10-301/601: Introduction to Machine Learning Lecture 23: Clustering

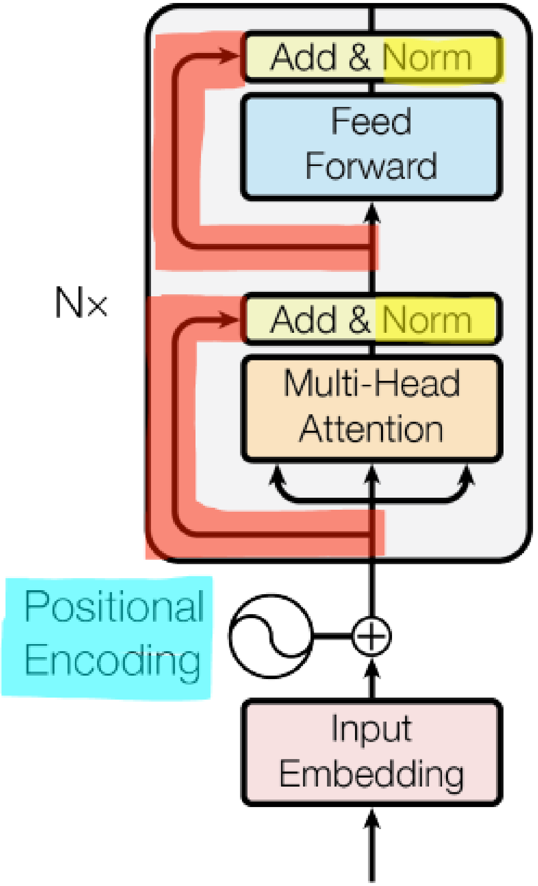
Henry Chai

6/4/25

Front Matter

- Announcements
  - HW5 released on 6/3, due 6/6 at 11:59 PM
  - Schedule change: two recitations this week
    - **Recitation on 6/4 (today!) will be a PyTorch tutorial**
    - Recitation on 6/5 will be Quiz 3 preparation

Recall: Transformers



- In addition to multi-head attention, transformer architectures use
    1. Positional encodings
    2. Layer normalization
    3. Residual connections
    4. A fully-connected feed-forward network

Source: https://arxiv.org/pdf/1706.03762.pdf

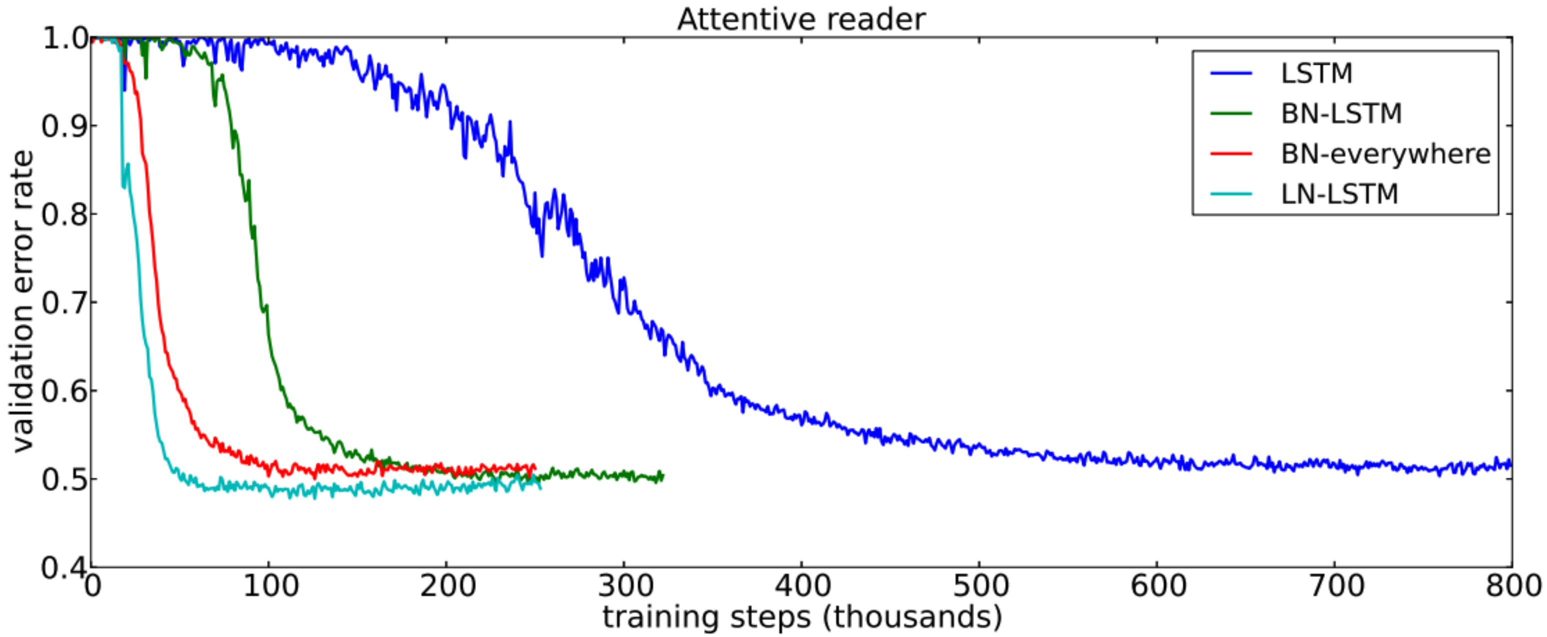
Layer Normalization

- Issue: for certain activation functions, the weights in later layers are **highly sensitive** to changes in the earlier layers
  - Small changes to weights in early layers are amplified so weights in deeper layers have to deal with massive dynamic ranges → slow optimization convergence
- Idea: normalize the output of a layer to always have the same (learnable) mean, β, and variance, γ^2

$$H' = \gamma\left(\frac{H - \mu}{\sigma}\right) + \beta$$

where μ is the mean and σ is the standard deviation of the values in the vector H

Layer Normalization



- Idea: normalize the output of a layer to always have the same (learnable) mean, β, and variance, γ^2

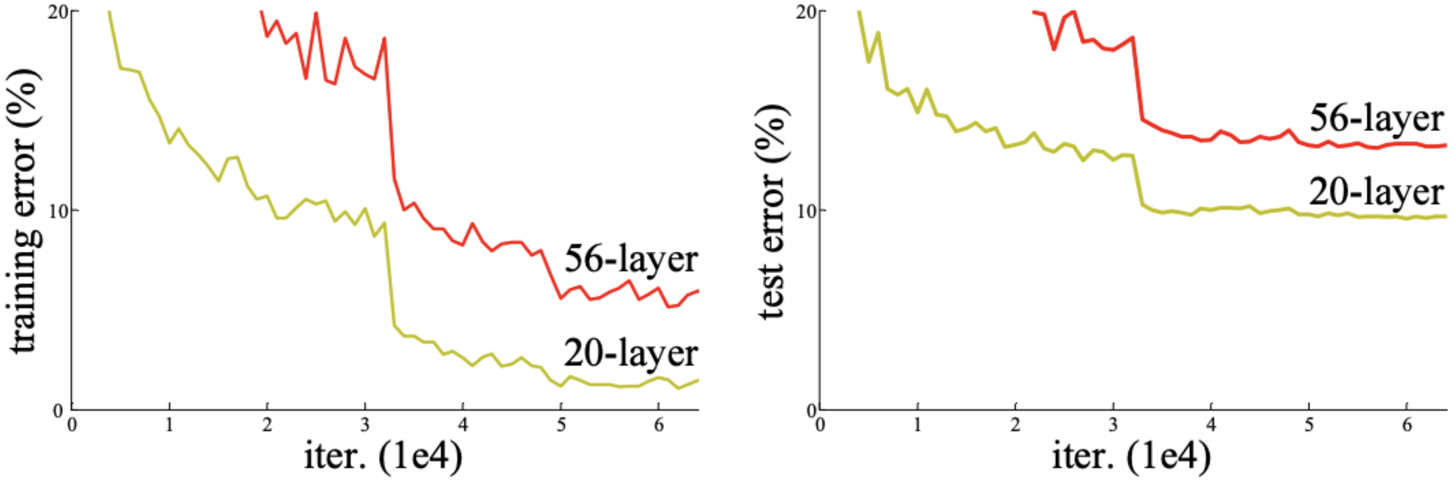
$$H' = \gamma\left(\frac{H - \mu}{\sigma}\right) + \beta$$

where μ is the mean and σ is the standard deviation of the values in the vector H

Source: https://arxiv.org/pdf/1607.06450.pdf

Residual Connections

- Observation: early deep neural networks suffered from the "degradation" problem where adding more layers actually made performance worse!



- Wait but this is ridiculous: if the later layers aren't helping, couldn't they just learn the identity transformation???
- Insight: neural network layers actually have a hard time learning the identity function

Source: https://arxiv.org/pdf/1512.03385.pdf

Residual Connections

- Observation: early deep neural networks suffered from the "degradation" problem where adding more layers actually made performance worse!
- Idea: add the input embedding back to the output of a layer

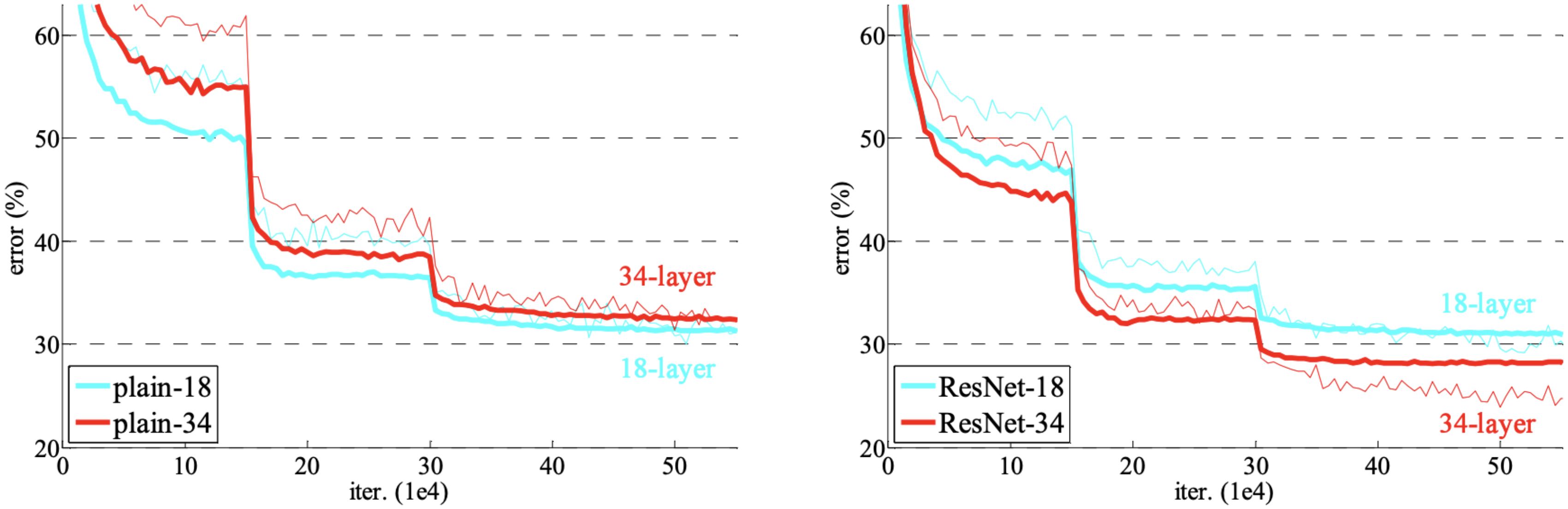
$$H' = H\left(x^{(i)}\right) + x^{(i)}$$

- Suppose the target function is f
  - Now instead of having to learn $f\left(x^{(i)}\right)$, the hidden layer just needs to learn the residual $r = f\left(x^{(i)}\right) - x^{(i)}$
  - If f is the identity function, then the hidden layer just needs to learn $r = 0$, which is easy for a neural network!

Residual Connections

- Observation: early deep neural networks suffered from the "degradation" problem where adding more layers actually made performance worse!
- Idea: add the input embedding back to the output of a layer

$$H' = H\left(x^{(i)}\right) + x^{(i)}$$



Source: https://arxiv.org/pdf/1512.03385.pdf

Learning Paradigms

- Supervised learning - $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(n)}, y^{(n)}\right)\right\}_{n=1}^{N}$
  - Regression - $y^{(n)} \in \mathbb{R}$
  - Classification - $y^{(n)} \in \{1, \ldots, C\}$
- Unsupervised learning - $\mathcal{D} = \left\{\boldsymbol{x}^{(n)}\right\}_{n=1}^{N}$
  - Clustering
  - Dimensionality reduction

Learning Paradigms

- Supervised learning - $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(n)}, y^{(n)}\right)\right\}_{n=1}^{N}$
  - Regression - $y^{(n)} \in \mathbb{R}$
  - Classification - $y^{(n)} \in \{1, \dots, C\}$
- Unsupervised learning - $\mathcal{D} = \left\{\boldsymbol{x}^{(n)}\right\}_{n=1}^{N}$
  - **Clustering**
  - Dimensionality reduction

Clustering

- Goal: split an unlabeled data set into groups or clusters of "similar" data points
- Use cases:
  - Organizing data
  - Discovering patterns or structure
  - Preprocessing for downstream machine learning tasks
- Applications:

Recall: Similarity for kNN

- Intuition: ~~predict the label of a data point to be the label of the "most similar" training point~~ two points are "similar" if the distance between them is small
- Euclidean distance: $d(\boldsymbol{x}, \boldsymbol{x}') = \|\boldsymbol{x} - \boldsymbol{x}'\|_2$

Partition-Based Clustering

- Given a desired number of clusters, K, return a partition of the data set into K groups or clusters, $\{C_1, \ldots, C_K\}$, that optimize some objective function

1. What objective function should we optimize?

2. How can we perform optimization in this setting?

Option A

Option B

Which do you prefer?

General Recipe for Machine Learning

- Define a model and model parameters
- Write down an objective function
- Optimize the objective w.r.t. the model parameters

Recipe for K-means

- Define a model and model parameters
  - Assume K clusters and use the Euclidean distance
  - Parameters: $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$ and $z^{(1)}, \ldots, z^{(N)}$
- Write down an objective function

$$\sum_{n=1}^{N} \left\| \boldsymbol{x}^{(n)} - \boldsymbol{\mu}_{z^{(n)}} \right\|_2$$

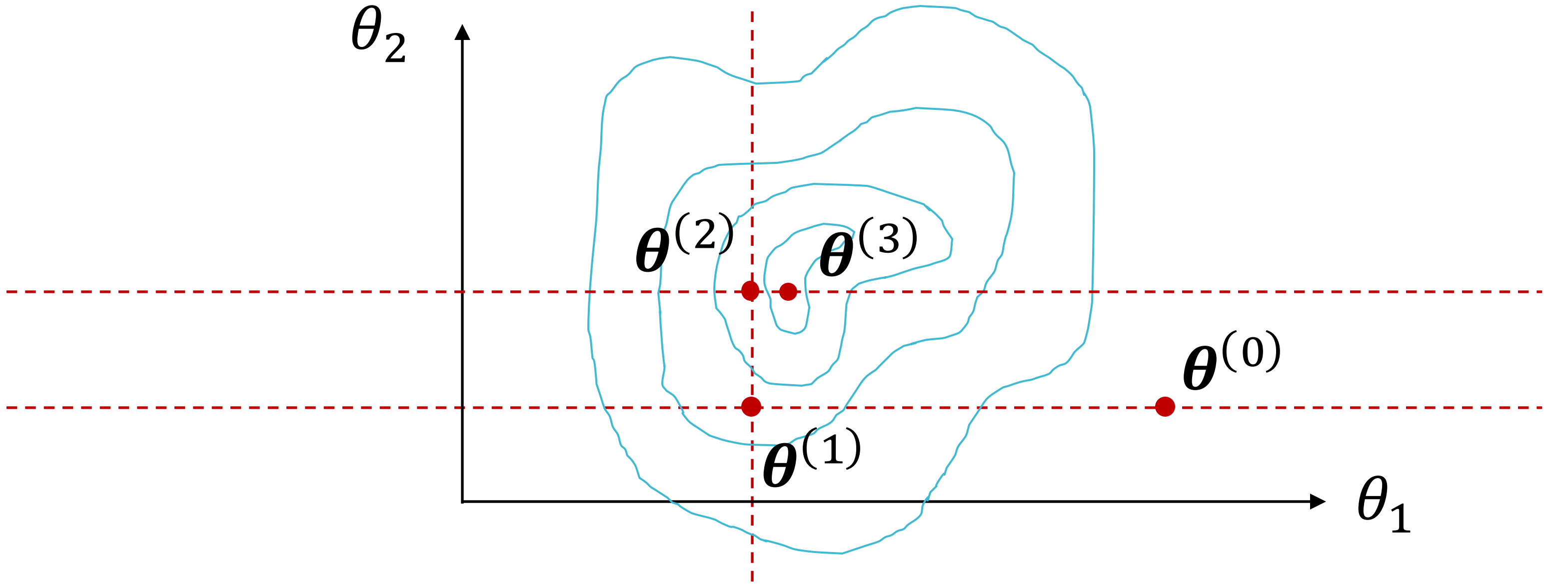
- Optimize the objective w.r.t. the model parameters
  - Use (block) coordinate descent

Coordinate Descent

- Goal: minimize some objective

$$\hat{\boldsymbol{\theta}} = \text{argmin } J(\boldsymbol{\theta})$$

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



Block Coordinate Descent

- Goal: minimize some objective

$$\hat{\boldsymbol{\alpha}}, \hat{\boldsymbol{\beta}} = \operatorname{argmin} \ J(\boldsymbol{\alpha}, \boldsymbol{\beta})$$

- Idea: iteratively pick one *block* of variables ($\boldsymbol{\alpha}$ or $\boldsymbol{\beta}$) and minimize the objective w.r.t. that block, keeping the other(s) fixed.
  - Ideally, blocks should be the largest possible set of variables *that can be efficiently optimized simultaneously*

Optimizing the K-means objective

$$\hat{\boldsymbol{\mu}}_1, \dots, \hat{\boldsymbol{\mu}}_K, z^{(1)}, \dots, z^{(\mathrm{N})} = \operatorname{argmin} \sum_{n=1}^{N} \left\| \boldsymbol{x}^{(n)} - \boldsymbol{\mu}_{z^{(n)}} \right\|_2$$

- If $\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_K$ are fixed

$$\hat{z}^{(n)} = \operatorname*{argmin}_{k \in \{1, \dots, K\}} \left\| \boldsymbol{x}^{(n)} - \boldsymbol{\mu}_k \right\|_2$$

- If $z^{(1)}, \dots, z^{(N)}$ are fixed

$$\hat{\boldsymbol{\mu}}_k = \operatorname*{argmin}_{\boldsymbol{\mu}} \sum_{n \,: z^{(n)} = k} \left\| \boldsymbol{x}^{(n)} - \boldsymbol{\mu} \right\|_2$$

$$= \frac{1}{N_k} \sum_{n \,: z^{(n)} = k} \boldsymbol{x}^{(n)}$$

K-means Algorithm

- Input: $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(n)}\right)\right\}_{n=1}^{N}, K$

1. Initialize cluster centers $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$
2. While NOT CONVERGED
   a. Assign each data point to the cluster with the nearest cluster center:
   $$z^{(n)} = \underset{k}{\operatorname{argmin}} \left\|\boldsymbol{x}^{(n)} - \boldsymbol{\mu}_k\right\|_2$$
   b. Recompute the cluster centers:
   $$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{n\,:\,z^{(n)}=k} \boldsymbol{x}^{(n)}$$
   where N_k is the number of data points in cluster k

- Output: cluster centers $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$ and cluster assignments $z^{(1)}, \ldots, z^{(N)}$

K-means: Example ($K = 3$)

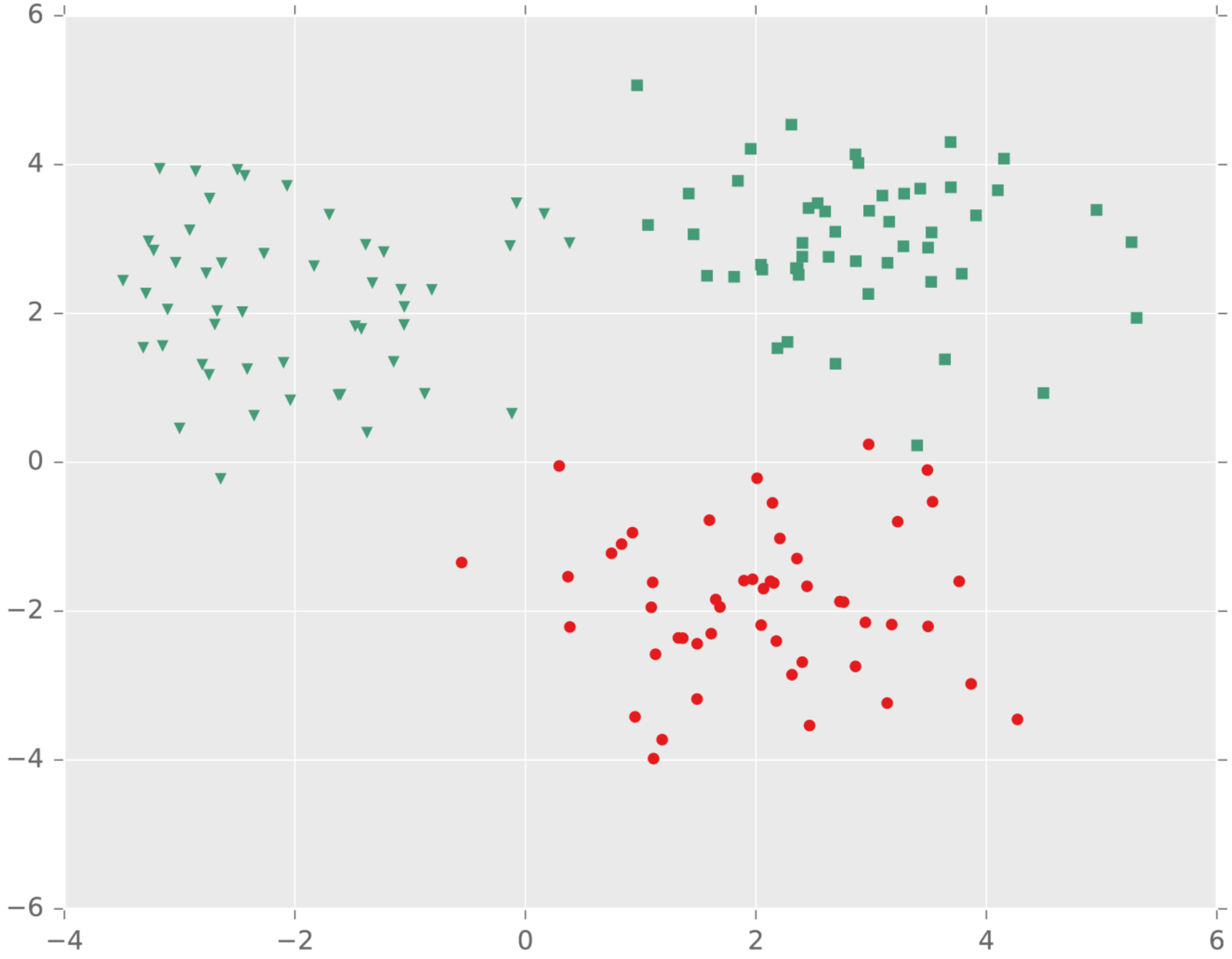


Figure courtesy of Matt Gormley

K-means: Example ($K = 3$)

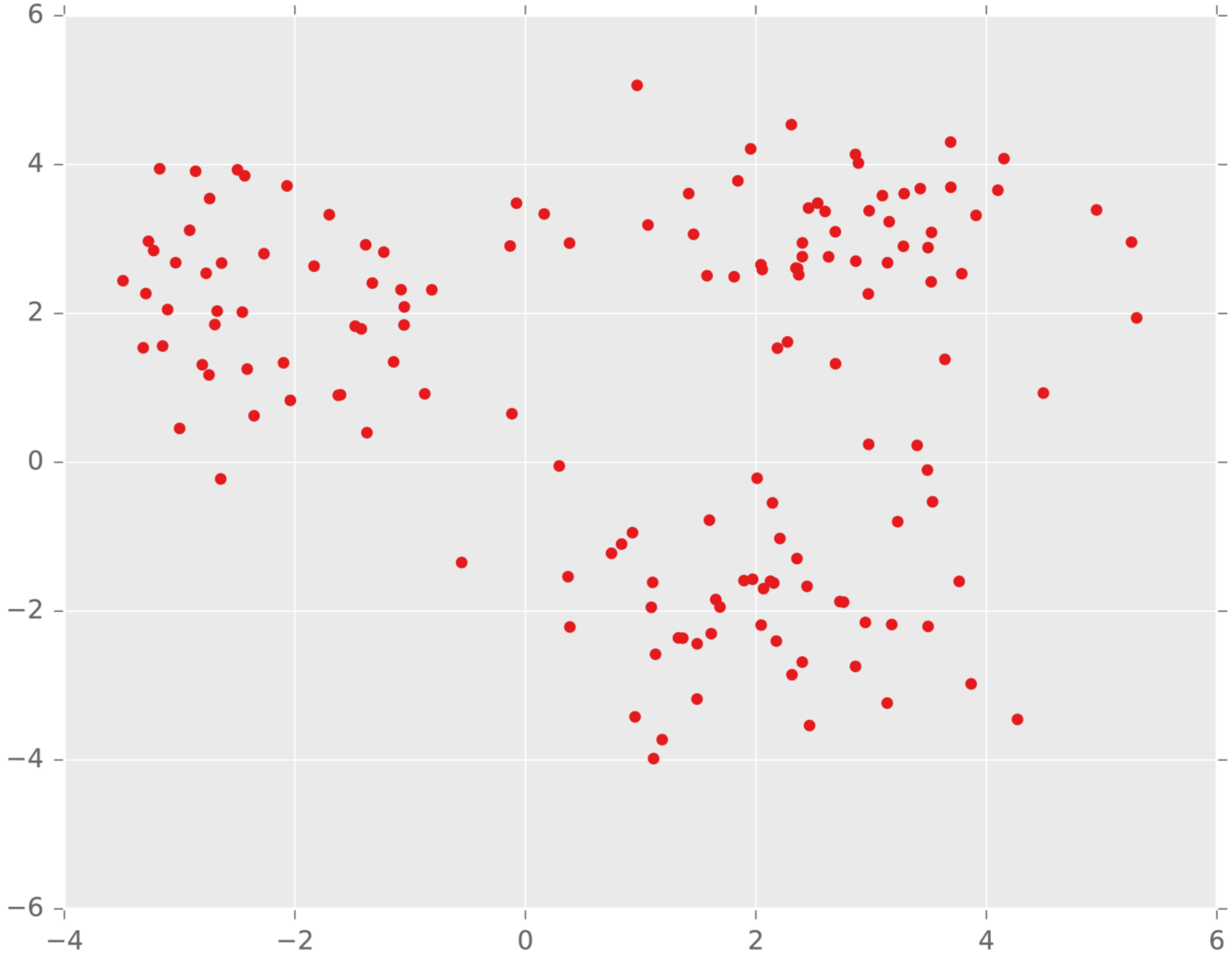


Figure courtesy of Matt Gormley

K-means: Example ($K = 3$)

Figure courtesy of Matt Gormley

K-means: Example ($K = 3$)

Figure courtesy of Matt Gormley

K-means: Example ($K = 3$)

Figure courtesy of Matt Gormley

K-means: Example ($K = 3$)

Figure courtesy of Matt Gormley

K-means: Example ($K = 3$)

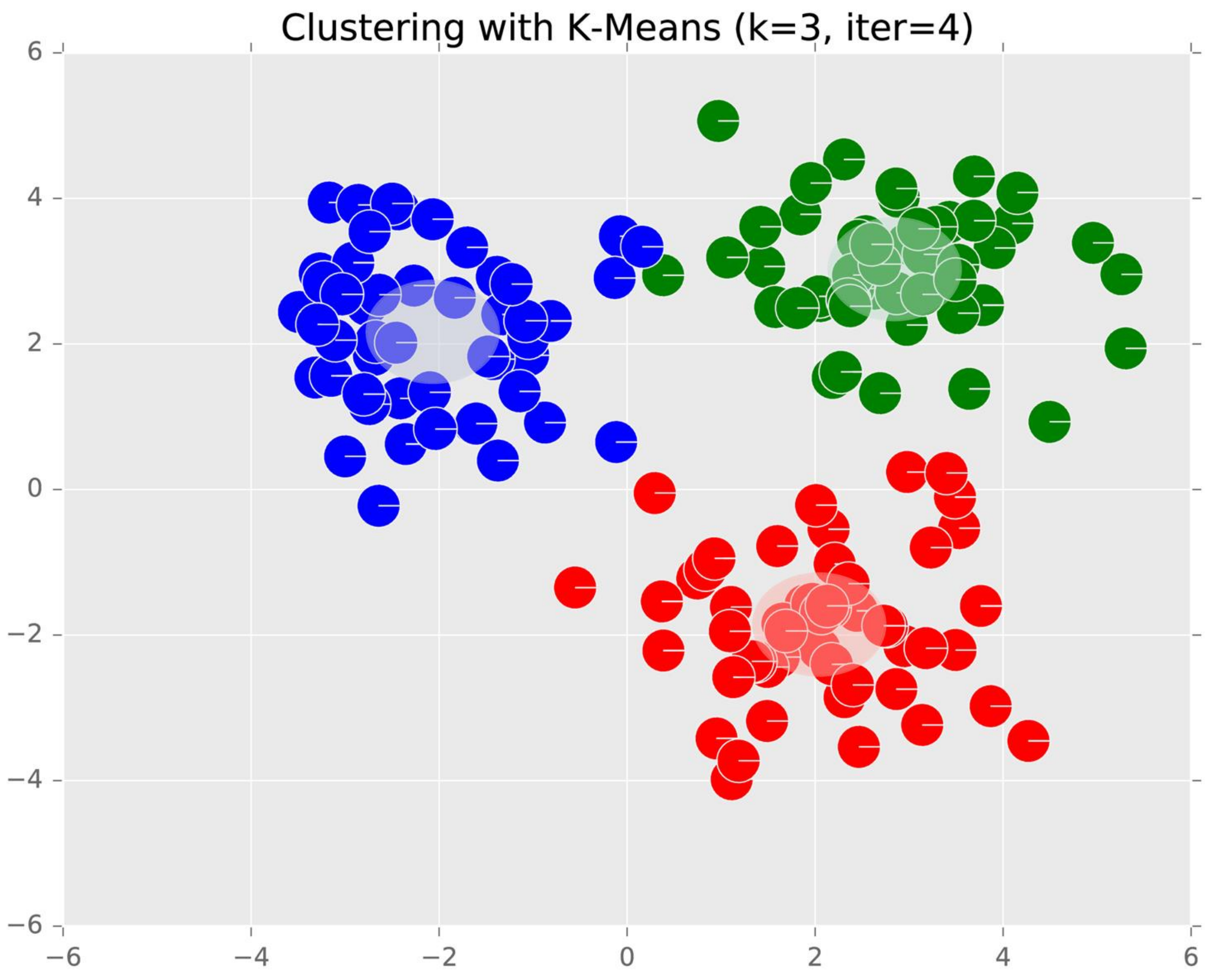


Figure courtesy of Matt Gormley

K-means: Example ($K = 3$)

Figure courtesy of Matt Gormley

K-means: Example ($K = 2$)

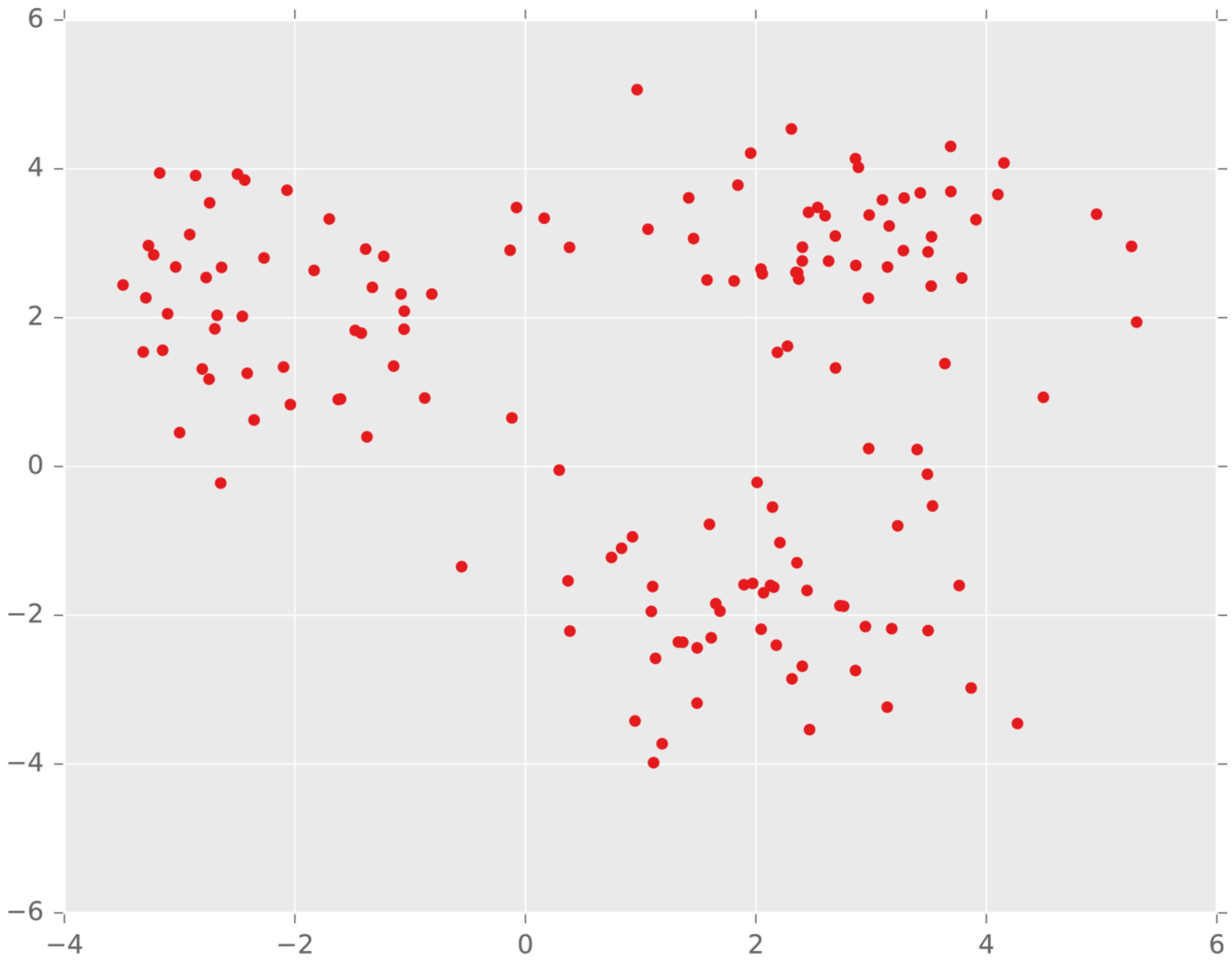


Figure courtesy of Matt Gormley

K-means: Example ($K = 2$)

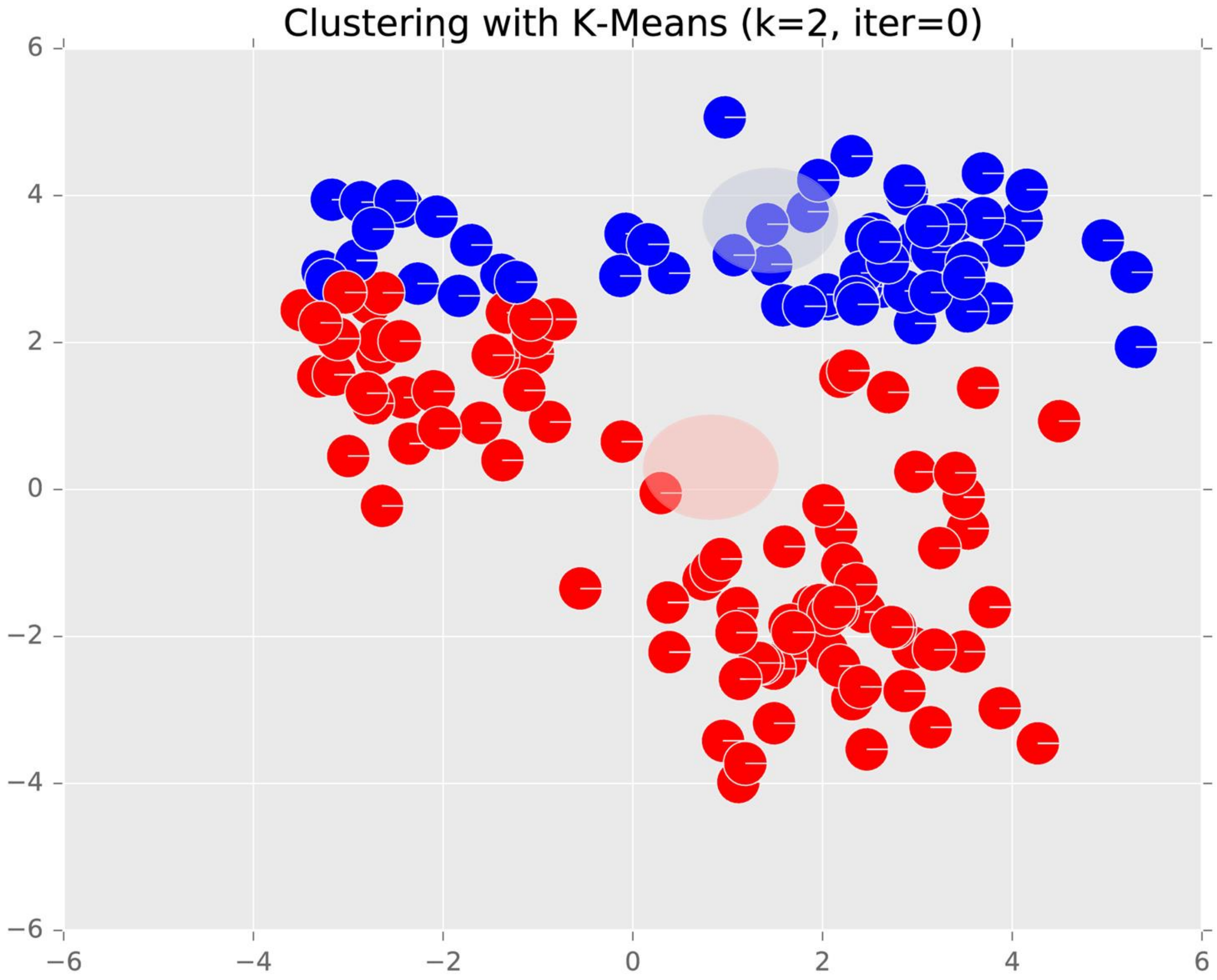


Figure courtesy of Matt Gormley

K-means: Example ($K = 2$)

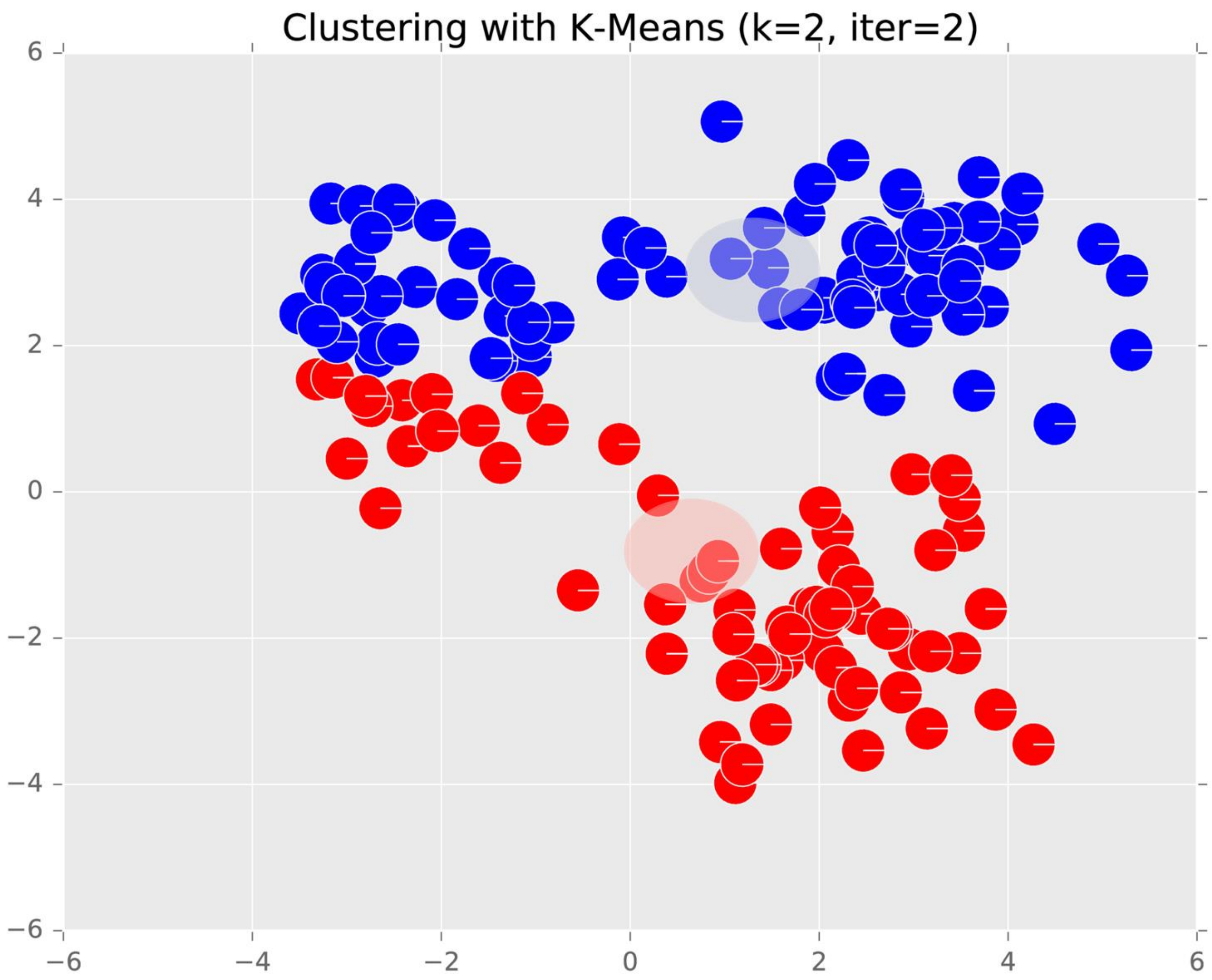


Figure courtesy of Matt Gormley

K-means: Example ($K = 2$)

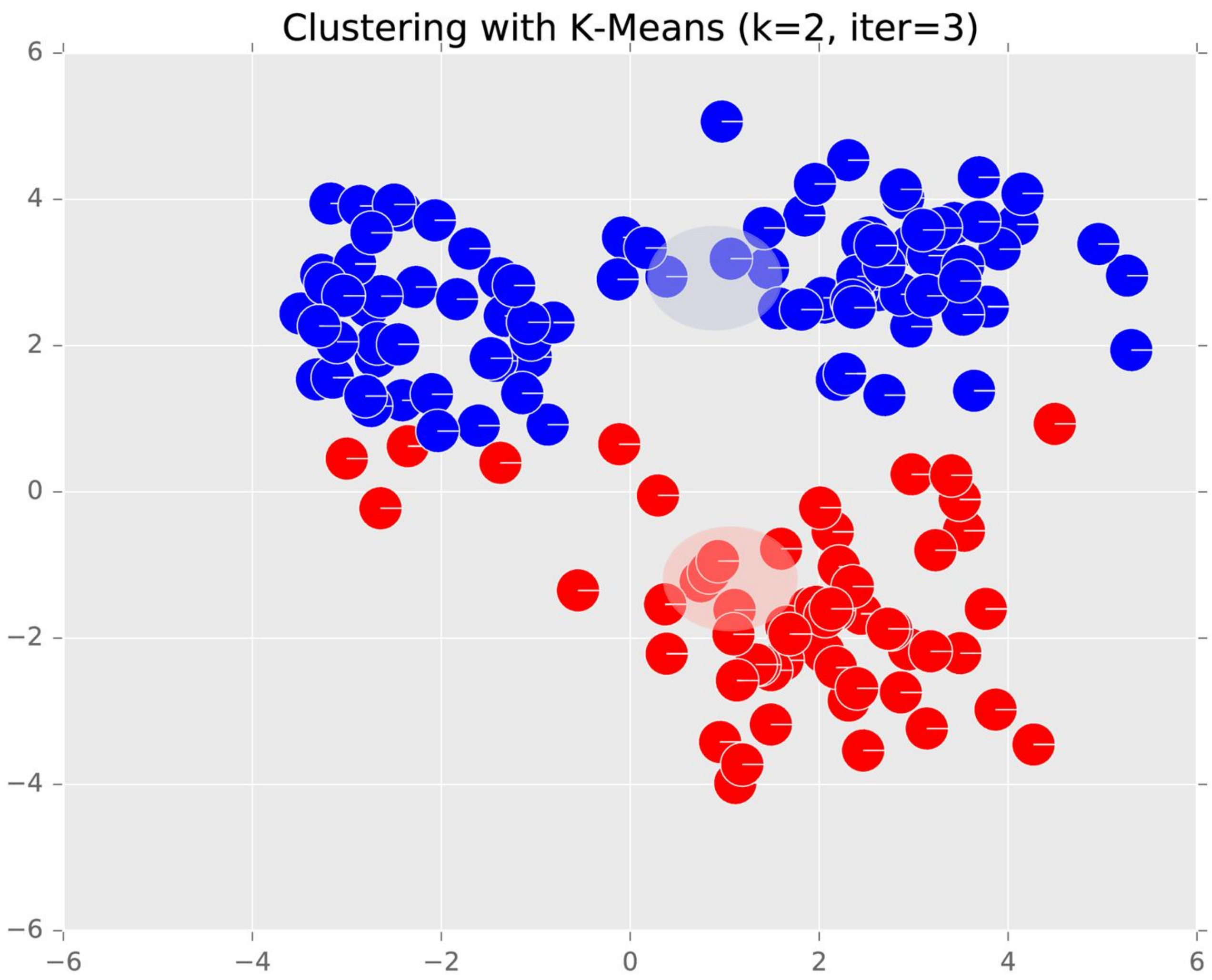


Figure courtesy of Matt Gormley

K-means: Example ($K = 2$)

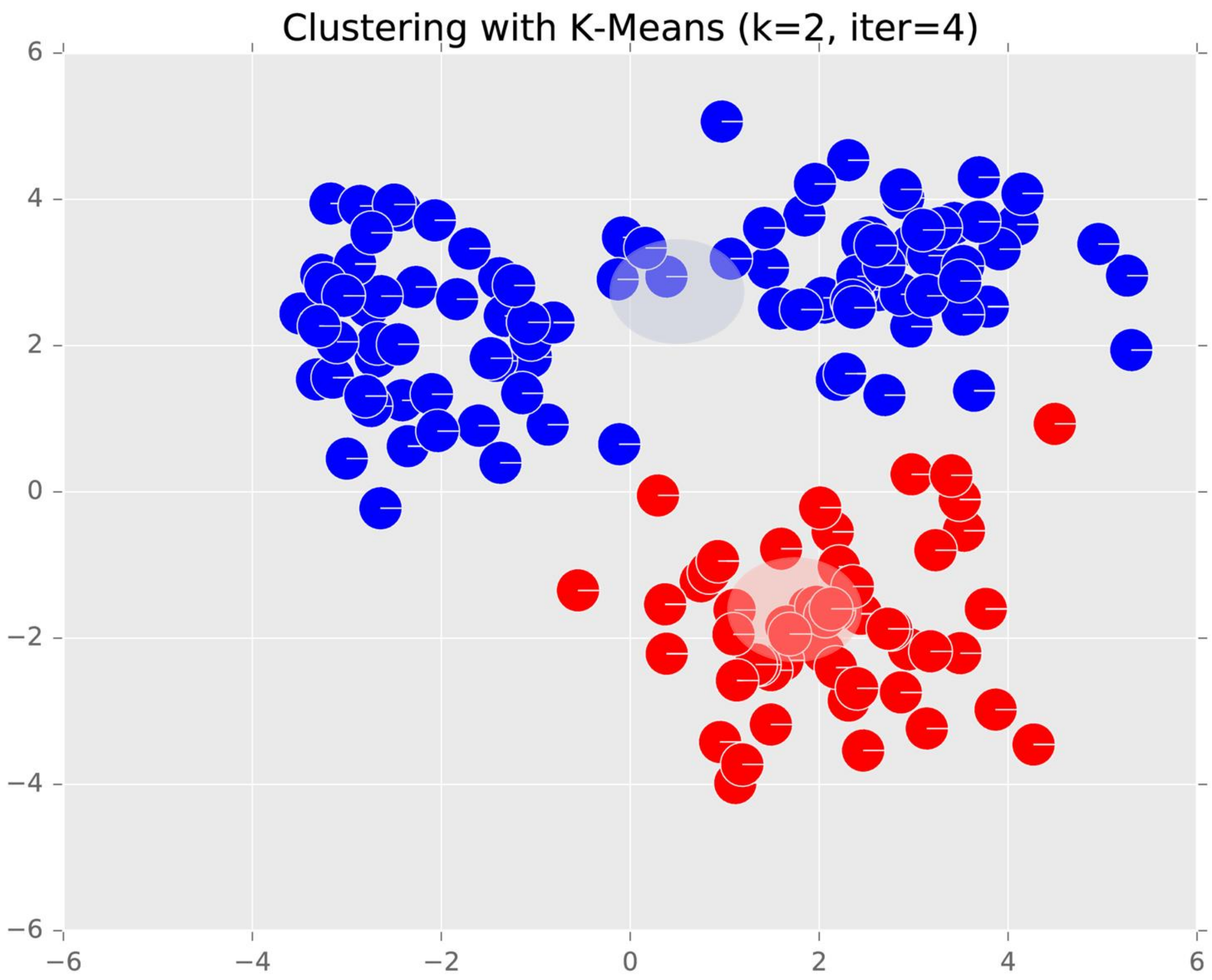


Figure courtesy of Matt Gormley

K-means: Example ($K = 2$)

Figure courtesy of Matt Gormley

K-means: Example ($K = 2$)

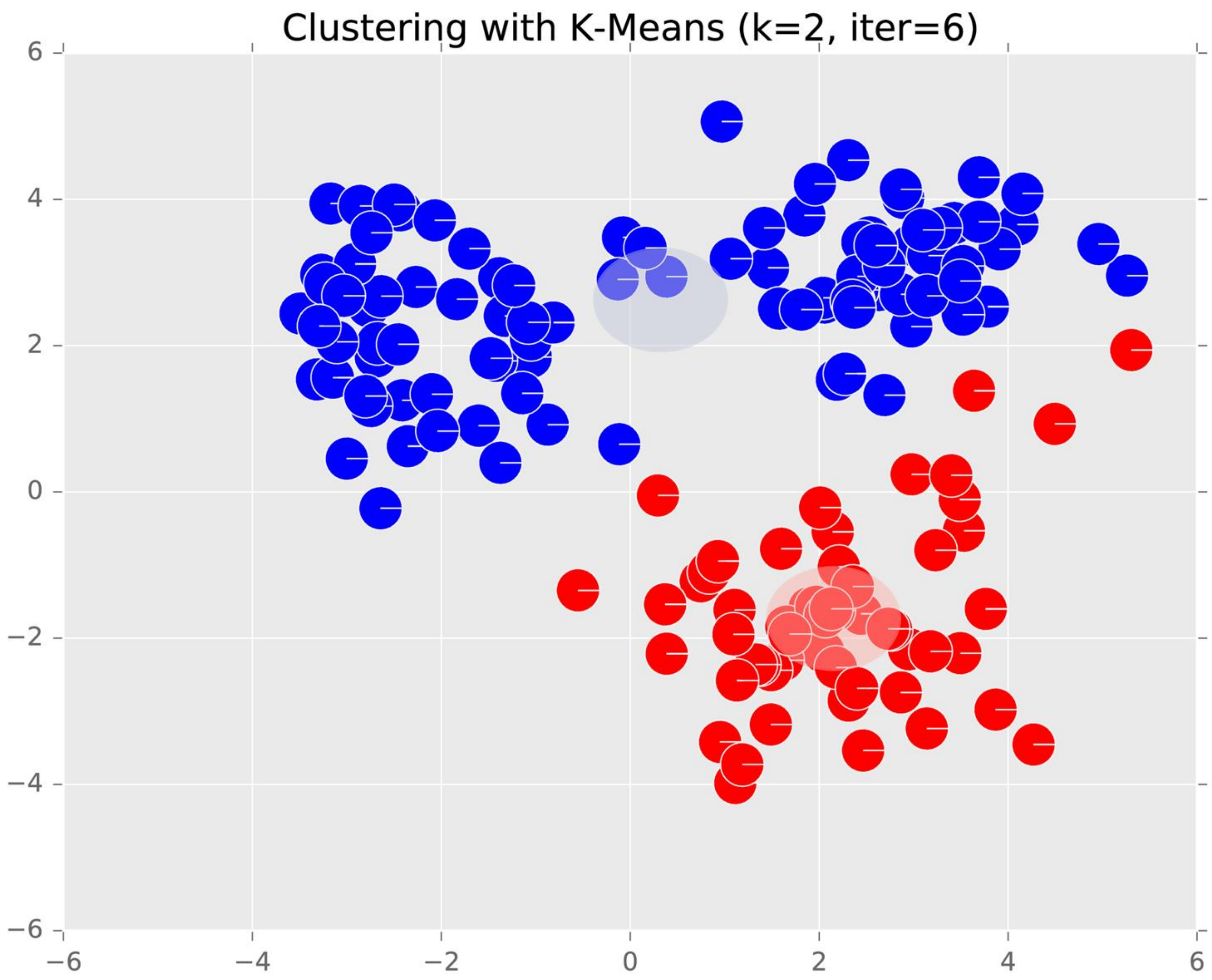


Figure courtesy of Matt Gormley

K-means: Example ($K = 2$)

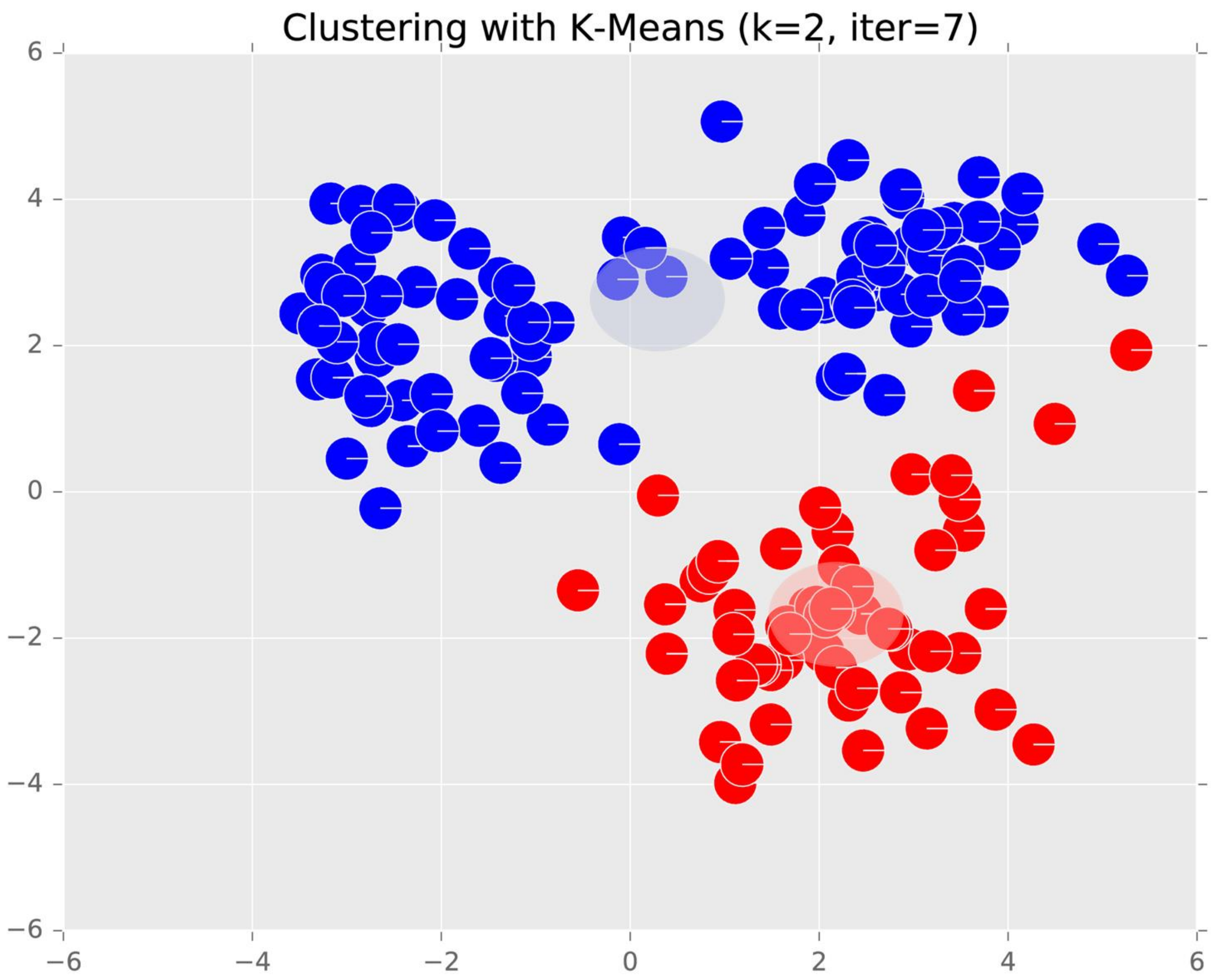
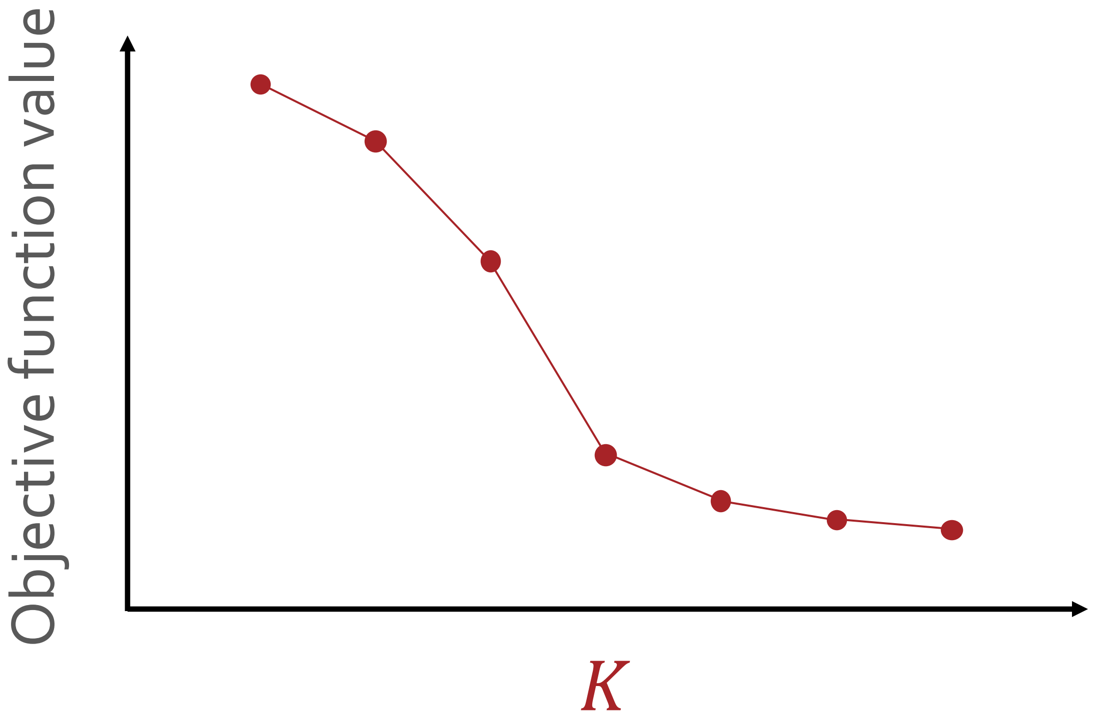


Figure courtesy of Matt Gormley

Setting K

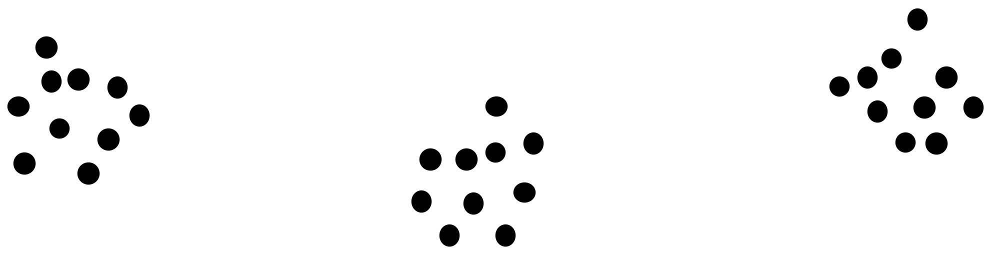
- Idea: choose the value of K that minimizes the objective function



- Better Idea: look for the characteristic "elbow" or largest decrease when going from $K-1$ to K

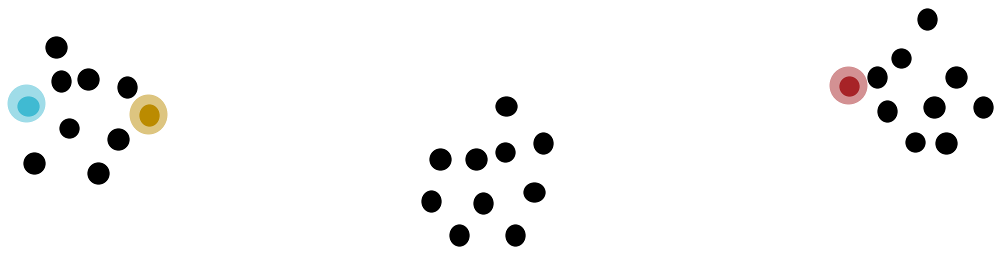
Initializing K-means

- Common choice: choose K data points at random to be the initial cluster centers (Lloyd's method)



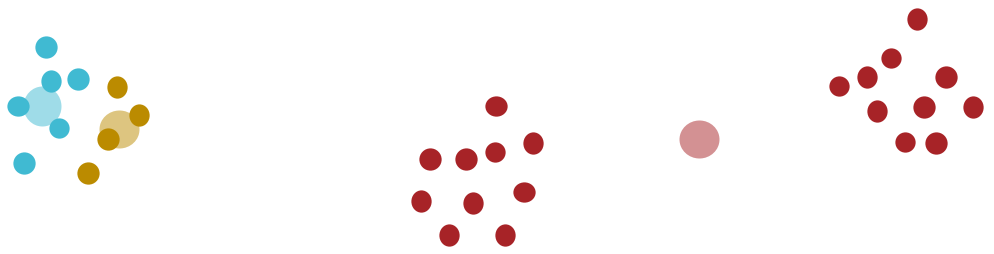
Initializing K-means

- Common choice: choose K data points at random to be the initial cluster centers (Lloyd's method)



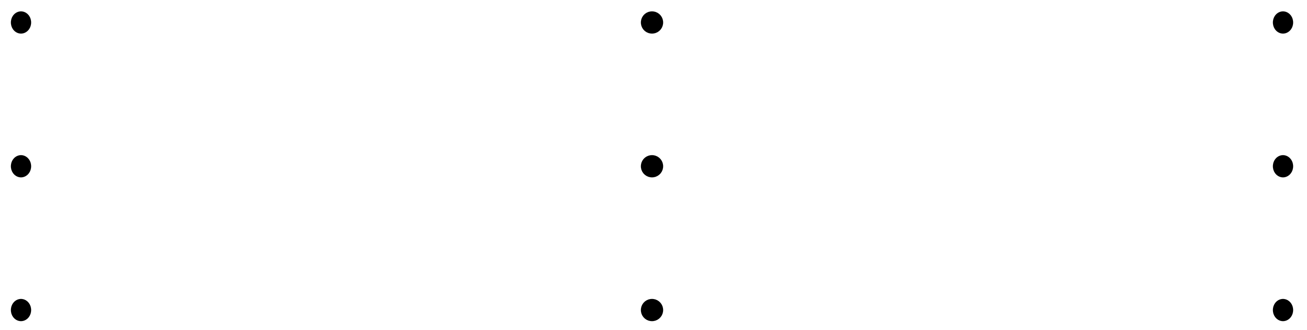
Initializing K-means

- Common choice: choose K data points at random to be the initial cluster centers (Lloyd's method)



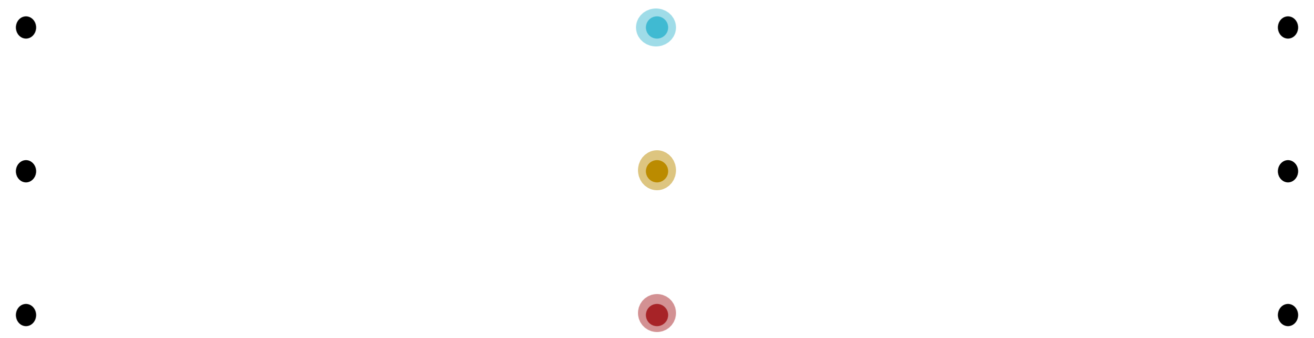
Initializing K-means

- Common choice: choose K data points at random to be the initial cluster centers (Lloyd's method)



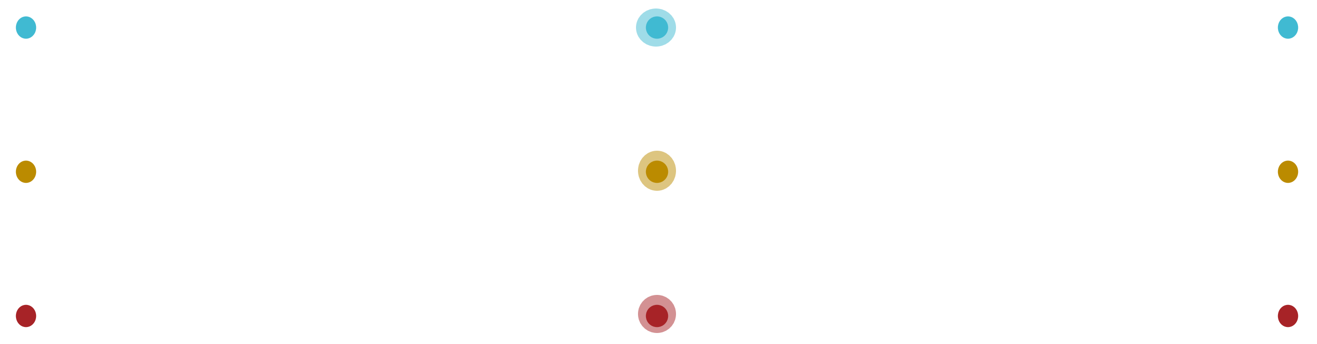
Initializing K-means

- Common choice: choose K data points at random to be the initial cluster centers (Lloyd's method)



Initializing K-means

- Common choice: choose K data points at random to be the initial cluster centers (Lloyd's method)



- Lloyd's method converges to a local minimum and that local minimum can be arbitrarily bad (relative to the optimal clusters)
- Intuition: want initial cluster centers to be far apart from one another

K-means++ (Arthur and Vassilvitskii, 2007)

1. Choose the first cluster center randomly from the data points.
2. For each other data point $\boldsymbol{x}$, compute $D(\boldsymbol{x})$, the distance between $\boldsymbol{x}$ and the closest cluster center.
3. Select the next cluster center proportional to $D(\boldsymbol{x})^2$.
4. Repeat 2 and 3 $K - 1$ times.

- K-means++ achieves a $O(\log K)$ approximation to the optimal clustering in expectation
- Both Lloyd's method and K-means++ can benefit from multiple random restarts.

Key Takeaways

- K-means objective function & model parameters
- Block-coordinate descent
- Setting K
- Initializing K means