DATA SCIENCE II:
Machine Learning
MTH 9899
Baruch College

Lecture 2: Introduction to Machine Learning

Adrian Sisser Dmytro Karabash

April 6, 2016

Outline

Outline

Supervised Learning

Least Squares
$$\min_{\beta \in \mathbb{R}^p} \left\{ \|y - X\beta\|_2^2 \right\} \tag{1}$$

Ridge
$$\min_{\beta \in \mathbb{R}^p} \left\{ \|y - X\beta\|_2^2 + \lambda_2 \|\beta\|_2^2 \right\} \tag{2}$$

Lasso
$$\min_{\beta \in \mathbb{R}^p} \left\{ \|y - X\beta\|_2^2 + \lambda_1 \|\beta\|_1 \right\} \tag{3}$$

Elastic-Net
$$\min_{\beta \in \mathbb{R}^p} \left\{ \|y - X\beta\|_2^2 + \lambda_1 \|\beta\|_1 + \lambda_2 \|\beta\|_2^2 \right\} \tag{4}$$

```
Least Squares:
  model.add(Dense(1, input_dim=K, activation='linear'))
Ridge        :
  model.add(Dense(1, input_dim=K, activation='linear', W_regularizer=l2(0.01)))
Lasso        :
  model.add(Dense(1, input_dim=K, activation='linear', W_regularizer=l1(0.01)))
Elastic-Net  :
  model.add(Dense(1, input_dim=K, activation='linear', W_regularizer=l1l2(0.01,0.01)))

Example:
  model = Sequential()
  model.add(Dense(1, input_dim=K, activation='linear'))
  model.compile(loss='mse', optimizer=sgd)
  model.fit(x1, y1, nb_epoch=2,batch_size=100)
```

A bit more on SGD:

```
SGD(lr=0.01, decay=1e-6, nesterov=True, momentum=0.95)
```

Update procedure for weights W, momentum M, and learning_rate . For each batch we do the following:

$$\begin{aligned} M_{t+1} &= \mu \cdot M_t - \epsilon_t \cdot \sum_{i \in \text{batch}} \nabla \mathrm{f}_i(W_t + 1_{Nesterov} \mu \cdot M_t) \\ W_{t+1} &= W_{t+1} + M_{t+1} \\ \epsilon_{t+1} &= (1 - decay) \cdot \epsilon_t \end{aligned}$$

where $\mu = \text{momentum} = 0.95$ is momentum, $\epsilon = lr = 0.01$ and f_i is loss function for i^{th} example, and $1_{Nesterov}$ indicating that this term is used in case momentum is Nesterov.

Outline

Types of Selection

- Forward Selection Step : add highest correlation variable
- Stepwise Forward Selection Step: add ϵ to β of highest correlation variable
- LARS Step: add as much to β of highest correlation variable as needed to make it tie with second highest.
- These are one pass algorithms in linear case models.
- For general more non-linear models forward selection can be used.
- You can also do multi-pass selection in case you have lots of data (otherwise prone to be useless due to overfit).

Outline

Various things can be done done with regression

$$y = \beta_0 + \sum \beta_i x_i$$

- Replace x_i with $f(x_i)$, examples of $f(x_i)$ can be polynomial without necesserily including all intermediate powers.
- Add some higher power like x_i^3 to the model for all or some x_i
- Add cross-terms like $x_i x_j$

This can be done in stages and together with feature selection at each step.
(WARNING: Adding to many variables is prone to overfitting; hence this is usually usefull only when your N=number of samples is very large; even then feature selection in intermediate steps is a good idea.)

Outline

Principal Component Analysis and Dimensionality Reduction

- Principal Component Analysis can be viewed as another sort of feature selection algorithm but the emphasis here is not on the fit but on explanation of the variance in the data itself.
- In fact PCA can be viewed as iteratively finding directions that explain most of the variance of the data.
- Formulaicly PCA is transformation $T = XW$ where W is the set of eigenvectors of $X^T X$ sorted by eigenvalue.
- PCA-based dimensionality reduction is simply using W_k correspodning to first k eigenvalues, hence $T = XW_k$.

PCA, Factor Models

- In PCA the transformation is determined by W that corresponds to eigenvectors
- In factor models, explanatory factors give you W and are preditermined by some factor model.
- The factor models have advantage of providing some meaning to each of the new transformed direction.

Outline

K-Means Introduction

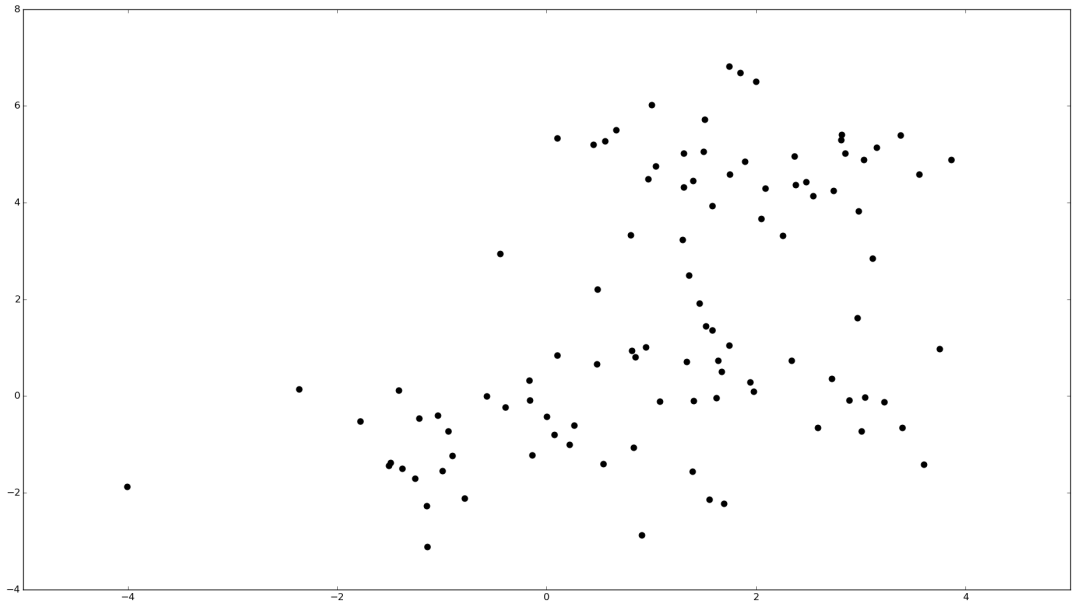
K-Means is a method of classifying points into 'groups' or 'clusters', based on their 'proximity'. For traditional k-means, the proximity measure is Euclidean distance, ie $\|\cdot\|_2$. If we want to form K clusters, we minimize as follows:

$$\arg\min_{S} \sum_{i=0}^{K} \sum_{x \in S_i} \|x - C_i\|$$

where C_i is the geometric center of all of the points belonging to S_i, the set of all points in cluster i. This is equivalent to assuming the points are normally distributed around each center.

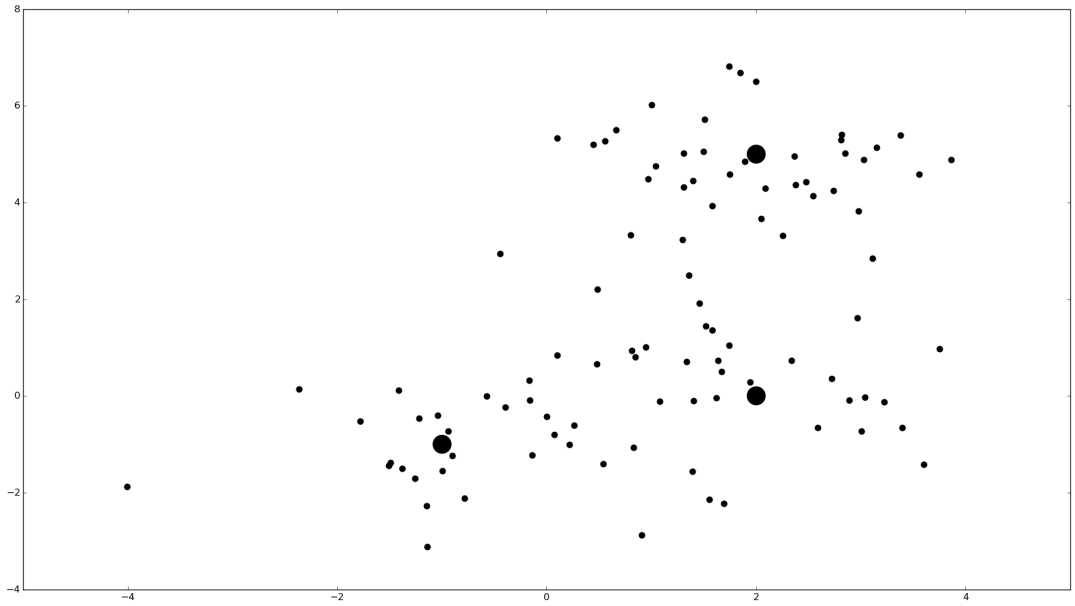
K-Means Introduction

Here is sample of clusters in 2 dimensions.



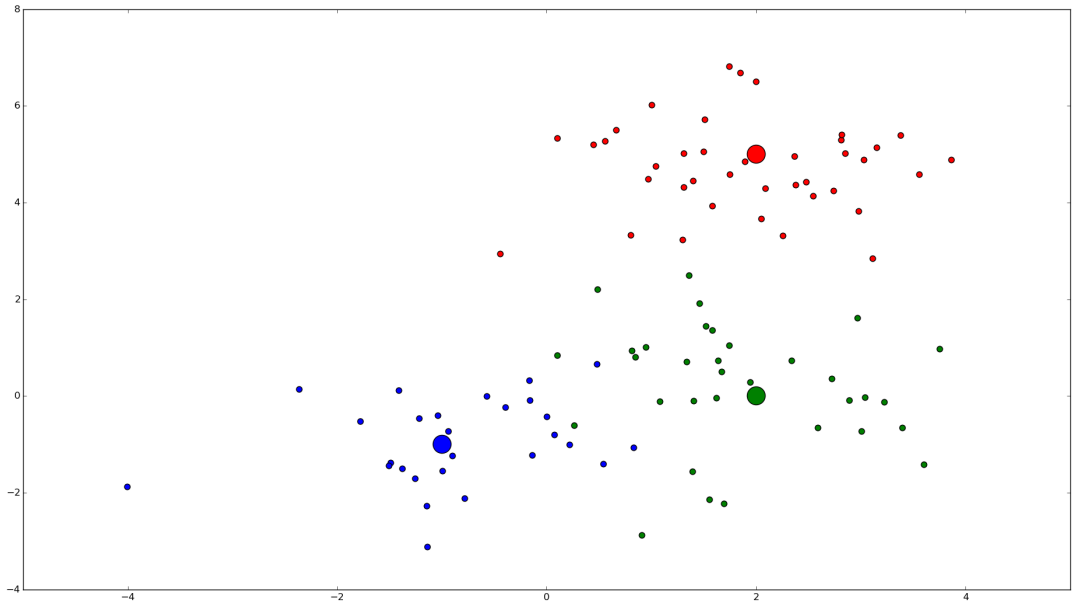
K-Means Introduction

Here is sample of clusters in 2 dimensions.



K-Means Introduction

Here is sample of clusters in 2 dimensions.



K-Means Algorithm

So how do we do this? We need an algorithm to solve the minimization problem from earlier:

$$\arg\min_{S} \sum_{i=0}^{K} \sum_{x \in S_i} \|x - C_i\|$$

Solving this problem directly isn't tractable - in fact, it's NP-hard for almost all cases.

KMeans Algorithm

Algorithm 1 KMeans Algorithm (Lloyd's Algorithm)

Require: $N > K > 1$

$centers \leftarrow$ select K random entries from $points$

repeat

for $i < N$ **do**

$assigned\_centers[i] \leftarrow$ find_nearest_center$(points[i])$

end for

for $i < K$ **do**

$centroids[i] \leftarrow$ find_centroid(i)

end for

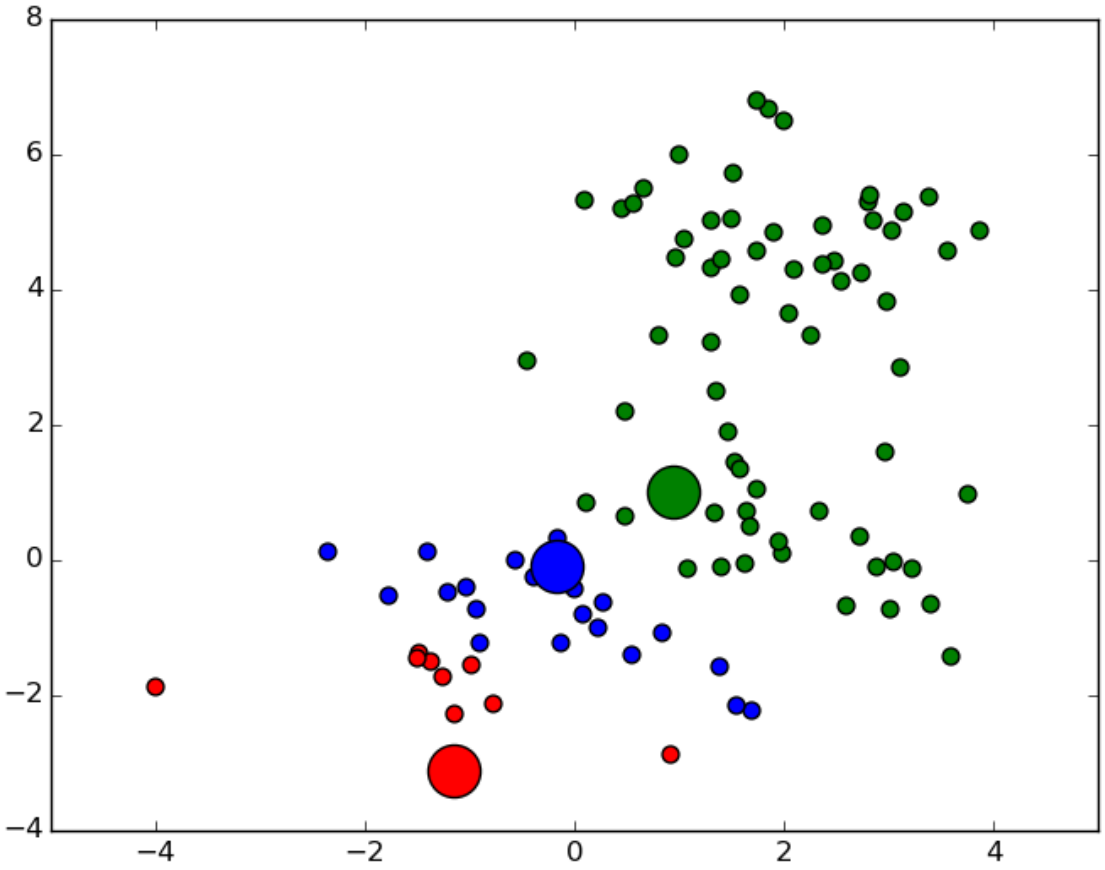
until $assigned\_centers$ does not change

Initialization

So how do we *randomly* assign the initial clusters? There are a few popular choices:

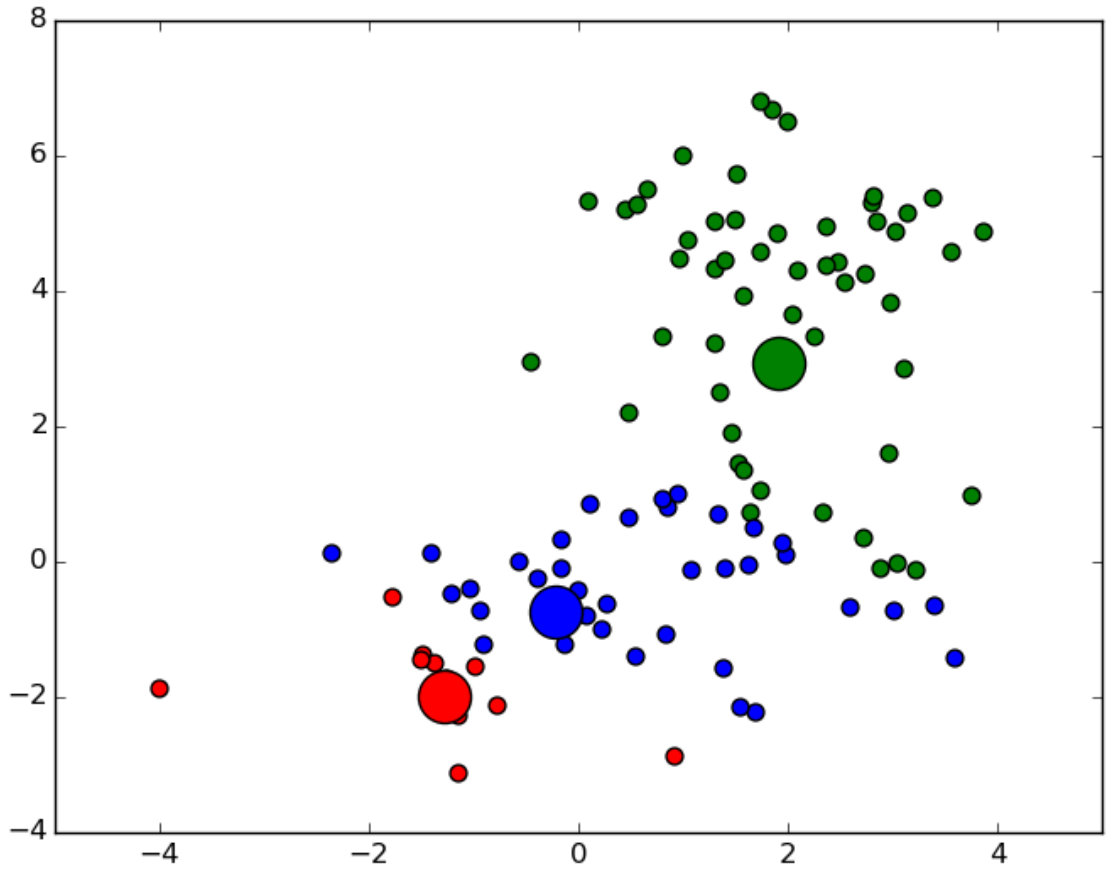
- Choose K random points from the initial list (Forgy Method).
- The Random partition method assigns each point a cluster at random, then calculates the centroids.

An example



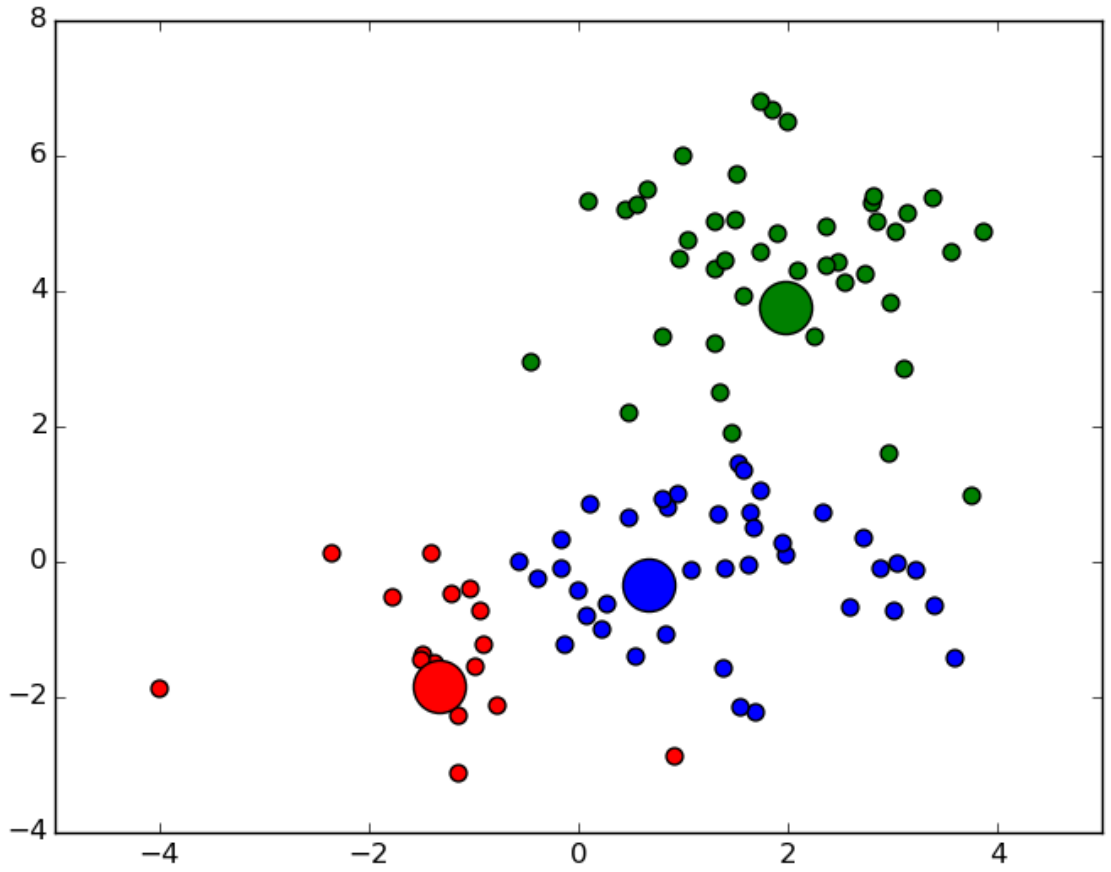
0 iterations

An example



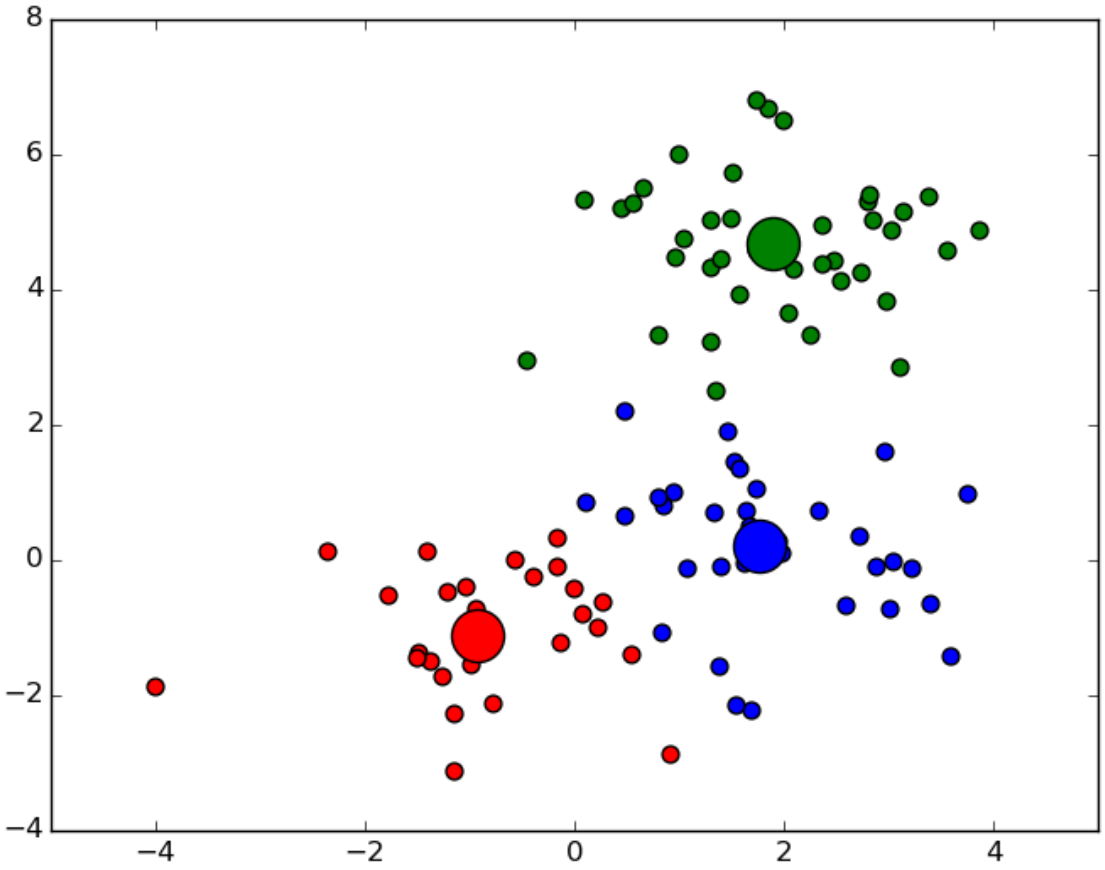
1 iterations

An example



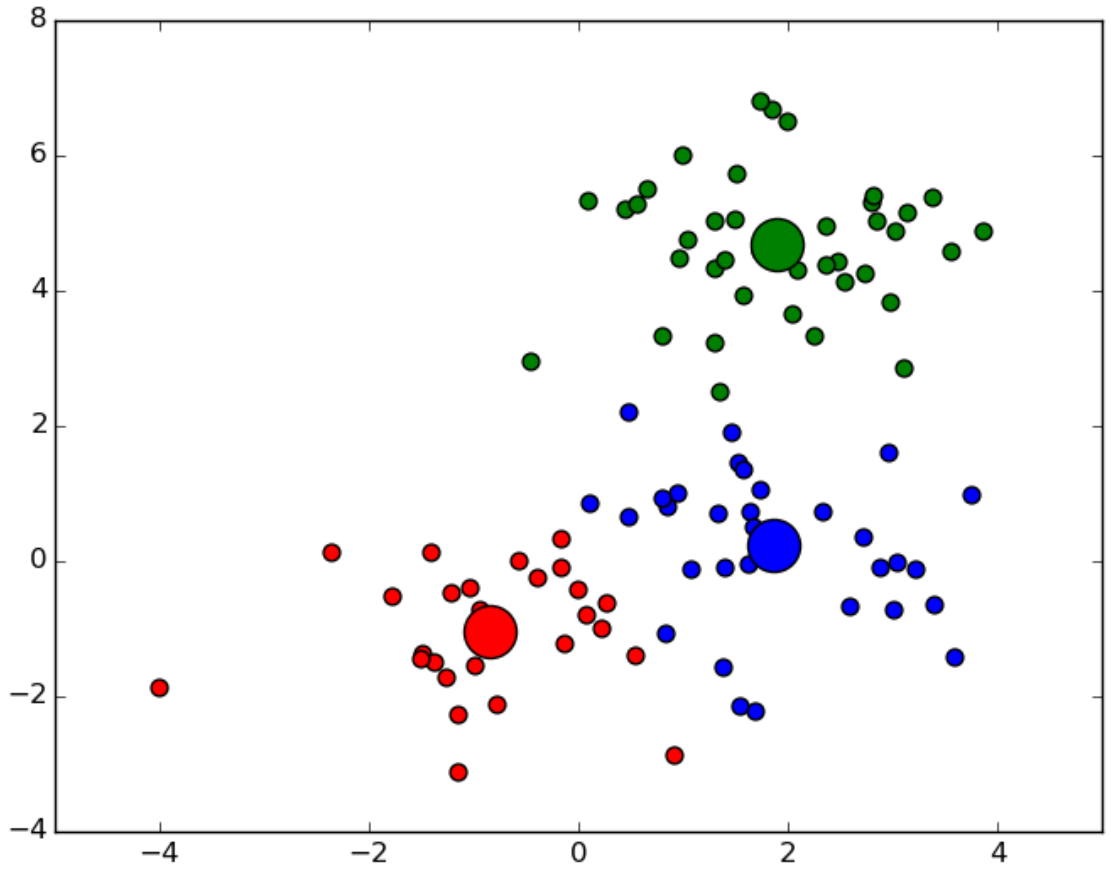
2 iterations

An example



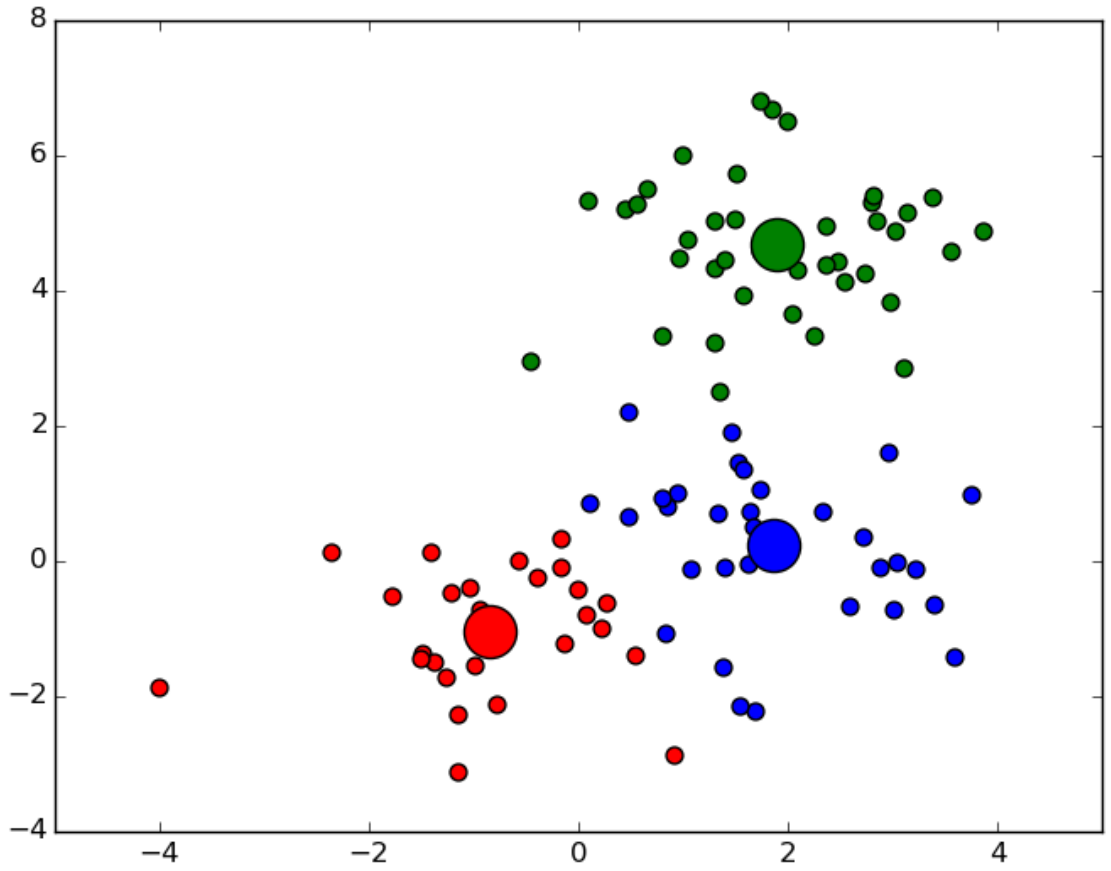
5 iterations

An example



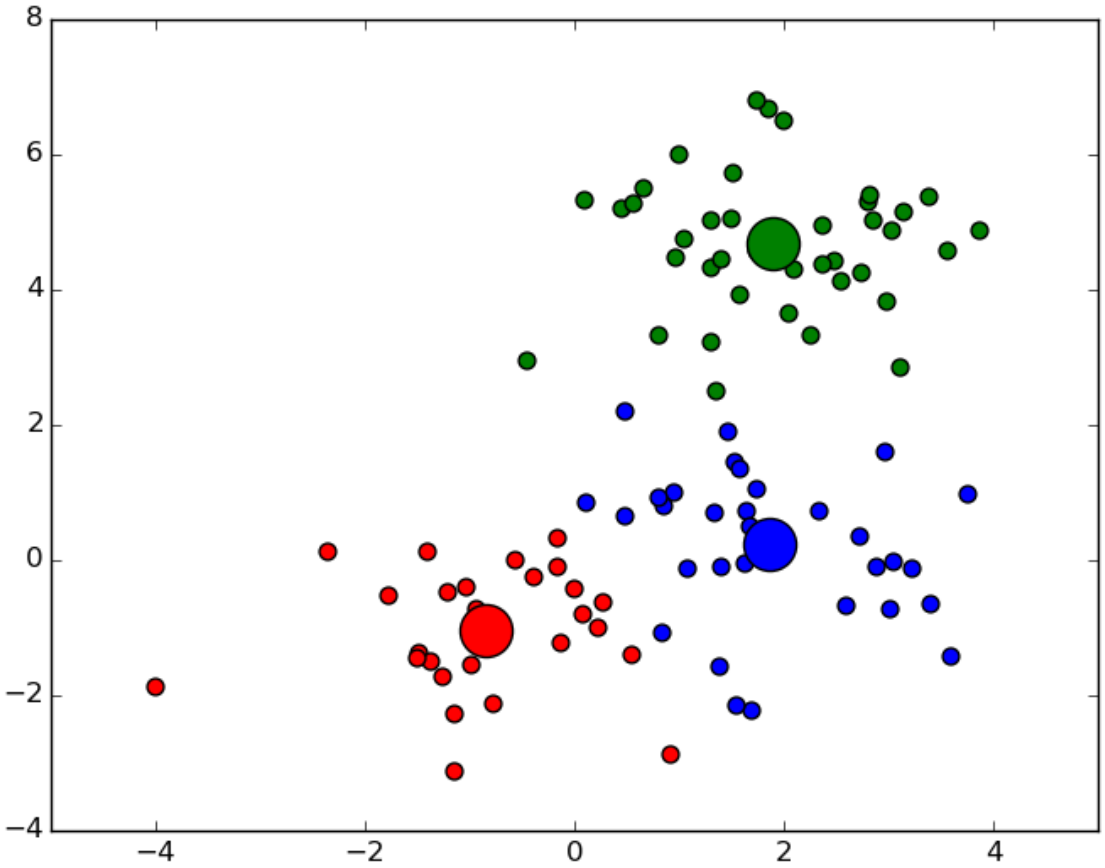
10 iterations

An example



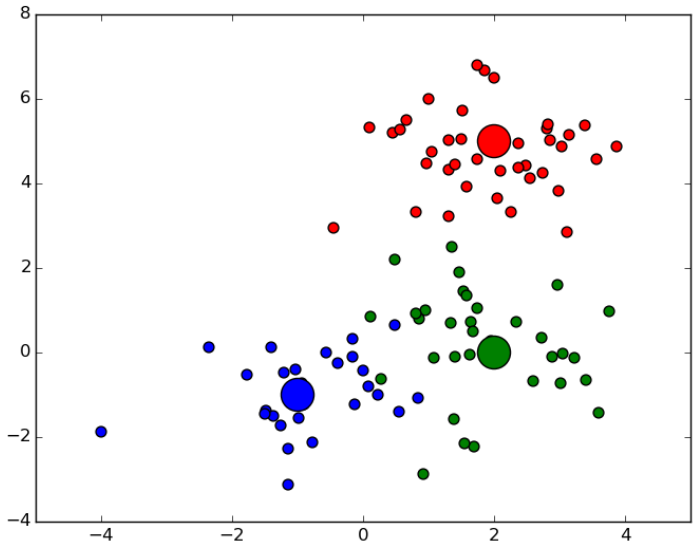
20 iterations

An example



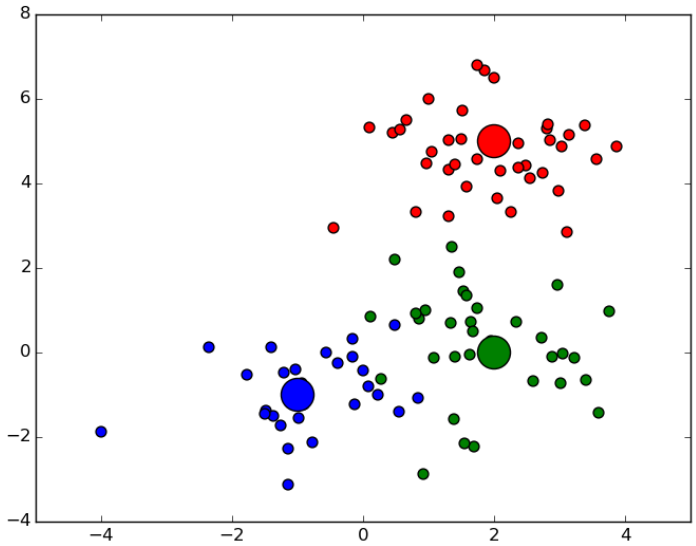
50 iterations

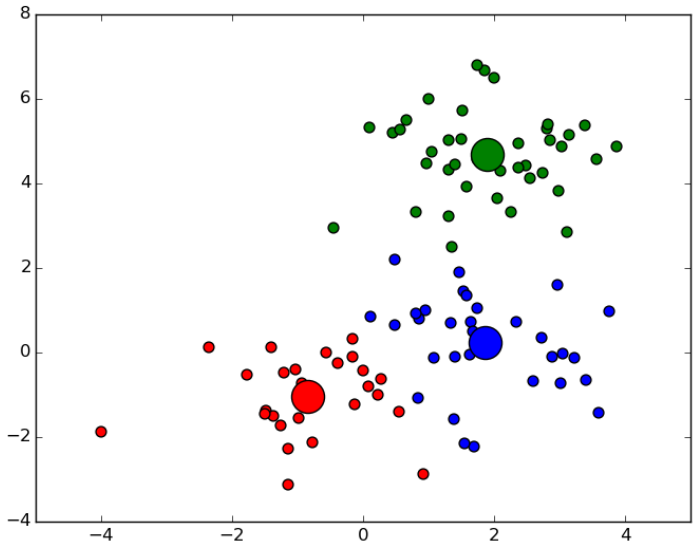
An example



Our original data

An example





Actual Centers		Calculated Centers
(2.0, 5.0)	$\rightarrow$	(1.9, 4.7)
(2.0, 0.0)	$\rightarrow$	(1.9, 0.2)
(-1.0, -1.0)	$\rightarrow$	(-0.8, -1.1)

Notes

- The algorithms discussed will only find a **LOCAL** minimum. To be sure we're getting a near-optimal solution, we should repeat this with different starting centroids.
- How do we know how many clusters, K, to look for? Adding more clusters will always improve the metrics.

G-Means offers a way for us to intuit K:

Algorithm 2 GMeans Algorithm

$K \leftarrow 0$
repeat
  $K \leftarrow K + 1$
  $centers \leftarrow \text{KMeans}(points, K)$
until $(points - centers) \sim \mathcal{N}$

We can test to see if the residuals are distributed normally through one of many tests, like Jarque-Bera or Kolmogorov-Smirnov.

Outline

The EM Algorithm

"Expectation-Maximization" is a generic algorithm for estimating MLE parameters. The derivation is complex, and we will go through it quickly here. An excellent reference is Andrew Ng's ML Notes.

$$
\begin{aligned}
X &= \{x_0, x_1, \ldots, x_{n-2}, x_{n_1}\} \\
Z &= \{z_0, z_1, \ldots, z_{n-2}, z_{n_1}\} \text{ \# These are are our latent variables} \\
\mathcal{L}(\theta|X,Z) &= \prod_{i=0}^{N} P(x_i;\theta) \\
\ell(\theta|X,Z) &= \sum_{i=0}^{N} \log P(x_i;\theta) \\
\ell(\theta|X,Z) &= \sum_{i=0}^{N} \log \sum_{j=0}^{K} P(x_i, z_i;\theta)
\end{aligned}
$$

The EM Algorithm

Let's define Q_i as a probability distribution of z_i. Now we can say:

$$\begin{aligned}
\ell(\theta|X,Z) &= \sum_{i=0}^{N} \log \sum_{j=0}^{K} P(x_i, z_i; \theta) \\
\ell(\theta|X,Z) &= \sum_{i=0}^{N} \log \sum_{j=0}^{K} Q_i(z_j) \frac{P(x_i, z_i; \theta)}{Q_i(z_j)}
\end{aligned}$$

Let's define Q_i as a probability distribution of z_i. Now we can say:

$$\begin{aligned}
\ell(\theta|X,Z) &= \sum_{i=0}^{N} \log \sum_{j=0}^{K} P(x_i, z_i; \theta) \\
\ell(\theta|X,Z) &= \sum_{i=0}^{N} \log \sum_{j=0}^{K} Q_i(z_j) \frac{P(x_i, z_i; \theta)}{Q_i(z_j)} \\
\ell(\theta|X,Z) &= \sum_{i=0}^{N} \log \mathbb{E}_{z_j \sim Q_i(z_j)} \frac{P(x_i, z_i; \theta)}{Q_i(z_j)}
\end{aligned}$$

The EM Algorithm

Let's define Q_i as a probability distribution of z_i. Now we can say:

$$\begin{aligned}
\ell(\theta|X,Z) &= \sum_{i=0}^{N} \log \sum_{j=0}^{K} P(x_i, z_i; \theta) \\
\ell(\theta|X,Z) &= \sum_{i=0}^{N} \log \sum_{j=0}^{K} Q_i(z_j) \frac{P(x_i, z_i; \theta)}{Q_i(z_j)} \\
\ell(\theta|X,Z) &= \sum_{i=0}^{N} \log \mathbb{E}_{z_j \sim Q_i(z_j)} \frac{P(x_i, z_i; \theta)}{Q_i(z_j)} \\
\ell(\theta|X,Z) &\geq \sum_{i=0}^{N} \mathbb{E}_{z_j \sim Q_i(z_j)} \log \frac{P(x_i, z_i; \theta)}{Q_i(z_j)} \\
\ell(\theta|X,Z) &\geq \sum_{i=0}^{N} \sum_{j=0}^{K} Q_i(z_j) \log \frac{P(x_i, z_i; \theta)}{Q_i(z_j)}
\end{aligned}$$

The EM Algorithm

The next steps are tricky (again, refer to Andrew Ng's ML Notes for more details). We said that Q_i was a PDF for z_i, so let's choose a good one:

$$\begin{aligned} Q_i(z_i) &= \frac{P(x_i, z_i; \theta)}{\sum_j P(x_i, z_j; \theta)} \\ &= P(z_i | x_i; \theta) \end{aligned}$$

Now, we're ready to look at the algorithm itself.

Algorithm 3 EM Algorithm

$\theta^0 =$ initial guess

$m \leftarrow 1$

repeat

$Q_i^m = P(z_i|x_i;\theta^m)$

$\theta^{m+1} = \arg\max_\theta \sum_{i=0}^{N}\sum_{j=0}^{K} Q_i^m(z_j)\log\frac{P(x_i,z_i;\theta^{m+1})}{Q_i^m(z_j)}$

$m \leftarrow m+1$

until convergence of ℓ

Take careful note of θ in the MLE step. Proof of convergence can be found in the Ng reference mentioned above.

An EM Application: Soft KMeans

Let's look at this in the context of a 'soft' KMeans Algorithm with 2 clusters. This means that instead of assuming each point is in a given cluster, C, we'll assign a probability that it's in each cluster. Here's our setup:

$$\begin{aligned} X &= \{x_0, x_1, ..., x_n\} \\ Z &= \{z_0, z_1, ..., z_n\} \\ \theta &= \{\mu_0, \sigma_0^2, \mu_1, \sigma_1^2, \pi_0, \pi_1\} \end{aligned}$$

Soft KMeans: The “E” Step

$$
\begin{aligned}
Q_i(z_j) &= \frac{P(x_i, z_j; \theta)}{\sum_m P(x_i, z_m; \theta)} \\
&= \frac{\phi_j(x_i; \theta)}{\sum_m \phi_m(x_i; \theta)}
\end{aligned}
$$

$Q_i(z_j)$ is the probability that point i belongs to C_j. Since we don’t make a hard assignment to any cluster, this is why we call this a ’Soft K-Means’ algorithm.

Soft KMeans: The "M" Step

To make notation simpler, now that we've done the "E" step, we'll say $w_{i,j}$ is the probability that point i is in C_j. The "M" step is:

$$\arg\max_{\theta} \quad \sum_{i=0}^{N} \sum_{j=0}^{K} \quad w_{i,j} \log \frac{P(x_i, z_j; \theta)}{w_{i,j}}$$

$$\arg\max_{\theta} \quad \sum_{i=0}^{N} \sum_{j=0}^{K} \quad w_{i,j} \log \frac{P(x_i | z_j; \theta) P(z_j)}{w_{i,j}}$$

$$\arg\max_{\theta} \quad \sum_{i=0}^{N} \sum_{j=0}^{K} \quad w_{i,j} \log \frac{\phi_{j;\theta}(x_i) \pi_j}{w_{i,j}}$$

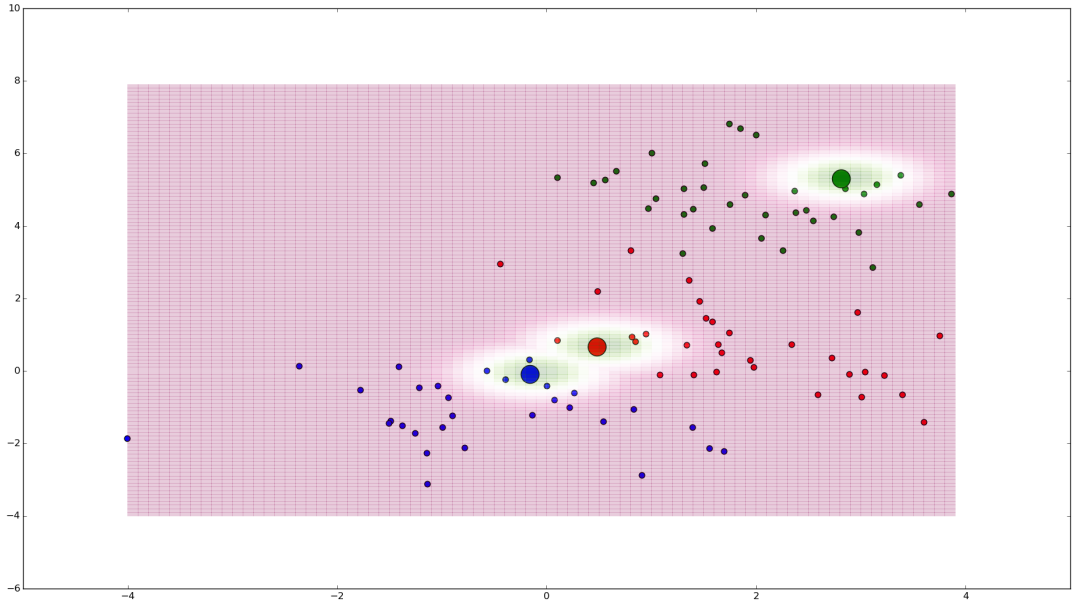
Soft KMeans: The "M" Step

$$\arg\max_{\theta} \quad \sum_{i=0}^{N} \sum_{j=0}^{K} \quad w_{i,j} \log \frac{\phi_{j;\theta}(x_i)\pi_j}{w_{i,j}}$$

If we take our function from before, and take some derivatives, we get very simple update rules:

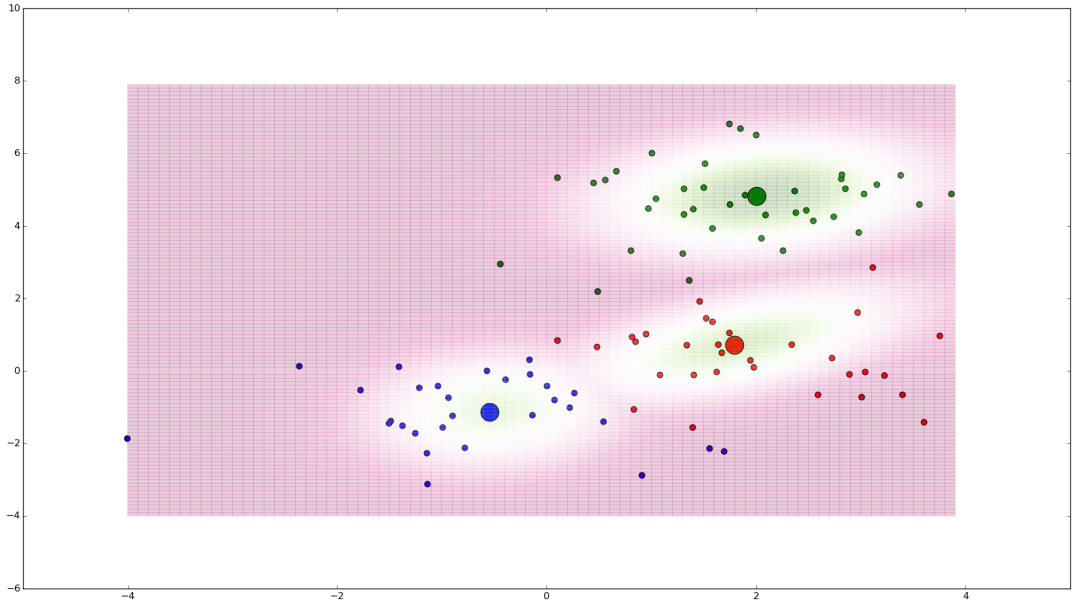
$$\begin{aligned}
\mu_j &= \frac{\sum_{i=0}^{N} w_{ij} x_i}{\sum_{i=0}^{N} w_{ij}} \\
\pi_j &= \frac{1}{N} \sum_{i} w_{ij} \\
\sigma_j^2 &= \frac{\sum_{i=0}^{N} w_{ij} \|x_i - \mu_j\|_2}{\sum_{i=0}^{N} w_{ij}}
\end{aligned}$$

An example



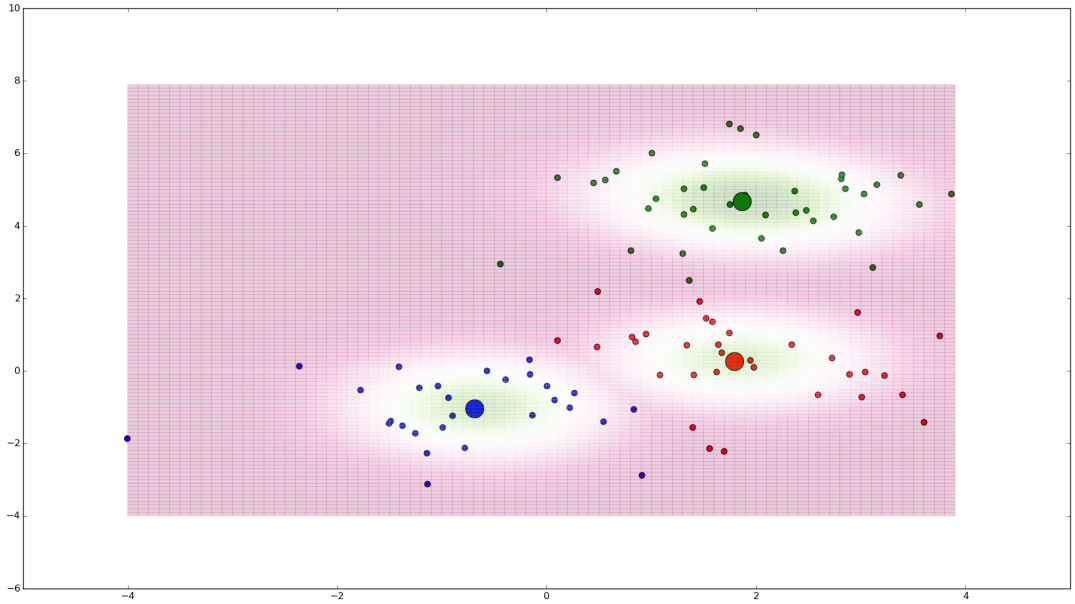
0 iterations

An example



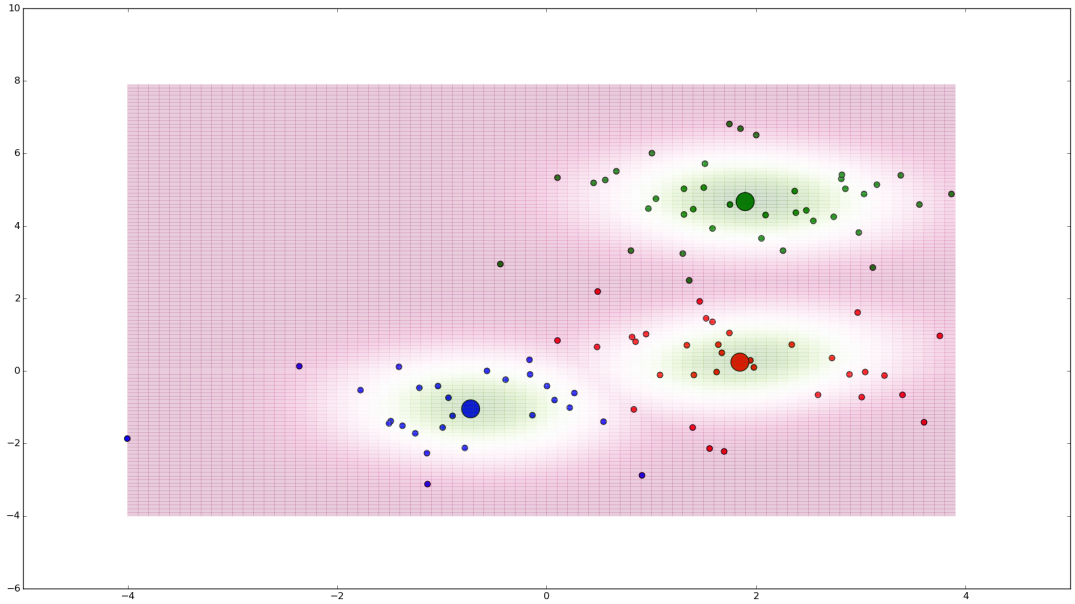
1 iterations

An example



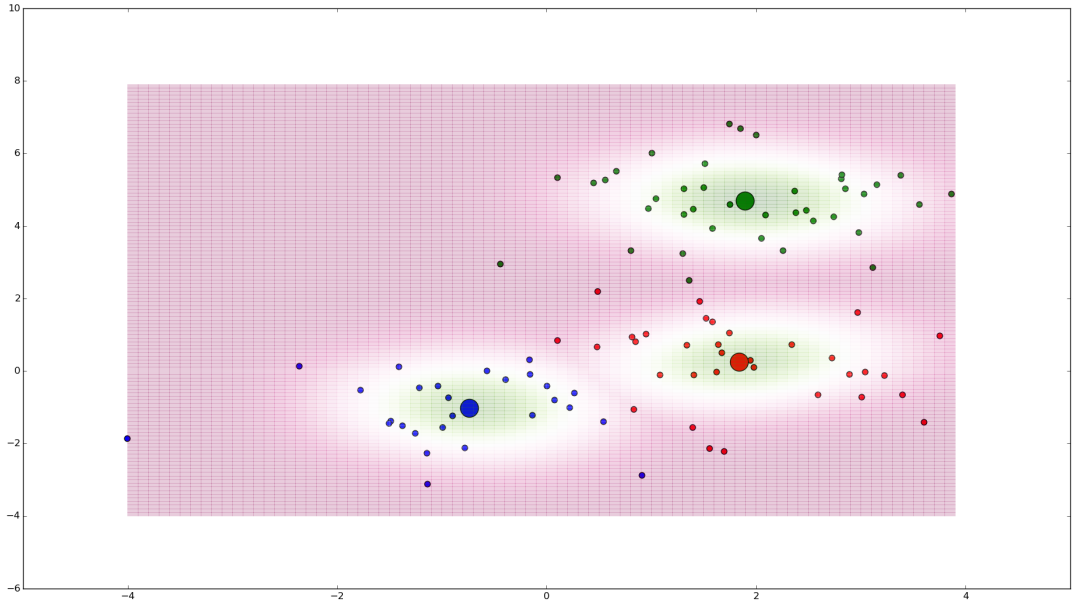
2 iterations

An example



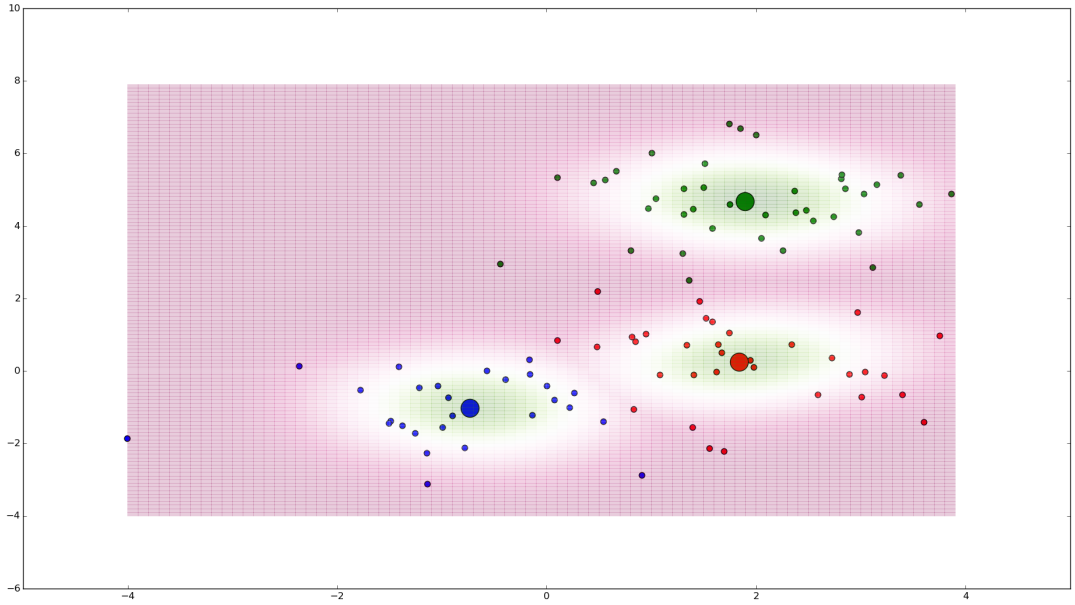
5 iterations

An example



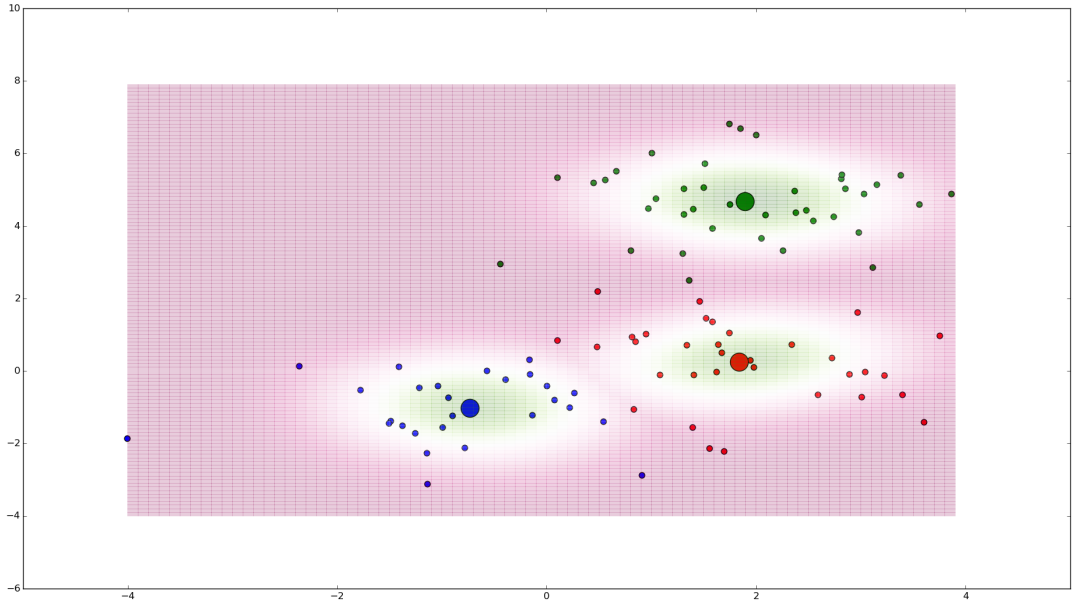
10 iterations

An example



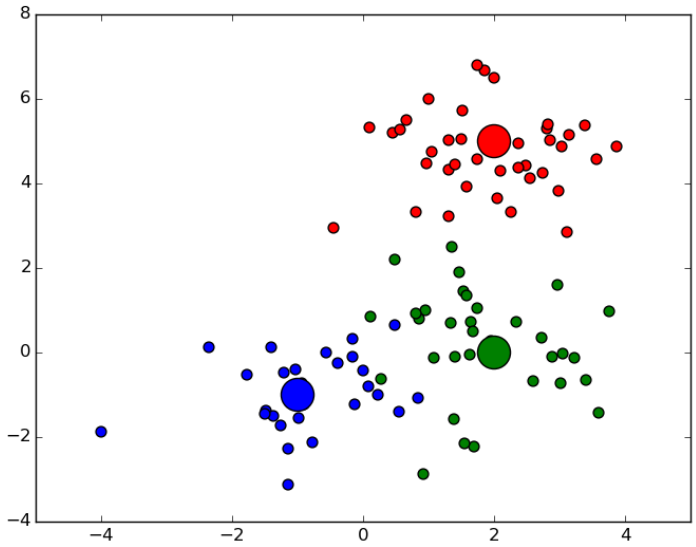
20 iterations

An example



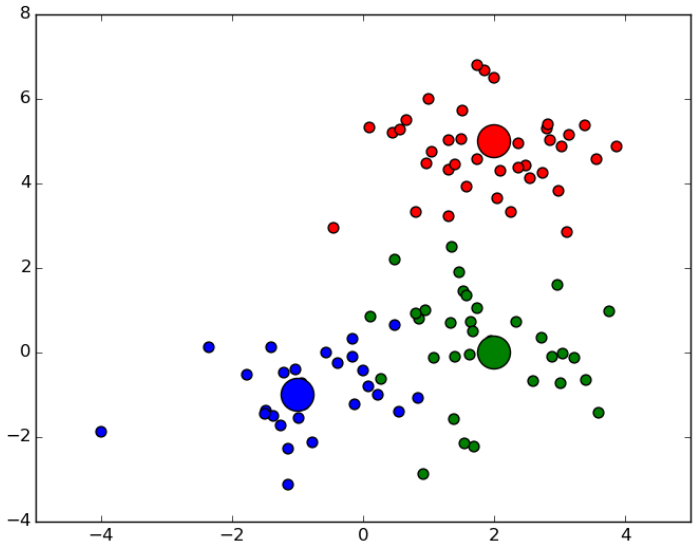
50 iterations

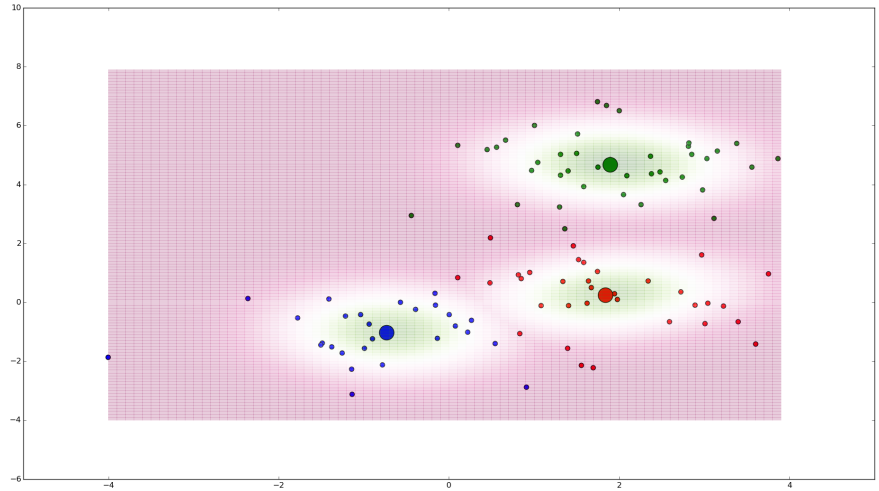
An example



Our original data

An example





Actual Centers		Calculated Centers
(2.0, 5.0)	$\rightarrow$	(1.9, 4.7)
(2.0, 0.0)	$\rightarrow$	(1.8, 0.2)
(-1.0, -1.0)	$\rightarrow$	(-0.7, -1.0)