10-301/601 Introduction to Machine Learning

Machine Learning Department
School of Computer Science
Carnegie Mellon University

K-Means + Ensemble Methods + Recommender Systems +

Matt Gormley
Lecture 26
Apr. 20, 2022

Reminders

- Homework 8: Reinforcement Learning
  - Out: Tue, Apr. 12
  - Due: Thu, Apr. 21 at 11:59pm
- Homework 9: Learning Paradigms
  - Out: Thu, Apr. 21
  - Due: Wed, Apr. 27 at 11:59pm
  - Can only use up to 2 grace/late days, so we can return grades before final exam
- Exam 3 Practice Problems
  - Out: Wed, Apr. 27
- Mock Exam 3
  - Out: Wed, Apr. 27
  - Due: Mon, May 2 at 11:59pm
- Exam 3
  - Tue, May 3 (9:30am – 11:30am)

Q&A

Q: I've had such a great experience with this class, especially with your excellent TAs: how can I be more like them and contribute to future iterations of this class?

A: You can apply to be TA for this course next semester (S22)!

Details will be posted to Piazza this week.

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:

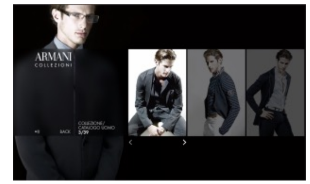
- Automatically organizing data.
- Understanding hidden structure in data.
- Preprocessing for further analysis.
  - Representing high-dimensional data in a low-dimensional space (e.g., for visualization purposes).

Slide courtesy of Nina Balcan

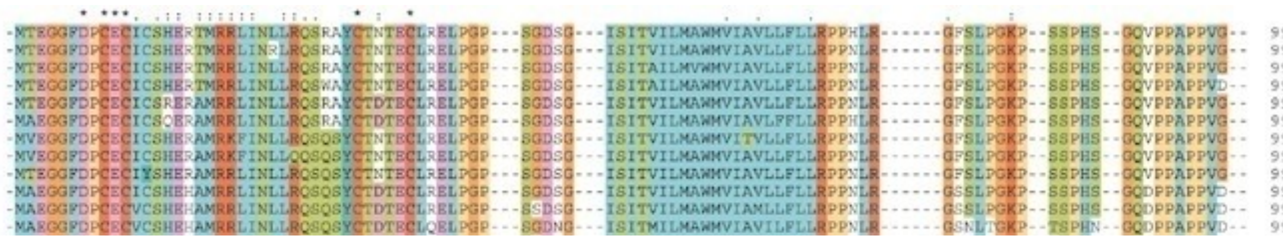
Applications (Clustering comes up everywhere…)

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

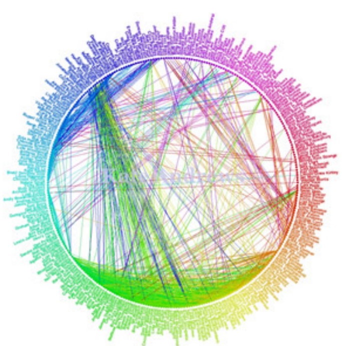


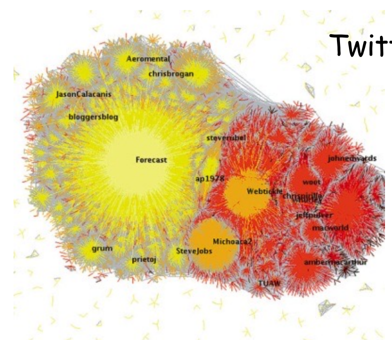
- Cluster protein sequences by function or genes according to expression profile.

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



Facebook network



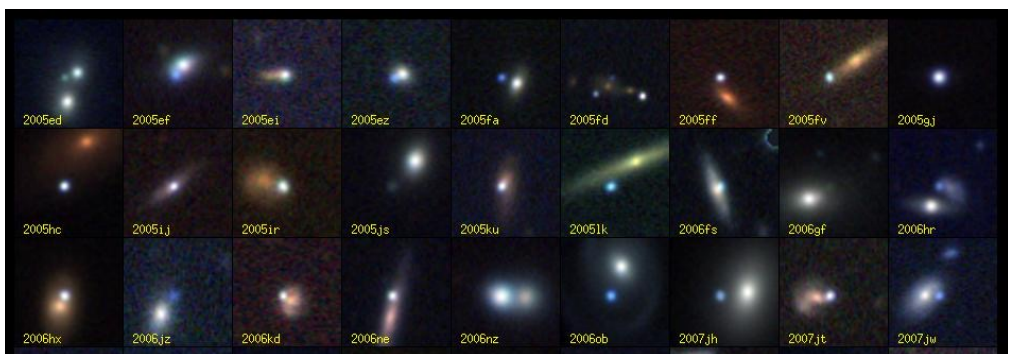
Twitter Network

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)

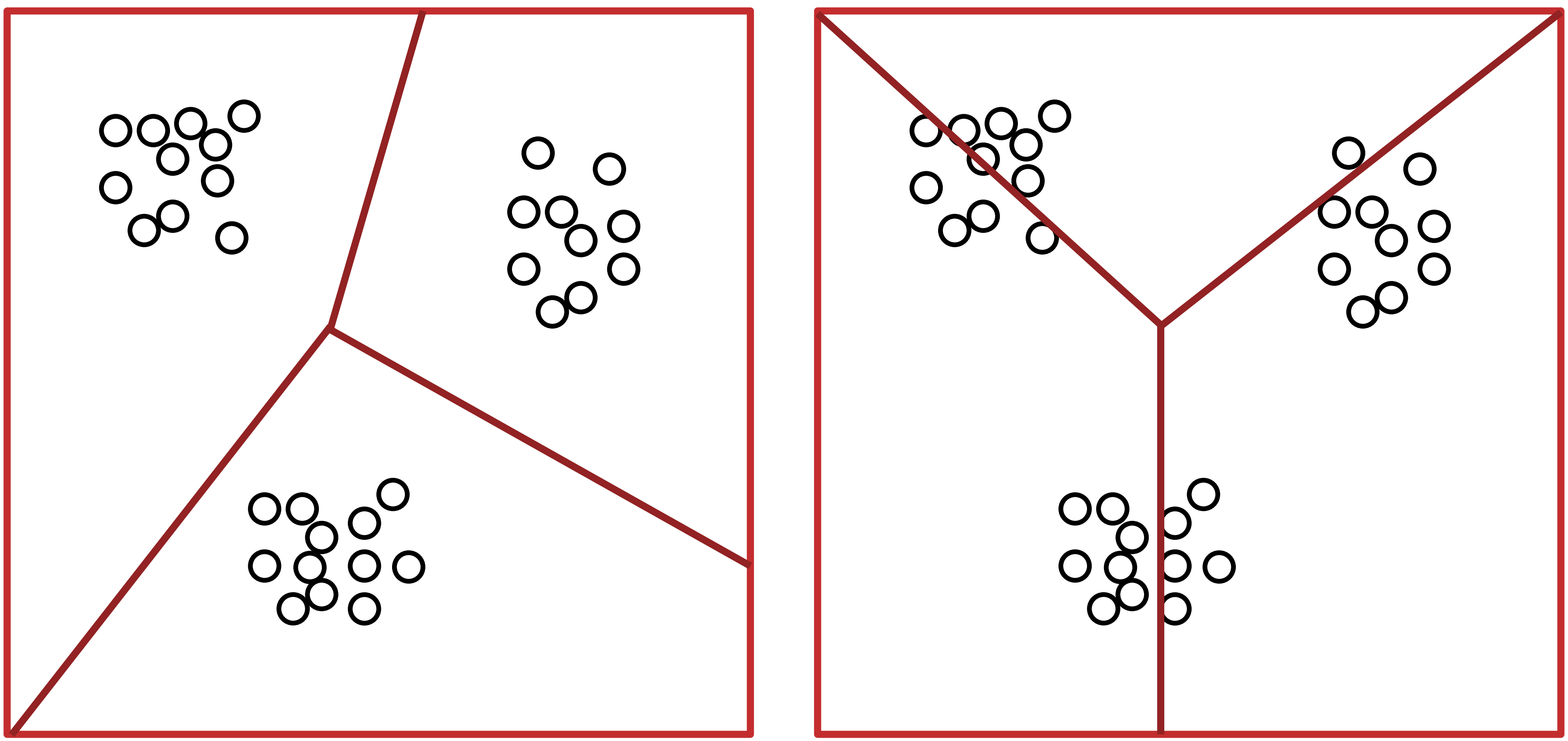


- And many many more applications….

Slide courtesy of Nina Balcan

Clustering

Question: Which of these partitions is "better"?



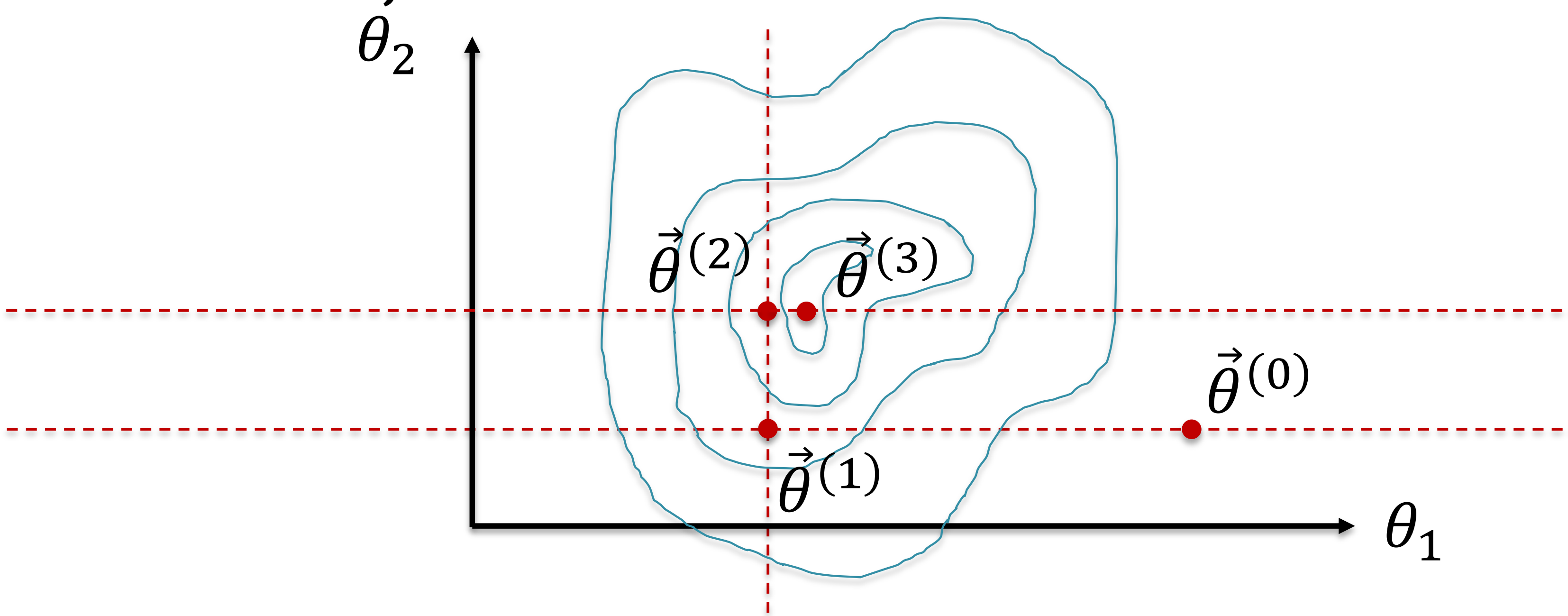
OPTIMIZATION BACKGROUND

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

Optimization Background

Whiteboard:

- Coordinate Descent
- Block Coordinate Descent

K-MEANS

K-Means

Whiteboard:

- K-means recipe
  - K-means model parameters
  - K-means objective function

K-Means Algorithm

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

K-Means

Whiteboard:

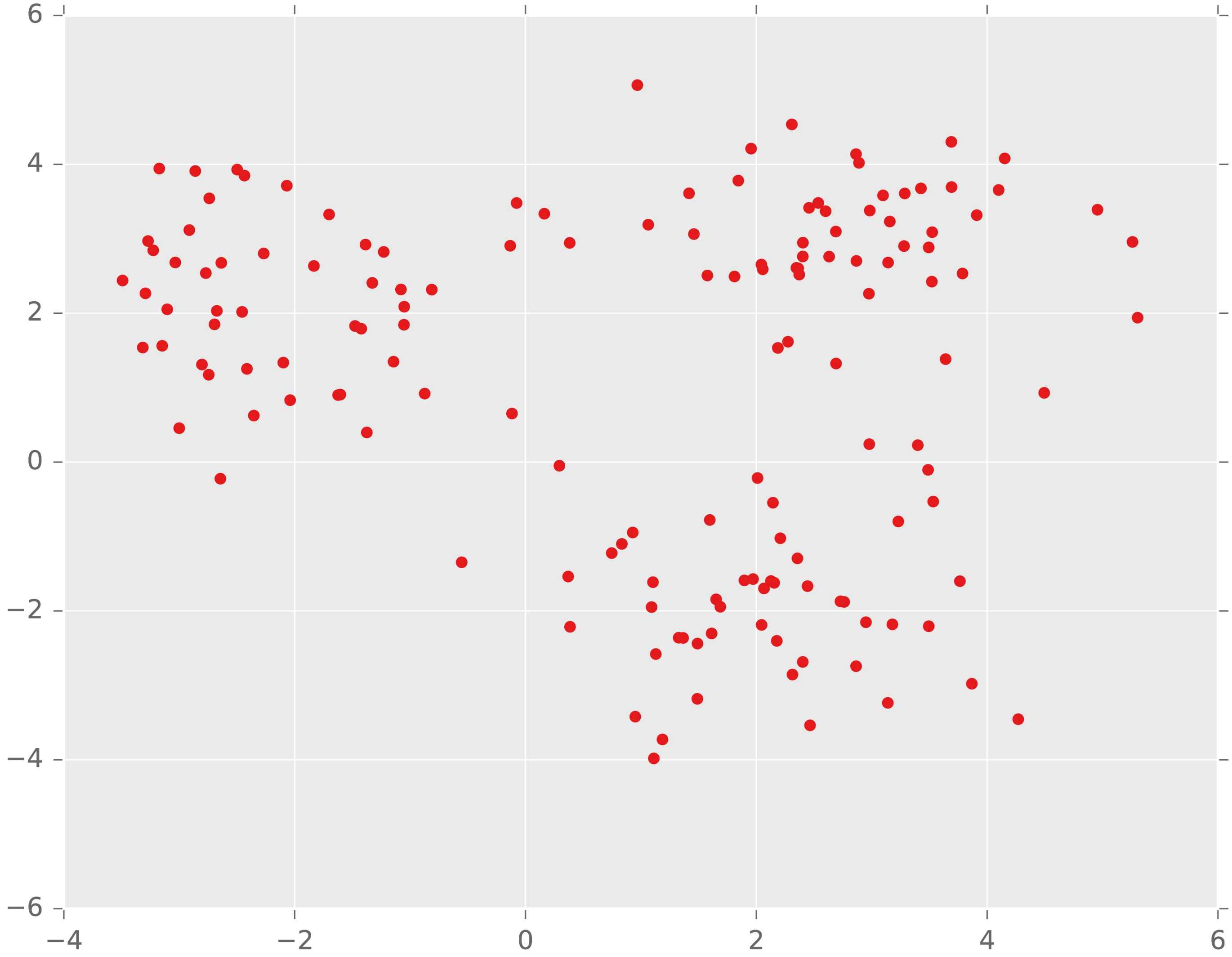
– Clustering: Inputs and Outputs
– Objective-based Clustering
– K-Means Objective
– Computational Complexity
– K-Means Algorithm / Lloyd’s Method

K=3 cluster centers

K-MEANS EXAMPLE

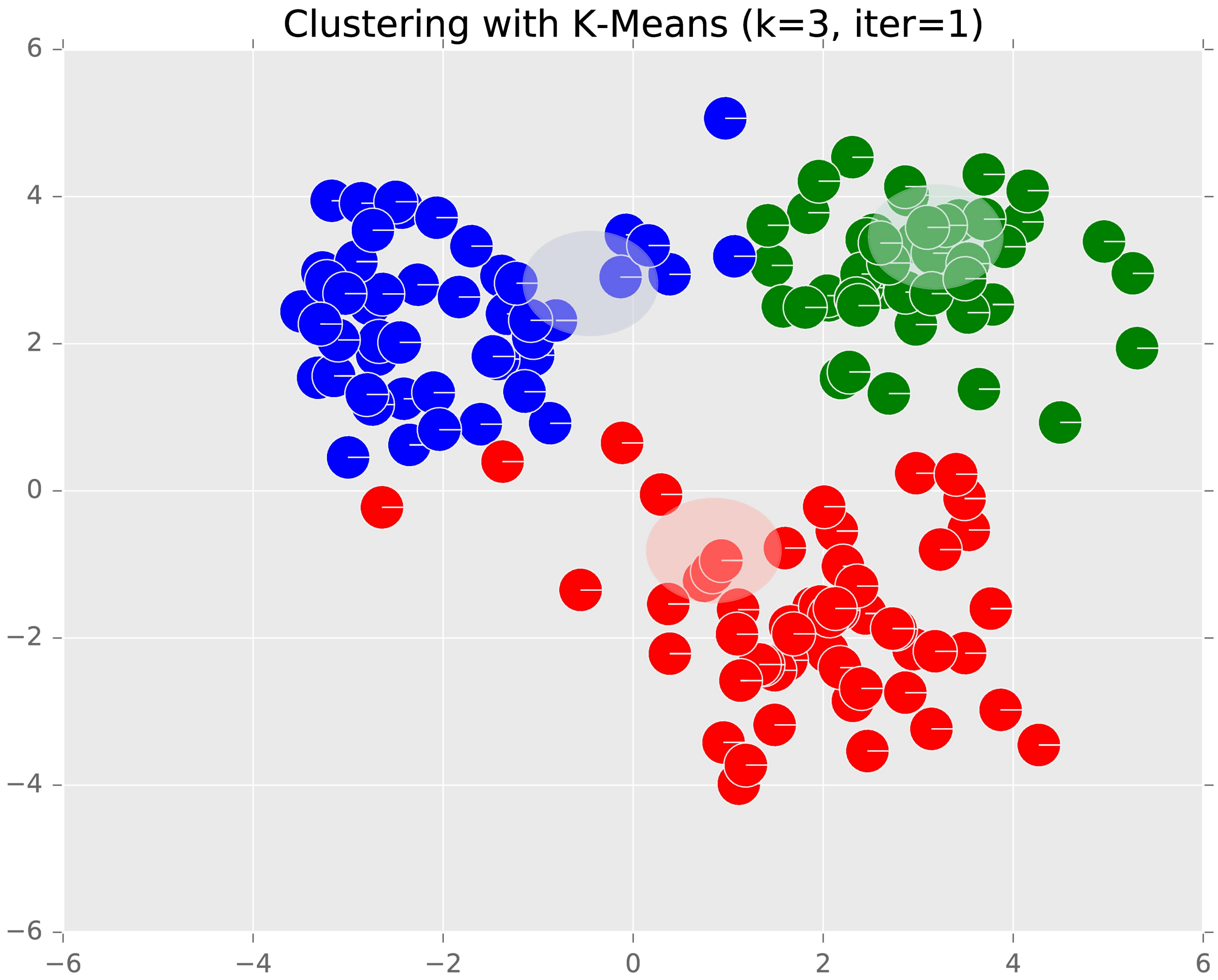
Example: K-Means

Example: K-Means

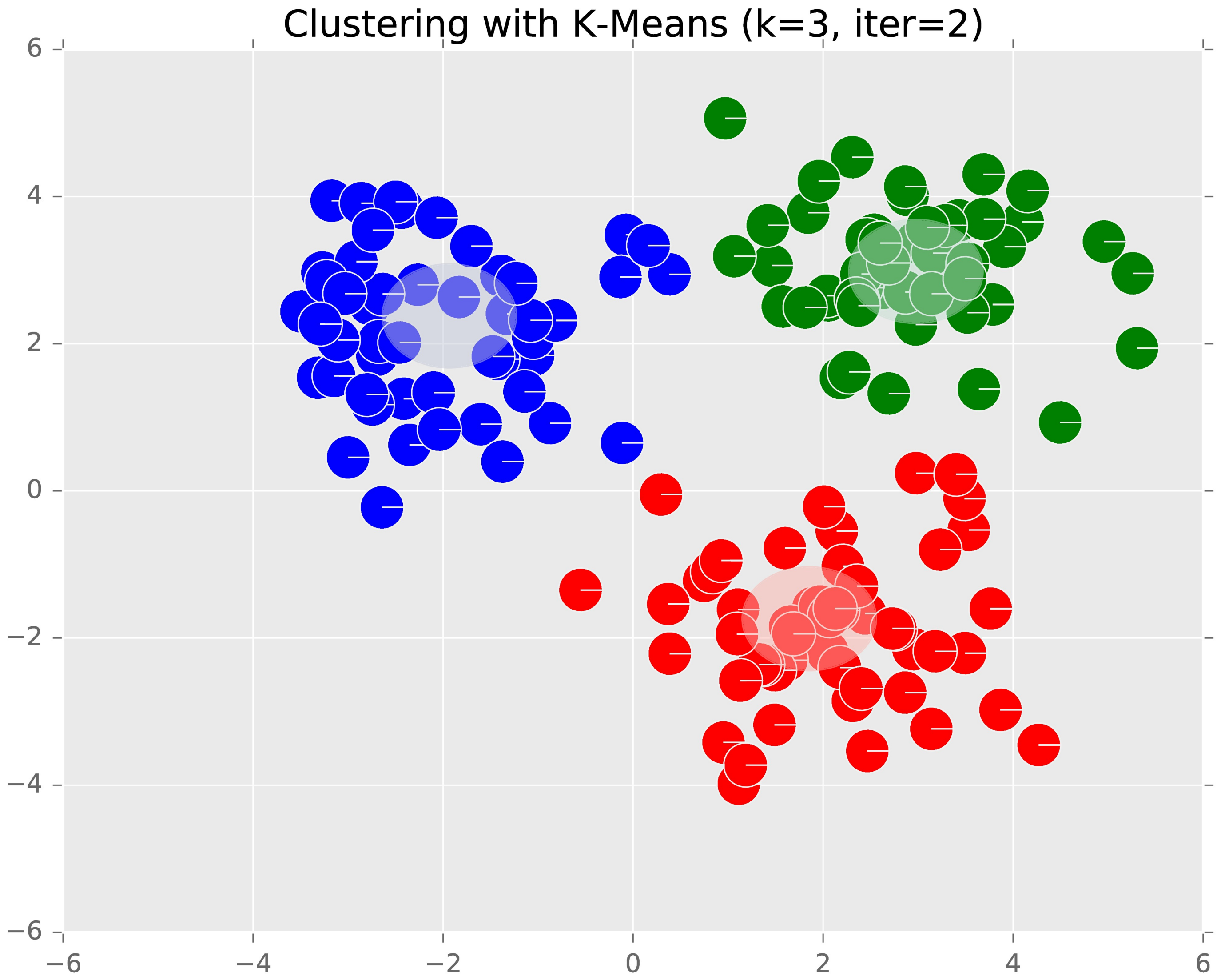


Example: K-Means

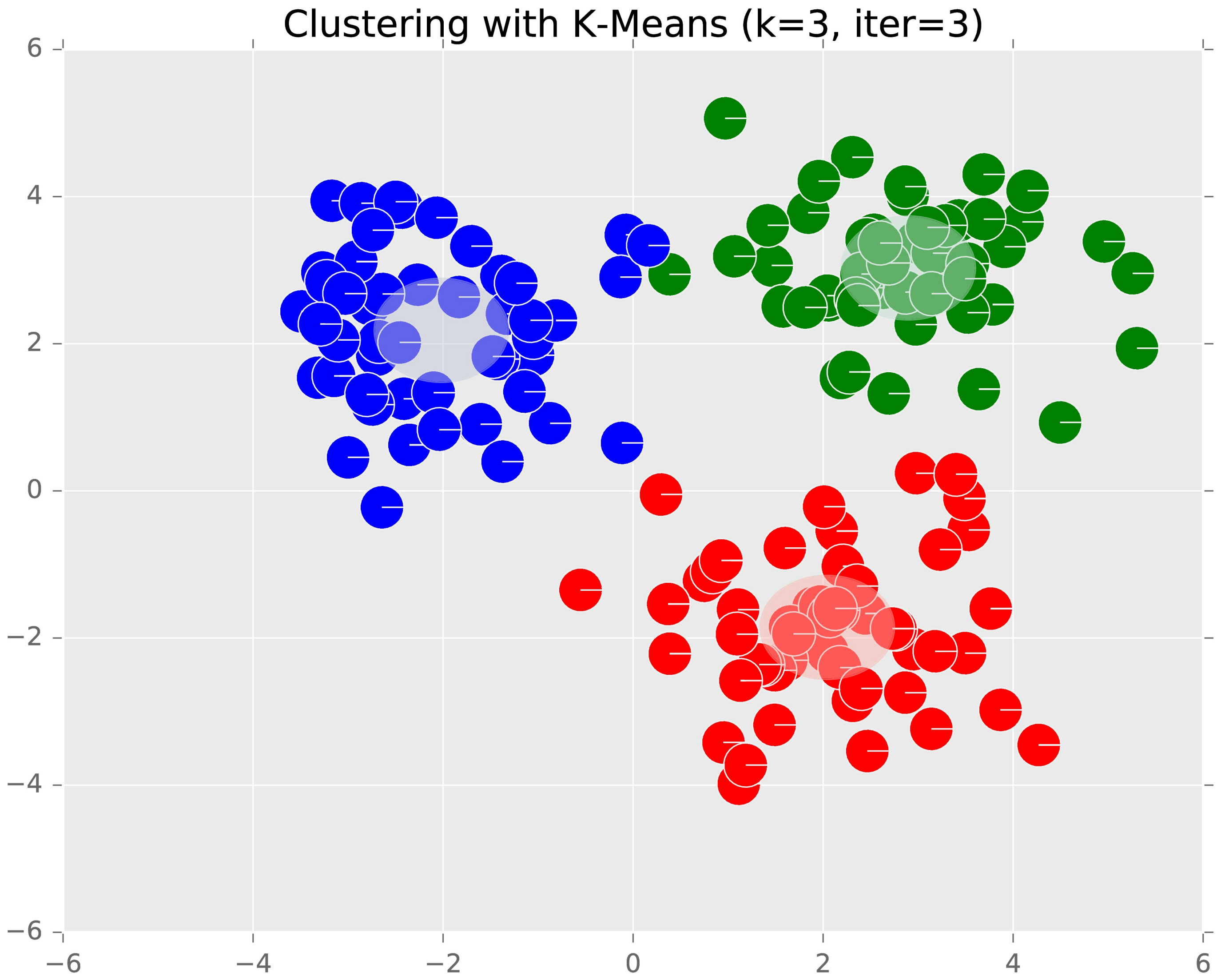
Example: K-Means



Example: K-Means

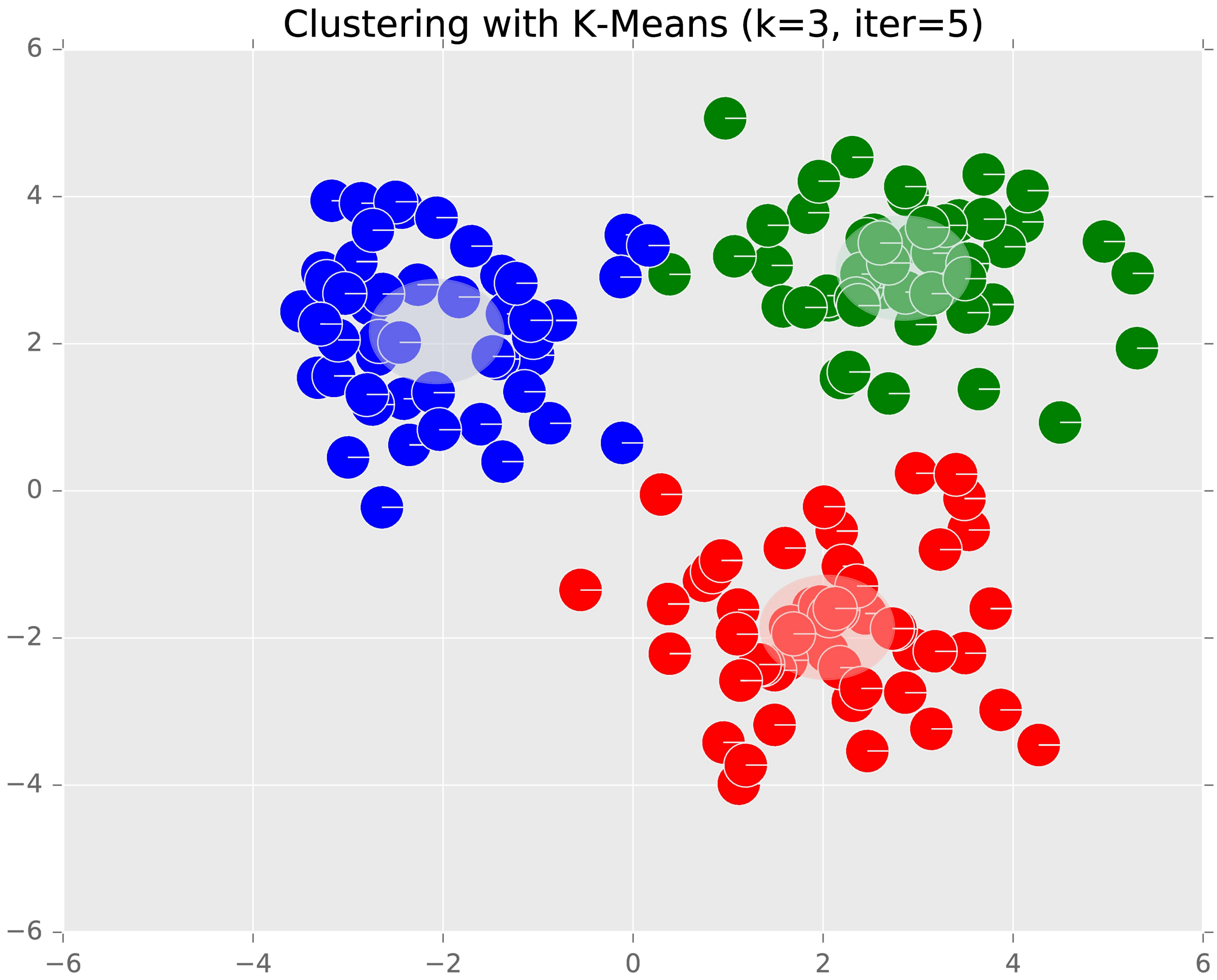


Example: K-Means



Example: K-Means

Example: K-Means

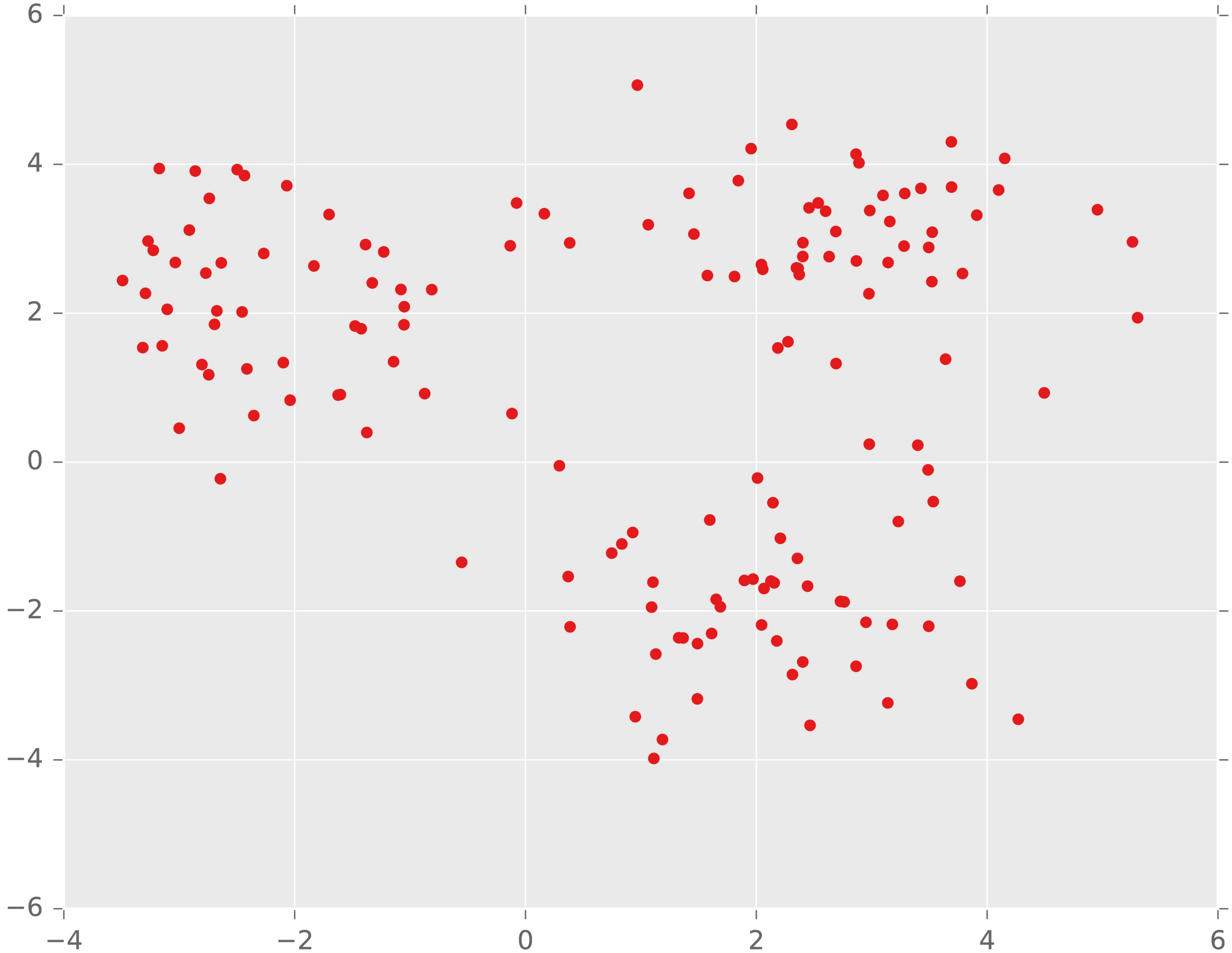


K=2 cluster centers

K-MEANS EXAMPLE

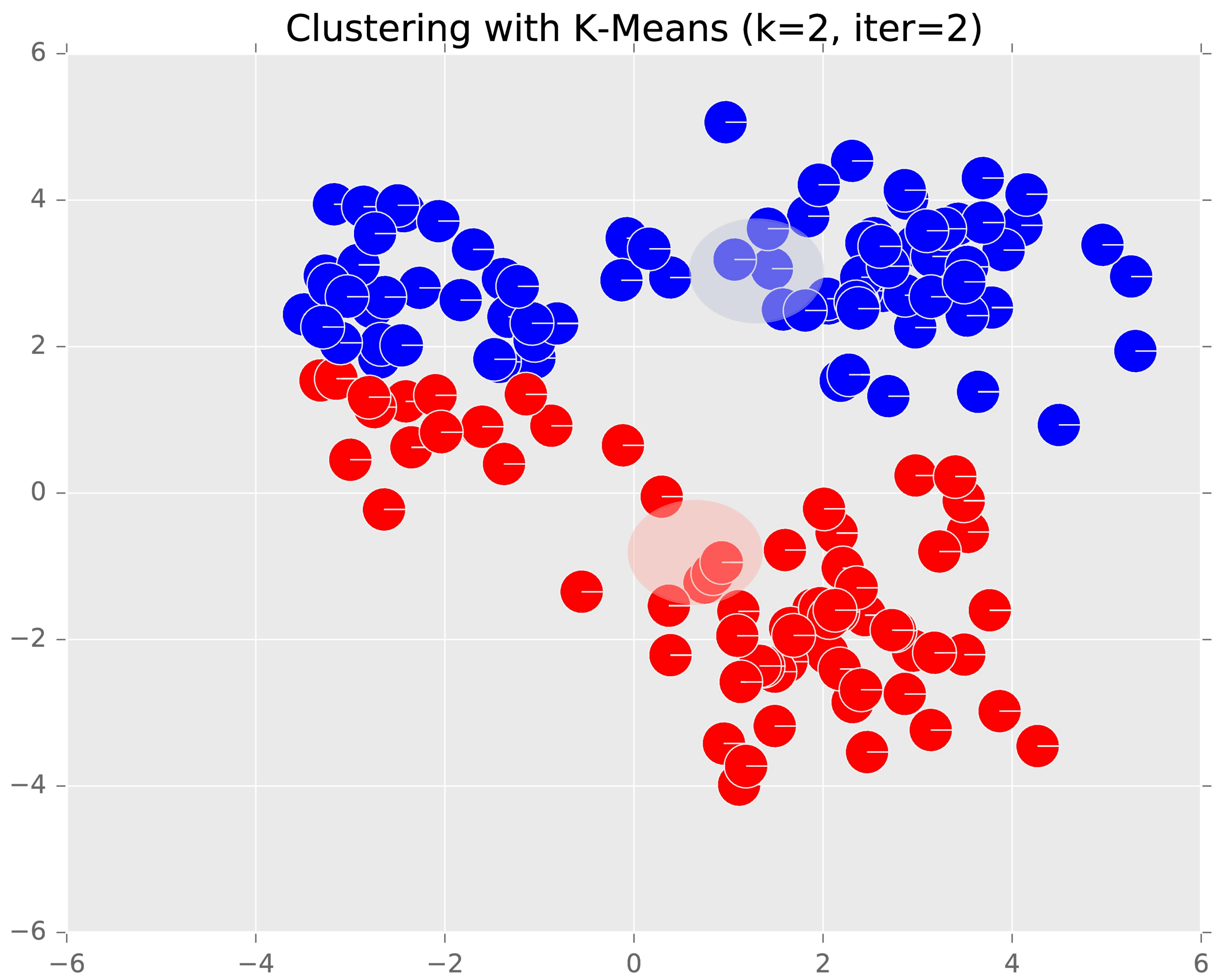
Example: K-Means

Example: K-Means

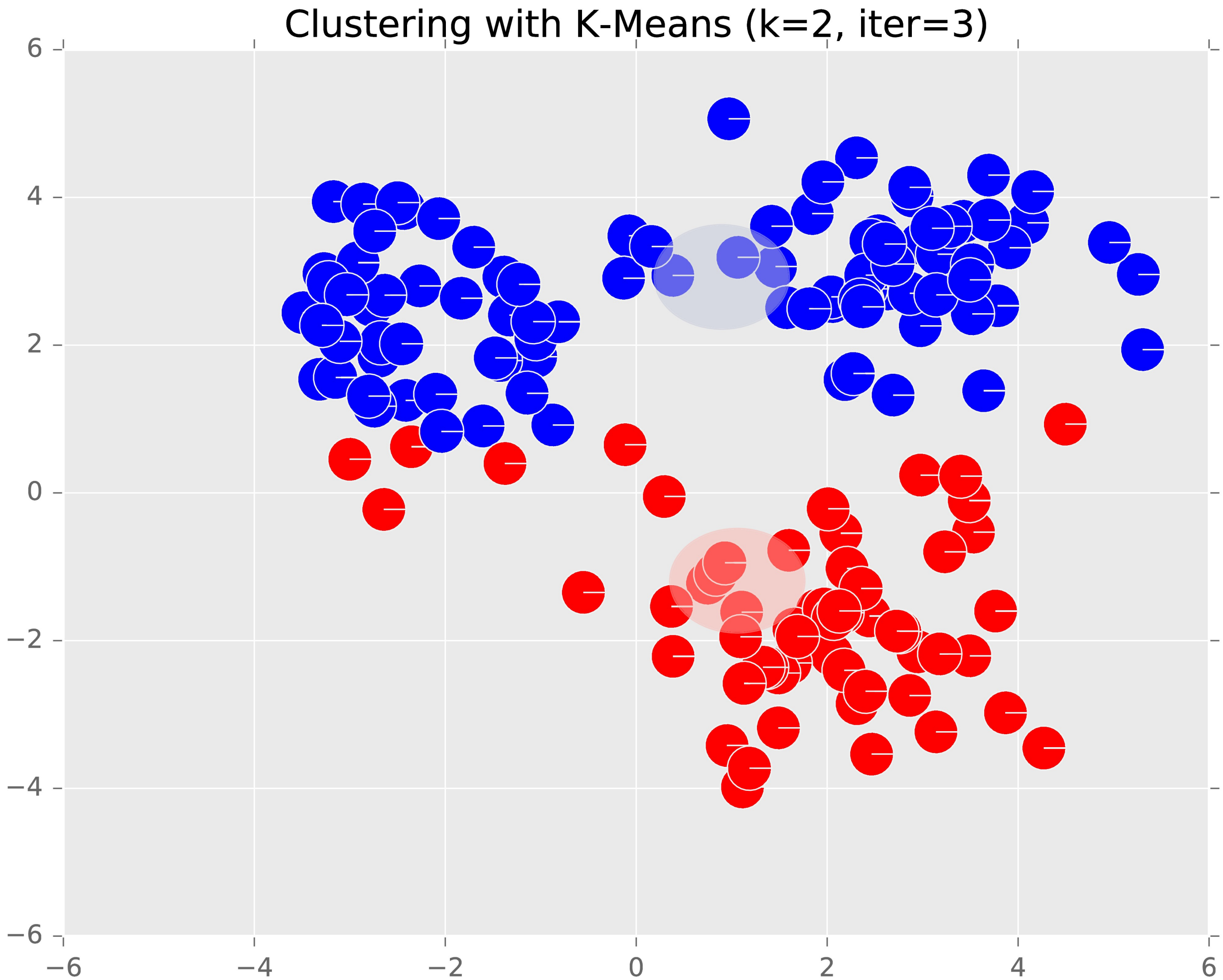


Example: K-Means

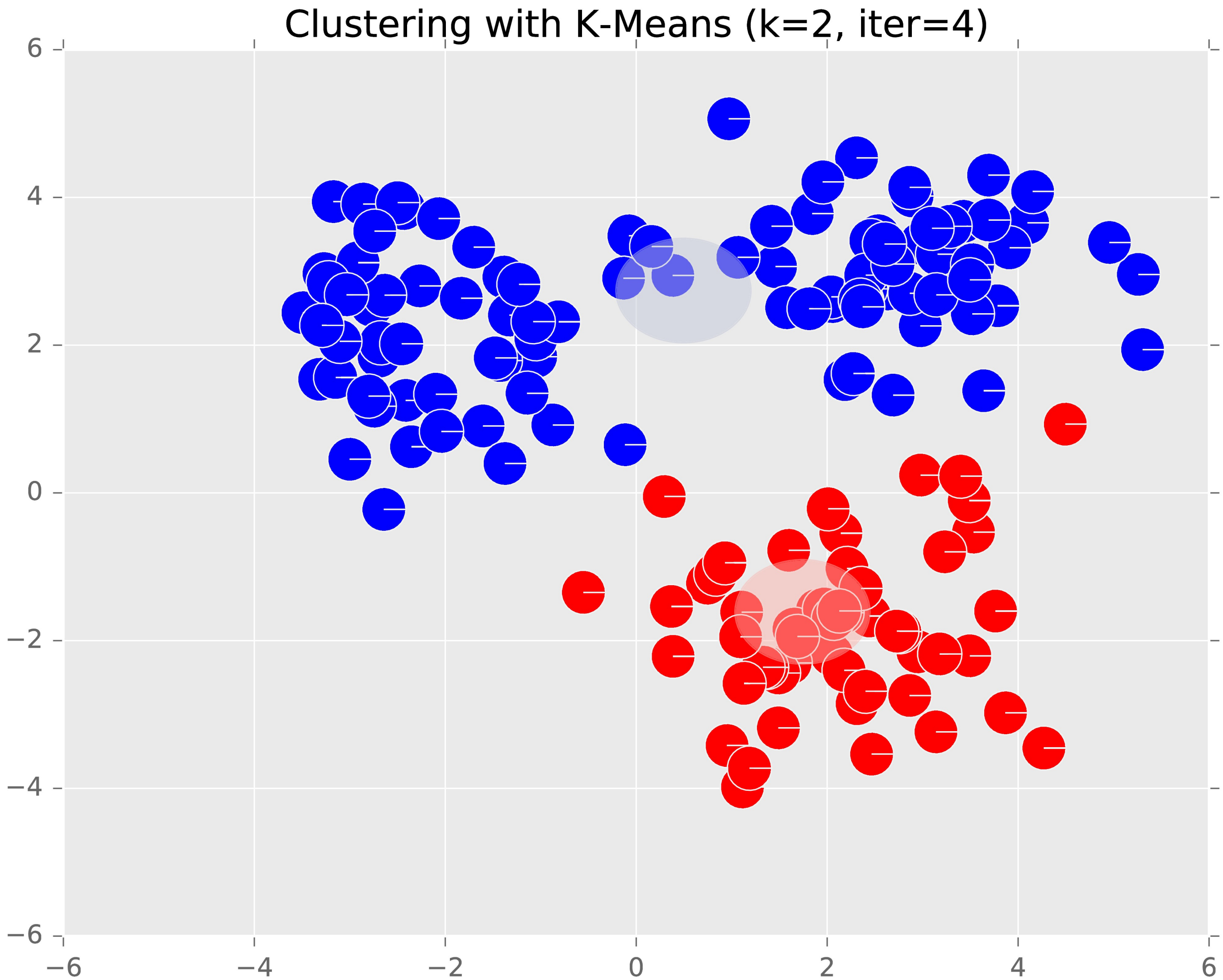
Example: K-Means



Example: K-Means

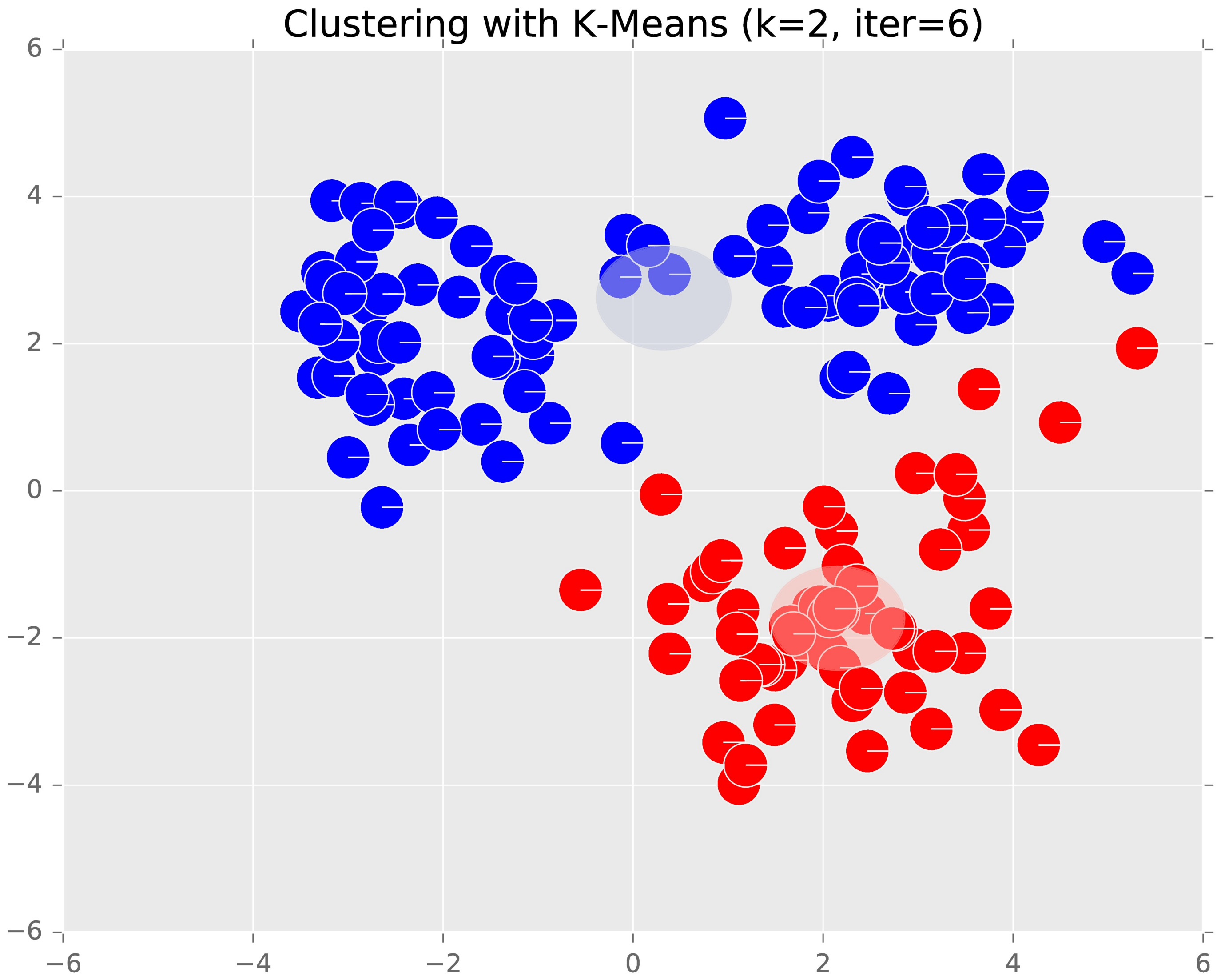


Example: K-Means

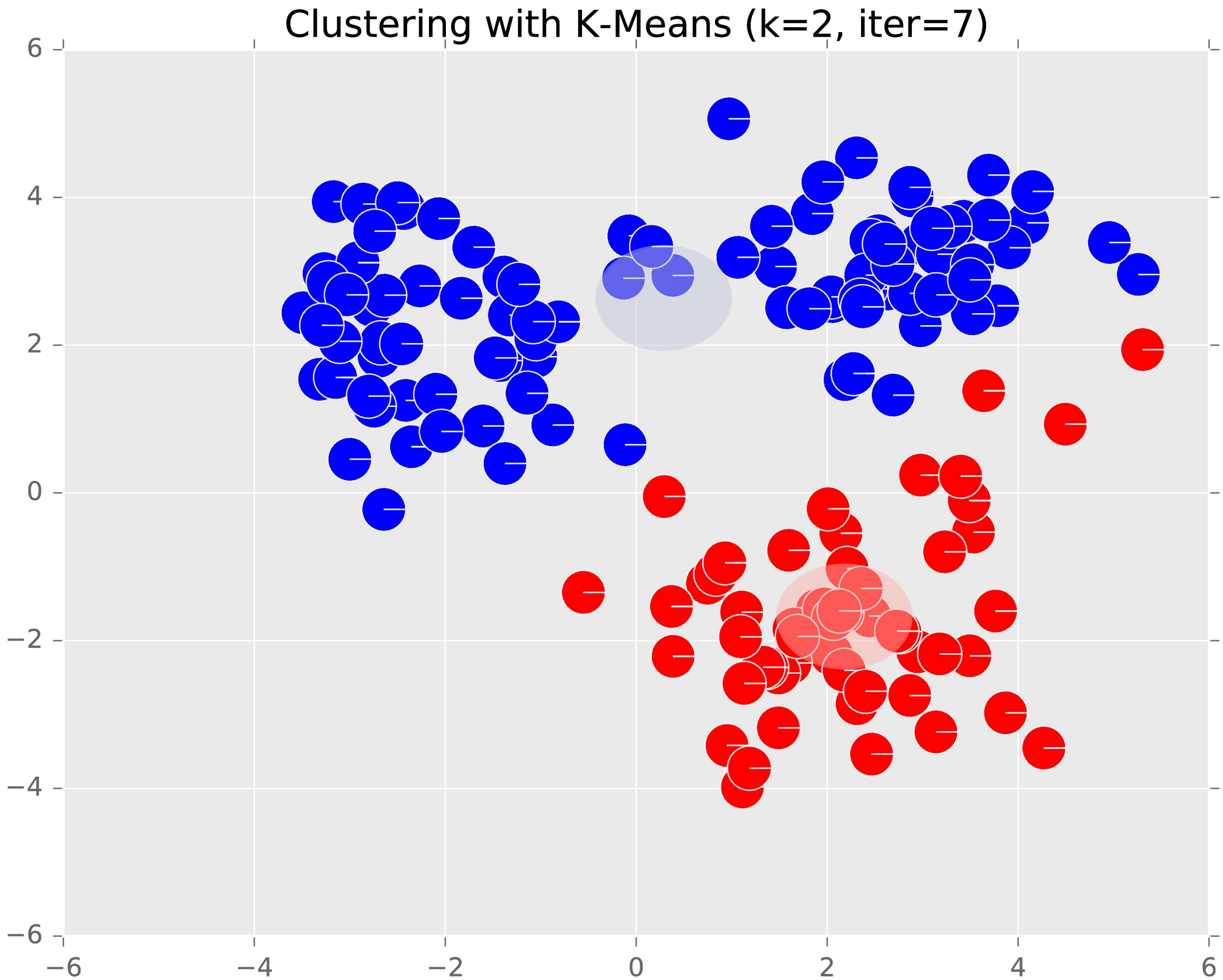


Example: K-Means

Example: K-Means



Example: K-Means



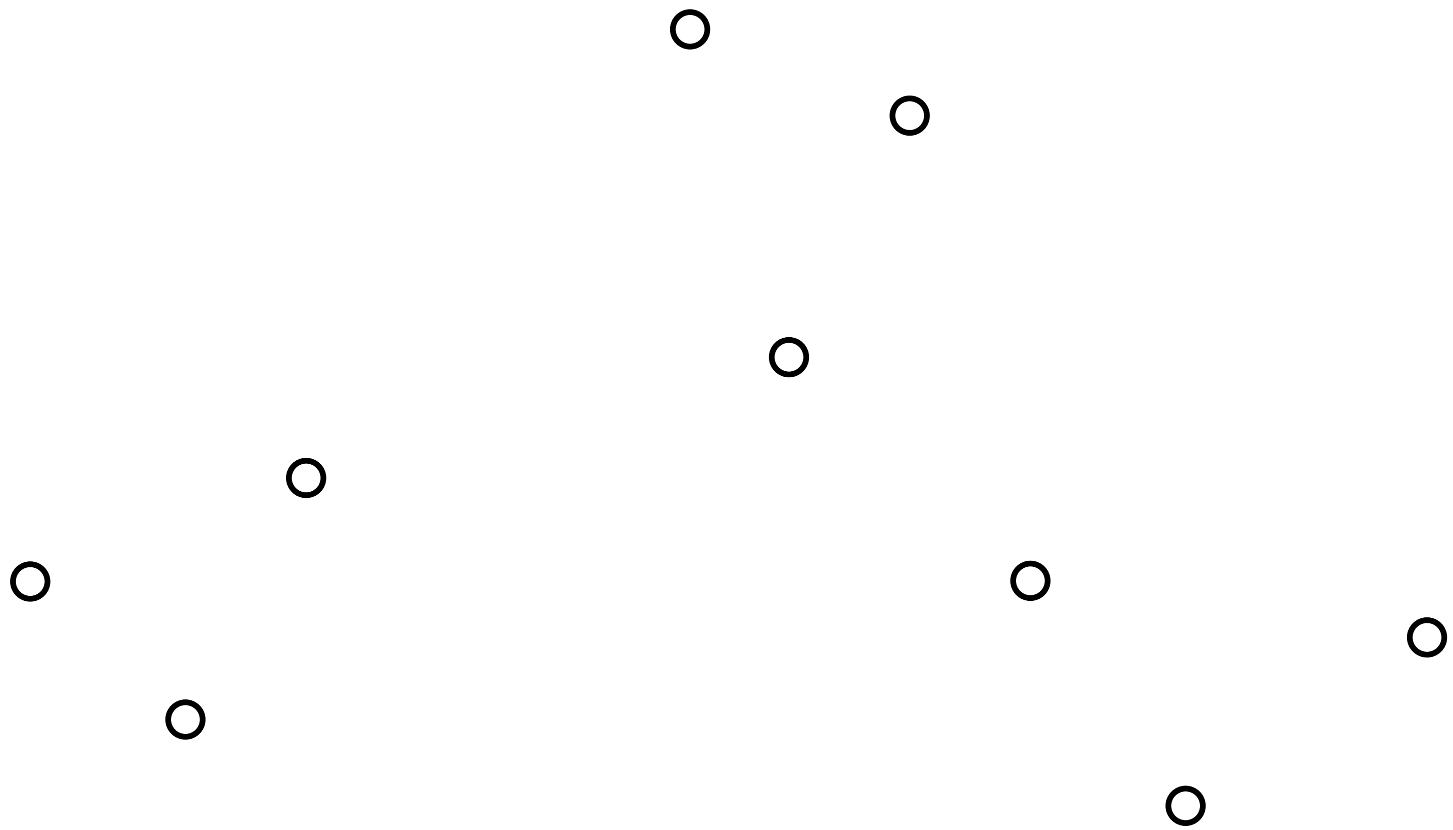
INITIALIZING K-MEANS

Initialization for K-Means

- Initialization is crucial (how fast it converges, quality of solution output)
- Techniques commonly used in practice
  - Random centers from the datapoints (repeat a few times)
  - Furthest traversal
  - K-means ++ (works well and has provable guarantees)

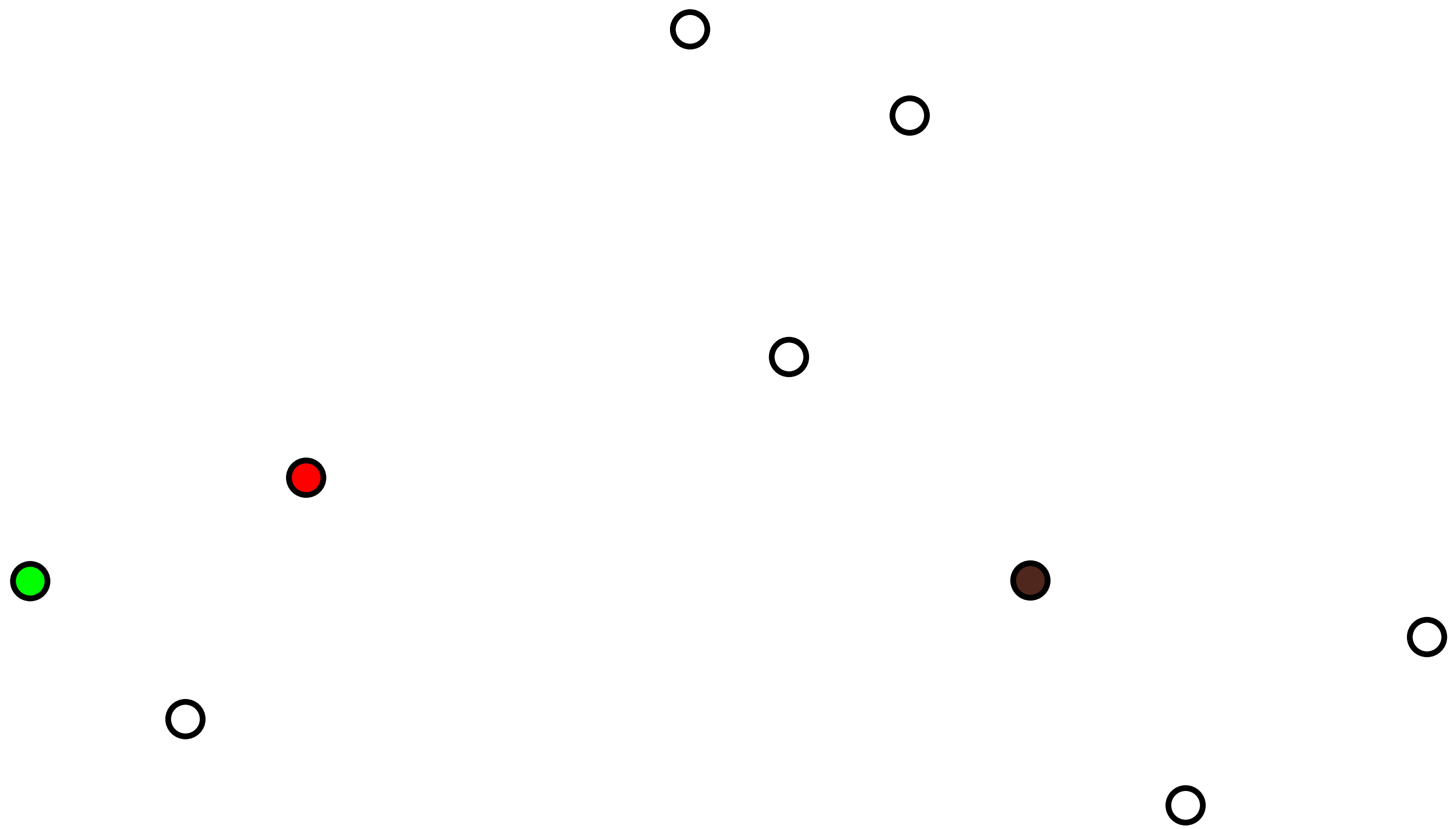
K-Means: Random Initialization

Given a set of data points



K-Means: Random Initialization

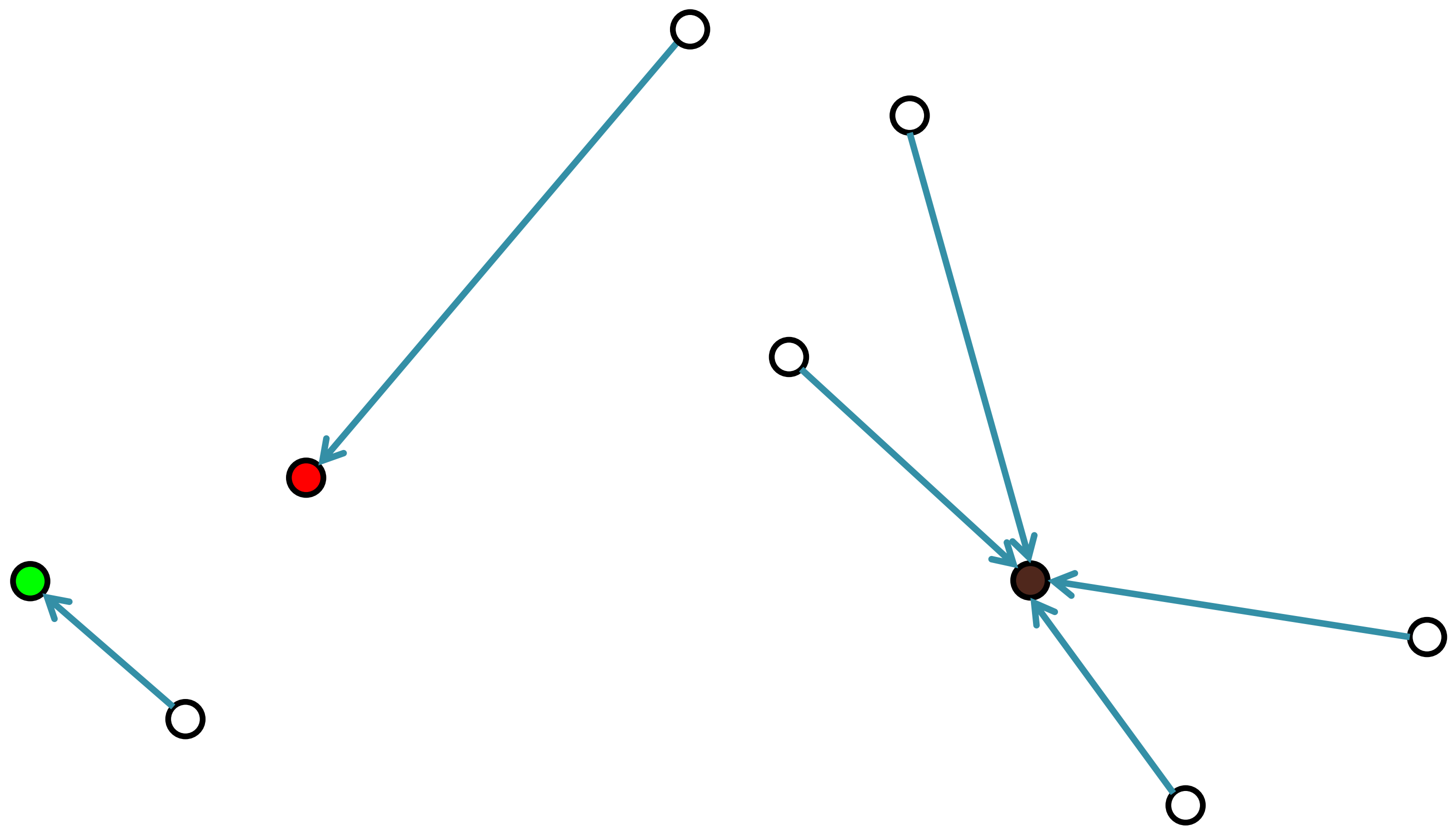
Select initial centers at random from amongst the data points



Slide courtesy of Nina Balcan

K-Means: Random Initialization

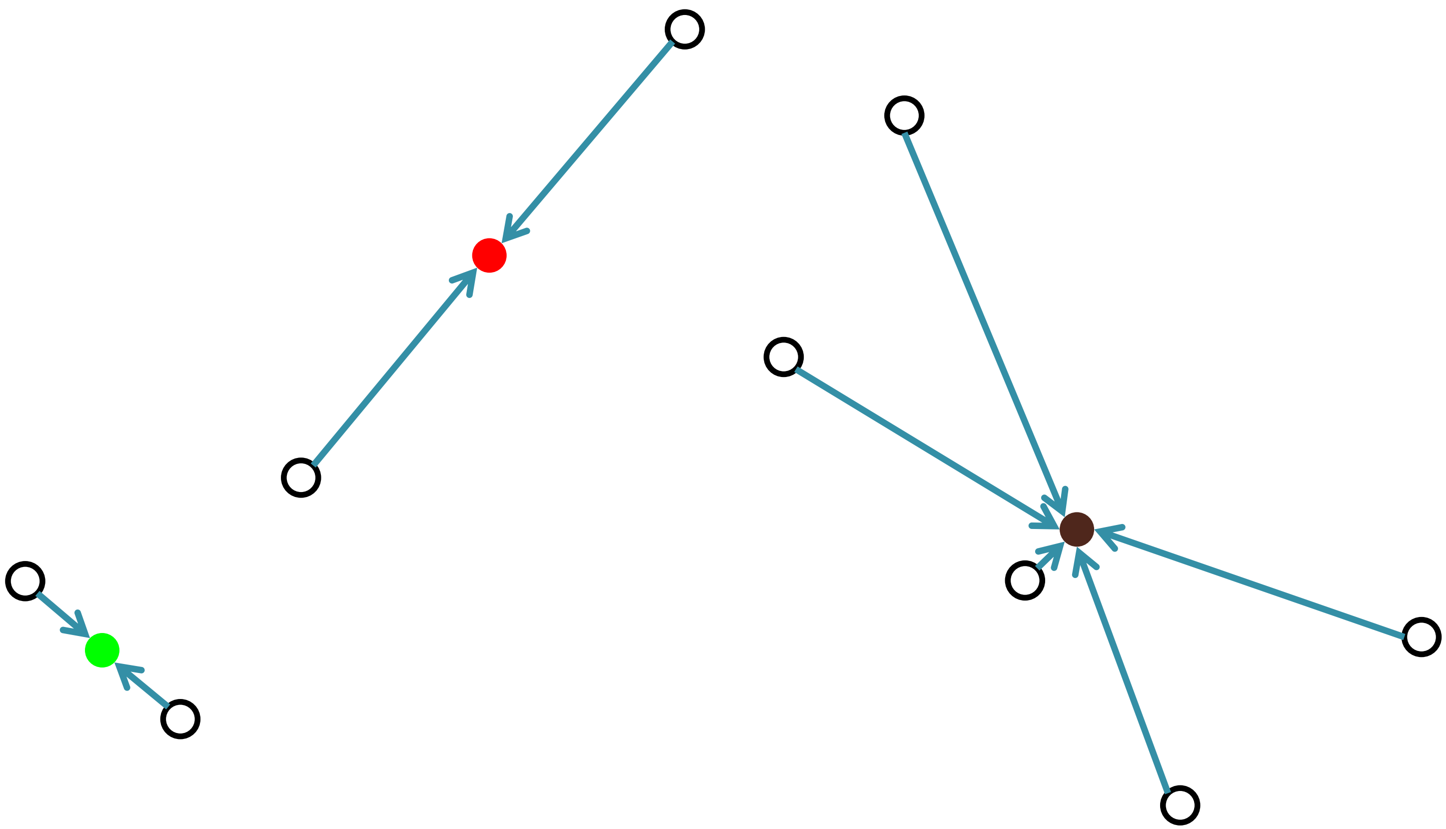
Assign each point to its nearest center



Slide courtesy of Nina Balcan

K-Means: Random Initialization

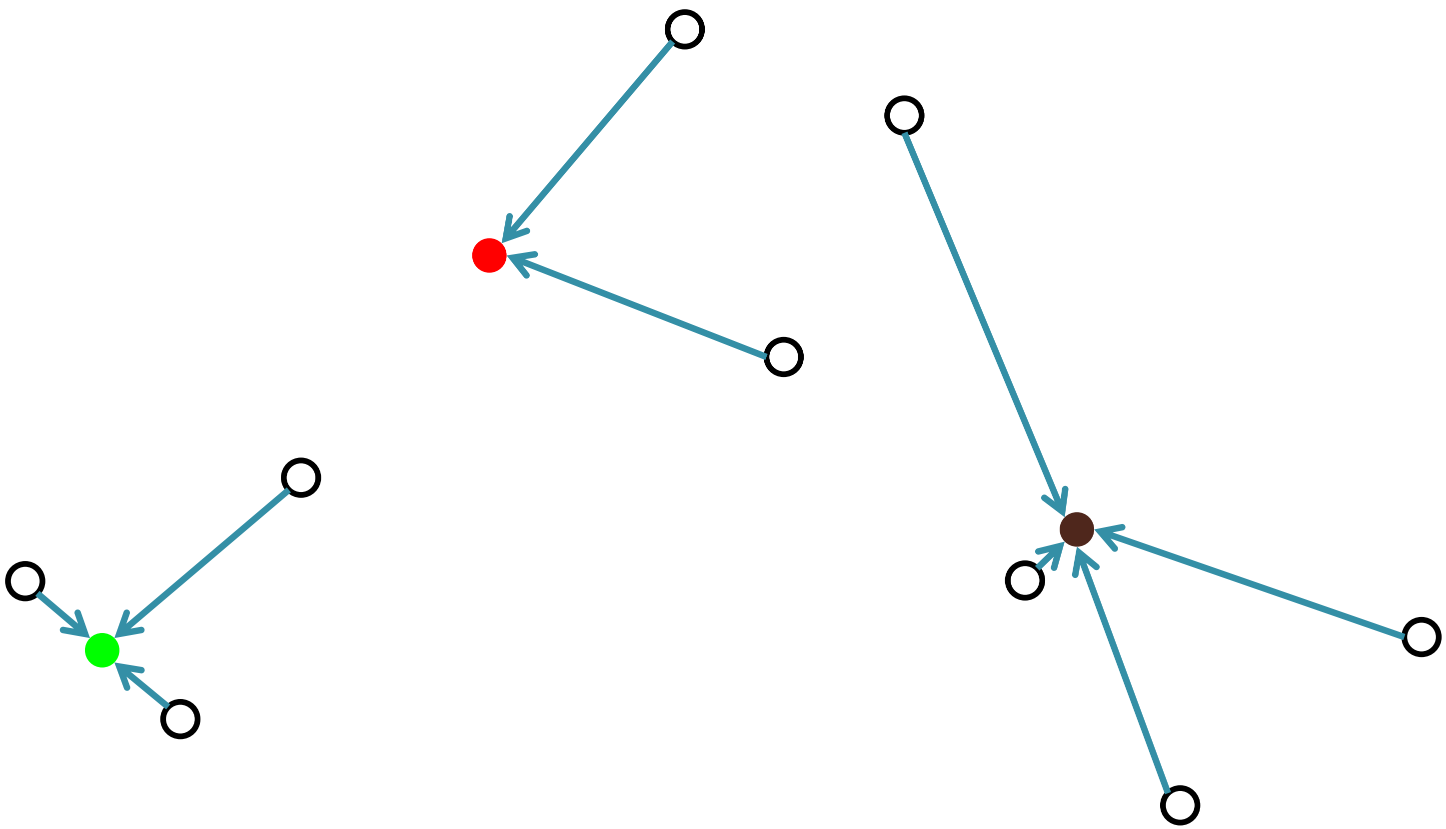
Recompute optimal centers given a fixed clustering



Slide courtesy of Nina Balcan

K-Means: Random Initialization

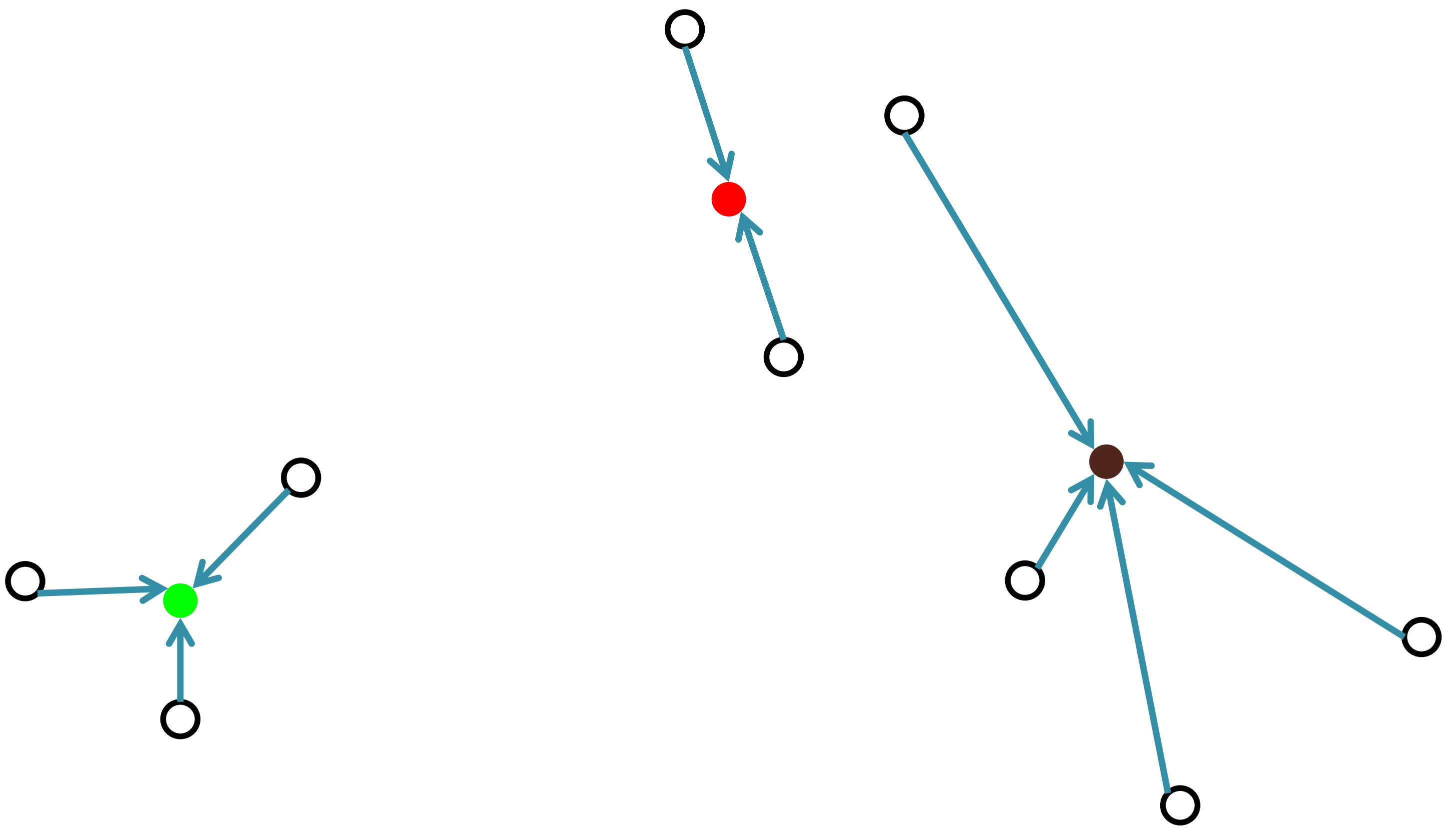
Assign each point to its nearest center



Slide courtesy of Nina Balcan

K-Means: Random Initialization

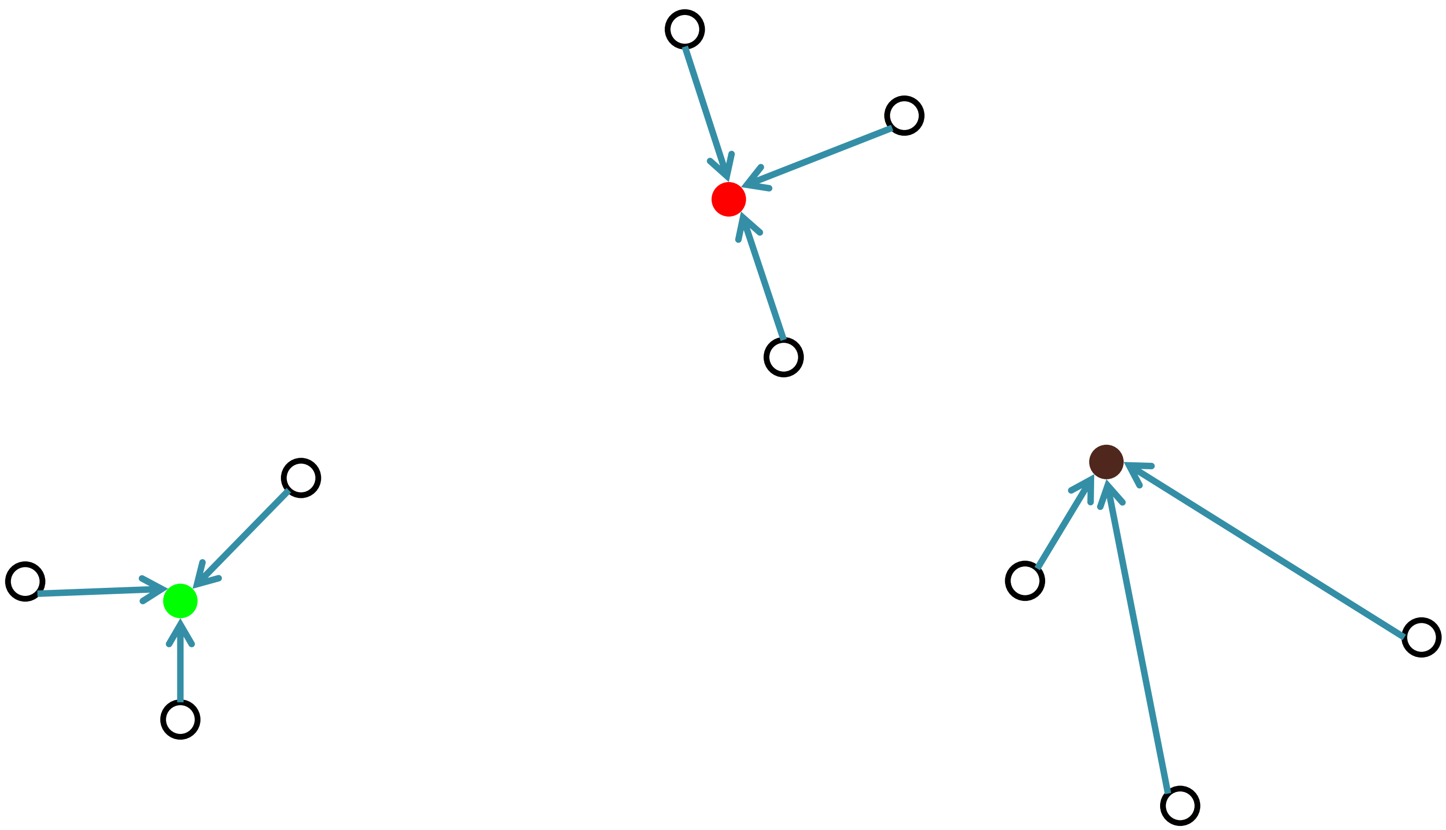
Recompute optimal centers given a fixed clustering



Slide courtesy of Nina Balcan

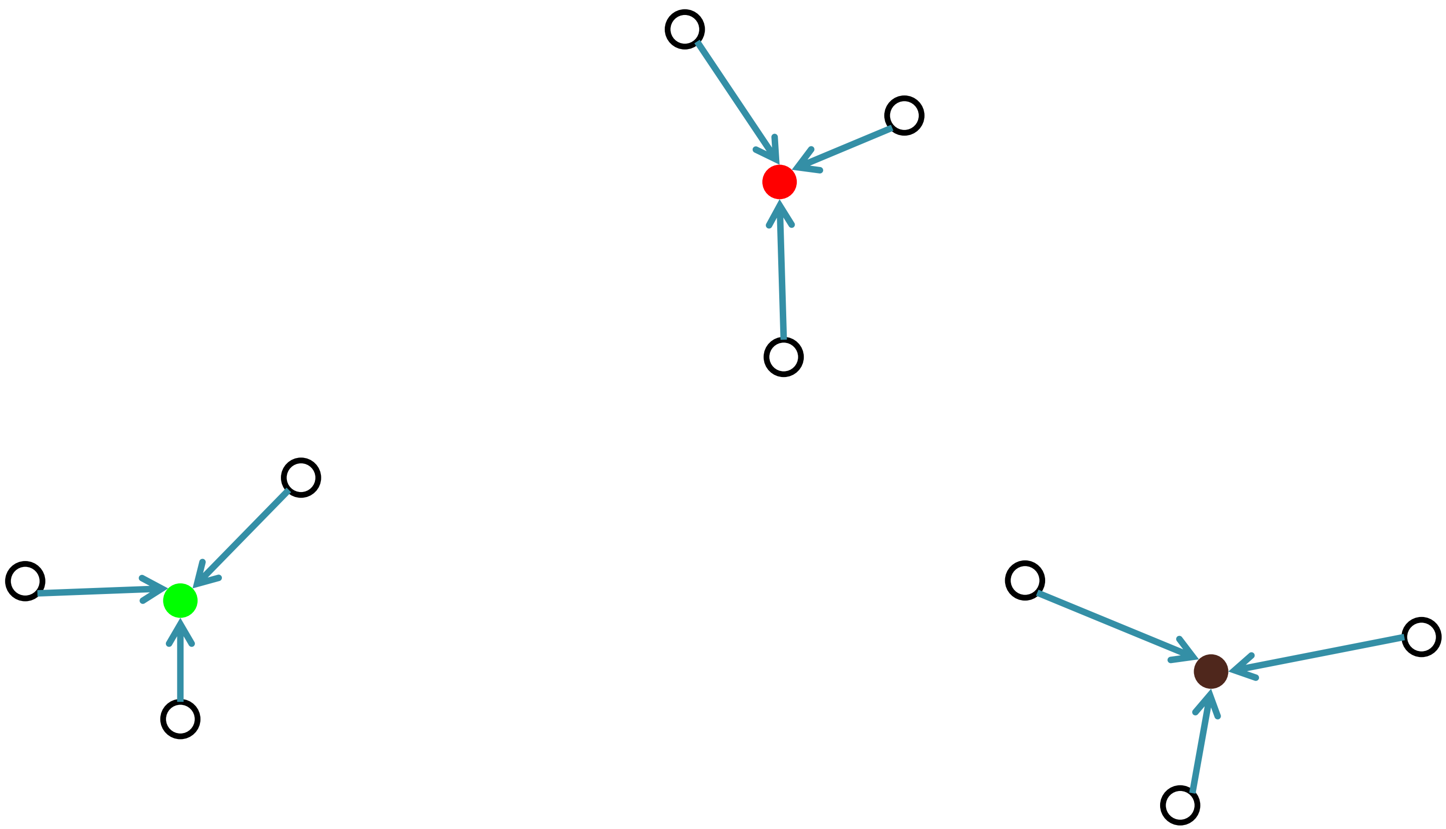
K-Means: Random Initialization

Assign each point to its nearest center



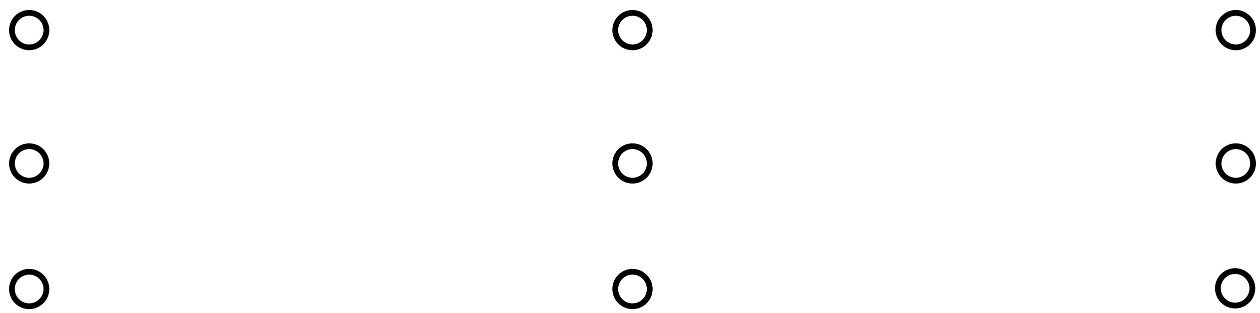
K-Means: Random Initialization

Recompute optimal centers given a fixed clustering



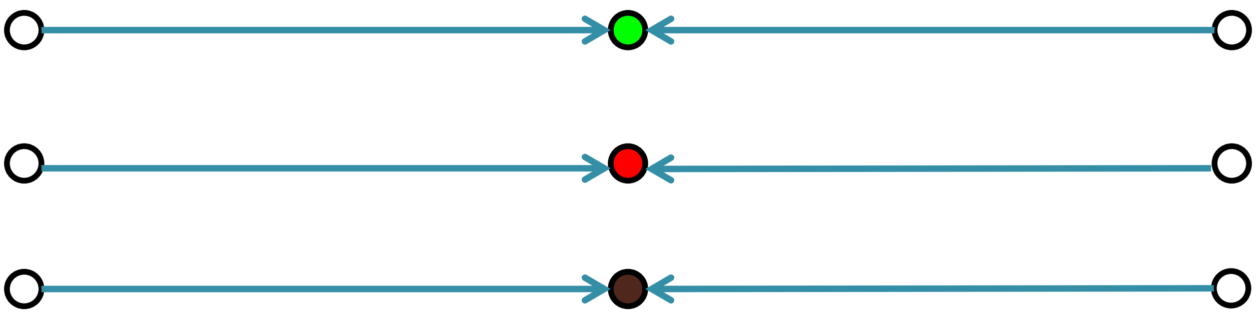
Good quality solution in this example

K-Means: Performance



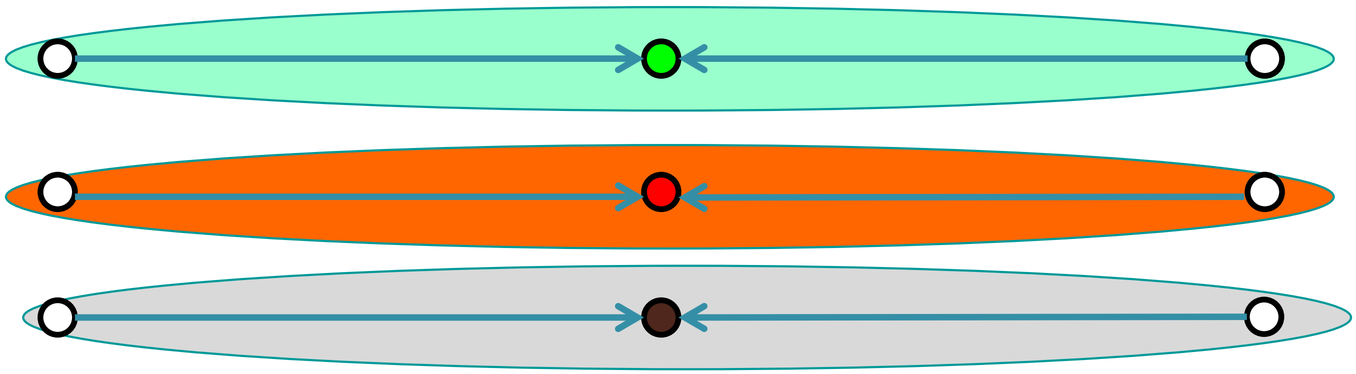
Slide courtesy of Nina Balcan

K-Means: Performance



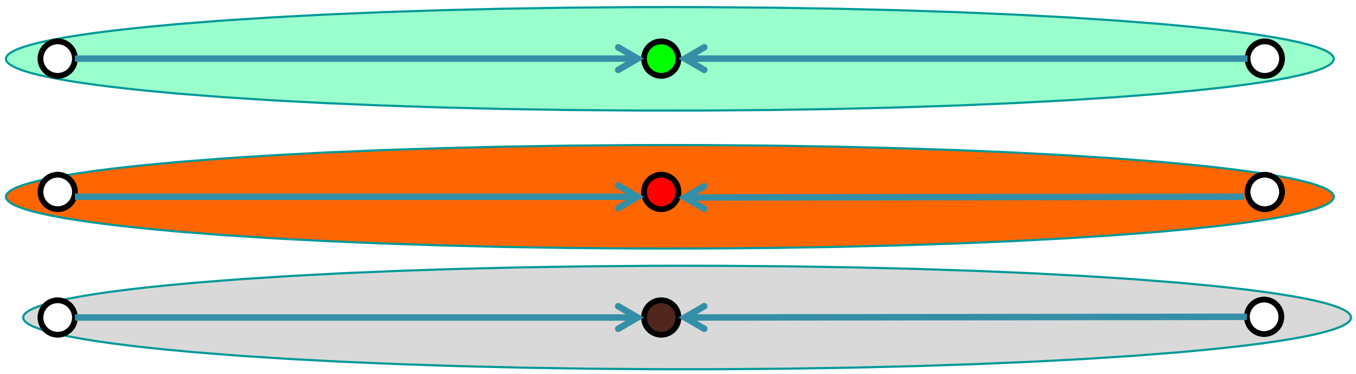
Slide courtesy of Nina Balcan

K-Means: Performance



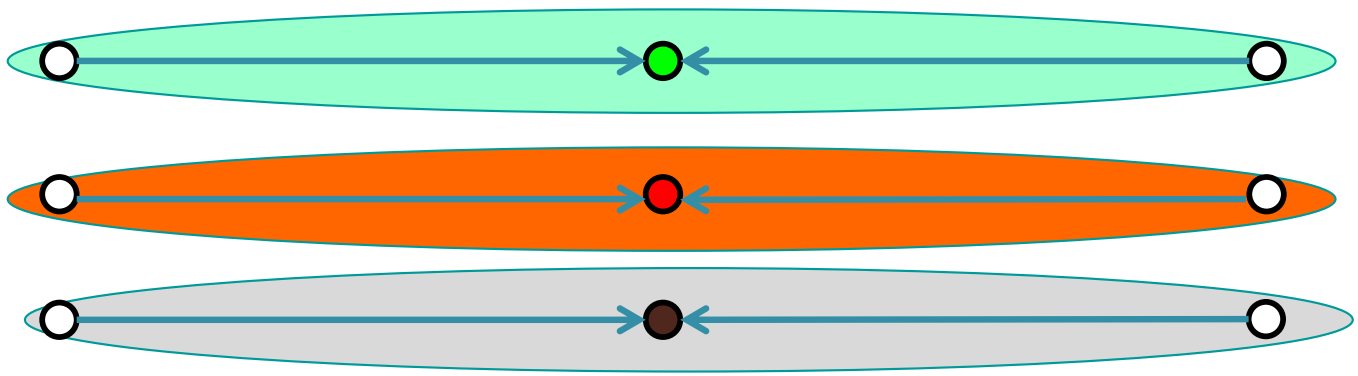
Always converges but may converge to a local optimum that is different from the global optimum, and in fact could be arbitrarily worse in terms of its score.

K-Means: Performance

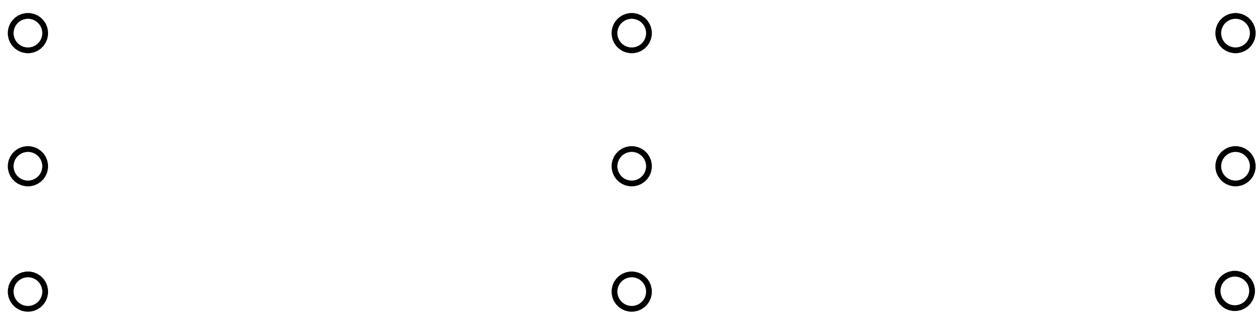


Local optimum: every point is assigned to its nearest center and every center is the mean value of its points.

K-Means: Performance

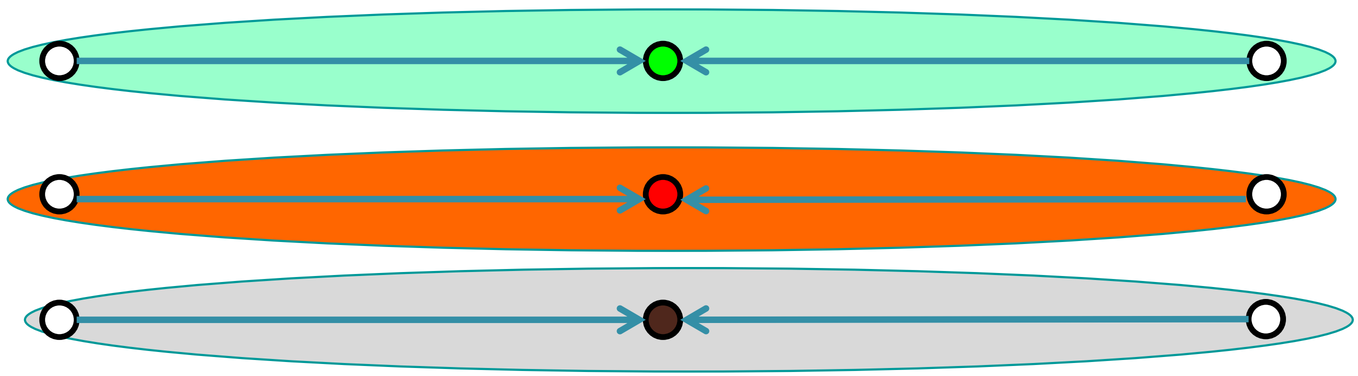


Can be arbitrarily worse than the optimum solution…

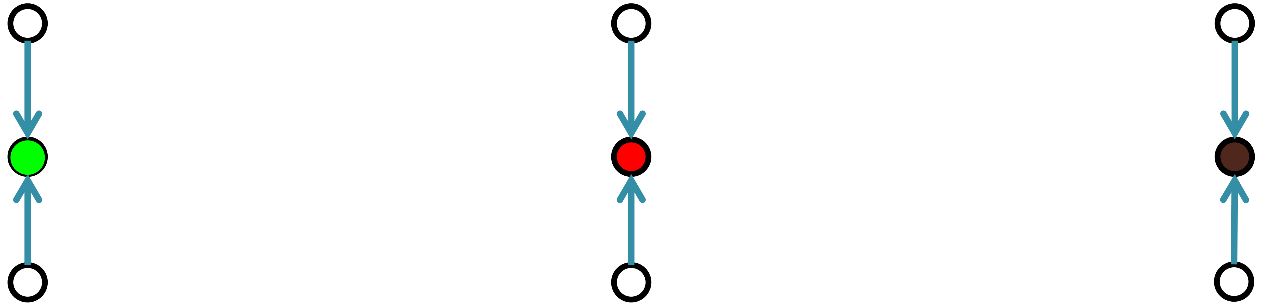


Slide courtesy of Nina Balcan

K-Means: Performance

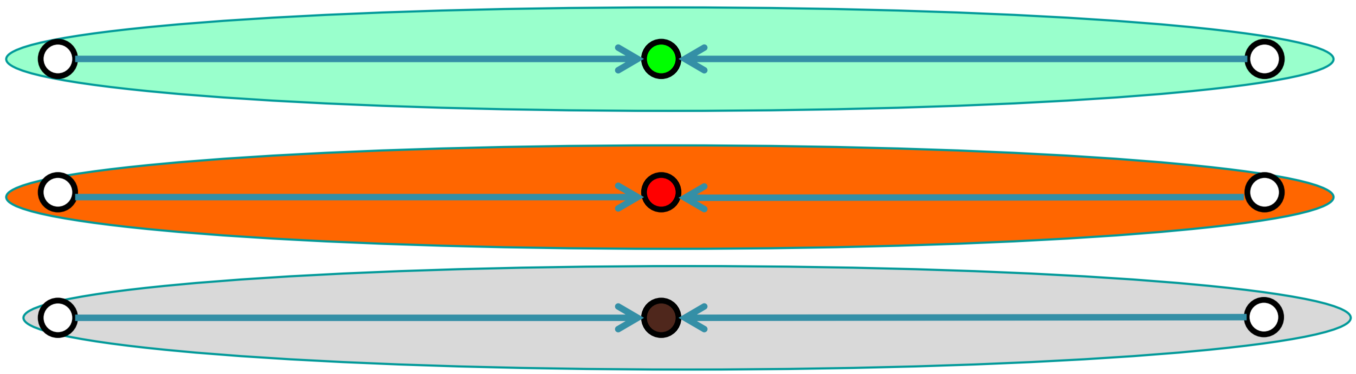


Can be arbitrarily worse than the optimum solution…

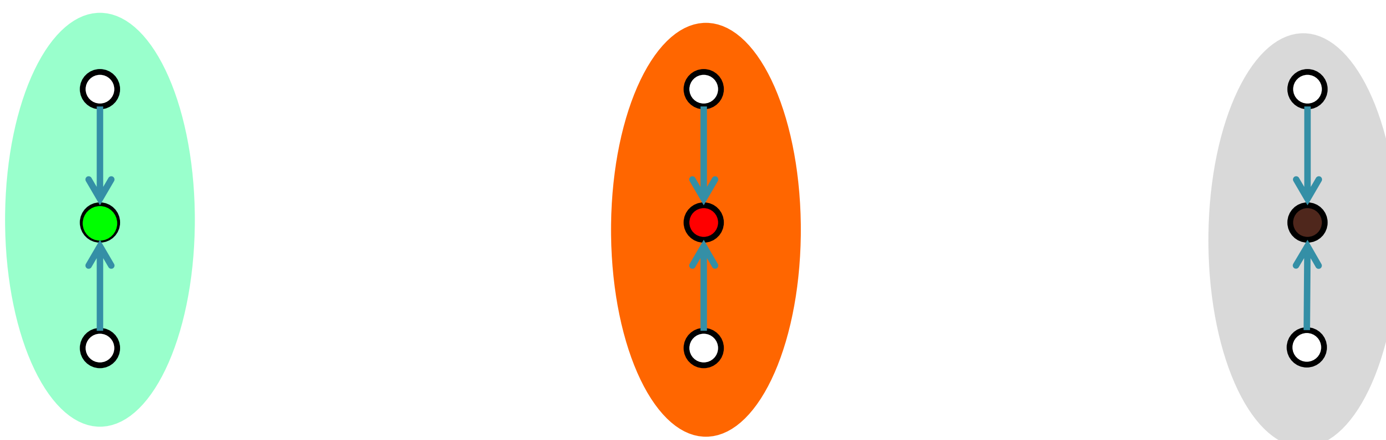


Slide courtesy of Nina Balcan

K-Means: Performance

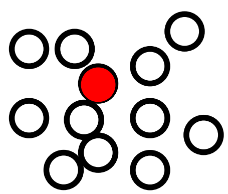


Can be arbitrarily worse than the optimum solution…



Slide courtesy of Nina Balcan

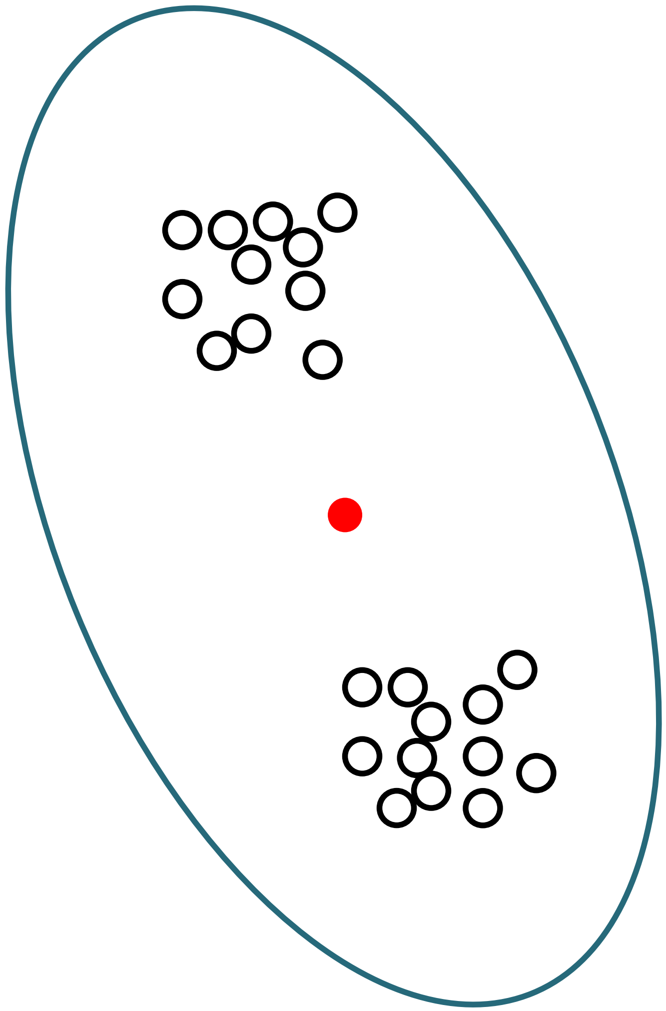
K-Means: Performance



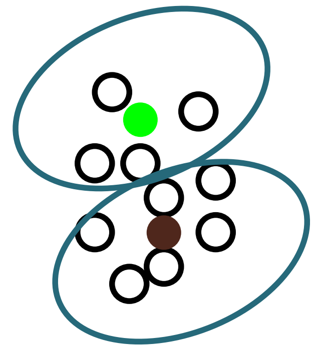
This bad performance, can happen even with well separated Gaussian clusters.

Slide courtesy of Nina Balcan

K-Means: Performance



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



K-Means: Performance

- 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

Another Initialization Idea: Furthest Point Heuristic

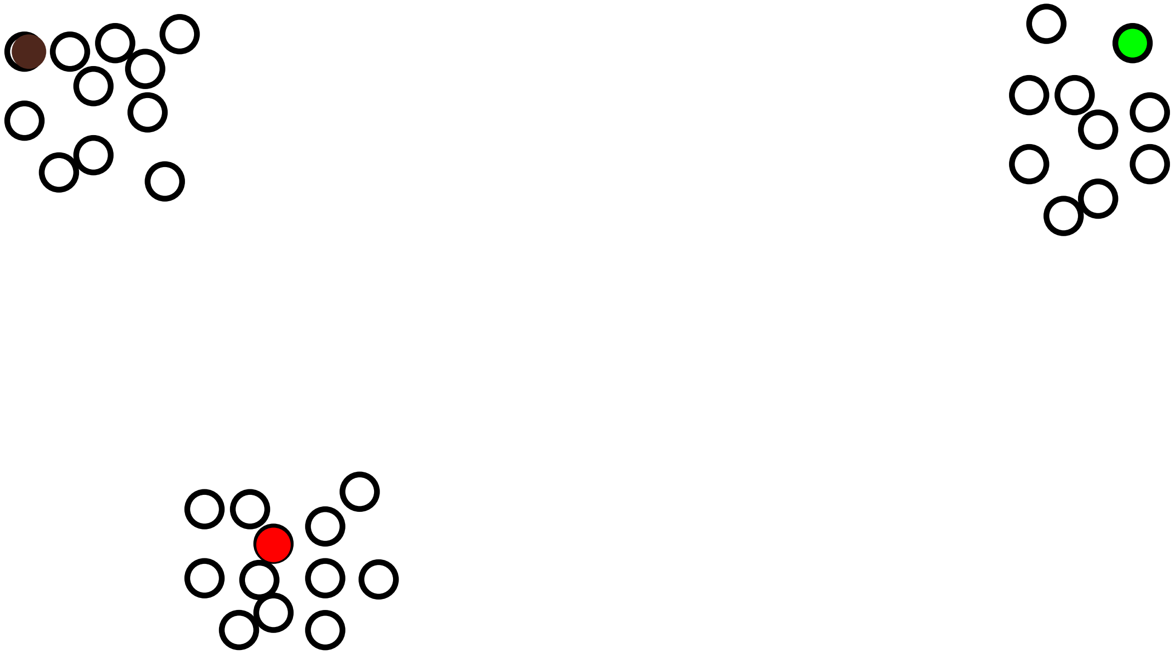
Choose $\mathbf{c_1}$ arbitrarily (or at random).

- For $\mathrm{j} = 2, \dots, \mathrm{k}$
  - Pick $\mathbf{c_j}$ among datapoints $\mathbf{x^1}, \mathbf{x^2}, \dots, \mathbf{x^n}$ that is farthest from previously chosen $\mathbf{c_1}, \mathbf{c_2}, \dots, \mathbf{c}_{\boldsymbol{j-1}}$

Fixes the Gaussian problem. But it can be thrown off by outliers….

Slide courtesy of Nina Balcan

Furthest point heuristic does well on previous example

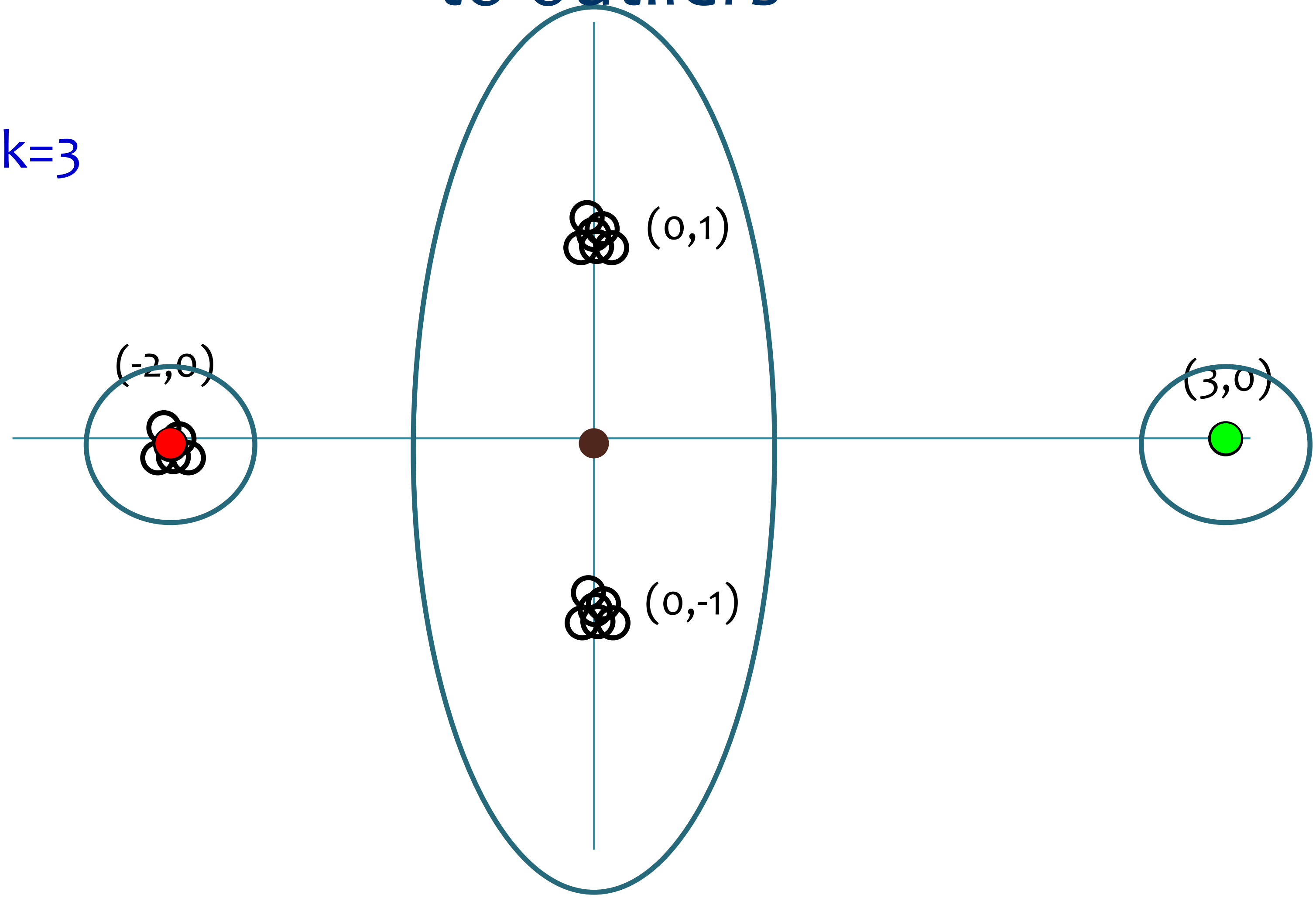


Furthest point initialization heuristic sensitive to outliers

Assume k=3

Furthest point initialization heuristic sensitive to outliers

Assume k=3



Slide courtesy of Nina Balcan

K-means++ Initialization: D^2 sampling [AV07]

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

- Choose $\mathbf{c_1}$ at random.
- For $\mathrm{j} = 2, \dots, \mathrm{k}$
  - Pick $\mathbf{c_j}$ among $\mathbf{x}^1, \mathbf{x}^2, \dots, \mathbf{x}^n$ according to the distribution

$$\mathbf{Pr}(\mathbf{c_j} = \mathbf{x^i}) \propto \mathbf{min}_{\mathbf{j'<j}} \left\|\mathbf{x^i} - \mathbf{c_{j'}}\right\|^2 \quad \mathrm{D}^2(\mathbf{x^i})$$

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

Running K-Means can only further improve the cost.

Slide courtesy of Nina Balcan

K-means++ Idea: D^2 sampling

- Interpolate between random and furthest point initialization
- Let $\mathrm{D}(\mathbf{x})$ be the distance between a point x and its nearest center. Chose the next center proportional to $\mathrm{D}^{\alpha}(\mathbf{x})$.

  - $\alpha = 0$, random sampling
  - $\alpha = \infty$, furthest point (Side note: it actually works well for k-center)
  - $\alpha = 2$, k-means++

Side note: $\alpha = 1$, works well for k-median

Slide courtesy of Nina Balcan

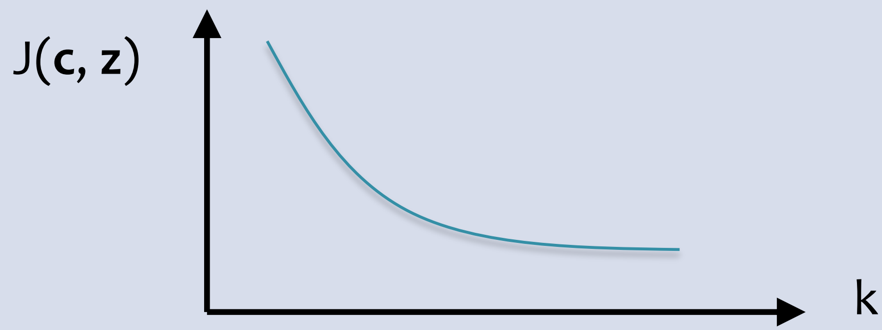
K-means++ handles the case on which furthest point heuristic failed

Slide courtesy of Nina Balcan

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

Learning Paradigms

Paradigm	**Data**
Supervised	$\mathcal{D} = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^N \qquad \mathbf{x} \sim p^*(\cdot)$ and $y = c^*(\cdot)$
$\hookrightarrow$ Regression	$y^{(i)} \in \mathbb{R}$
$\hookrightarrow$ Classification	$y^{(i)} \in \{1, \ldots, K\}$
$\hookrightarrow$ Binary classification	$y^{(i)} \in \{+1, -1\}$
$\hookrightarrow$ Structured Prediction	$\mathbf{y}^{(i)}$ is a vector
Unsupervised	$\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^N \qquad \mathbf{x} \sim p^*(\cdot)$
$\hookrightarrow$ Clustering	predict $\{z^{(i)}\}_{i=1}^N$ where $z^{(i)} \in \{1, \ldots, K\}$
$\hookrightarrow$ Dimensionality Reduction	convert each $\mathbf{x}^{(i)} \in \mathbb{R}^M$ to $\mathbf{u}^{(i)} \in \mathbb{R}^K$ with $K << M$
Semi-supervised	$\mathcal{D} = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^{N_1} \cup \{\mathbf{x}^{(j)}\}_{j=1}^{N_2}$
Online	$\mathcal{D} = \{(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), (\mathbf{x}^{(3)}, y^{(3)}), \ldots\}$
Active Learning	$\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^N$ and can query $y^{(i)} = c^*(\cdot)$ at a cost
Imitation Learning	$\mathcal{D} = \{(s^{(1)}, a^{(1)}), (s^{(2)}, a^{(2)}), \ldots\}$
Reinforcement Learning	$\mathcal{D} = \{(s^{(1)}, a^{(1)}, r^{(1)}), (s^{(2)}, a^{(2)}, r^{(2)}), \ldots\}$

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

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)

Application Areas

Key challenges?

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

Theoretical Foundations:

What principles guide learning?

- ❑ probabilistic
- ❑ information theoretic
- ❑ evolutionary search
- ❑ ML as optimization

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*

Outline for Today

We'll talk about two distinct topics:

1. **Ensemble Methods**: combine or learn multiple classifiers into one
   (i.e. a family of algorithms)
2. **Recommender Systems**: produce recommendations of what a user will like
   (i.e. the solution to a particular type of task)

We'll use a prominent example of a recommender systems (the Netflix Prize) to motivate both topics…

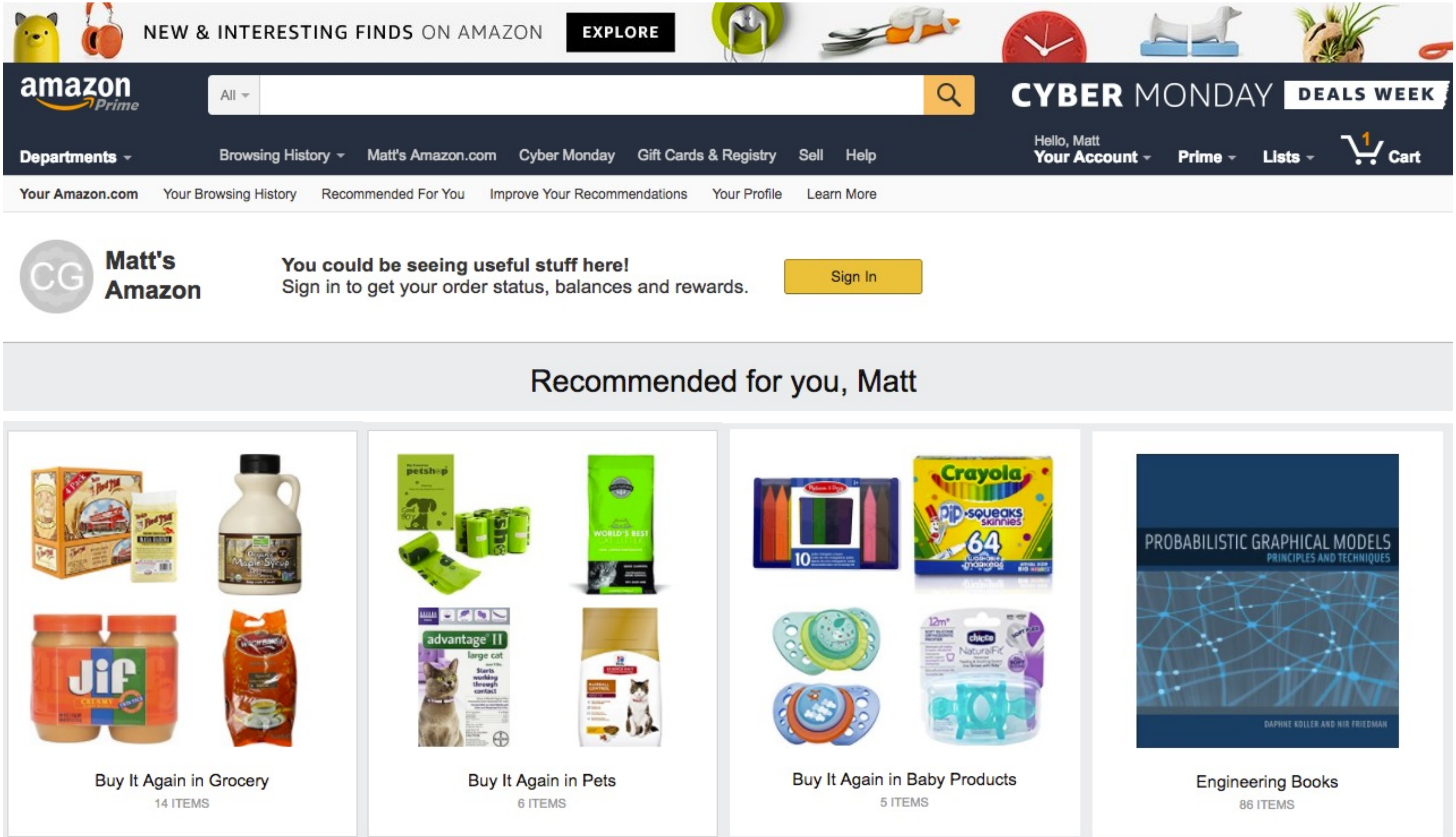
RECOMMENDER SYSTEMS

Recommender Systems

A Common Challenge:

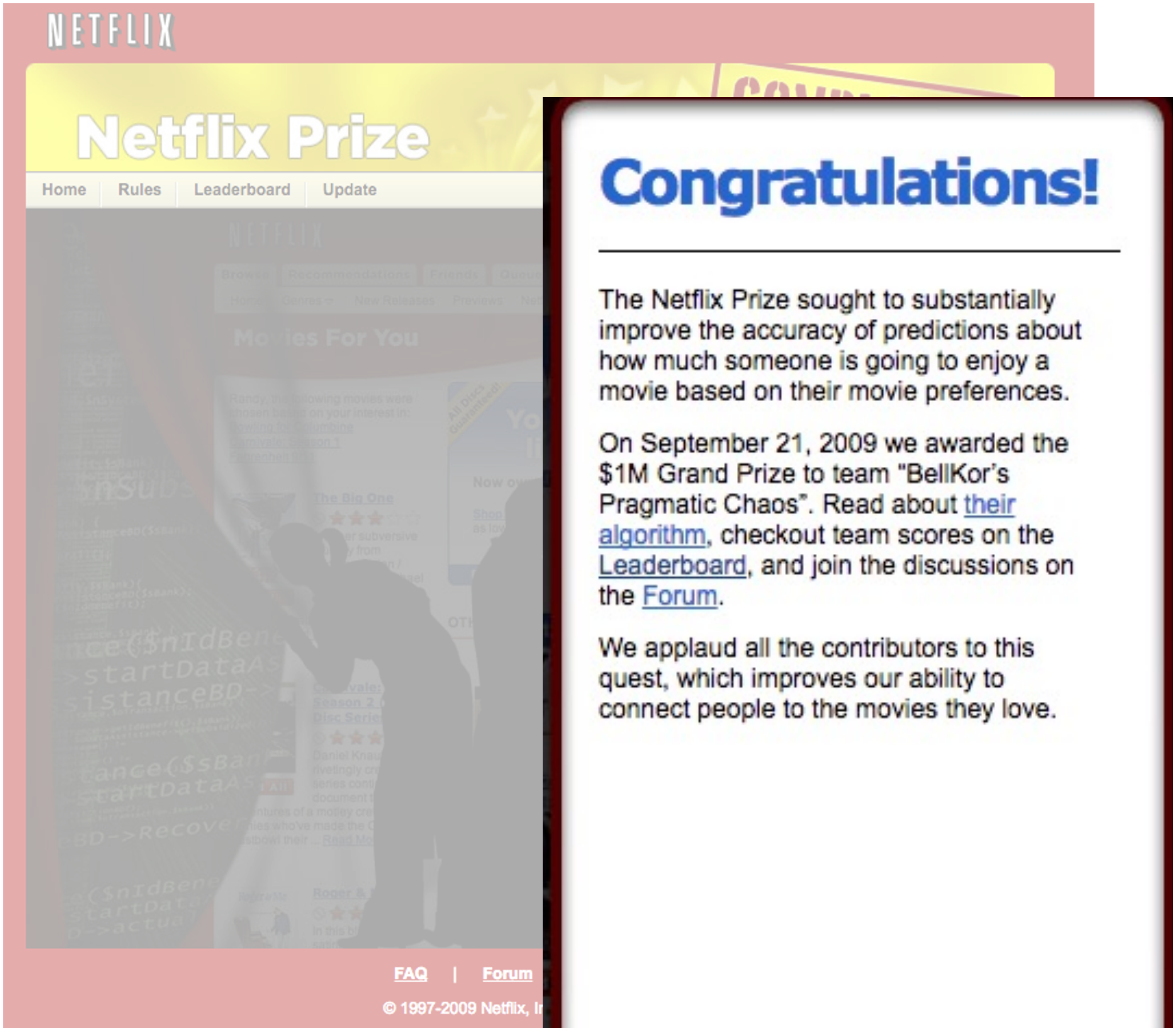
- Assume you're a company selling **items** of some sort: movies, songs, products, etc.
- Company collects millions of **ratings** from **users** of their **items**
- To maximize profit / user happiness, you want to **recommend** items that users are likely to want

Recommender Systems



Recommender Systems

Recommender Systems



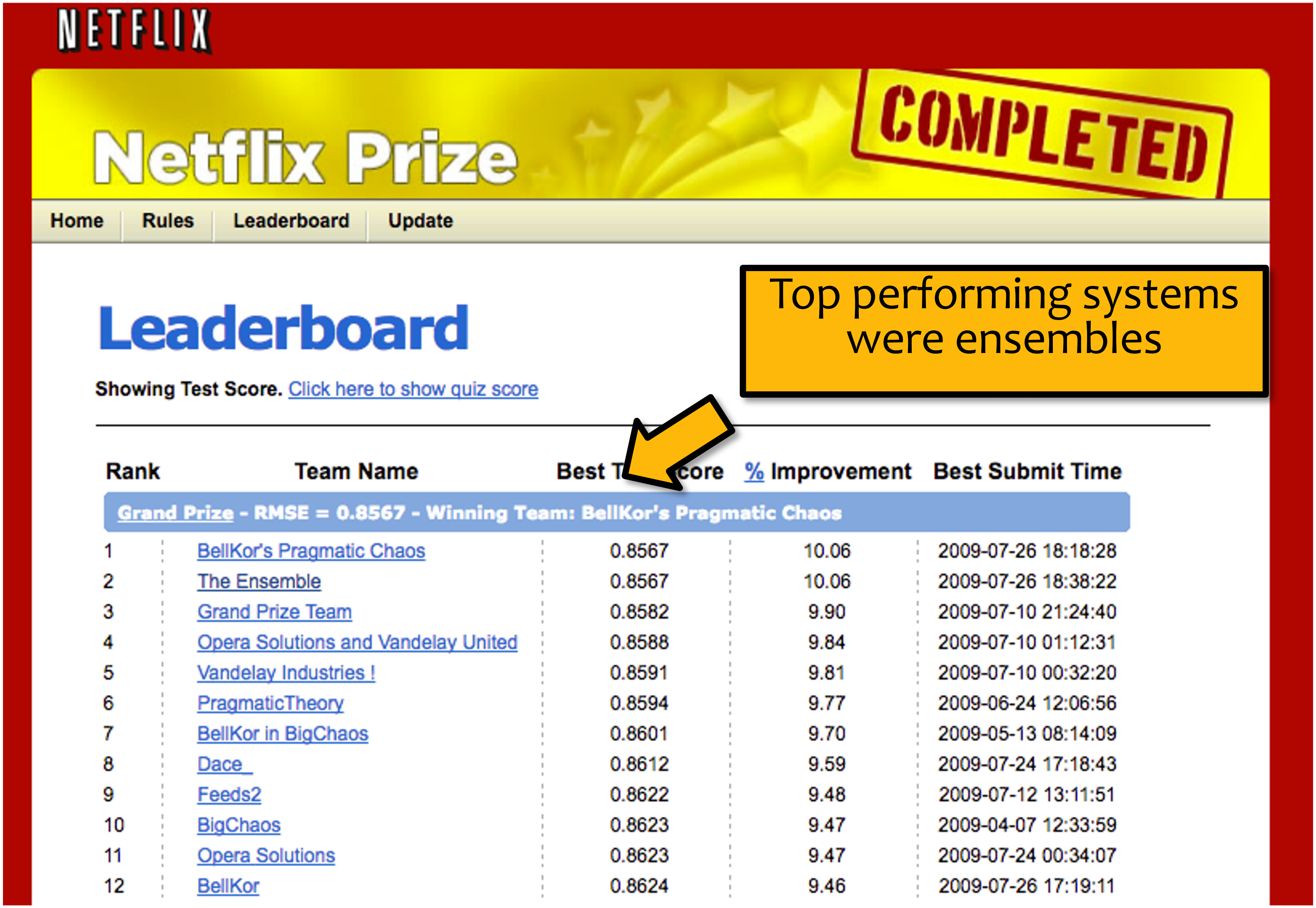
Recommender Systems

Problem Setup

- 500,000 users
- 20,000 movies
- 100 million ratings
- Goal: To obtain lower root mean squared error (RMSE) than Netflix's existing system on 3 million held out ratings

ENSEMBLE METHODS

Recommender Systems



NETFLIX

Netflix Prize

COMPLETED

Home | Rules | Leaderboard | Update

Leaderboard

Showing Test Score. Click here to show quiz score

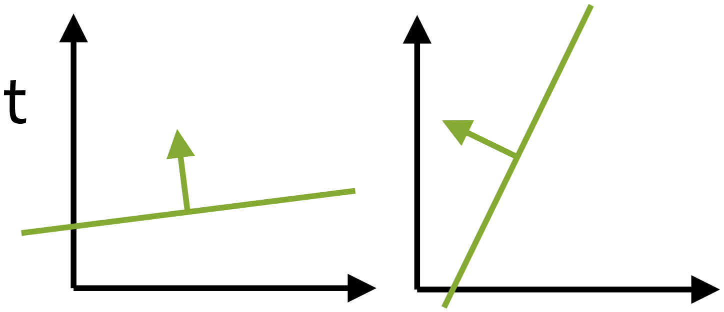
Rank	Team Name	Best T core	% Improvement	Best Submit Time
Grand Prize - RMSE = 0.8567 - Winning Team: BellKor's Pragmatic Chaos				
1	BellKor's Pragmatic Chaos	0.8567	10.06	2009-07-26 18:18:28
2	The Ensemble	0.8567	10.06	2009-07-26 18:38:22
3	Grand Prize Team	0.8582	9.90	2009-07-10 21:24:40
4	Opera Solutions and Vandelay United	0.8588	9.84	2009-07-10 01:12:31
5	Vandelay Industries !	0.8591	9.81	2009-07-10 00:32:20
6	PragmaticTheory	0.8594	9.77	2009-06-24 12:06:56
7	BellKor in BigChaos	0.8601	9.70	2009-05-13 08:14:09
8	Dace_	0.8612	9.59	2009-07-24 17:18:43
9	Feeds2	0.8622	9.48	2009-07-12 13:11:51
10	BigChaos	0.8623	9.47	2009-04-07 12:33:59
11	Opera Solutions	0.8623	9.47	2009-07-24 00:34:07
12	BellKor	0.8624	9.46	2009-07-26 17:19:11

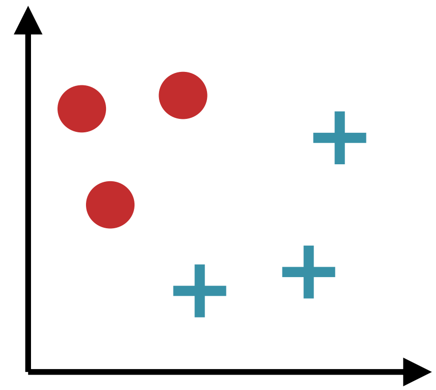
Top performing systems were ensembles

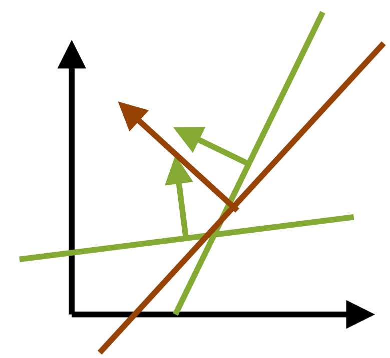
Weighted Majority Algorithm

(Littlestone & Warmuth, 1994)

- **Given**: pool *A* of binary classifiers (that you know nothing about)
- **Data:** stream of examples (i.e. online learning setting)
- **Goal:** design a new learner that uses the predictions of the pool to make new predictions
- **Algorithm**:
  - Initially weight all classifiers equally
  - Receive a training example and predict the (weighted) majority vote of the classifiers in the pool
  - Down-weight classifiers that contribute to a mistake by a factor of β







Weighted Majority Algorithm

(Littlestone & Warmuth, 1994)

Suppose we have a pool of T binary classifiers $\mathcal{A} = \{h_1, \ldots, h_T\}$ where $h_t : \mathbb{R}^M \to \{+1, -1\}$. Let α_t be the weight for classifier h_t.

Algorithm 1 Weighted Majority Algorithm

procedure WEIGHTEDMAJORITY($\mathcal{A}$, β)
  Initialize classifier weights $\alpha_t = 1, \ \forall t \in \{1, \ldots, T\}$
  for each training example $(\mathbf{x}, y)$ **do**
    Predict majority vote class (splitting ties randomly)

$$\hat{h}(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

    if a mistake is made $\hat{h}(x) \neq y$ **then**
      for each classifier $t \in \{1, \ldots, T\}$ **do**
        If $h_t(x) \neq y$, then $\alpha_t \leftarrow \beta\alpha_t$

Weighted Majority Algorithm

Theorems (Littlestone & Warmuth, 1994)

For the general case where *WM* is applied to a pool $\mathcal{A}$ of algorithms we show the following upper bounds on the number of mistakes made in a given sequence of trials:

1. $O(\log|\mathcal{A}|+m)$, if one algorithm of $\mathcal{A}$ makes at most m mistakes.

2. $O(\log\frac{|\mathcal{A}|}{k}+m)$, if each of a subpool of k algorithms of $\mathcal{A}$ makes at most m mistakes.

3. $O(\log\frac{|\mathcal{A}|}{k}+\frac{m}{k})$, if the total number of mistakes of a subpool of k algorithms of $\mathcal{A}$ is at most m.

These are "mistake bounds" of the variety we saw for the Perceptron algorithm

ADABOOST

Comparison

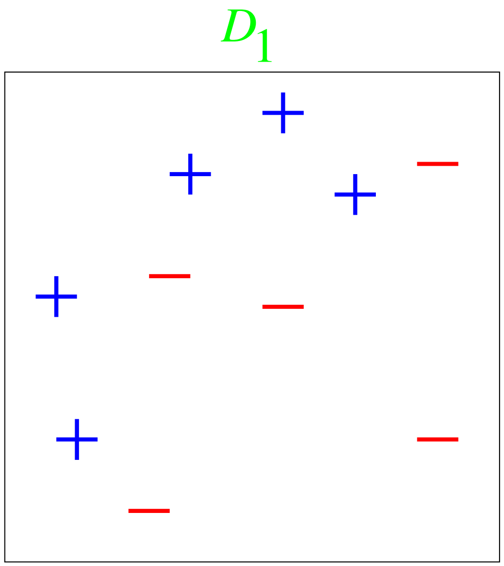
Weighted Majority Algorithm

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

AdaBoost

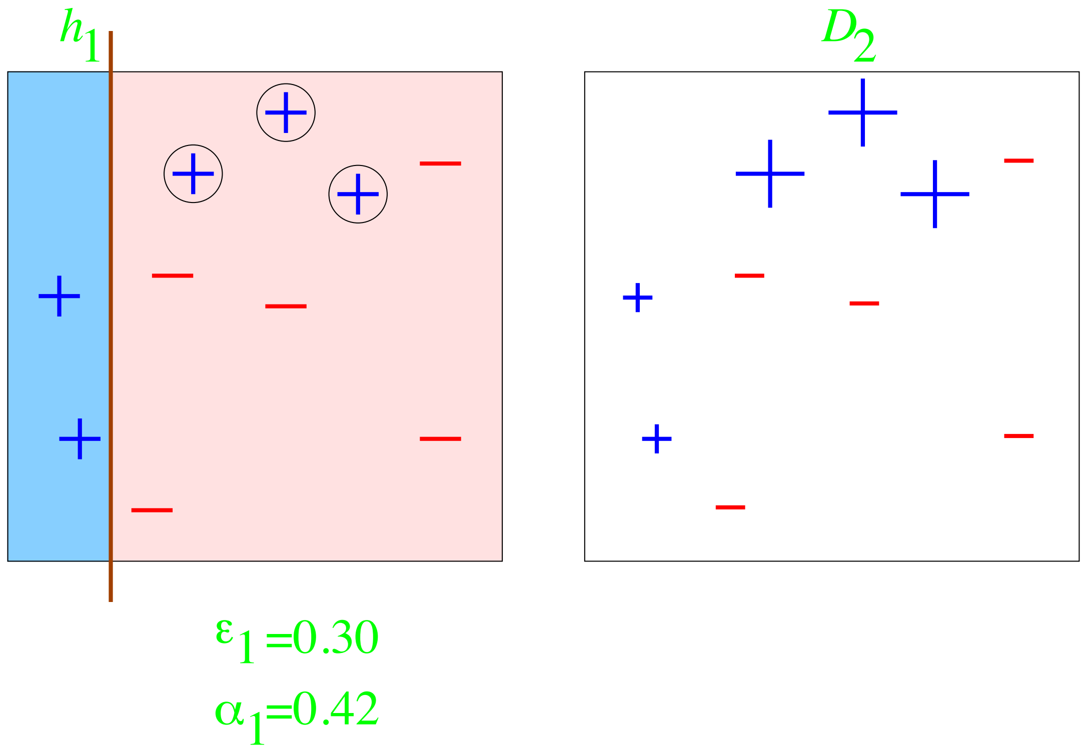
- an example of a boosting method
- simultaneously learns:
  - the classifiers themselves
  - (majority vote) weight for each classifiers

AdaBoost: Toy Example



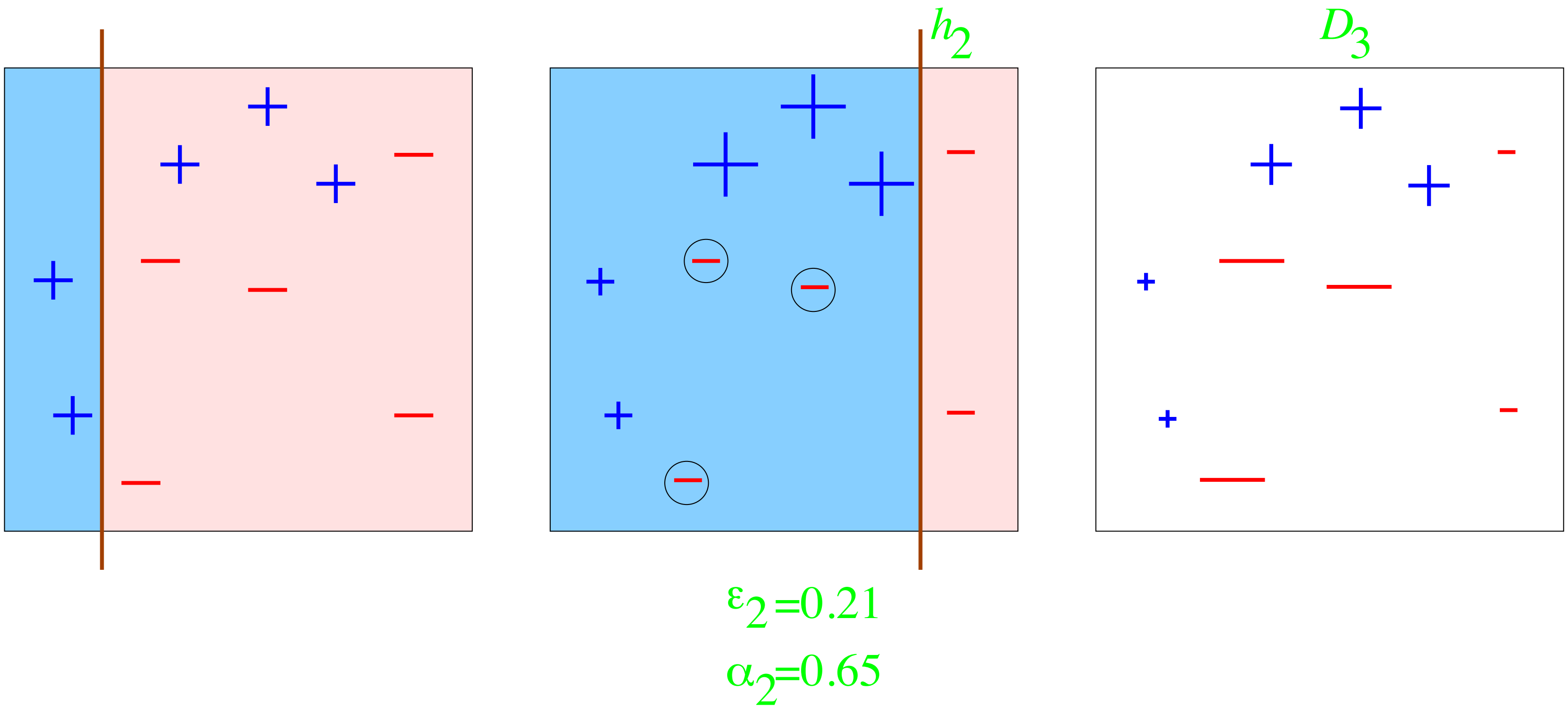
weak classifiers = vertical or horizontal half-planes

AdaBoost: Toy Example



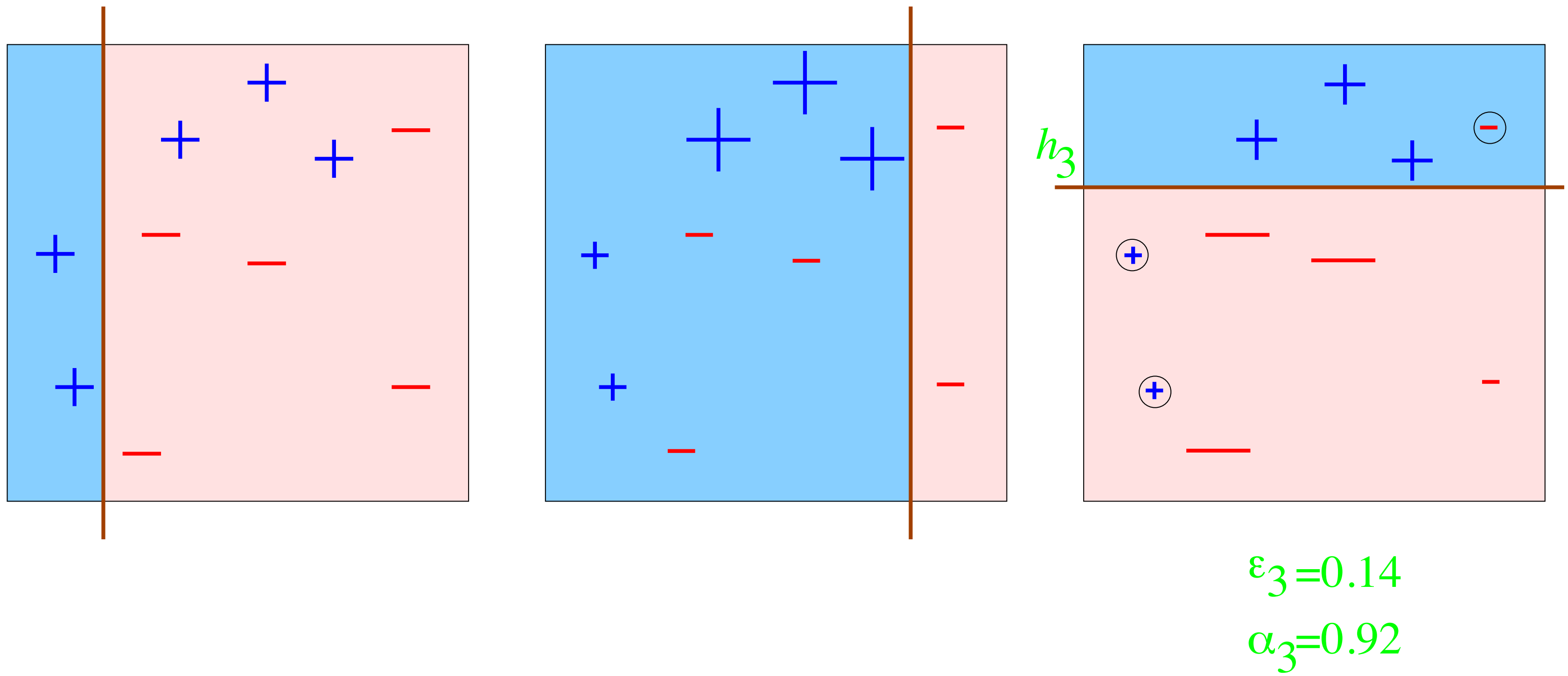
Slide from Schapire NIPS Tutorial

AdaBoost: Toy Example



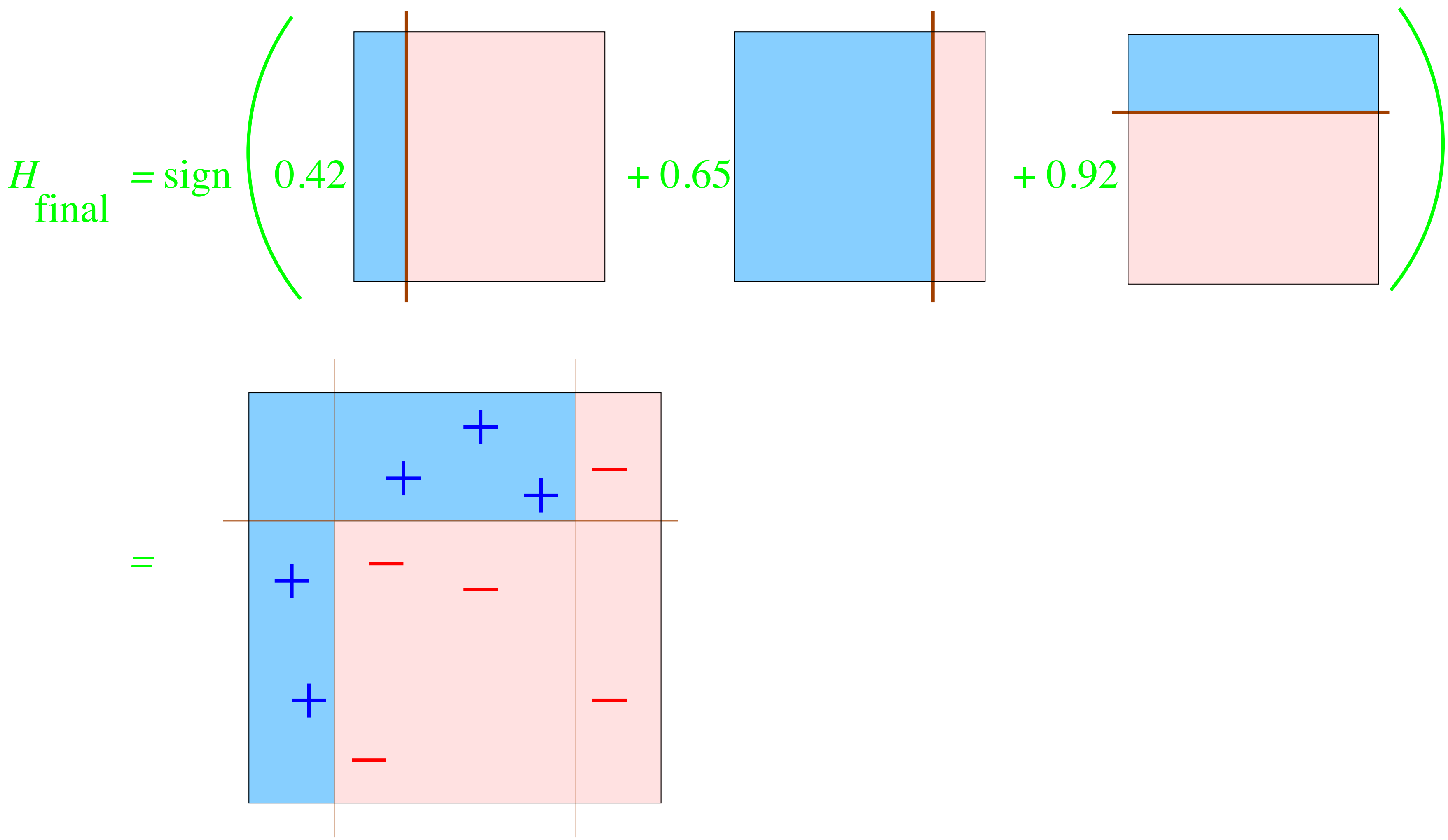
Slide from Schapire NIPS Tutorial

AdaBoost: Toy Example



Slide from Schapire NIPS Tutorial

AdaBoost: Toy Example



AdaBoost

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in X, y_i \in Y = \{-1, +1\}$
Initialize $D_1(i) = 1/m$.
For $t = 1, \ldots, T$:

- Train weak learner using distribution D_t.
- Get weak hypothesis $h_t : X \to \{-1, +1\}$ with error

$$\epsilon_t = \Pr_{i \sim D_t}\left[h_t(x_i) \neq y_i\right].$$

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

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

where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

Output the final hypothesis:

$$H(x) = \operatorname{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right).$$

Algorithm from (Freund & Schapire, 1999)

AdaBoost

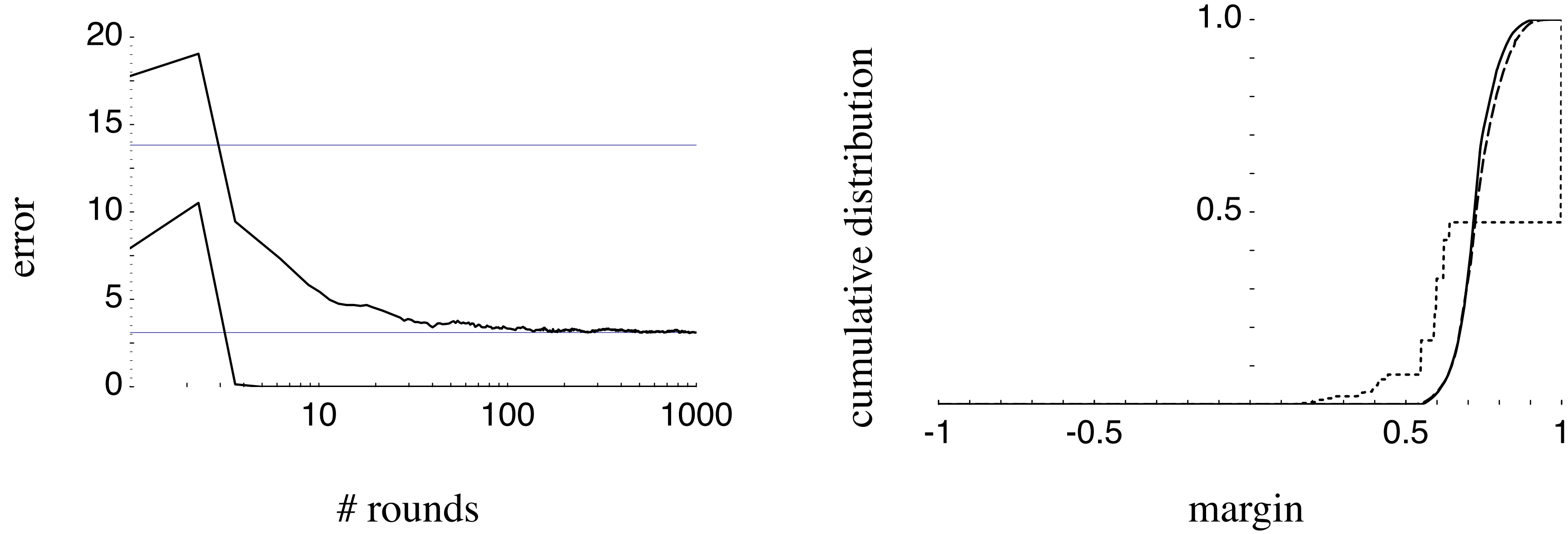


Figure 2: Error curves and the margin distribution graph for boosting C4.5 on the letter dataset as reported by Schapire et al. [41]. *Left*: the training and test error curves (lower and upper curves, respectively) of the combined classifier as a function of the number of rounds of boosting. The horizontal 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. Implement the Weighted Majority Algorithm
2. Implement AdaBoost
3. Distinguish what is learned in the Weighted Majority Algorithm vs. Adaboost
4. Contrast the theoretical result for the Weighted Majority Algorithm to that of Perceptron
5. Explain a surprisingly common empirical result regarding Adaboost train/test curves