ColumbiaX: Machine Learning

Lecture 14

.

Prof. John Paisley

Department of Electrical Engineering
& Data Science Institute

Columbia University

Unsupervised Learning

SUPERVISED LEARNING (focussing on so far)

Framework of supervised learning

Given: Pairs $(x_1, y_1), \ldots, (x_n, y_n)$. Think of x as input and y as output.

Learn: A function $f(x)$ that accurately predicts $y_i \approx f(x_i)$ on this data.

Goal: Use the function $f(x)$ to predict new y_0 given x_0. [we learned on training data]

Some underlying distribution that we don't get to observe. The nature is using to generate labelled pairs of x and y.

Probabilistic motivation

If we think of (x, y) as a random variable with joint distribution $p(x, y)$, then supervised learning seeks to learn the conditional distribution $p(y|x)$. *

where we directly defined this conditional distribution on the label/output, given an input. For ex: with logistic regression.

This can be done either directly or indirectly:

Directly: e.g., with logistic regression where $p(y|x) =$ sigmoid function

Indirectly: e.g., with a Bayes classifier **

** where we use Bayes rule to turn y|x into x|y times prior on y.

class conditional distribution

prior

$$y = \arg\max_k \; p(y = k|x) = \arg\max_k \; p(x|y = k)p(y = k)$$

* So there is some joint distribution, and we get to see some pairs from the joint distribution. But then we want to learn the conditional distribution so that we can predict y given x.

UNSUPERVISED LEARNING

Equivalently we can use exactly the same rules of probability to say that the joint distribution of both x and y is the conditional likelihood of y given x times a prior on x.

Some motivation

- The Bayes classifier factorizes the joint density as $p(x, y) = p(x|y)p(y)$.
- The joint density can also be written as $p(x, y) = p(y|x)p(x)$.
- *Unsupervised learning* focuses on the term $p(x)$ — learning $p(x|y)$ on a class-specific subset has the same "feel." What should this be?
- This implies an underlying classification task, but often there isn't one.

Unsupervised learning

Given: A data set $x_1, \ldots, x_n$, where $x_i \in \mathcal{X}$, e.g., $\mathcal{X} = \mathbb{R}^d$

Define: Some model of the data (probabilistic or non-probabilistic).

Goal: Learn structure within the data set *as defined by the model*.

- Supervised learning has a clear performance metric: accuracy
- Unsupervised learning is often (but not always) more subjective

Unclear what exactly is the measure of quality is going to be.

* With supervised learning we either focussed on this term $\circ$ or we focussed on learning both of these terms together.

→ Now, develop models / define models that allow us just to define some sort of distribution on x. Or you could even think learning some class conditional distribution which we need to define.

→ we assume that this $p(x|y)$ was a simple multi-variate gaussian previously, but maybe there are more complex distributions that are better suited to the data we're modelling that we might want to learn.

** ① What's going on with this dataset?
② What are the statistical properties of this dataset etc.

SOME TYPES OF UNSUPERVISED LEARNING

Overview of second half of course

We will discuss a few (3) types of unsupervised learning approaches in the second half of the course.

Clustering models: Learn a partition of data $x_1, \ldots, x_n$ into groups.

Applications:
- Image segmentation, data quantization, preprocessing for other models

Matrix factorization: Learn an underlying dot-product representation of data.

Useful for:
- User preference modeling, topic modeling* (text documents), movie prediction, movie rating prediction.

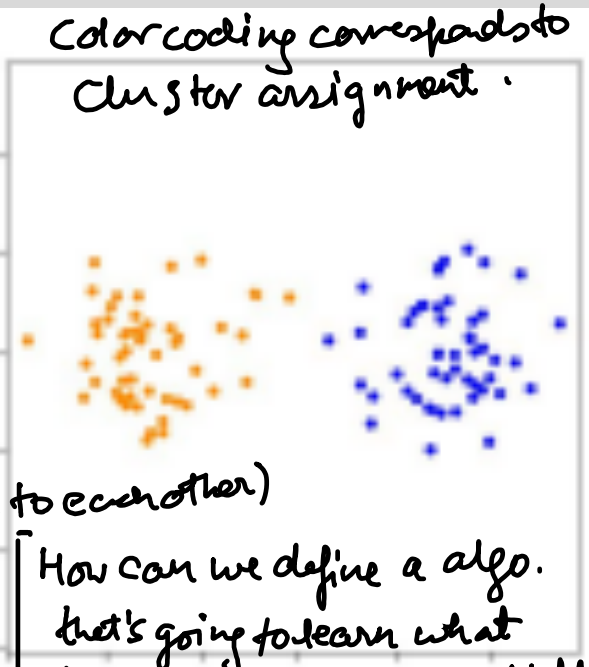
Sequential models: Learn a model based on sequential information. either the data is sequential in nature / we're going to use some sort of a sequential way of representing the data.

Applications:
- Learn how to rank objects, target tracking (in pairwise comparisons)

As will become evident, an unsupervised model can often be interpreted as a supervised model, or very easily turned into one. (division unclear, rough)

* learning what the underlying themes in these documents and so on.

CLUSTERING (in general, one specific algorithm called k means)



Problem

- Given data $x_1, \ldots, x_n$, partition it into groups called *clusters*.
- Find the clusters, given only the data. & some modelling assumption.
- Observations in same group $\Rightarrow$ "similar," (for ex: close to eachother)
  different groups $\Rightarrow$ "different." (somehow far apart)
- We will set how many clusters we learn.

Learn clusters such that

How can we define a algo. that's going to learn what these clusters are and tell us for each individual data point what cluster it belongs to?

To do this we are going to assume a latent variable. c ?

Cluster assignment representation

For K clusters, encode cluster assignments as an indicator $c \in \{1, \ldots, K\}$,

$$c_i = k \iff x_i \text{ is assigned to cluster } k$$

auxiliary variable of what cluster the observation x_i belongs to.

Clustering feels similar to classification in that we "label" an observation by its cluster assignment. The difference is that there is no ground truth.

* Assume that there are k clusters underlying our dataset, and this is going to be the parameter we set. (Later discuss some ways of deciding this no.)

** But we don't know in advance what these labels are. So we want to simultaneously learn the clusters and learn the labels.

THE K-MEANS ALGORITHM

CLUSTERING AND K-MEANS

K-means is the simplest and most fundamental clustering algorithm.

Input: $x_1, \ldots, x_n$, where $x \in \mathbb{R}^d$.

Output: Vector $\boldsymbol{c}$ of cluster assignments, and K mean vectors $\boldsymbol{\mu}$

(1) ▶ $\boldsymbol{c} = (c_1, \ldots, c_n), \quad c_i \in \{1, \ldots, K\}$

- If $c_i = c_j = k$, then x_i and x_j are *clustered together* in cluster k.

(2) ▶ $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_K), \quad \mu_k \in \mathbb{R}^d$ (same space as x_i)

- Each μ_k (called a *centroid*) defines a cluster. (belongs to)

As usual, we need to define an *objective function*. We pick one that:

1. Tells us what are good $\boldsymbol{c}$ and $\boldsymbol{\mu}$, and
2. That is easy to optimize.

↑ kmeans comes from what obj. function we define

K-MEANS OBJECTIVE FUNCTION

The K-means objective function can be written as

$$\boldsymbol{\mu}^*, \boldsymbol{c}^* = \arg\min_{\boldsymbol{\mu}, \boldsymbol{c}} \sum_{i=1}^{n} \sum_{k=1}^{K} \mathbb{1}\{c_i = k\} \|x_i - \mu_k\|^2$$

don't care how far we are from other clusters.

(only assign one cluster to each datapoint.)

Some observations:

- K-means uses the squared Euclidean distance of x_i to the centroid μ_k.
- It only penalizes the distance of x_i to the centroid it's assigned to by c_i.

$$\mathcal{L} = \sum_{i=1}^{n} \sum_{k=1}^{K} \mathbb{1}\{c_i = k\} \|x_i - \mu_k\|^2 = \sum_{k=1}^{K} \sum_{i:c_i=k} \|x_i - \mu_k\|^2$$

- The objective function is "non-convex"*
  - This means that we can't actually find the *optimal* $\boldsymbol{\mu}^*$ and $\boldsymbol{c}^*$.
  - We can only derive an *algorithm* for finding a *local optimum* (more later).

* We can only derive an algorithm to minimize this locally.

meaning we can't take the derivative of this thing and find the perfect values from mu and c. But we can derive an algorithm that is going to continually improve this objective function and continually minimize it until it converges to a local minimum.

Gradient-based optimization

We can't optimize the K-means objective function exactly by taking derivatives and setting to zero, so we use an iterative algorithm.

However, the algorithm we will use is different from gradient methods:

$$w \leftarrow w - \eta \nabla_w \mathcal{L} \qquad \text{(gradient descent)}$$

Recall: With gradient descent, when we update a parameter "w" we move in the direction that decreases the objective function, but

- It will almost certainly not move to the *best* value for that parameter.
- It may not even move to a better value if the step size η is too big.
- We also need the parameter w to be continuous-valued.

Our algorithm can't involve gradient methods because even though μ_j is a real vector, c_i is discrete valued variable that we want to set. And can take one of k values, and so it doesn't make sense to take derivative with respect to c. Also, not being able to derive a gradient method.

(specific example for k-means.)

Coordinate descent

We will discuss a new and widely used optimization procedure in the context of K-means clustering. We want to minimize the objective function

$$\mathcal{L} = \sum_{i=1}^{n} \sum_{k=1}^{K} \mathbb{1}\{c_i = k\} \|x_i - \mu_k\|^2. \quad *$$

We split the variables into two unknown sets $\boldsymbol{\mu}$ and $\boldsymbol{c}$. We can't find their best values *at the same time* to minimize $\mathcal{L}$. However, we will see that

- Fixing $\boldsymbol{\mu}$ we can find the best $\boldsymbol{c}$ exactly.
- Fixing $\boldsymbol{c}$ we can find the best $\boldsymbol{\mu}$ exactly.

**

This optimization approach is called *coordinate descent*: Hold one set of parameters fixed, and optimize the other set. Then switch which set is fixed.

* Instead of trying to minimize over everything at once, what we're going to do is split these model variables into 2 different sets:

① All vectors μ.
② all indicators or the cluster assignment c.

**

And so we're gonna derive an algorithm that's like that, where we hold one fixed, optimize it over another set of variables, then holding that other set fixed, optimize over the first set, and iterate back and forth.

So this is called coordinate descent.

And holding one coordinate fixed, we descend in the other coordinate.

Coordinate descent (in the context of K-means)

Input: $x_1, \ldots, x_n$ where $x_i \in \mathbb{R}^d$. Randomly initialize $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_K)$.

- Iterate back-and-forth between the following two steps:
  1. Given $\boldsymbol{\mu}$, find the best value $c_i \in \{1, \ldots, K\}$ for $i = 1, \ldots, n$.
  2. Given $\boldsymbol{c}$, find the best vector $\mu_k \in \mathbb{R}^d$ for $k = 1, \ldots, K$.

There's a circular way of thinking about why we need to iterate:

1. Given a particular $\boldsymbol{\mu}$, we may be able to find *the best* $\boldsymbol{c}$, but once we *change* $\boldsymbol{c}$ we can probably find a better $\boldsymbol{\mu}$.
2. Then find *the best* $\boldsymbol{\mu}$ for the new-and-improved $\boldsymbol{c}$ found in #1, but now that we've changed $\boldsymbol{\mu}$, there is probably a better $\boldsymbol{c}$.

And then keep iterating back and forth.

We have to iterate because the values of $\boldsymbol{\mu}$ and $\boldsymbol{c}$ *depend on each other*. This happens very frequently in unsupervised models.

if we hold mu fixed, we're going to be able to say that we can find the best cluster assignments in the vector c that minimizes the objective function for that particular μ.

But now that we've changed c, maybe there's a better mu that we can find.
Similarly, if we find the best most optimal setting for mu,
given a particular setting for c,
We've now changed mu, and so there's possibly a better c.
So it's a circular way of thinking, that every time we optimize a variable
in the model, it's only optimal for the setting of the other variables.
And then by optimizing it,
those other variables perhaps now can be optimized even more.

And the whole reason why we iterate back and forth is because mu and
c, they depend on each other in the objective function.
So they aren't separated from each other.
The setting of c is going to impact mu, and the setting of mu will impact c.
So when we change one, we're going to impact how good the other one is.

K-MEANS ALGORITHM: UPDATING $\boldsymbol{c}$

Assignment step

Given $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_K)$, update $\boldsymbol{c} = (c_1, \ldots, c_n)$. By rewriting $\mathcal{L}$, we notice the independence of each c_i given $\boldsymbol{\mu}$,

So here's the sum over each cluster assignment for the 1st data point. through the sum over each cluster assignment for the nth data point.

$$\mathcal{L} = \underbrace{\left(\sum_{k=1}^{K} \mathbb{1}\{c_1 = k\} \|x_1 - \mu_k\|^2 \right)}_{\text{distance of } x_1 \text{ to its assigned centroid}} + \cdots + \underbrace{\left(\sum_{k=1}^{K} \mathbb{1}\{c_n = k\} \|x_n - \mu_k\|^2 \right)}_{\text{distance of } x_n \text{ to its assigned centroid}}.$$

goal

We can see that whatever we set c_i, has no impact what we set c_n. We can view these n completely separate and independent problems, given the setting for $\mu_1 \cdots \mu_K$.

We can minimize $\mathcal{L}$ with respect to each c_i by minimizing each term above separately. The solution is to assign x_i to the closest centroid

$$c_i = \arg\min_k \|x_i - \mu_k\|^2. \quad *$$

Because there are only K options for each c_i, there are no derivatives. Simply calculate all the possible values for c_i and pick the best (smallest) one.

For ith data point to assign it to a cluster. We are just searching all of the clusters, and finding the one that ith data point is closest to. (Assignment step) (index)

So if we wanna minimize c1, we wanna minimize the objective over c1.
We wanna assign the first observation to one of the k clusters.
All we need to do is focus in on this term here, and minimize this thing independently of all of the other Terms and the objective.
So really the optimization of, for example, CI is an independent optimization where we pick out the I sum in this sequence.
And we now minimize.
We minimize the squared distance from the ith data point to the kth centroid over the subscript k.
So ci is simply going to be assigned to the cluster that has the minimum squared distance here.

K-MEANS ALGORITHM: UPDATING μ

We've updated c, we've optimized the objective over c for a particular μ. Because we have minimized each individual term over each datapoint point-wise, now we need to update μ.

Update step

Given $\boldsymbol{c} = (c_1, \ldots, c_n)$, update $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_K)$. For a given $\boldsymbol{c}$, we can break $\mathcal{L}$ into K clusters defined by $\boldsymbol{c}$ so that each μ_i is independent.

Cluster centroids.

This objective function is identical to the previous objective function, except now we've splitting the sum over k instead of over n.

$$\mathcal{L} = \underbrace{\left(\sum_{i=1}^{N} \mathbb{1}\{c_i = 1\}\|x_i - \mu_1\|^2\right)}_{\text{sum squared distance of data in cluster \#1}} + \cdots + \underbrace{\left(\sum_{i=1}^{N} \mathbb{1}\{c_i = K\}\|x_i - \mu_K\|^2\right)}_{\text{sum squared distance of data in cluster \#}K}.$$

minimize over all values of μ. *

For each k, we then optimize. Let $n_k = \sum_{i=1}^{n} \mathbb{1}\{c_i = k\}$. Then

We take the derivative of it with respect to μ_k and set it 0. And we find out we can analytically solve.

$$\mu_k = \arg\min_{\mu} \sum_{i=1}^{n} \mathbb{1}\{c_i = k\}\|x_i - \mu\|^2 \xrightarrow{\text{we end up getting}} \mu_k = \frac{1}{n_k} \sum_{i=1}^{n} x_i \mathbb{1}\{c_i = k\}.$$

That is, μ_k is the *mean* of the data assigned to cluster k.

**

Update for k^{th} centroid simply sums all of the data that was assigned to k^{th} centroid and then divides by the no. of data points in the k^{th} centroid

*

(1) And using exactly the same argument as before, notice that updating this,
for mu 1 is separate and independent of whatever the other values of mu are.

(2) Also important to remember is that ci can be only equal
to one value between one and K.
So for a particular value of ci, for a particular ci,
this would be zero for K minus one values, and
it will be equal to one for only one of these and only one of these terms,
corresponding to the cluster that ith the point was assigned to in the previous slide.

**

In other words, we take the mean,
you could think of it, of all of the data assigned to the kth centroid.
Hence the name k means cluster.

Algorithm: K-means clustering

Given: $x_1, \ldots, x_n$ where each $x \in \mathbb{R}^d$

Goal: Minimize $\mathcal{L} = \sum_{i=1}^{n} \sum_{k=1}^{K} \mathbb{1}\{c_i = k\} \|x_i - \mu_k\|^2$. K means objective function

- Randomly initialize $\mu = (\mu_1, \ldots, \mu_K)$. (centroids)
- Iterate until $\boldsymbol{c}$ and $\boldsymbol{\mu}$ stop changing → until d(objective function)

1. Update each c_i :

$$c_i = \arg\min_k \|x_i - \mu_k\|^2$$

[Index of cluster that x_i is closest to, in Euclidean sense.]

After updating each of these cluster assignments, we update the cluster centroid.

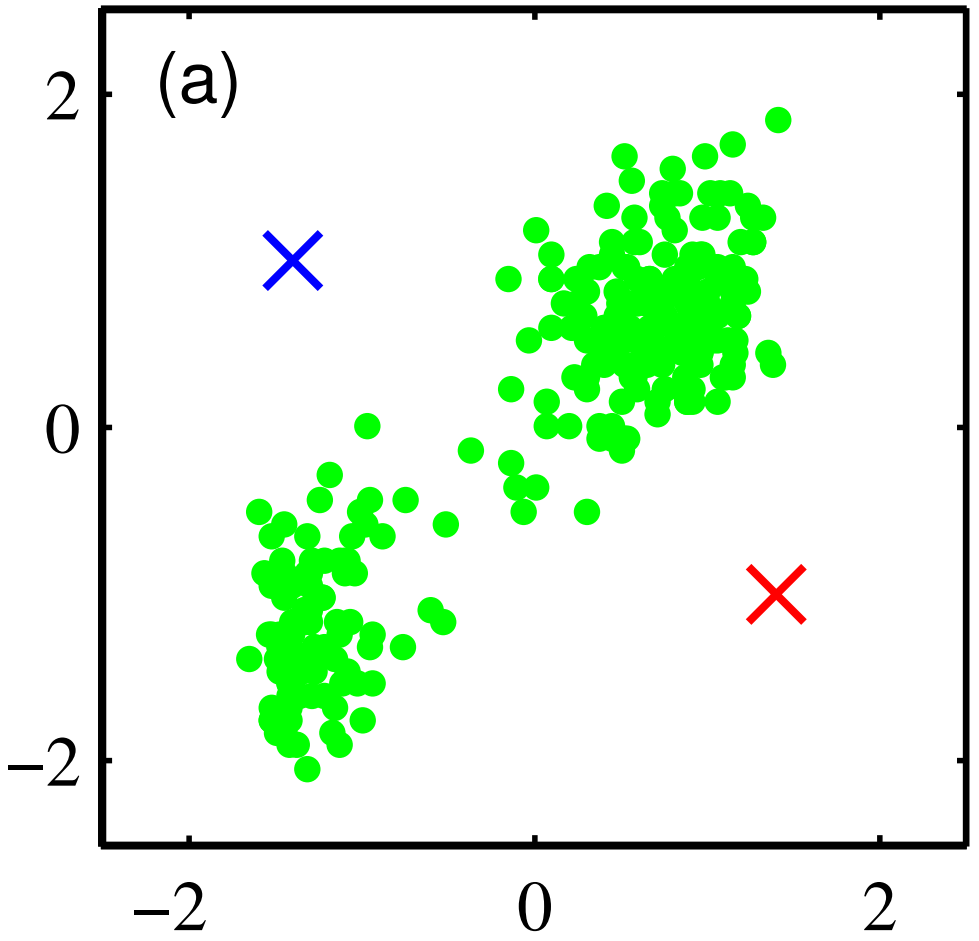
2. Update each μ_k : Set

$$n_k = \sum_{i=1}^{n} \mathbb{1}\{c_i = k\} \quad \text{and} \quad \mu_k = \frac{1}{n_k} \sum_{i=1}^{n} x_i \mathbb{1}\{c_i = k\}$$

avg. of all of the data assigned to k^{th} centroid.

according to the most recent update for the vector c.

K-MEANS ALGORITHM: EXAMPLE RUN



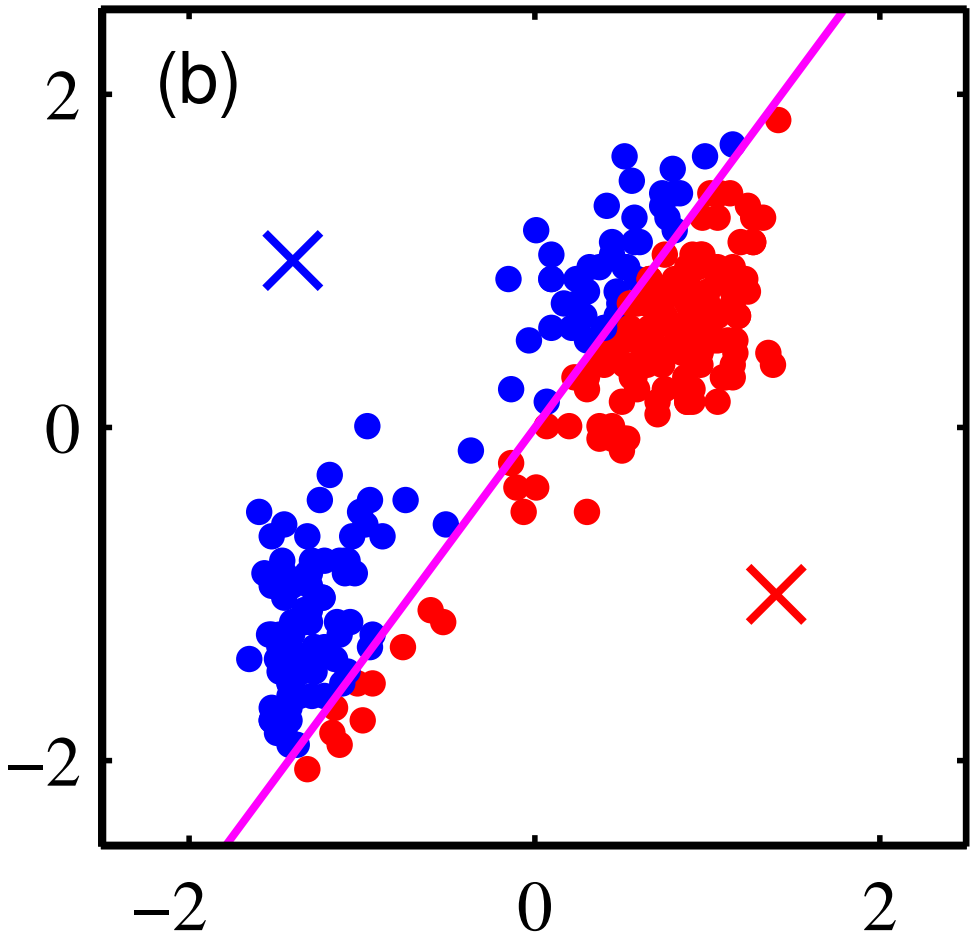
A random initialization

output:

① we want to output for each data point an index of what it was in the first cluster or the second cluster.

② And we also want to output the two centroids. (2 2-dimensional vectors.)

K-MEANS ALGORITHM: EXAMPLE RUN

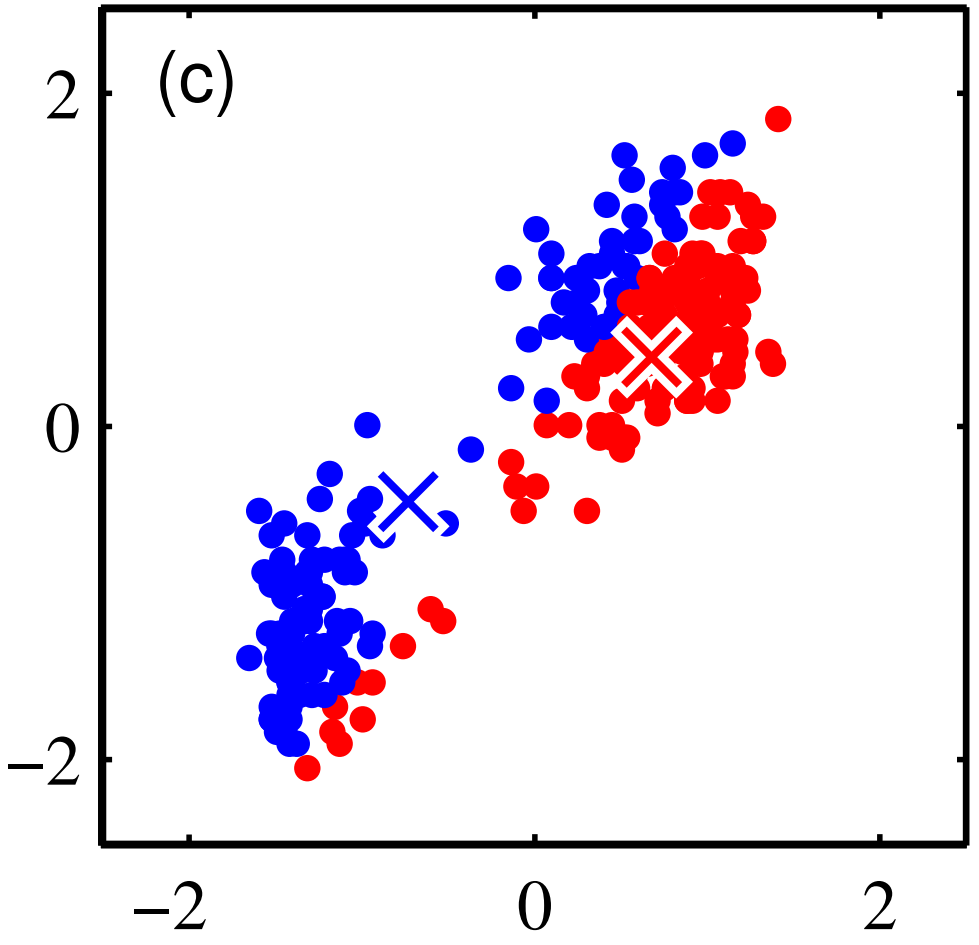


Iteration 1

Assign data to clusters

(all data to the nearest centroid.)

K-MEANS ALGORITHM: EXAMPLE RUN

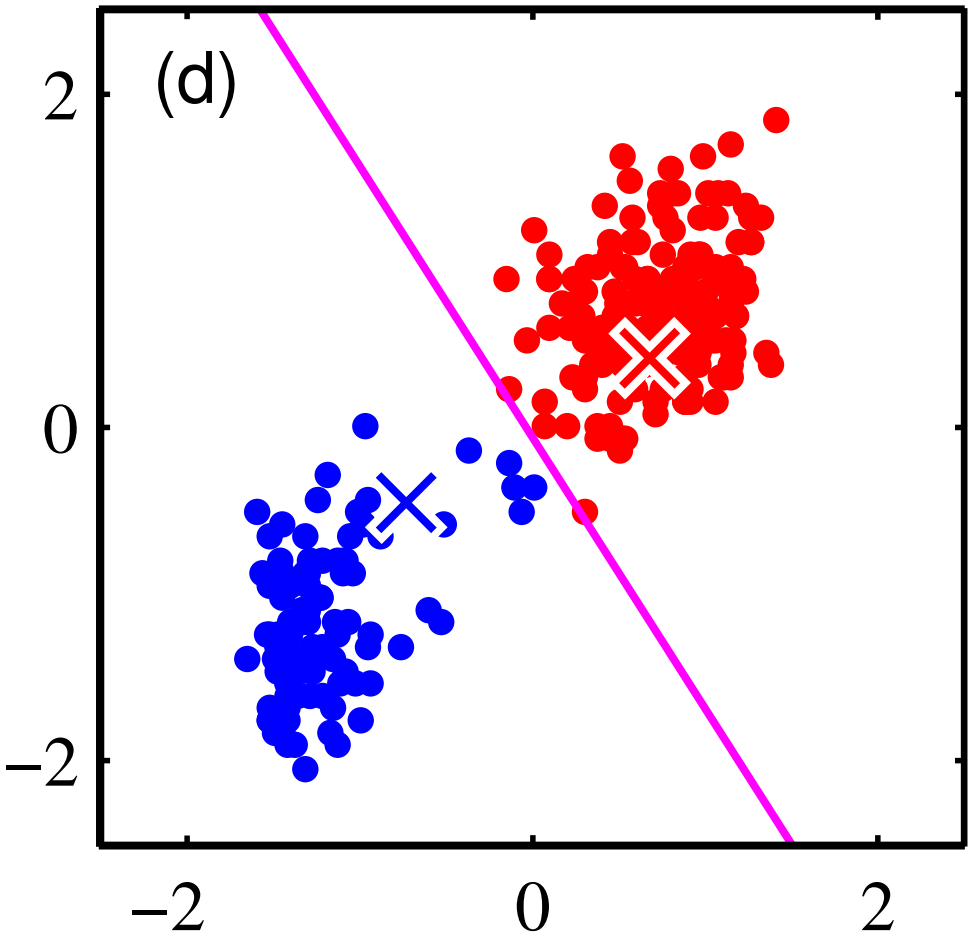


(avg. of all data points in a cluster.)

Iteration 1

Update the centroids

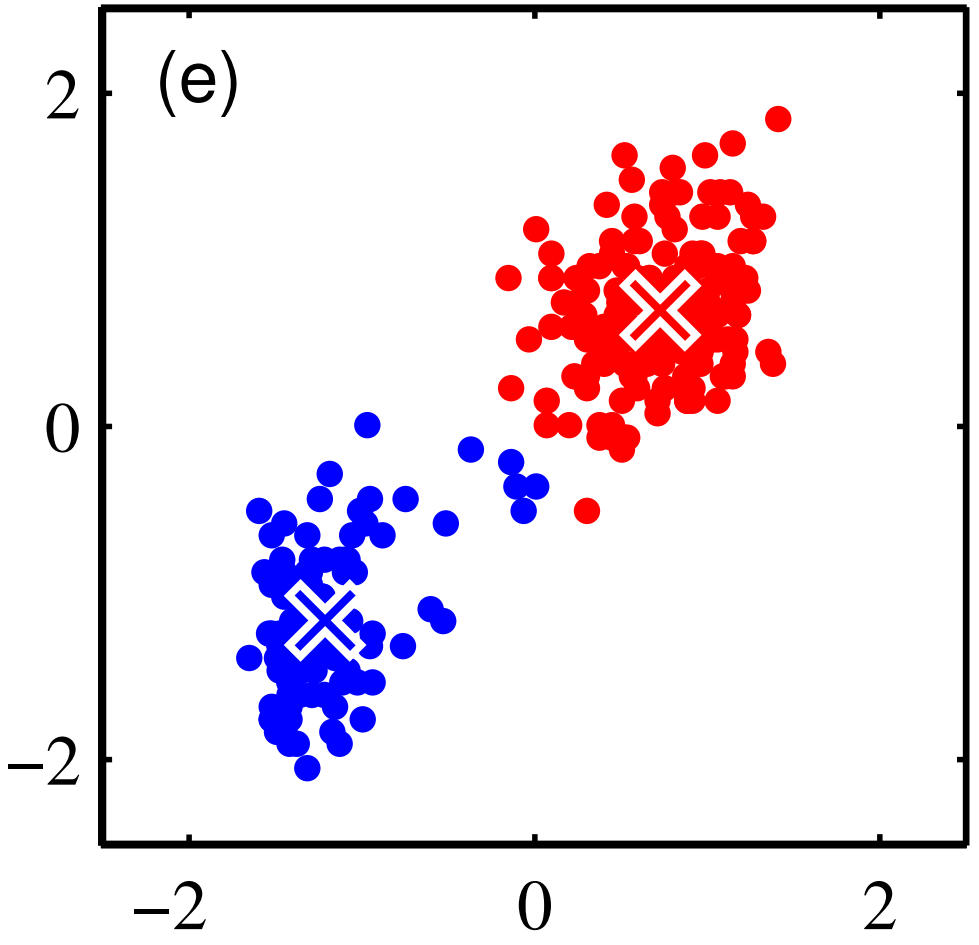
K-MEANS ALGORITHM: EXAMPLE RUN



Iteration 2

Assign data to clusters

K-means algorithm: Example run

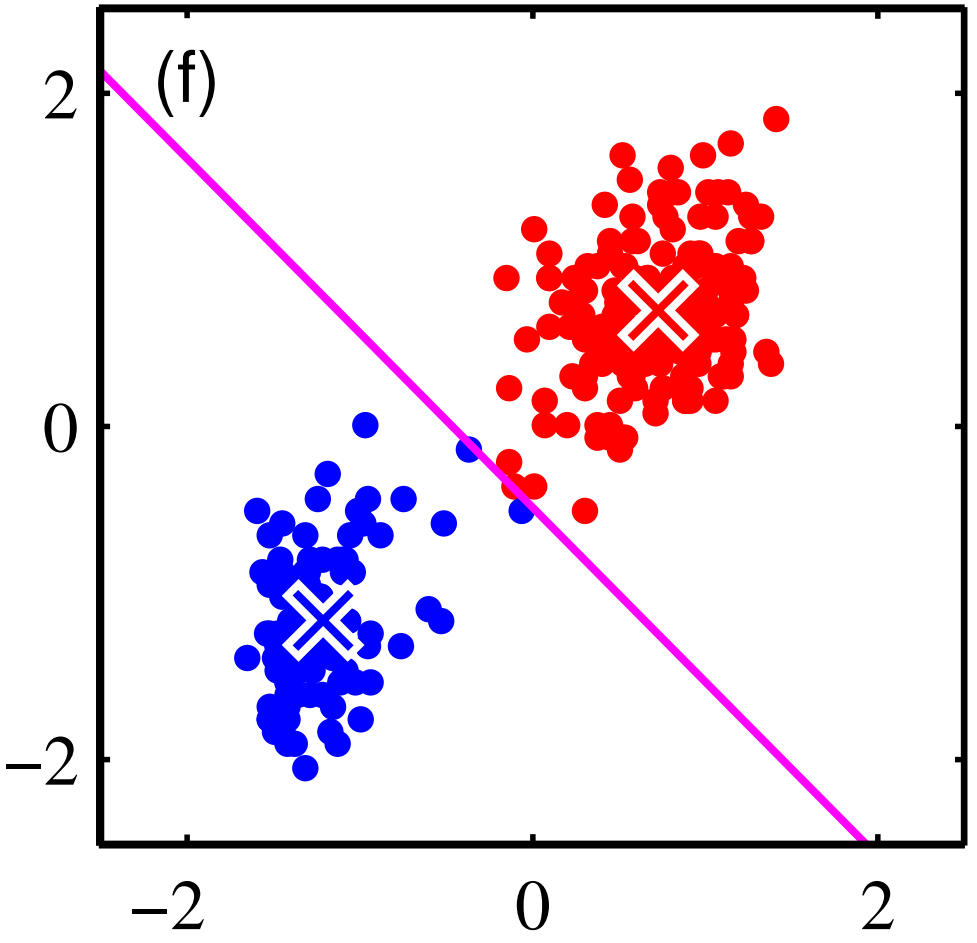


Iteration 2

Update the centroids

So we average all of the red points to get an update of the red centroid and average all of the blue points to get an update of the blue centroid. And we keep repeating this by assigning and then averaging.

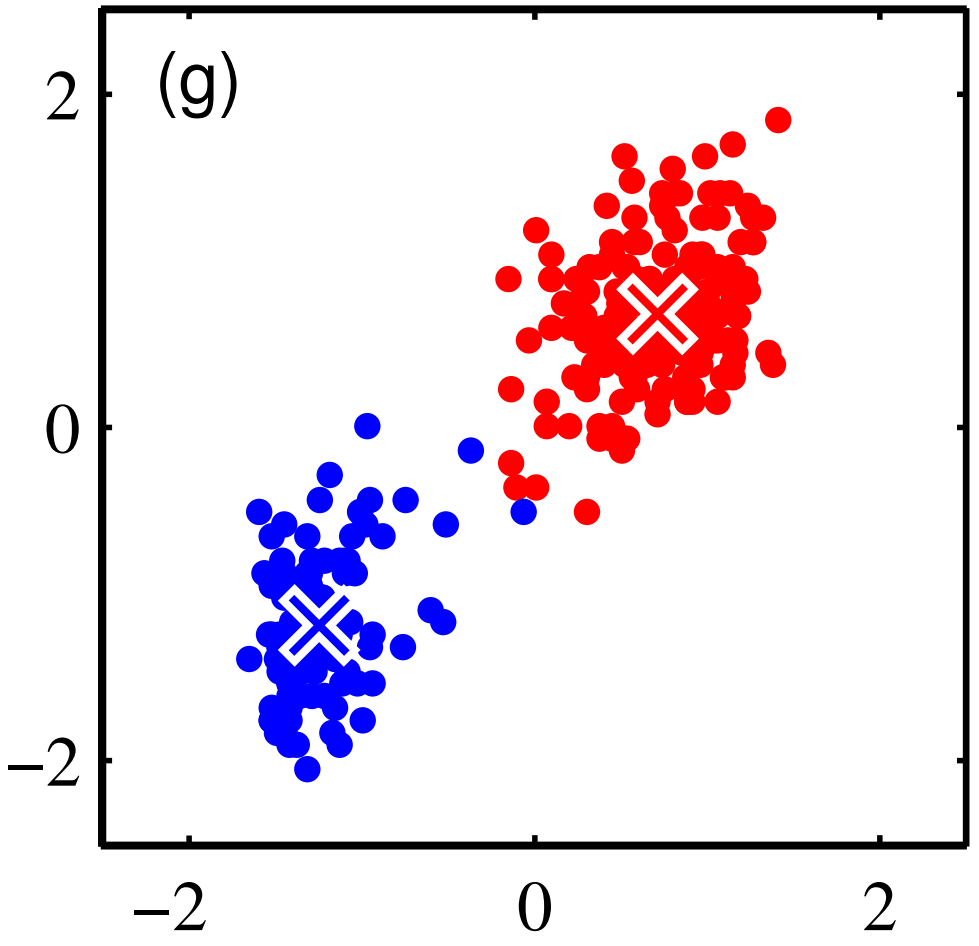
K-means algorithm: Example run



Iteration 3

Assign data to clusters

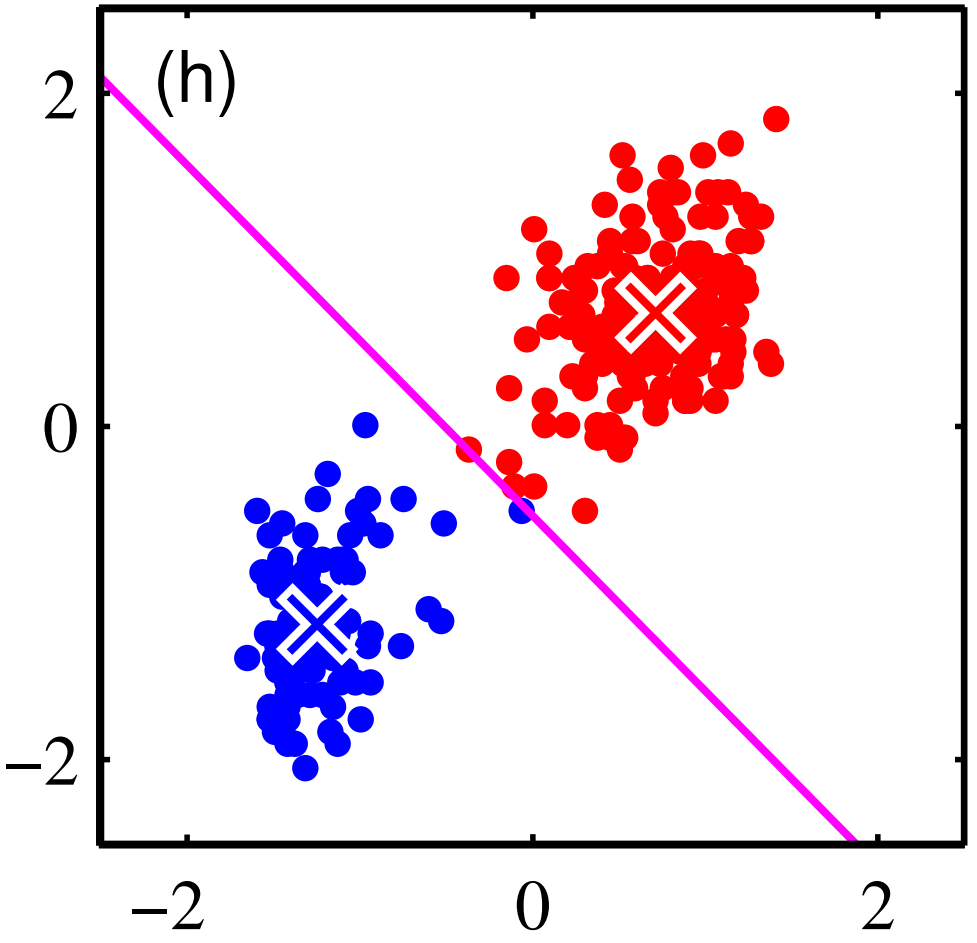
K-means algorithm: Example run



Iteration 3

Update the centroids

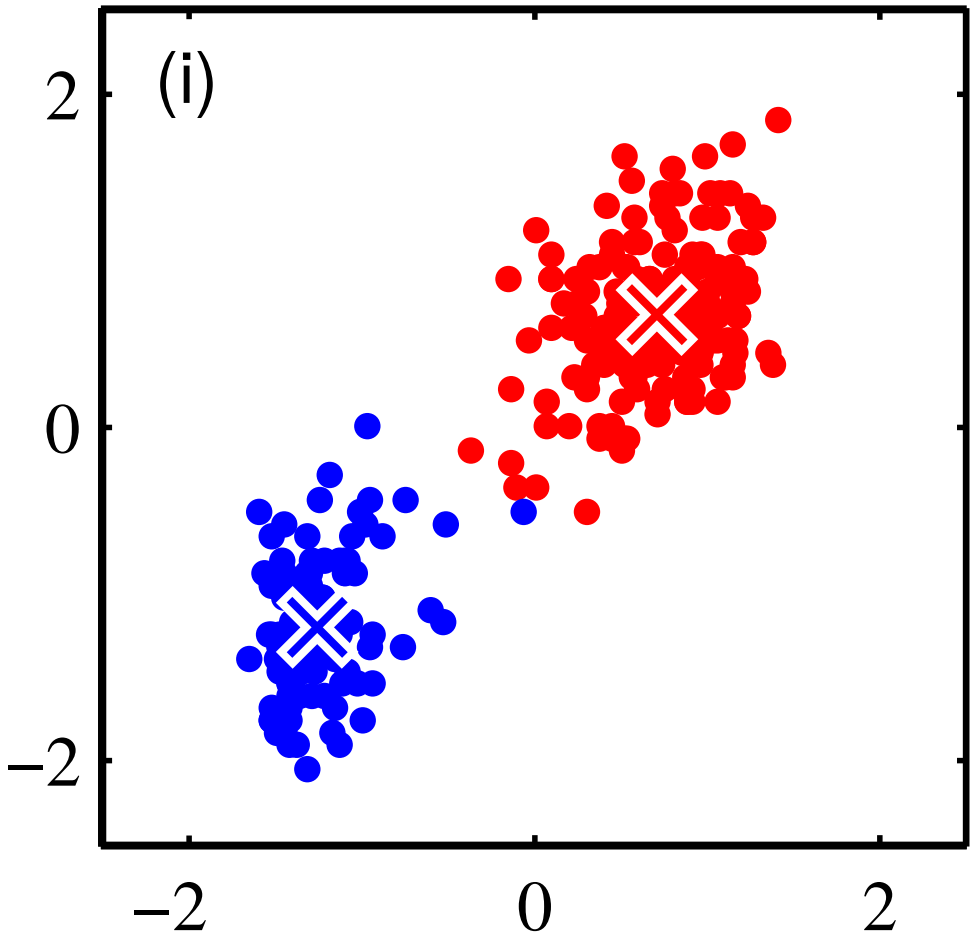
K-means algorithm: Example run



Iteration 4

Assign data to clusters

K-MEANS ALGORITHM: EXAMPLE RUN



Eventually we stop, the algorithm is now converged.

Iteration 4

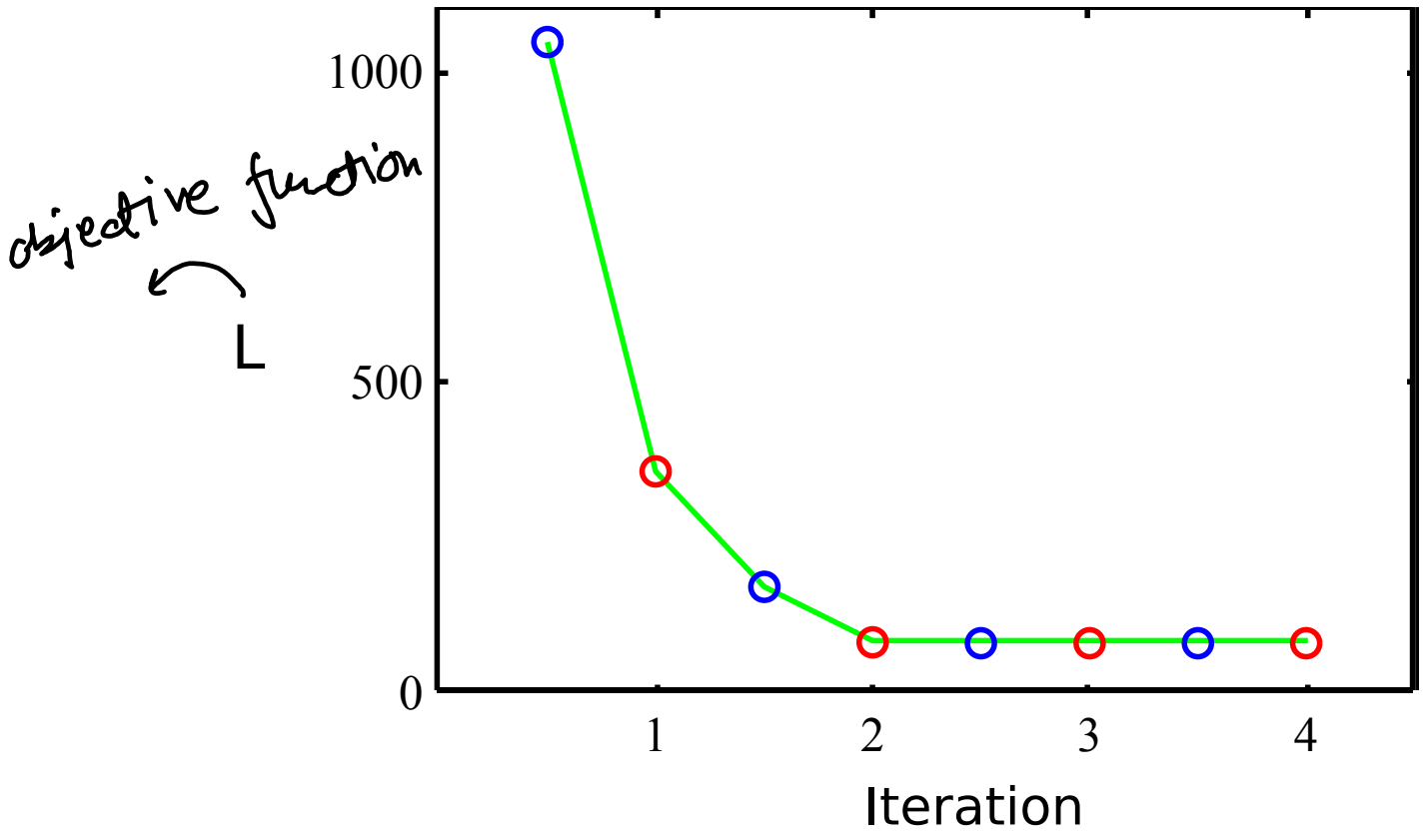
Update the centroids

And we also return for each data point, an indicator of which cluster it belonged to.

CONVERGENCE OF K-MEANS

blue dots → after updating c
red dots → after updating μ.

After each red dot we have completed 1 iteration.



Objective function after

- the "assignment" step (blue: corresponding to $\boldsymbol{c}$), and
- the "update" step (red: corresponding to $\boldsymbol{\mu}$).

CONVERGENCE OF K-MEANS

The outline of why this convergences is straightforward:

1. Every update to c_i or μ_k decreases $\mathcal{L}$ compared to the previous value. (for these parameters.)
2. Therefore, $\mathcal{L}$ is *monotonically decreasing*.*
3. $\mathcal{L} \geq 0$,** so Step 1 converges to some point (but probably not to 0).

One way to prove it has converged. Then μ is not going to change; if μ is not going to change then c's not going change.

When $\boldsymbol{c}$ stops changing, the algorithm has converged to a *local* optimal solution. This is a result of $\mathcal{L}$ not being convex. In c and μ. Maybe it's convex in one of them but not the other one. c & μ together converged to the local optimal sol^n not the global optimal.

Non-convexity means that different initializations will give different results. (final sol^{n}s, only locally optimal)

- Often the results will be similar in quality, but no guarantees.
- In practice, the algorithm can be run multiple times with different initializations. Then use the result with the lowest $\mathcal{L}$.

* Every update we make by the design of the algorithm, it's finding a new value of the parameter such that L is less than what it was previously.

** Also if we look at the objective It's bounded below by zero.
So, it can't be just by construction of the objective.
If you look at where sum in squared, squared distances and
so that's some has to be a positive number, it can't be a negative number.
So L has to converge at some point, but probably not to zero.

SELECTING K

(Another practical issue.)

We can't compare k means objective function across different settings of k. For the same setting of k, we can compare different runs of the k means objective function. But for different settings of k, we can compare different runs of the k means objective function.

We don't know how many clusters there are, but selecting K is tricky. The K-means objective function decreases as K increases,

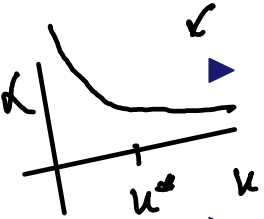
$$\mathcal{L} = \sum_{i=1}^{n} \sum_{k=1}^{K} \mathbb{1}\{c_i = k\} \|x_i - \mu_k\|^2.$$

For example, if $K = n$ then let $\mu_k = x_k$ and as a result $\mathcal{L} = 0$.

[Perfectly clustered all our data points, by putting each data point in it's own unique individual cluster]

Methods for choosing K include:

- Using advanced knowledge. e.g., if you want to split a set of tasks among K people, then you already know K.



- Looking at the *relative* decrease in $\mathcal{L}$. If K^* is best, then increasing K when $K \leq K^*$ should decrease $\mathcal{L}$ much more than when $K > K^*$. (Diminishing returns)
- Often the K-means result is part of a larger application. The main application may start to perform worse even though $\mathcal{L}$ is decreasing. (pre-processing step.)
- More advanced modeling techniques exist that address this issue. (Bayesian non-parametrics)

So for example if we want to take our continuous data and
we don't want to put a continuous model of it.
We don't want to learn a continuous model of it.
We maybe want to first discretize the data.
So we take each point, x_i, and instead of learning a continuous
distribution on that, we assign it to a cluster.
And now, each point is represented by an index between 1 and k.
So we've discretized the data, then we can learn a discrete model.
Often times, that's something that we'll wanna do.
In that case, that discrete model, is being used for its own application.
And so we can use whatever we're trying to use our larger application for.
That might have its own quality measures.
And we can then, for multiple discritizations, multiple values of k,
see how well we perform on some other task we're trying to do.

TWO APPLICATIONS OF K-MEANS

Lossy data compression

Approach: Vectorize 2×2 patches from an image (so data is $x \in \mathbb{R}^4$) and cluster them with K-means. Replace each patch with its assigned centroid.

(left) Original 1024×1024 image requiring 8 bits/pixel (1MB total)

(middle) Approximation using 200 clusters (requires 239KB storage)

(right) Approximation using 4 clusters (requires 62KB storage)

Data preprocessing (side comment)

K-means is also very useful for *discretizing* data as a preprocessing step. This allows us to recast a continuous-valued problem as a discrete one.

* We chop it up into 2 x 2 patches. (columns of width 2 and rows of width 2).
And then we treat each 2 by 2 patch as a 4 dimensional vector. So each 2 by 2 patch now becomes a data point in $\mathbb{R}^4$. And we have a bunch of these points that's what we're going to do k-means on.

We then take each 2 by 2 patch, we replace it in the image with the centroid that it was assigned to.

So in the original image, we have these four dimensional vectors that are vectorized versions of 2 by 2 patches, that can take almost any value. There are many different unique values.
Now if we look at this image and we look at any 2 by 2 patch, there are only 200 unique values that those patches can take corresponding to the 200 centroids that we learned.
, where we replace the patch with the centroid that it was assigned to as indicated by its corresponding value of c.

EXTENSIONS: K-MEDOIDS

Algorithm: K-medoids clustering

Input: Data $x_1, \ldots, x_n$ and distance measure $D(x, \mu)$. Randomly initialize μ.

- Iterate until $\boldsymbol{c}$ is no longer changing

1. For each c_i : Set

$$c_i = \arg\min_k \; D(x_i, \mu_k)$$

2. For each μ_k : Set

$$\mu_k = \arg\min_{\mu} \sum_{i:c_i=k} D(x_i, \mu)$$

Comment: Step #2 may require an algorithm.

K-medoids is a straightforward extension of K-means where the distance measure isn't the squared error. That is,

- K-means uses $D(x, \mu) = \|x - \mu\|^2$.
- Could set $D(x, \mu) = \|x - \mu\|_1$, which would be more robust to outliers.
- If $x \notin \mathbb{R}^d$, we could define $D(x, \mu)$ to be more complex.

So in that case we would have nested iterative algorithms.
And each iteration of the k medoids algorithm we would have to call an iterative algorithm to update each centroid m_1 k.

Problem:

However, if we wanted to make our algorithm more robust to outliers.
So we've seen some previous slides how if we have an outlier in our data set, and we calculate the mean.
That outlier can drag the centroid out quite a bit.

Solution:

If we want to define an algorithm that's going to be robust to these types of outliers, we might want to use the l 1 distance.

So it also can be used here to make this distance measure more robust.
So that outliers can't drag these centroids out towards them as much.

For example, if the data is a histogram of counts,
of the number of times a word appears in a document.
Then maybe we will want to define a more appropriate
distance measure that would take into account the support of a dataset.