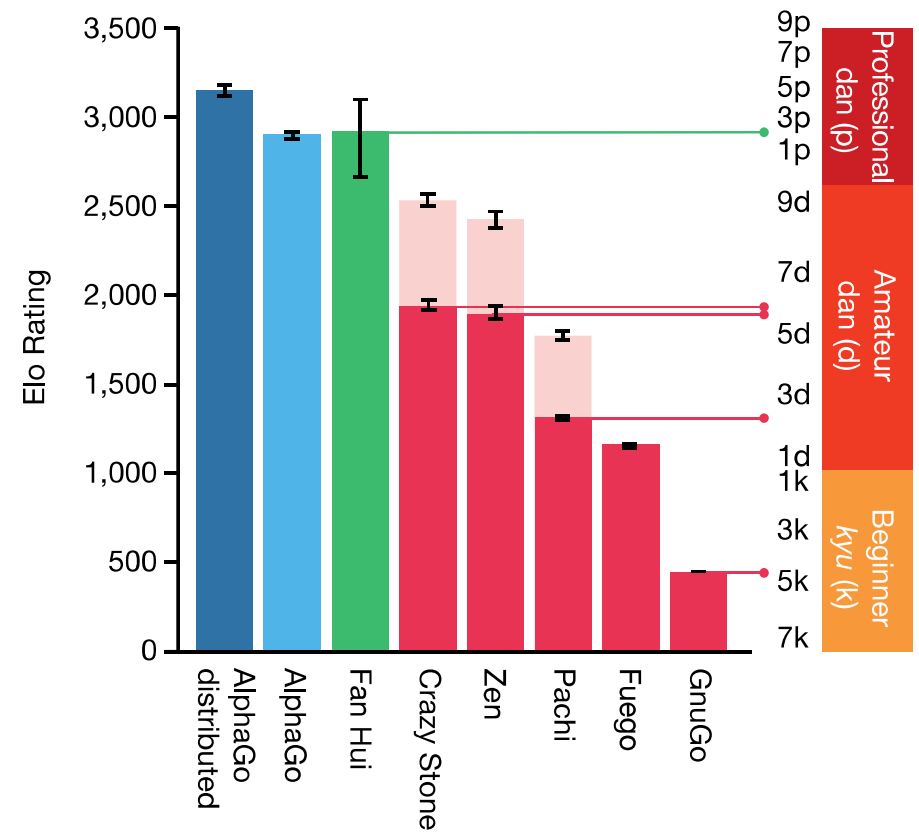


Figure from Silver et al. (2016)

Alpha Go

- Results of a tournament
- From Silver et al. (2016): “a 230 point gap corresponds to a 79% probability of winning”



Learning Objectives

Reinforcement Learning: Q-Learning

You should be able to...

1. Apply Q-Learning to a real-world environment
2. Implement Q-learning
3. Identify the conditions under which the Q-learning algorithm will converge to the true value function
4. Adapt Q-learning to Deep Q-learning by employing a neural network approximation to the Q function
5. Describe the connection between Deep Q-Learning and regression

Q1

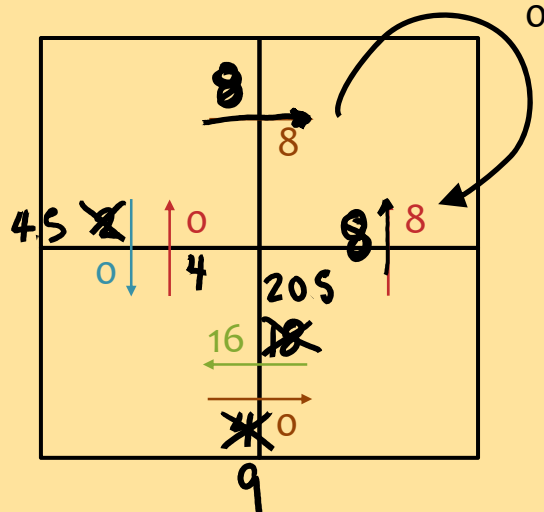
Q-Learning *A = calamity*

$$Q(s,a) \leftarrow r + (0.5) \max_{a'} Q(s',a')$$

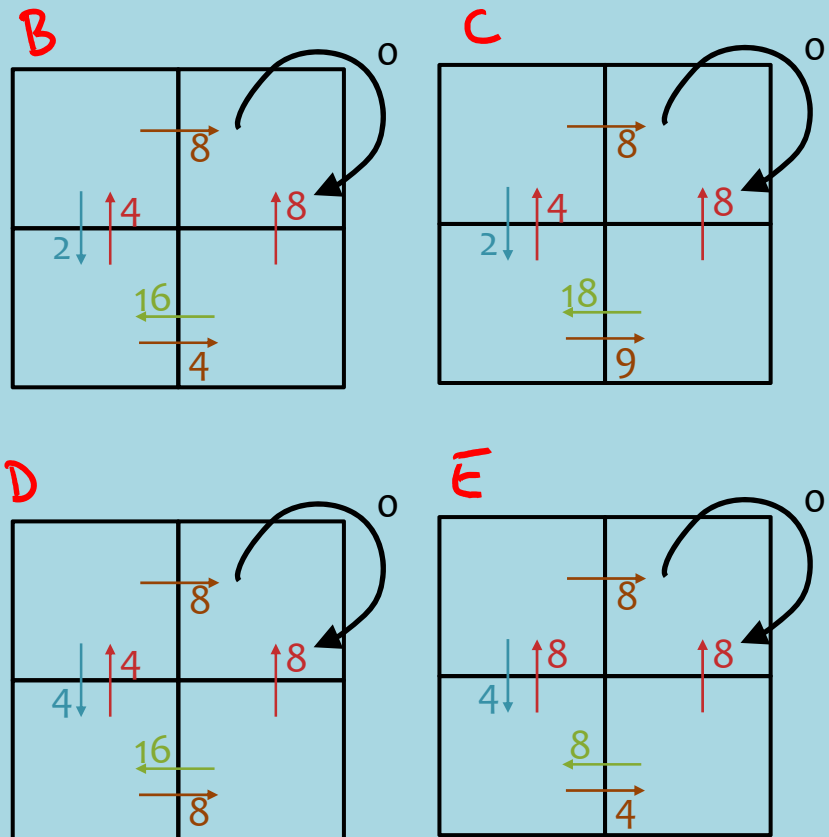
Question:

For the $R(s,a)$ values shown on the arrows below, which are the corresponding $Q^*(s,a)$ values?

Assume discount factor = 0.5.



Answer:



F = None of the above

ML Big Picture

Learning Paradigms:

What data is available and when? What form of prediction?

- supervised learning
- unsupervised learning
- semi-supervised learning
- reinforcement learning
- active learning
- imitation learning
- domain adaptation
- online learning
- density estimation
- recommender systems
- feature learning
- manifold learning
- dimensionality reduction
- ensemble learning
- distant supervision
- hyperparameter optimization

Theoretical Foundations:

What principles guide learning?

- ☐ probabilistic
- ☐ information theoretic
- ☐ evolutionary search
- ☒ ML as optimization

Problem Formulation:

What is the structure of our output prediction?

boolean	Binary Classification
categorical	Multiclass Classification
ordinal	Ordinal Classification
real	Regression
ordering	Ranking
multiple discrete	Structured Prediction
multiple continuous	(e.g. dynamical systems)
both discrete & cont.	(e.g. mixed graphical models)

Facets of Building ML Systems:

How to build systems that are robust, efficient, adaptive, effective?

1. Data prep
2. Model selection
3. Training (optimization / search)
4. Hyperparameter tuning on validation data
5. (Blind) Assessment on test data

Big Ideas in ML:

Which are the ideas driving development of the field?

- inductive bias
- generalization / overfitting
- bias-variance decomposition
- generative vs. discriminative
- deep nets, graphical models
- PAC learning
- distant rewards

Application Areas

Key challenges?

NLP, Speech, Computer Vision, Robotics, Medicine, Search

Learning Paradigms

Paradigm	Data
Supervised	$\mathcal{D} = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^N \quad \mathbf{x} \sim p^*(\cdot) \text{ and } y = c^*(\cdot)$
$\hookrightarrow$ Regression	$y^{(i)} \in \mathbb{R}$
$\hookrightarrow$ Classification	$y^{(i)} \in \{1, \dots, K\}$
$\hookrightarrow$ Binary classification	$y^{(i)} \in \{+1, -1\}$
$\hookrightarrow$ Structured Prediction	$\mathbf{y}^{(i)}$ is a vector
Unsupervised	$\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^N \quad \mathbf{x} \sim p^*(\cdot)$
Semi-supervised	$\mathcal{D} = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^{N_1} \cup \{\mathbf{x}^{(j)}\}_{j=1}^{N_2}$
Online	$\mathcal{D} = \{(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), (\mathbf{x}^{(3)}, y^{(3)}), \dots\}$
Active Learning	$\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^N$ and can query $y^{(i)} = c^*(\cdot)$ at a cost
Imitation Learning	$\mathcal{D} = \{(s^{(1)}, a^{(1)}), (s^{(2)}, a^{(2)}), \dots\}$
Reinforcement Learning	$\mathcal{D} = \{(s^{(1)}, a^{(1)}, r^{(1)}), (s^{(2)}, a^{(2)}, r^{(2)}), \dots\}$

K-Means Outline

- **Clustering: Motivation / Applications**
- **Optimization Background**
 - Coordinate Descent
 - Block Coordinate Descent
- **Clustering**
 - Inputs and Outputs
 - Objective-based Clustering
- **K-Means**
 - K-Means Objective
 - Computational Complexity
 - K-Means Algorithm / Lloyd's Method
- **K-Means Initialization**
 - Random
 - Farthest Point
 - K-Means++

CLUSTERING

Clustering, Informal Goals

Goal: Automatically partition **unlabeled** data into groups of similar datapoints.

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

Useful for:

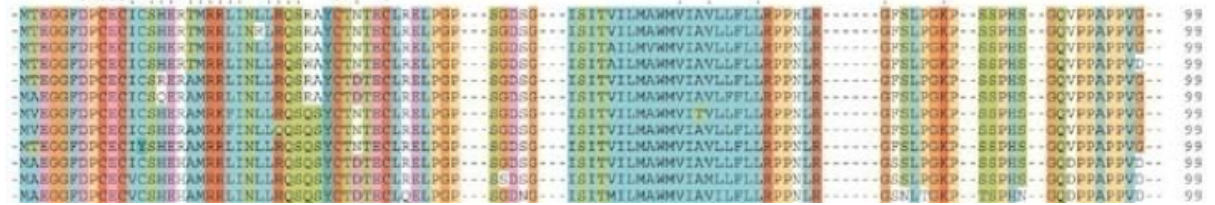
- Automatically organizing data.
- Understanding hidden structure in data.
- Preprocessing for further analysis.
 - Representing high-dimensional data in a low-dimensional space (e.g., for visualization purposes).

Applications (Clustering comes up everywhere...)

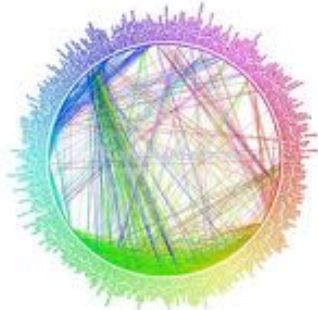
- Cluster news articles or web pages or search results by topic.



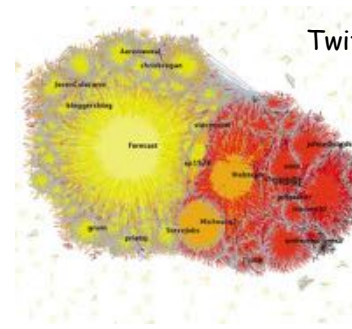
- Cluster protein sequences by function or genes according to expression profile.



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



Facebook network



Twitter Network

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....

Optimization Background

Whiteboard:

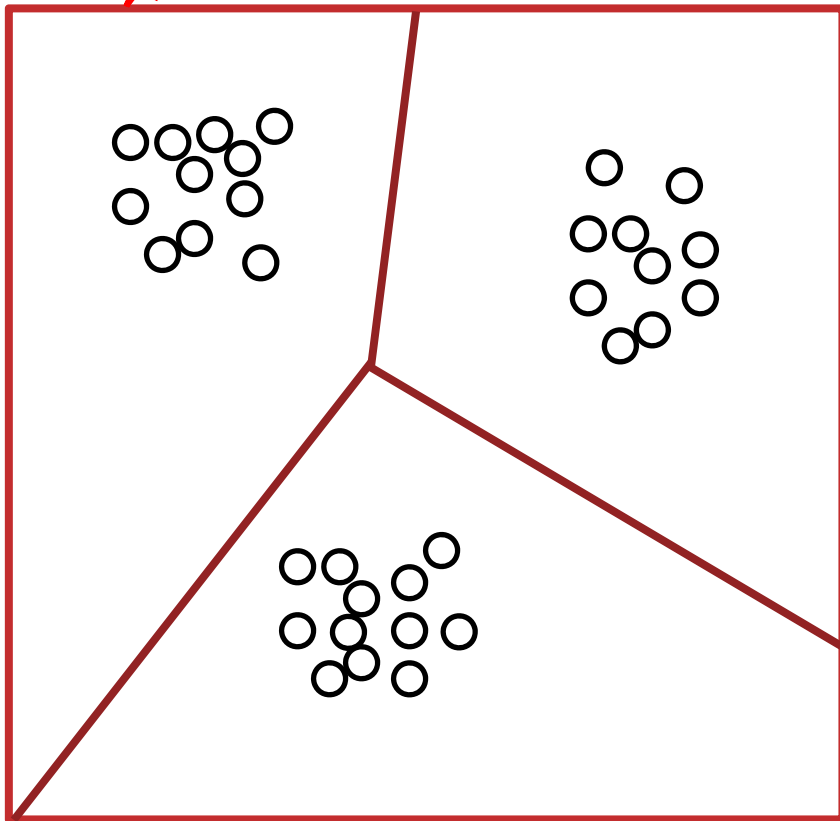
- Coordinate Descent
- Block Coordinate Descent

Q2
A = calamity

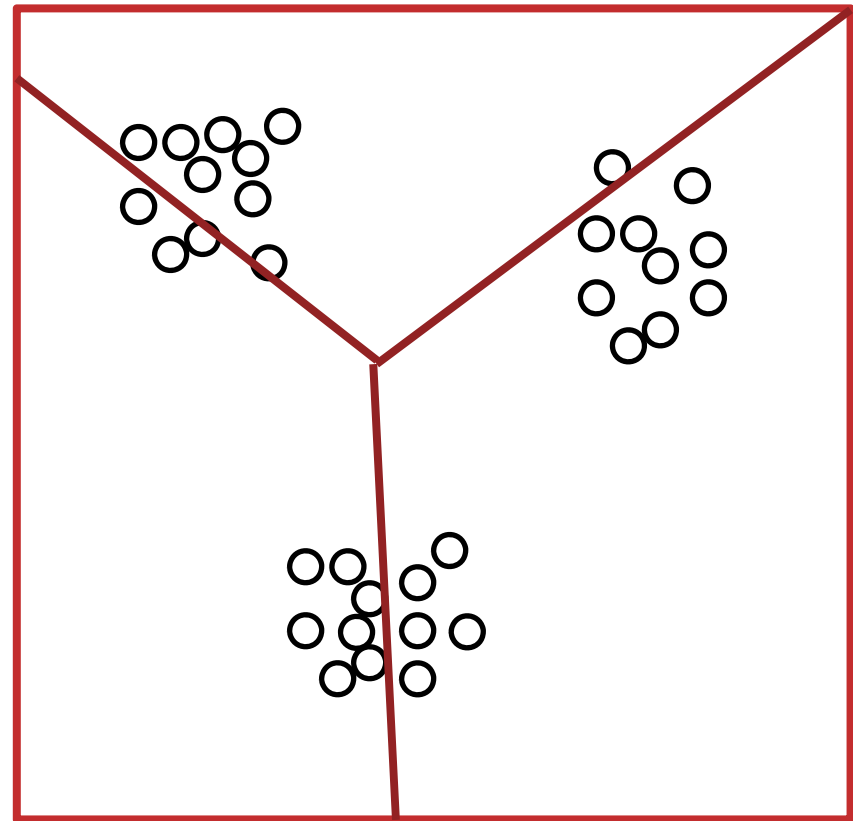
Clustering

Question: Which of these partitions is “better”?

B ✖



C



K-MEANS

K-Means Algorithm

- **Given** unlabeled feature vectors

$$D = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(N)}\}$$

- **Initialize** cluster centers $\mathbf{c} = \{\mathbf{c}^{(1)}, \dots, \mathbf{c}^{(K)}\}$
and cluster assignments $\mathbf{z} = \{z^{(1)}, z^{(2)}, \dots, z^{(N)}\}$

- **Repeat** until convergence:

A { for j in $\{1, \dots, K\}$

$\mathbf{c}^{(j)} = \text{mean}$ of **all** points assigned to cluster j

B { for i in $\{1, \dots, N\}$

$z^{(i)} = \text{index } j \text{ of cluster center nearest to } \mathbf{x}^{(i)}$

Example: Real-World Dataset



K-Means

Whiteboard:

- Clustering: Inputs and Outputs
- Objective-based Clustering
- K-Means Objective
- Computational Complexity
- K-Means Algorithm / Lloyd's Method

K-Means Algorithm

- **Given** unlabeled feature vectors
 $D = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(N)}\}$
- **Initialize** cluster centers $\mathbf{c} = \{\mathbf{c}^{(1)}, \dots, \mathbf{c}^{(K)}\}$
and cluster assignments $\mathbf{z} = \{z^{(1)}, z^{(2)}, \dots, z^{(N)}\}$
- **Repeat** until convergence:
 - for j in $\{1, \dots, K\}$
 $\mathbf{c}^{(j)} = \text{mean}$ of **all** points assigned to cluster j
 - for i in $\{1, \dots, N\}$
 $z^{(i)} = \text{index } j$ of cluster center **nearest** to $\mathbf{x}^{(i)}$

K-Means Initialization

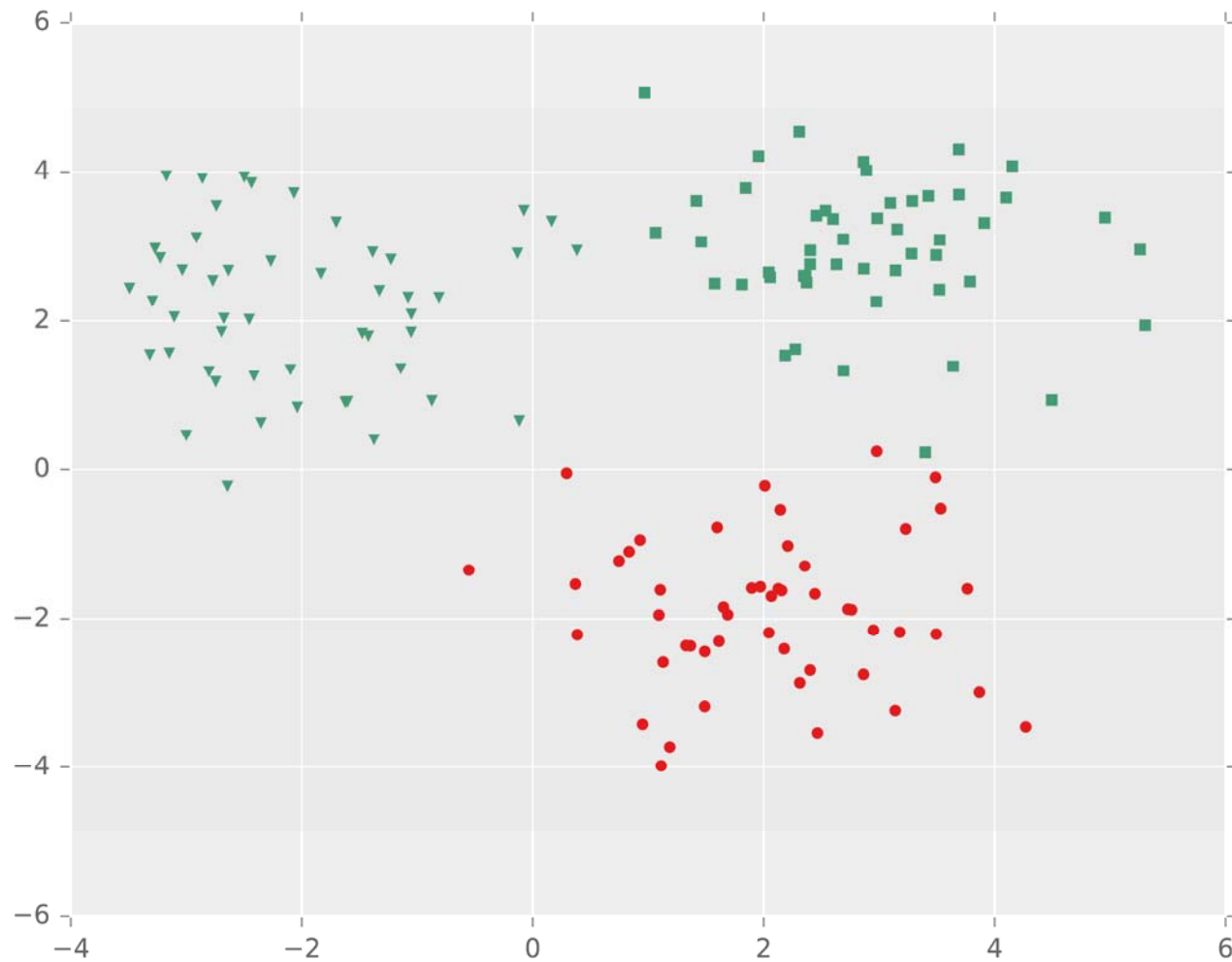
Whiteboard:

- Random
- Furthest Traversal
- K-Means++

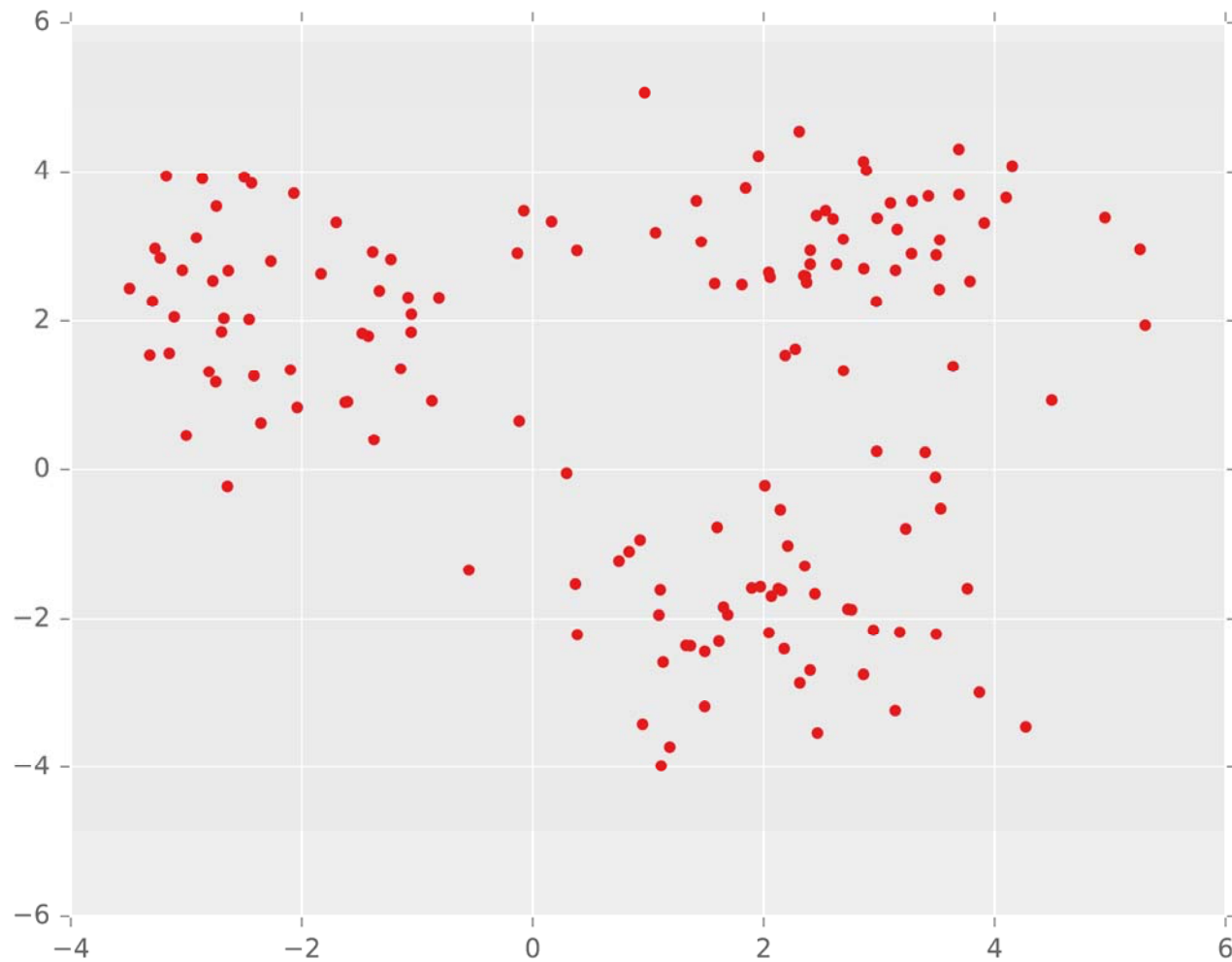
K=3 cluster centers

K-MEANS EXAMPLE

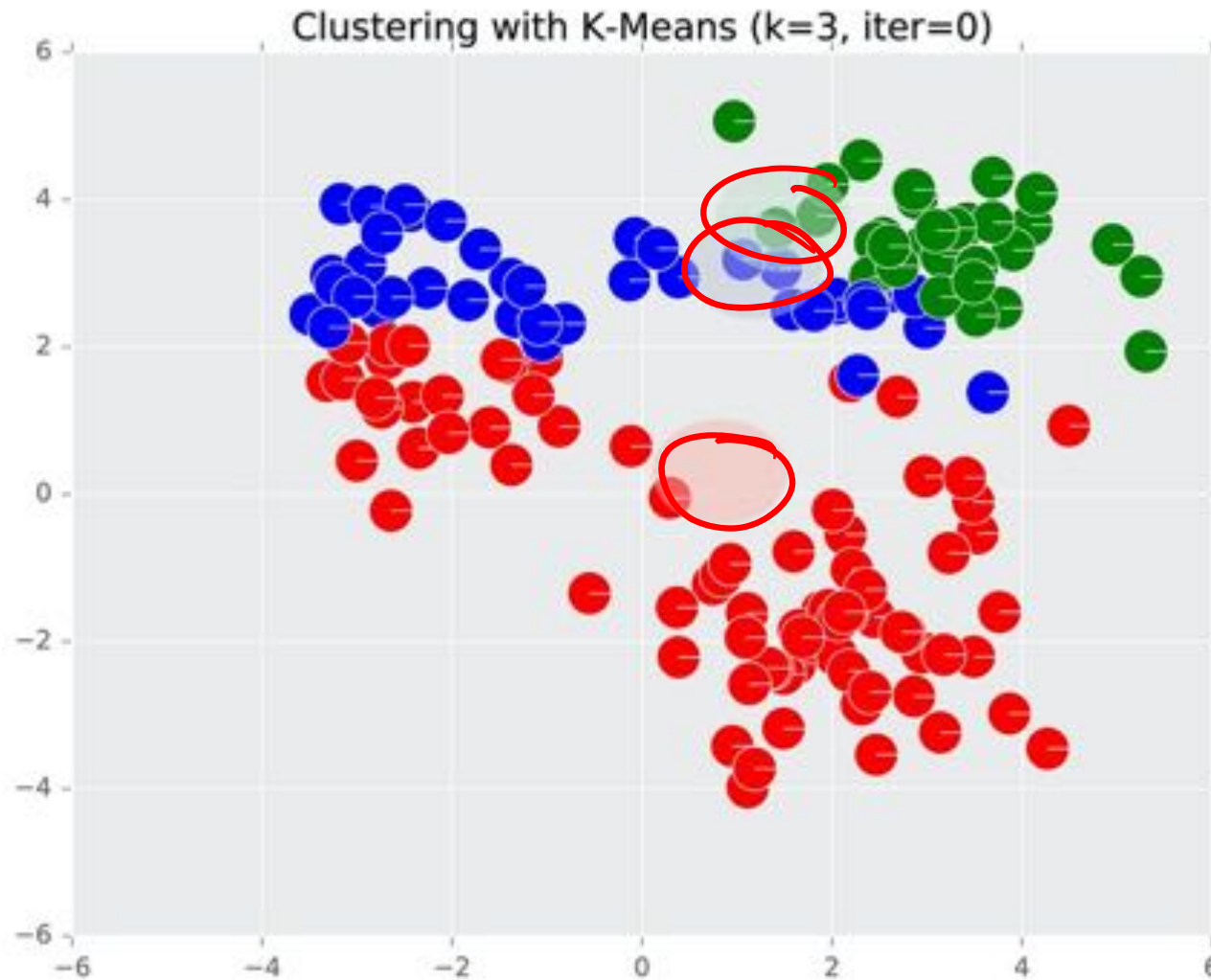
Example: K-Means



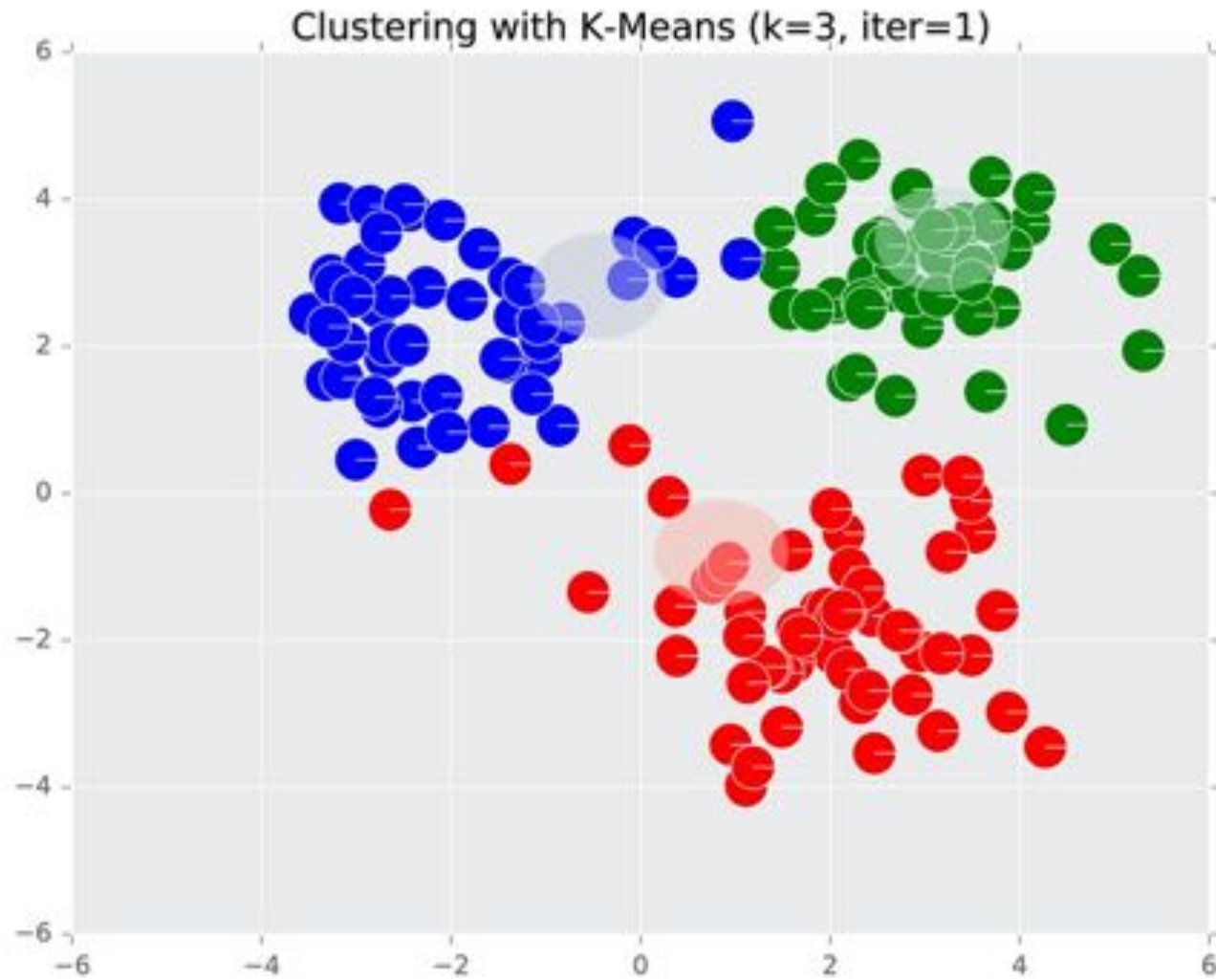
Example: K-Means



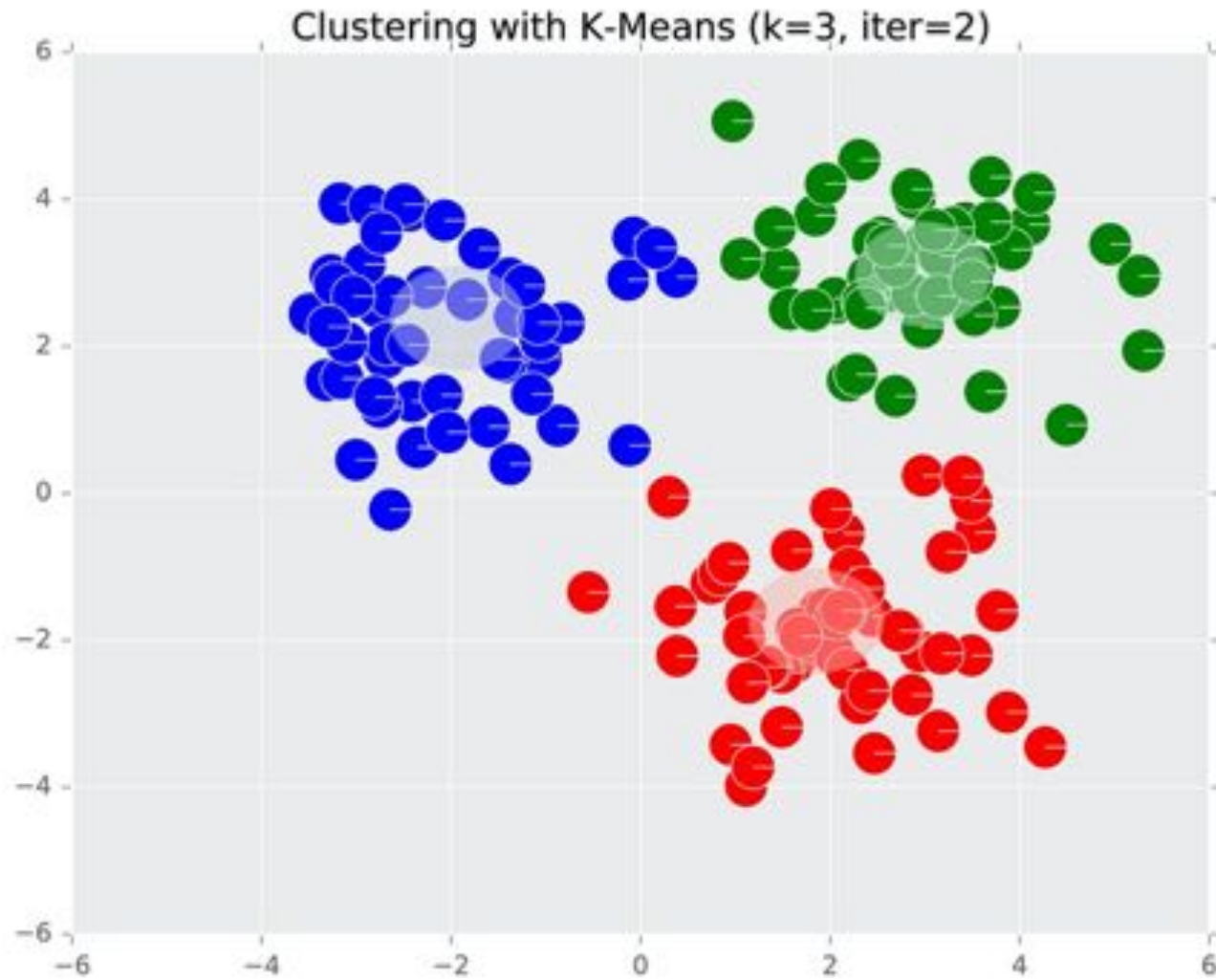
Example: K-Means



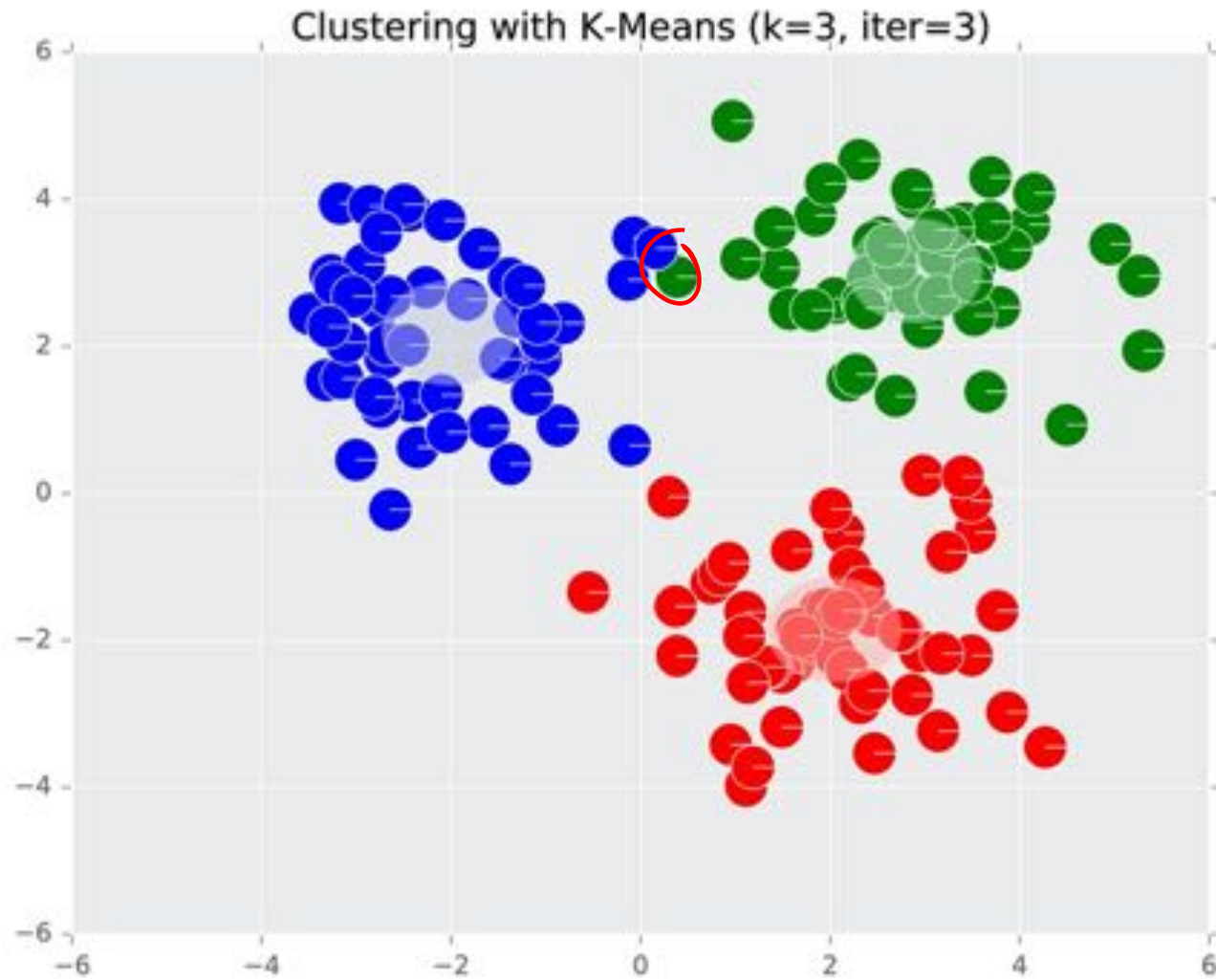
Example: K-Means



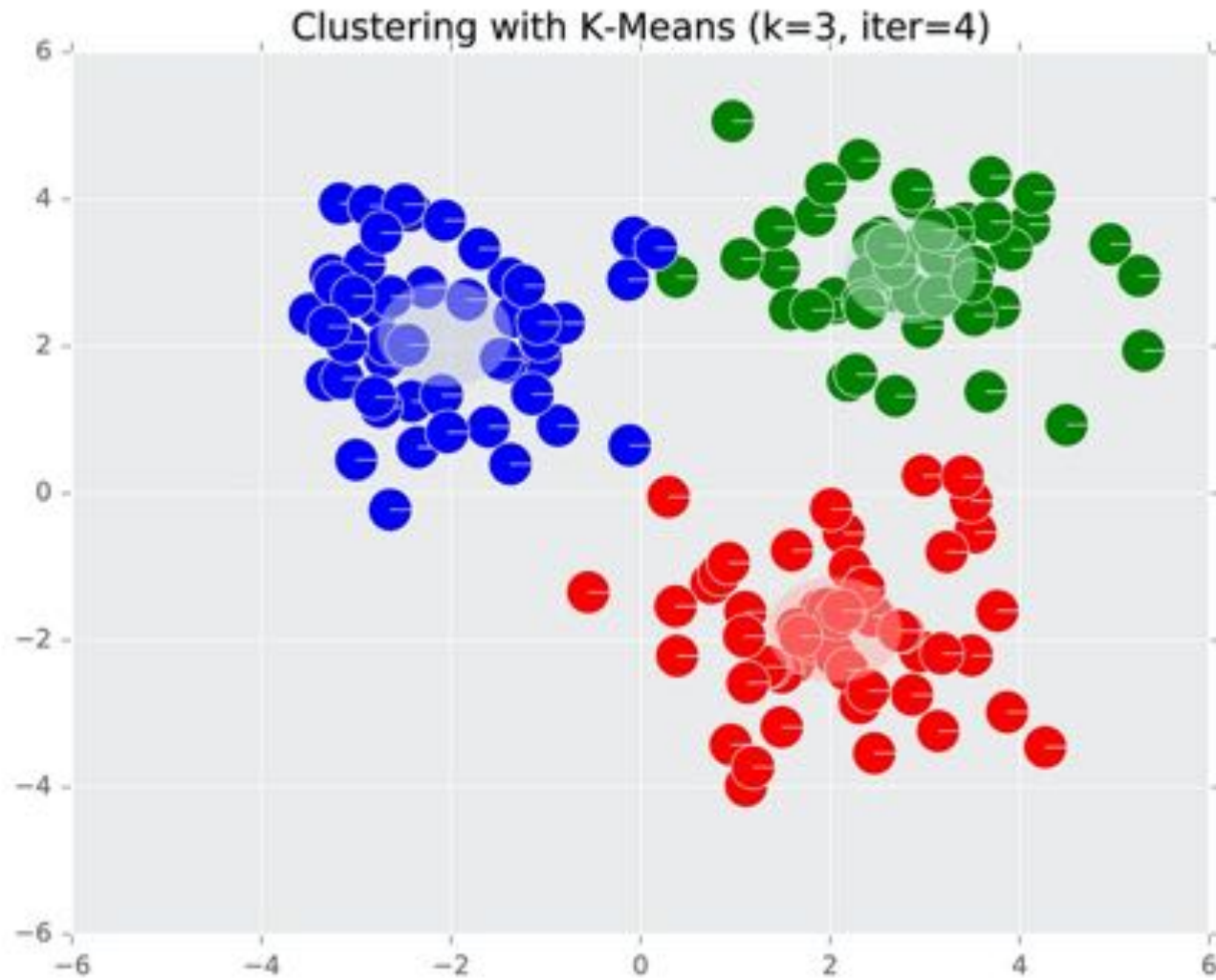
Example: K-Means



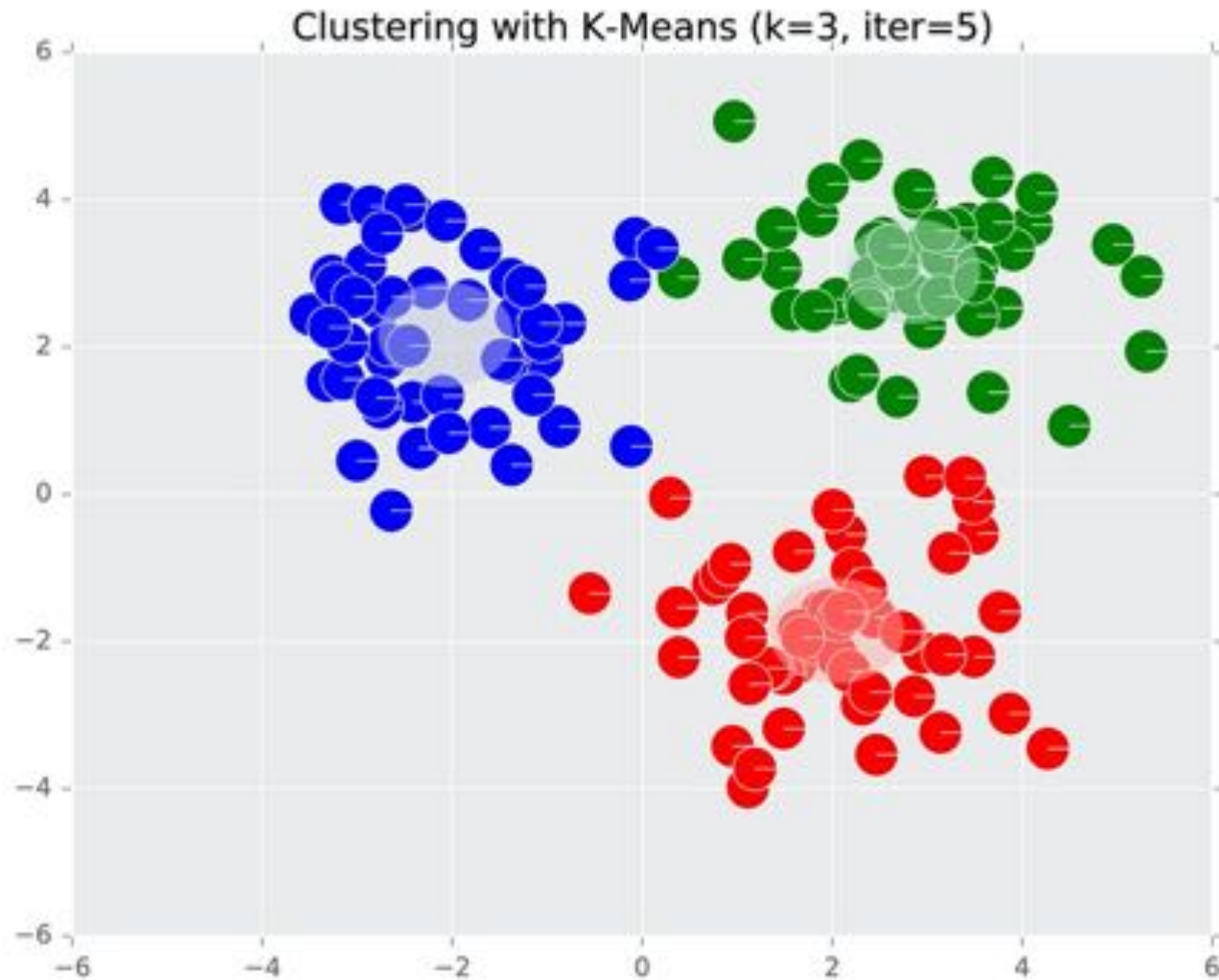
Example: K-Means



Example: K-Means



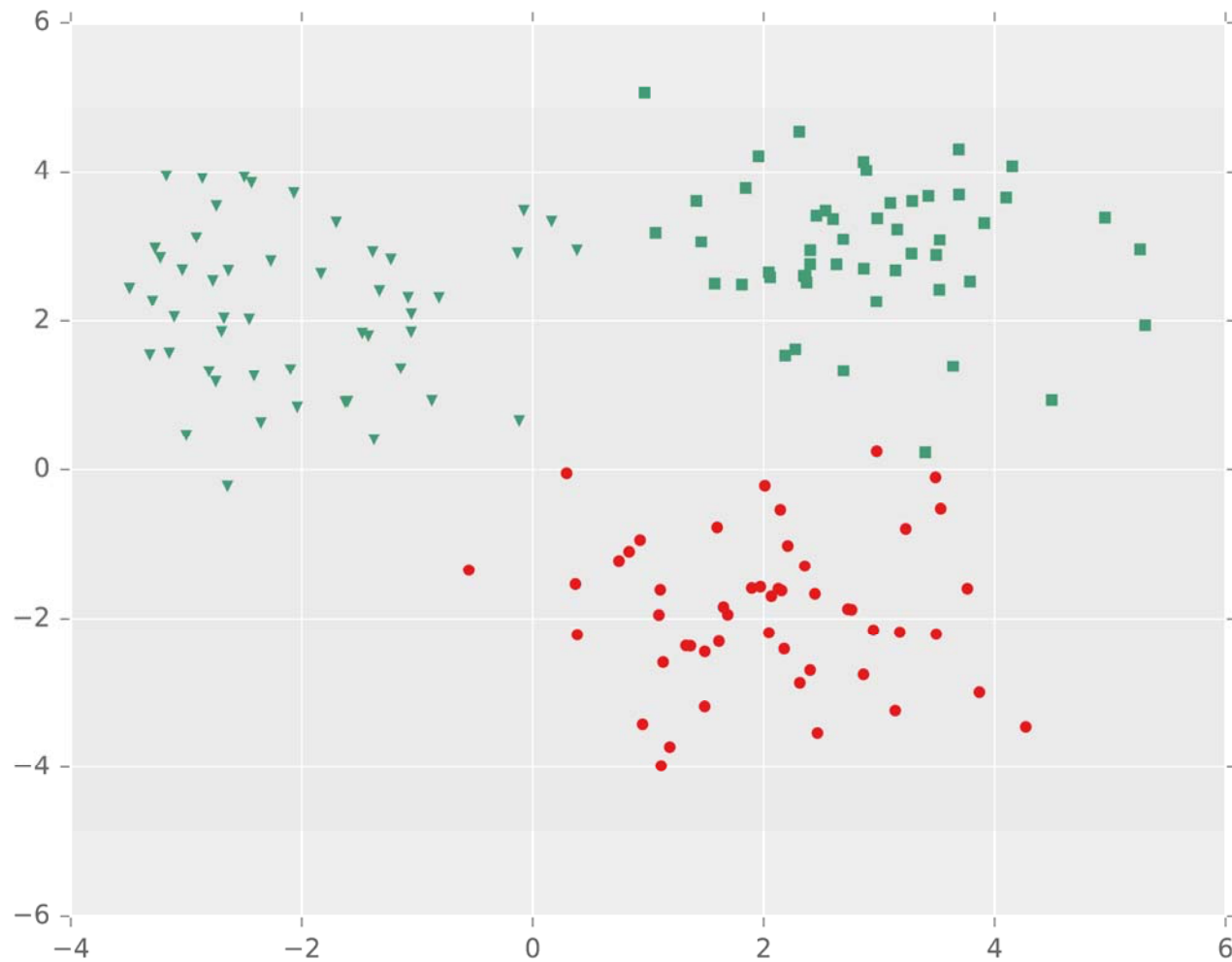
Example: K-Means



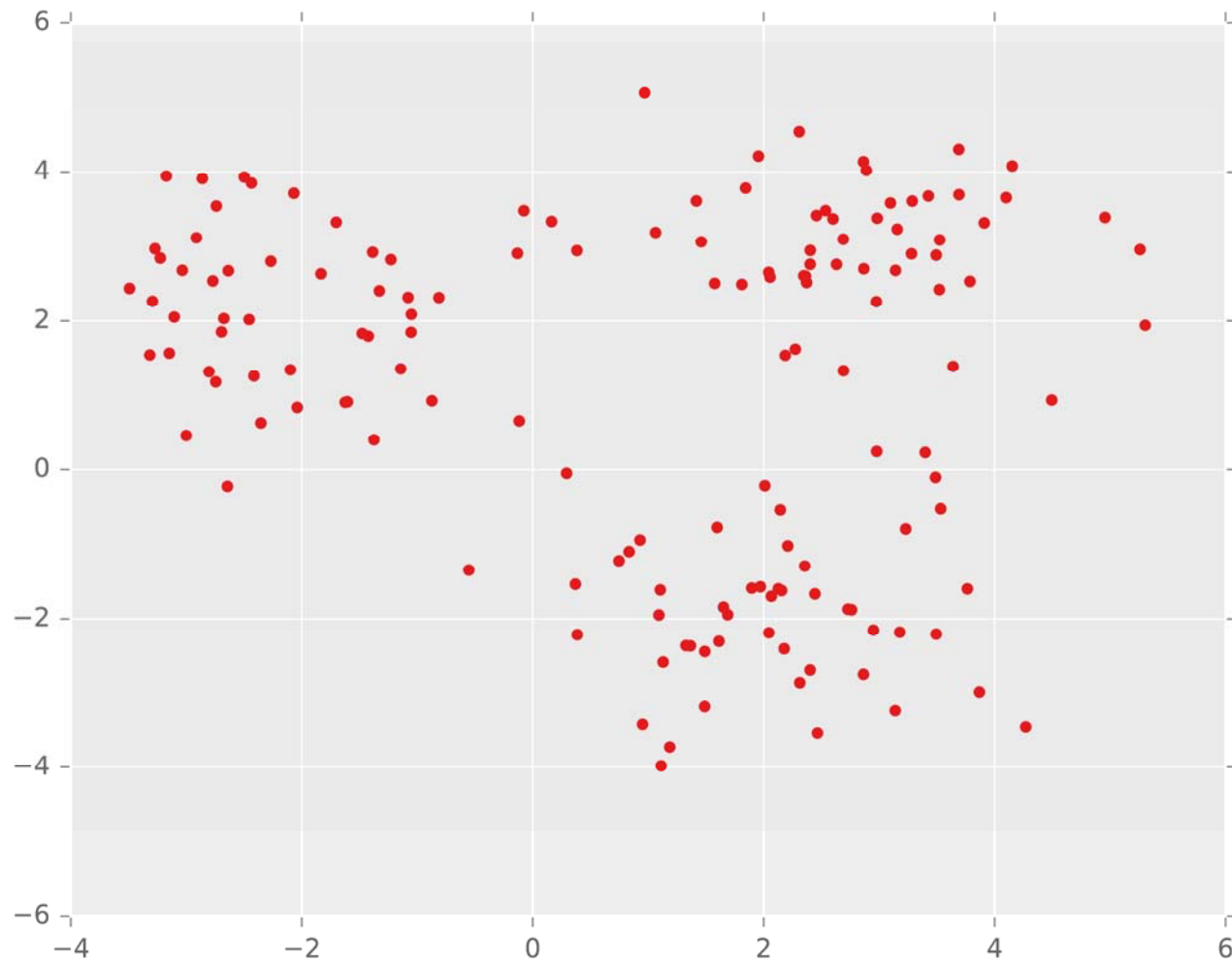
K=2 cluster centers

K-MEANS EXAMPLE

Example: K-Means



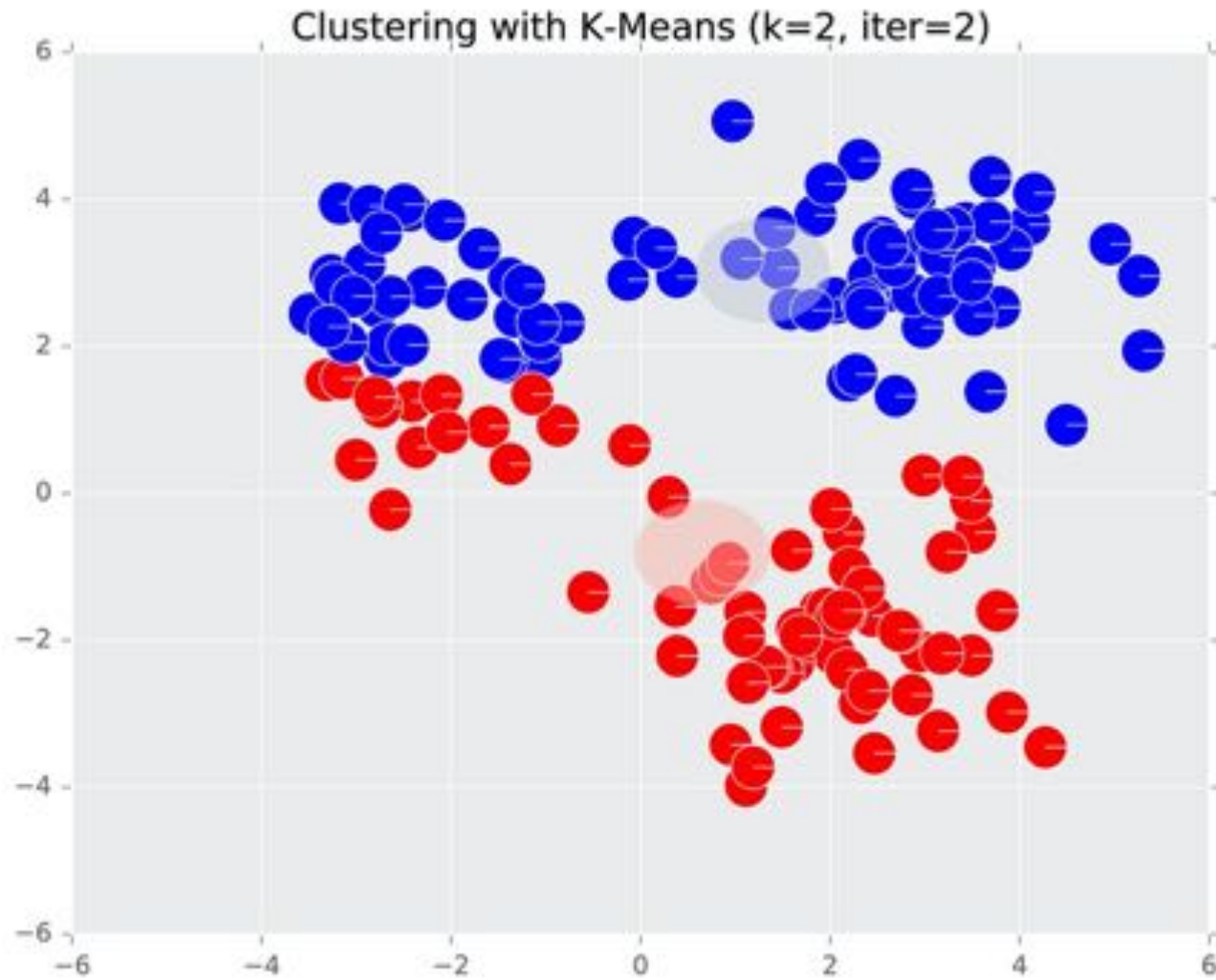
Example: K-Means



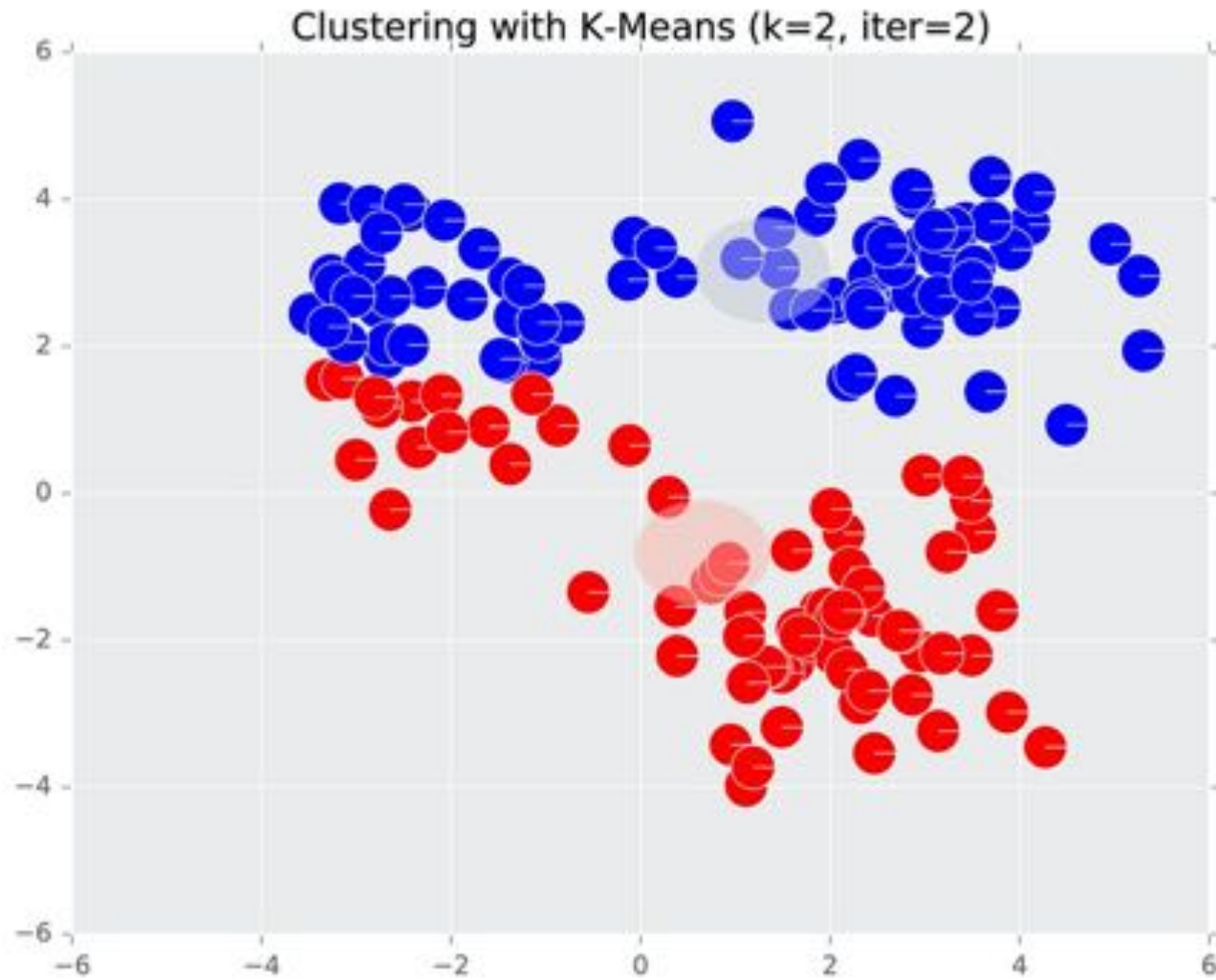
Example: K-Means



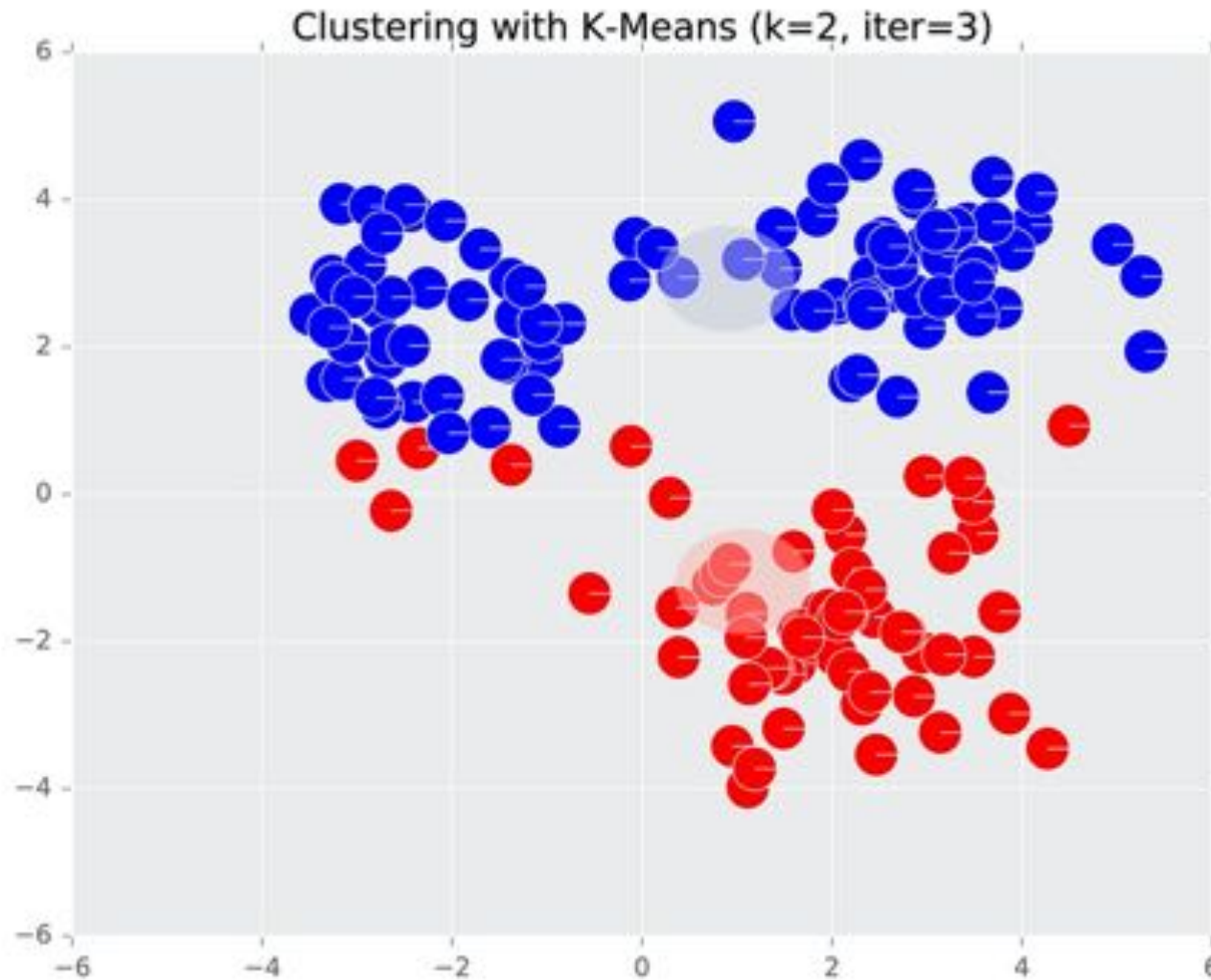
Example: K-Means



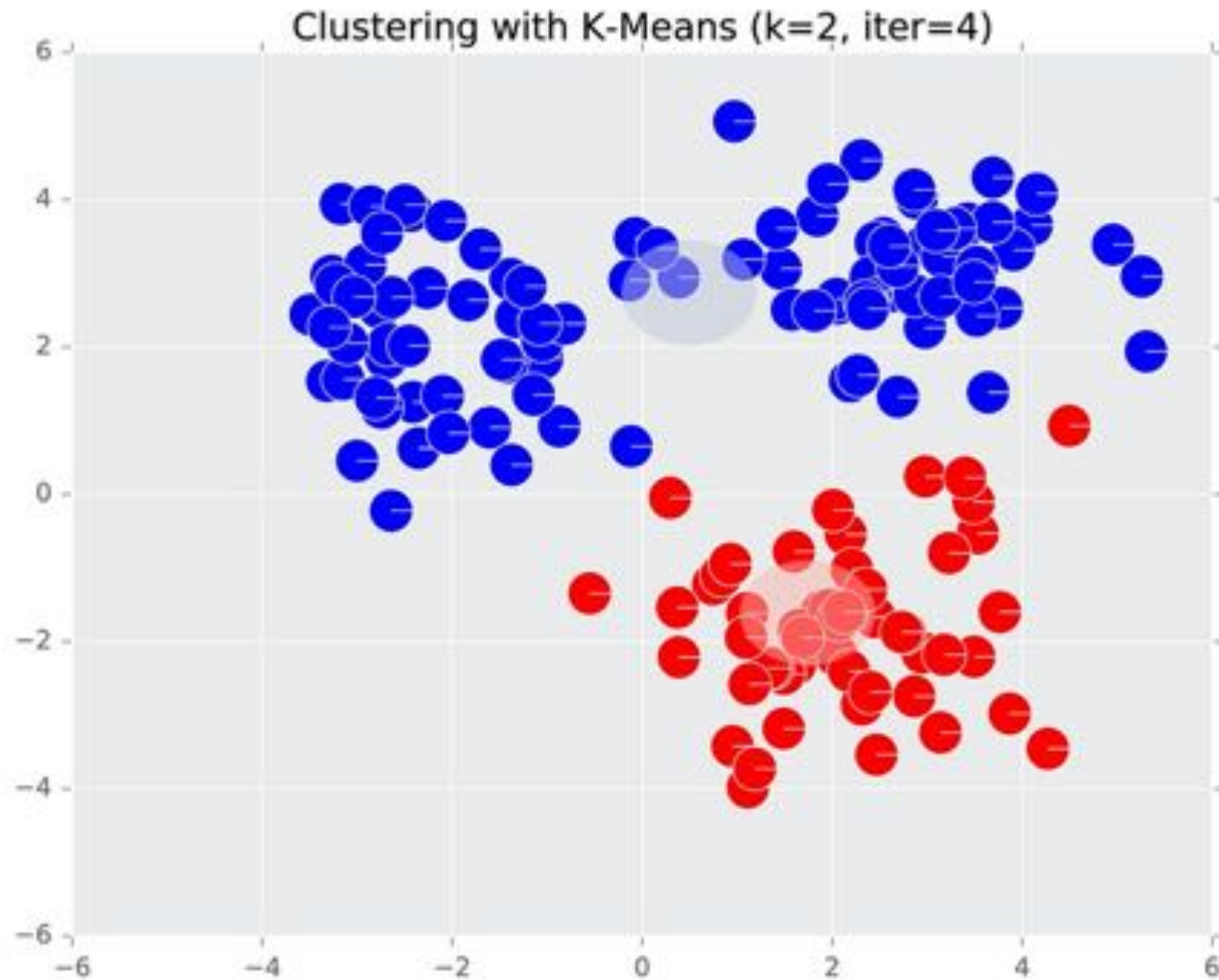
Example: K-Means



Example: K-Means



Example: K-Means



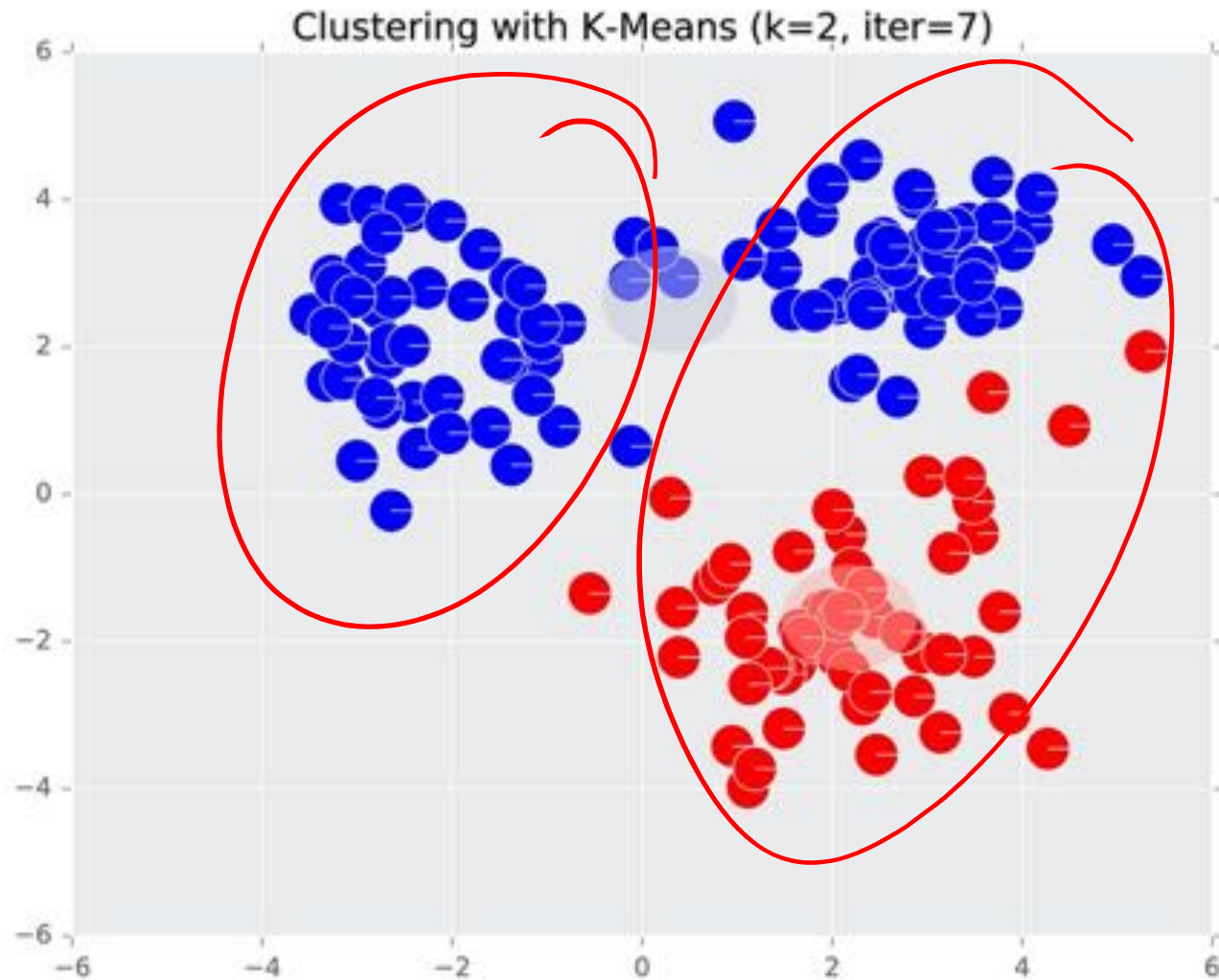
Example: K-Means



Example: K-Means



Example: K-Means



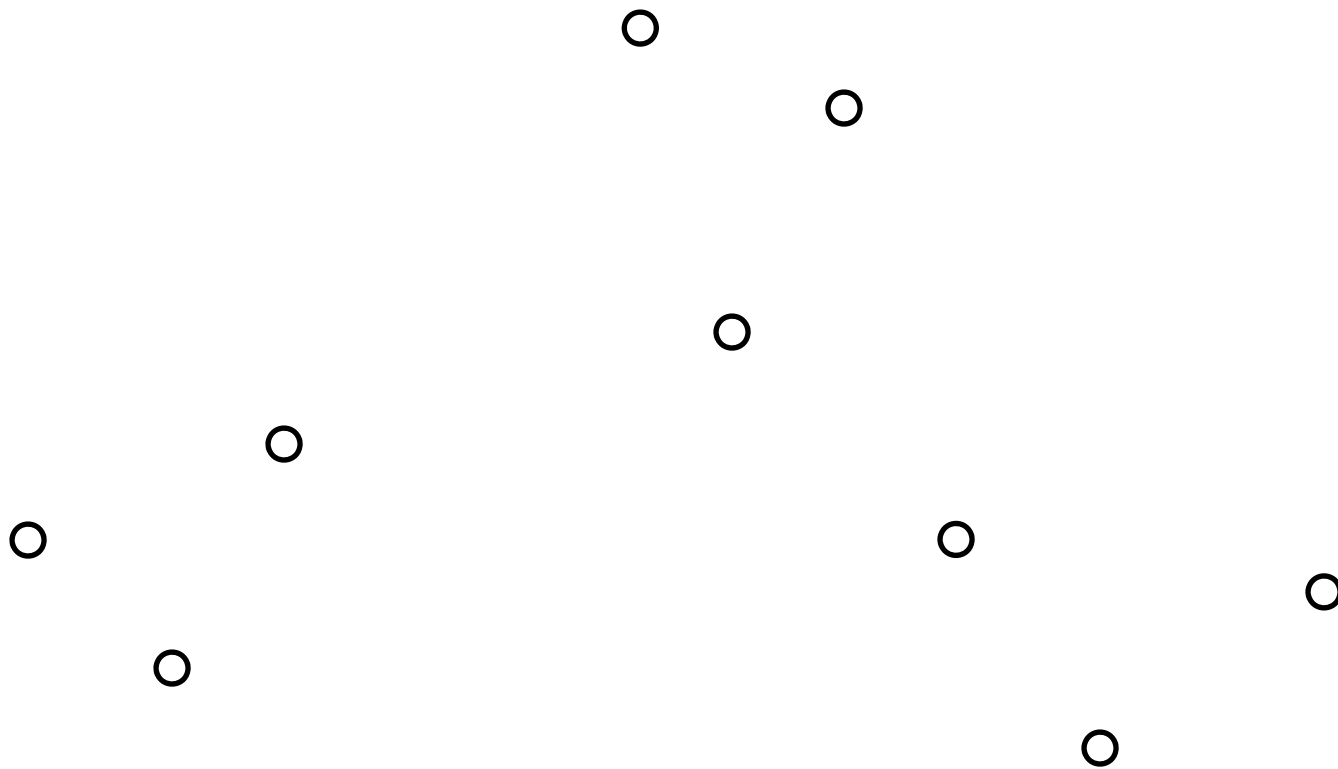
Example: K-Means



K-MEANS PERFORMANCE

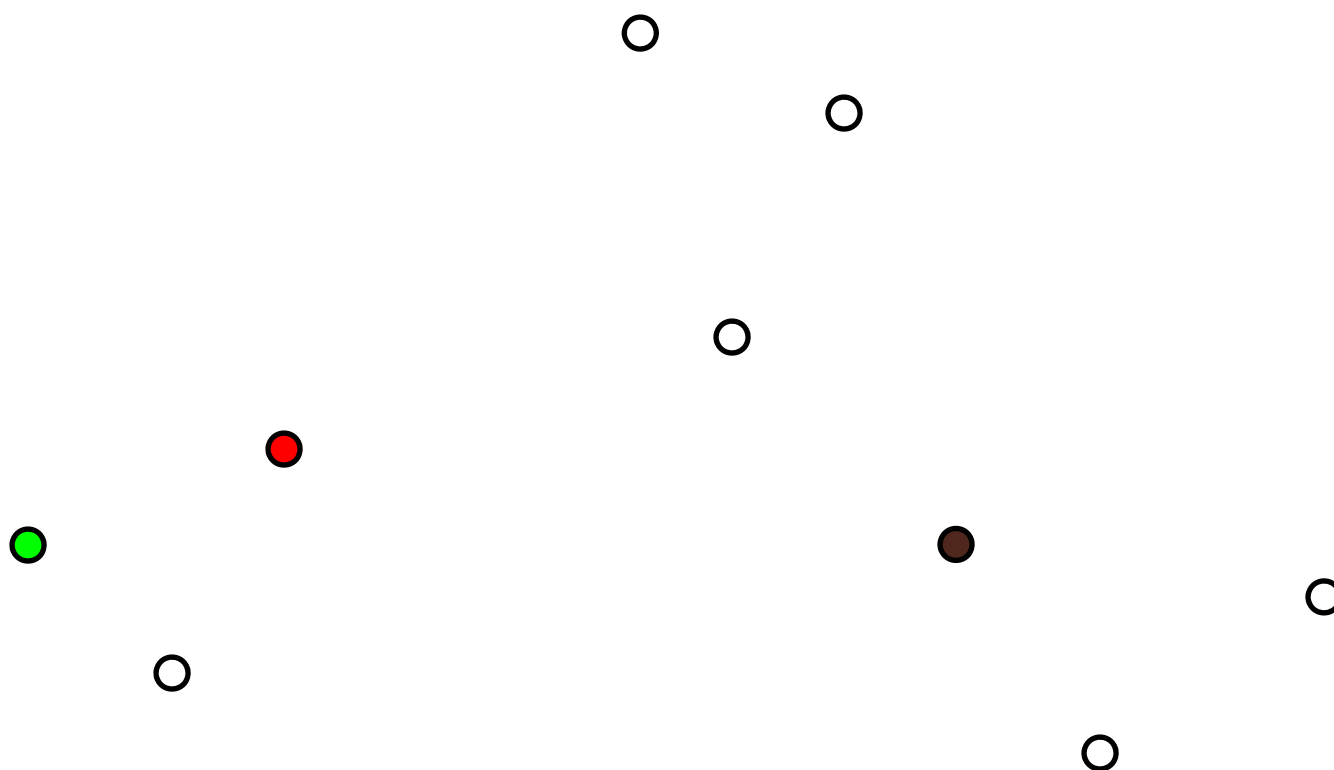
Lloyd's method: Random Initialization

Example: Given a set of datapoints



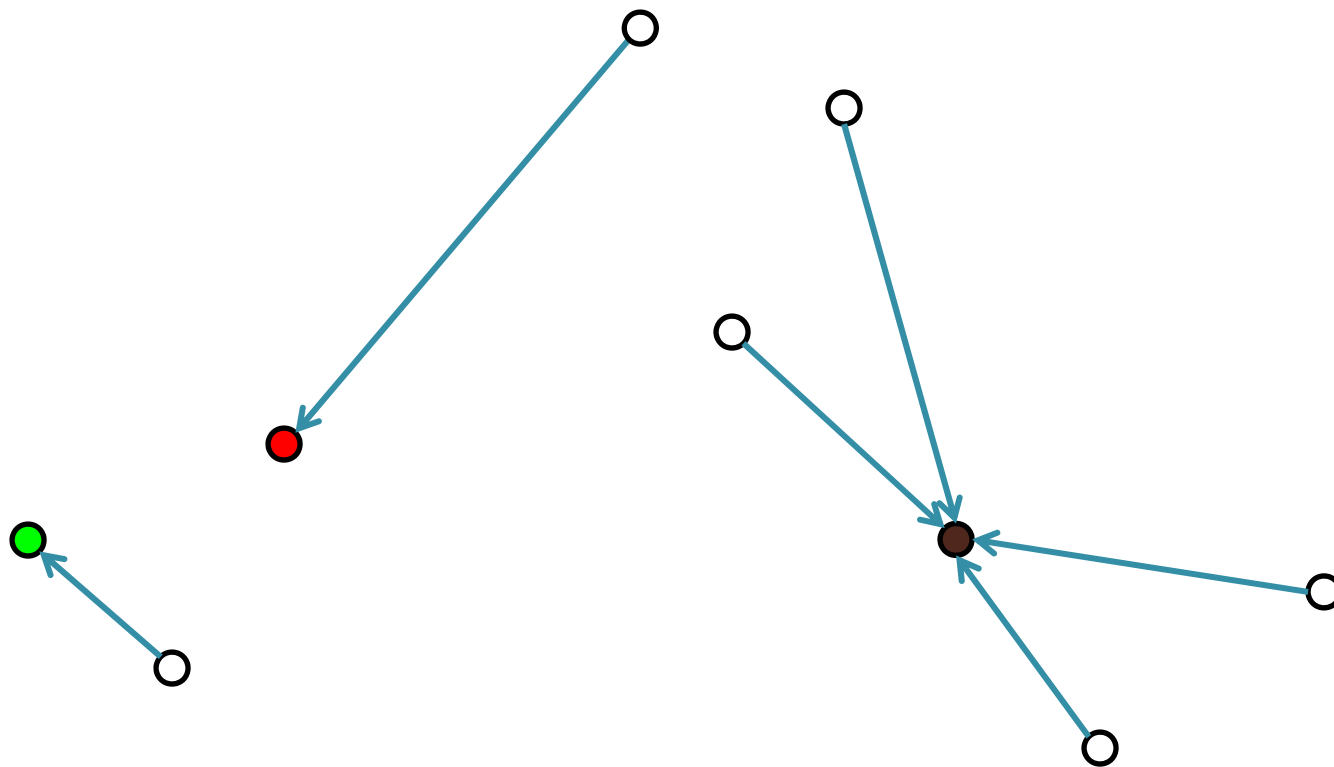
Lloyd's method: Random Initialization

Select initial centers at random



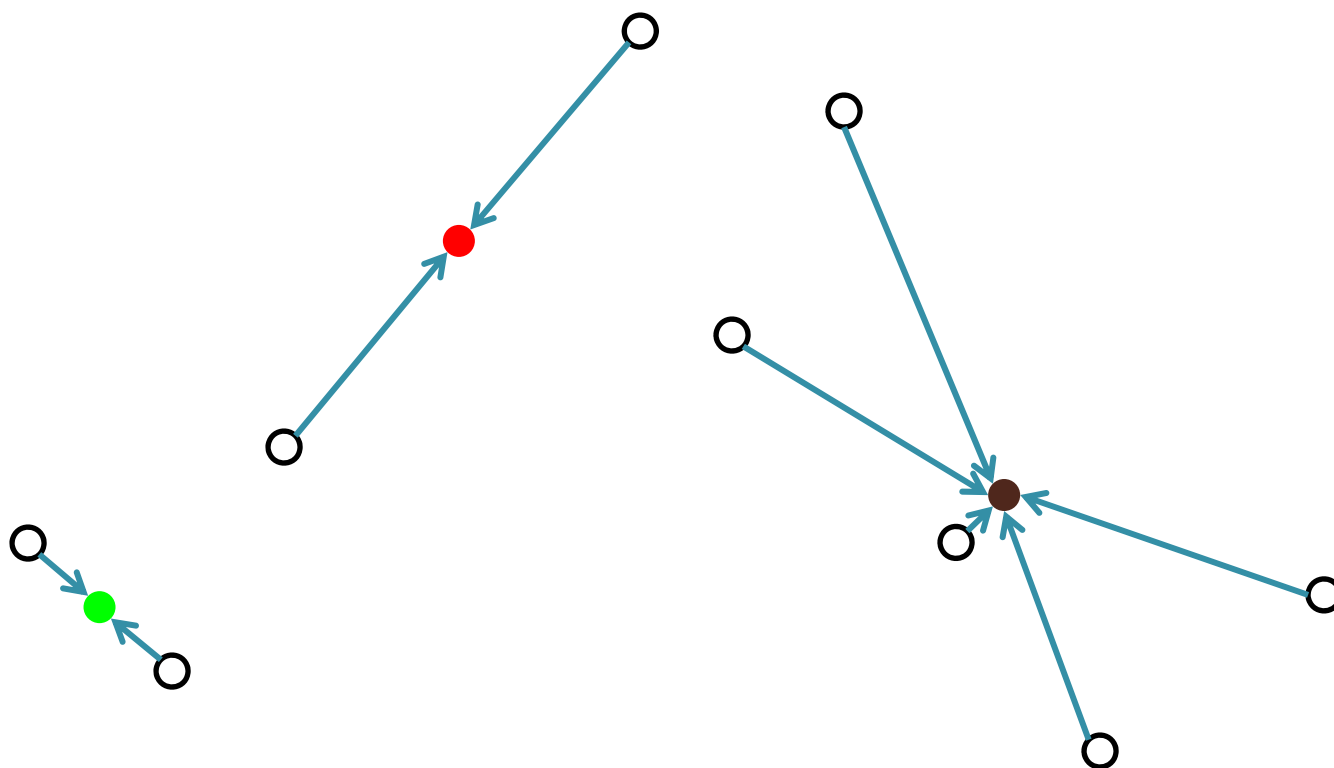
Lloyd's method: Random Initialization

Assign each point to its nearest center



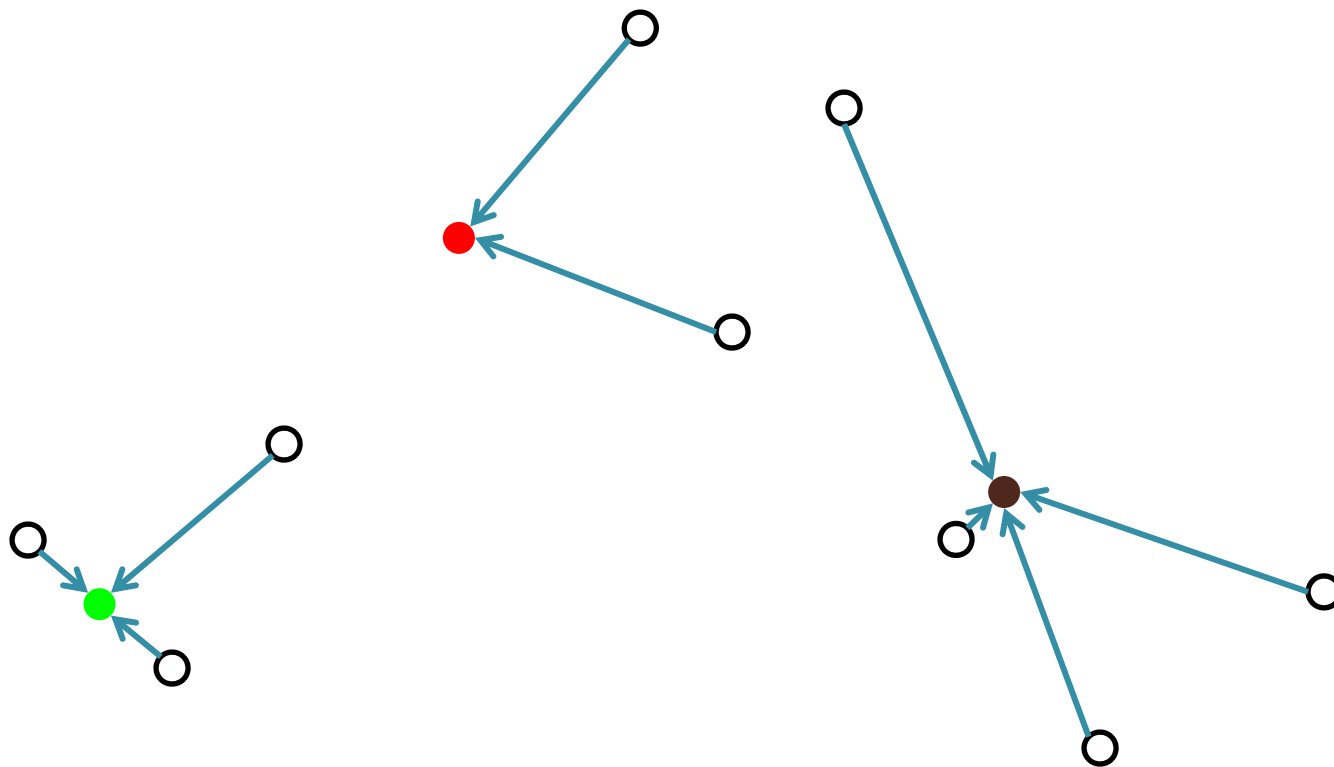
Lloyd's method: Random Initialization

Recompute optimal centers given a fixed clustering



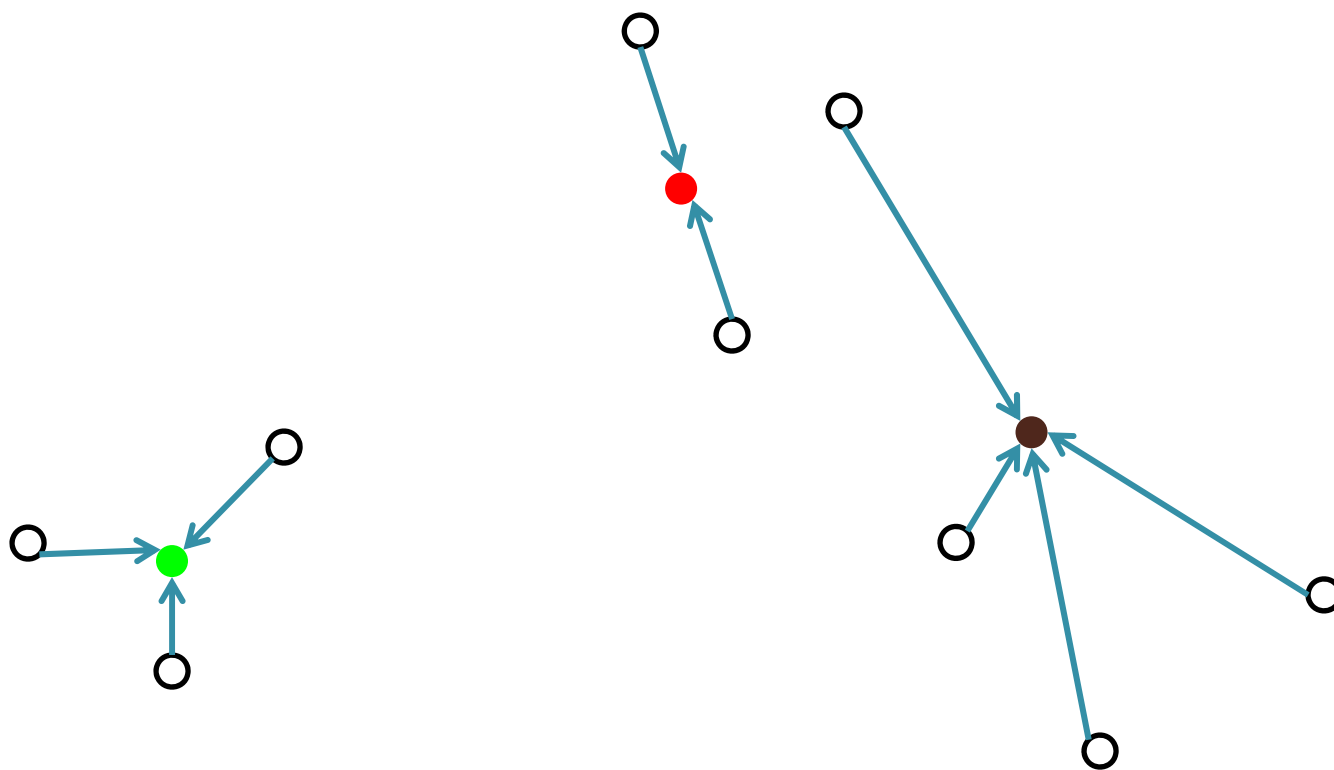
Lloyd's method: Random Initialization

Assign each point to its nearest center



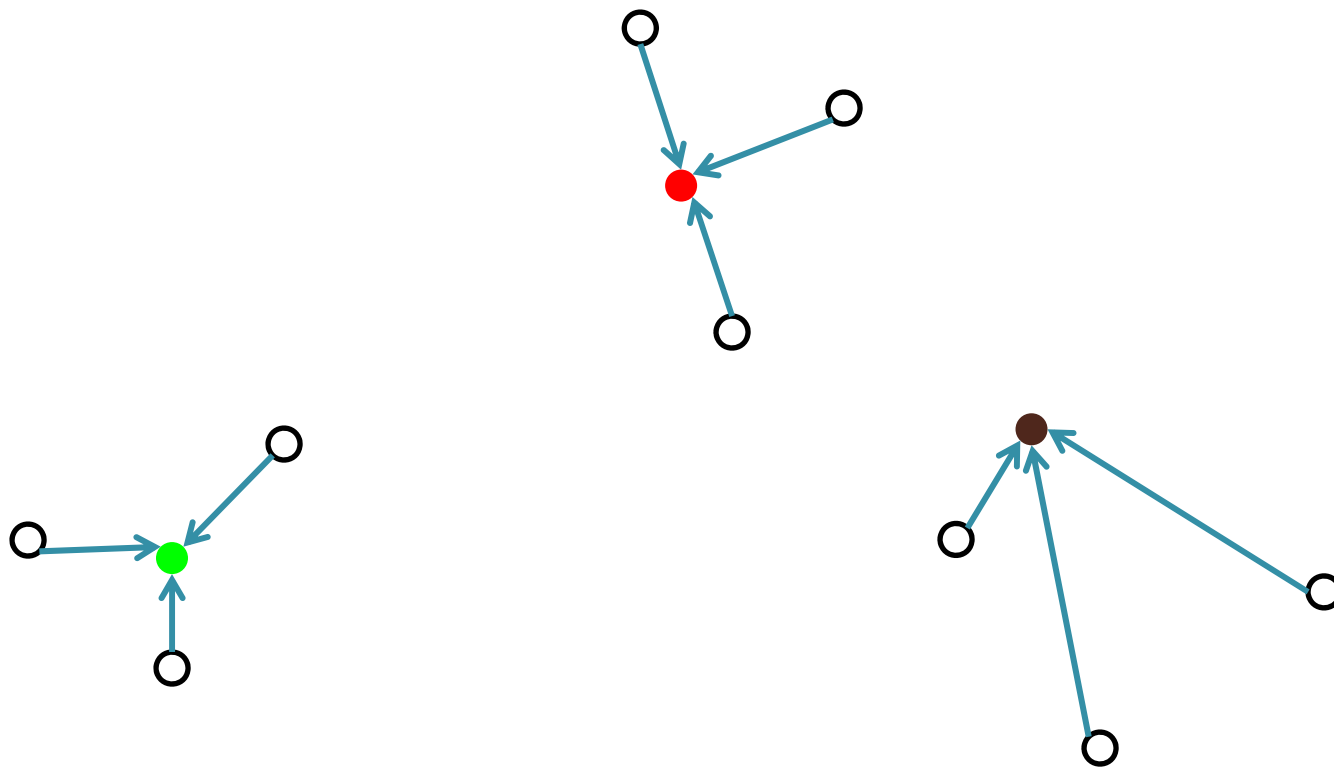
Lloyd's method: Random Initialization

Recompute optimal centers given a fixed clustering



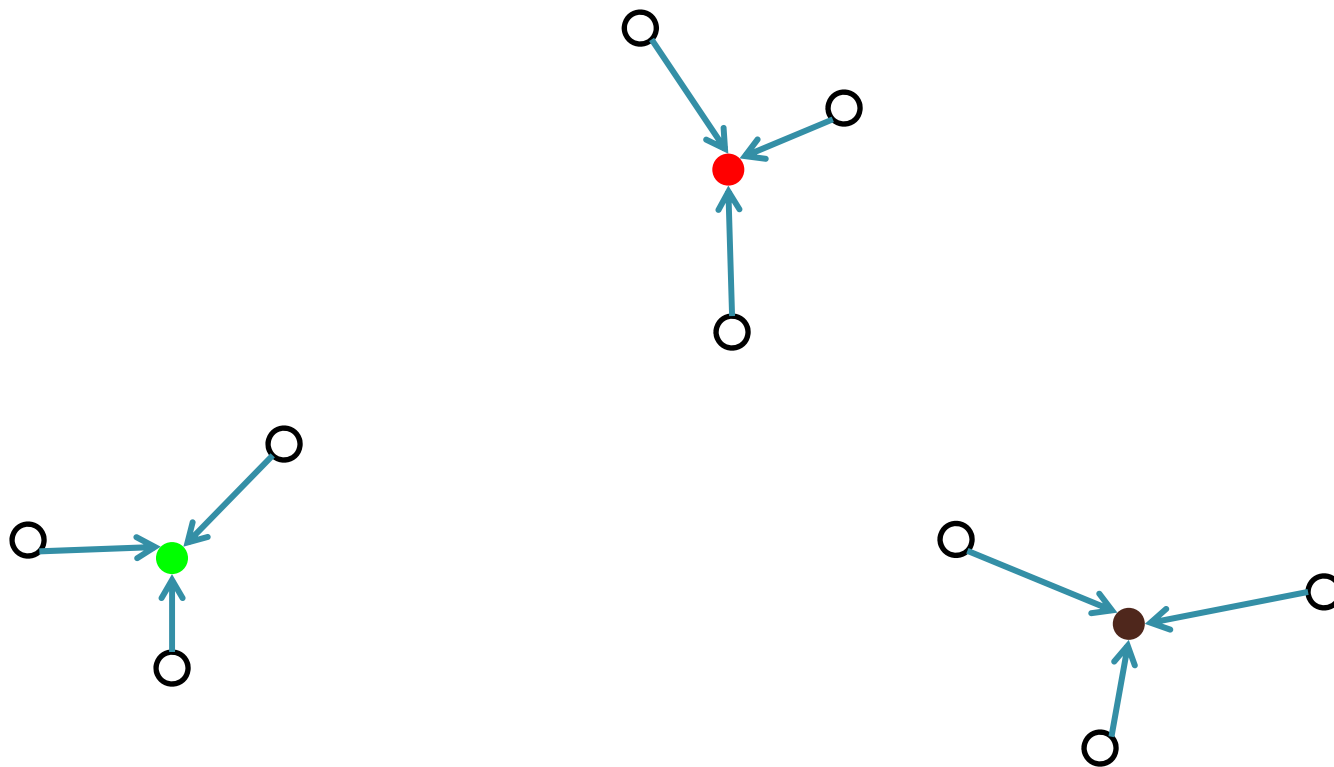
Lloyd's method: Random Initialization

Assign each point to its nearest center



Lloyd's method: Random Initialization

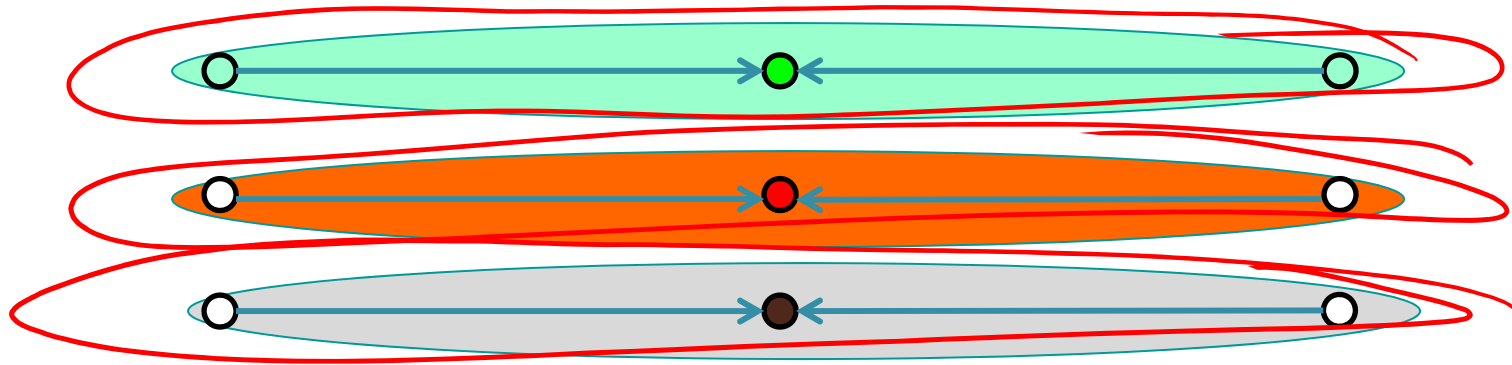
Recompute optimal centers given a fixed clustering



Get a good quality solution in this example.

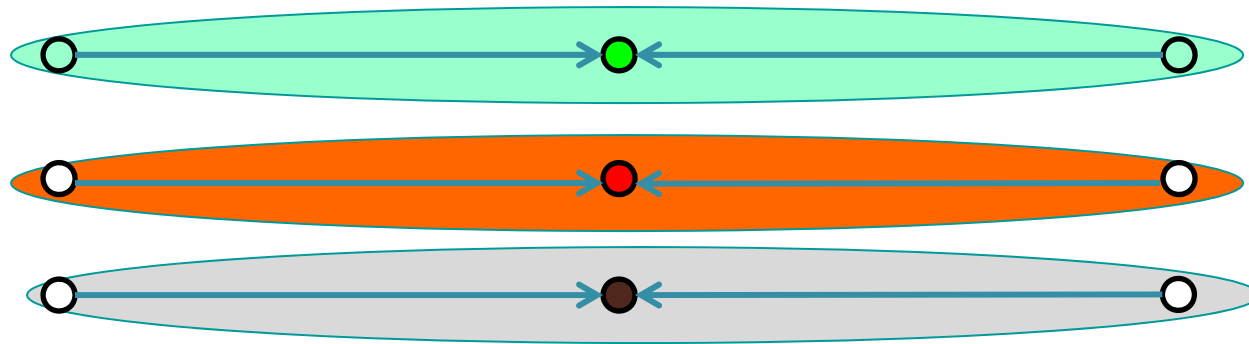
Lloyd's method: Performance

obj. fn. for K -means is nonconvex



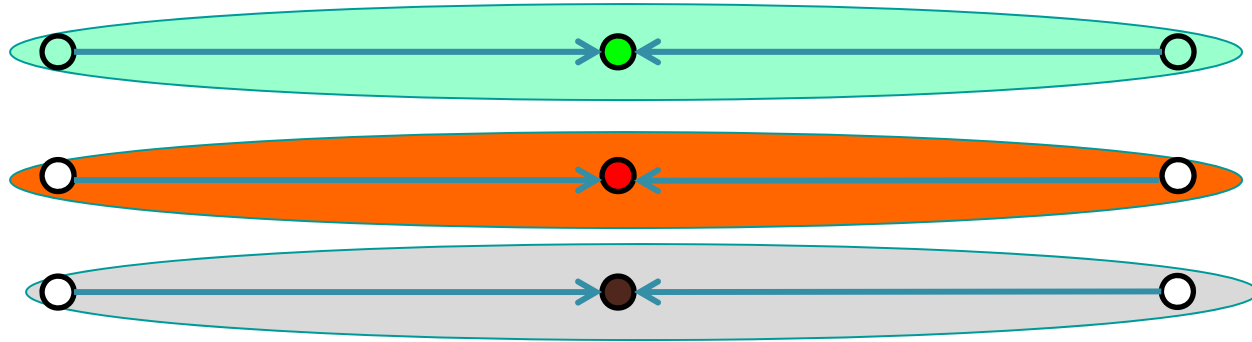
It always converges, but it may converge at a local optimum that is different from the global optimum, and in fact could be arbitrarily worse in terms of its score.

Lloyd's method: Performance

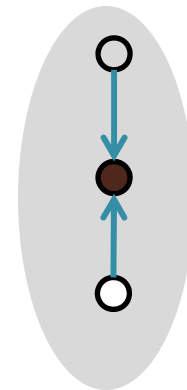
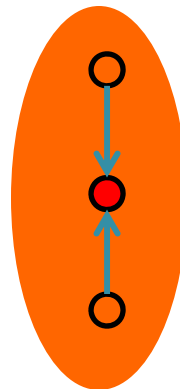
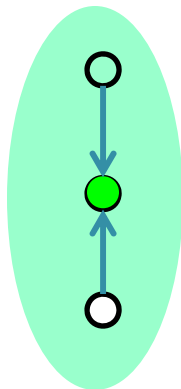


Local optimum: every point is assigned to its nearest center and every center is the mean value of its points.

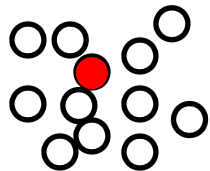
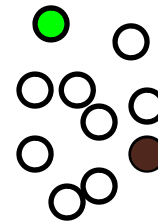
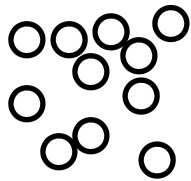
Lloyd's method: Performance



.It is arbitrarily worse than optimum solution....

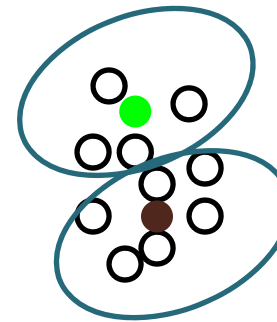
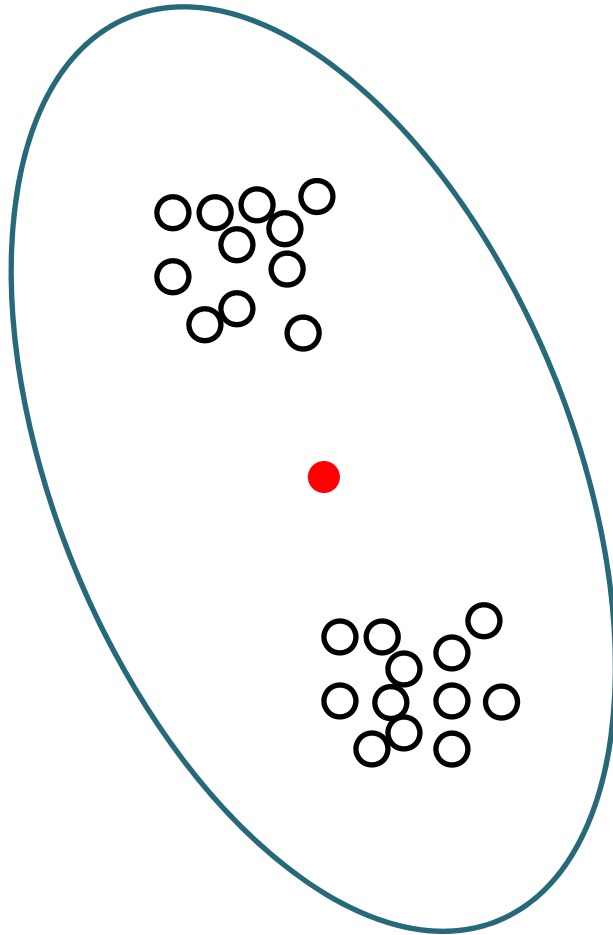


Lloyd's method: Performance



This bad performance, can happen even with well separated Gaussian clusters.

Lloyd's method: Performance



This bad performance, can happen even with well separated Gaussian clusters.

Some Gaussian are combined....



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 nonconvexity of the K-Means objective function with the (possibly) poor performance of random initialization