Linear Unsupervised Learning

Instructor: Hui Guan

Slides adapted from: https://github.com/jermwatt/machine_learning_refined

Outline

- Background: Fixed Spanning Sets, Orthonormality, and Projections
- The Linear Autoencoder and Principal Component Analysis
- K-means Clustering

8.2 Fixed Spanning Sets, Orthonormality, and Projections

- The fundamental principles of unsupervised learning all lie in some very rudimentary ideas from linear algebra combined with the cost-function based approach to learning we have seen previously with supervised methods.
- This is because unsupervised learning is about effectively representing datasets with no output, i.e., ***only input data***.
- Our goal with such data is simply to effectively represent it, and this is a task that has roots in the notion of a *spanning set of vectors* from basic linear algebra.

- In this Section we review the fundamental concepts from linear algebra surrounding the notion of a spanning set of vectors.
- This includes: linear independence, learning proper weights for a fixed basis set representation, orthogonal bases and representation, and projections.
- Reviewing these topics up front will allow us to much more effectively tackle and intuit unsupervised learning techniques since they can be thought of as direct extensions.

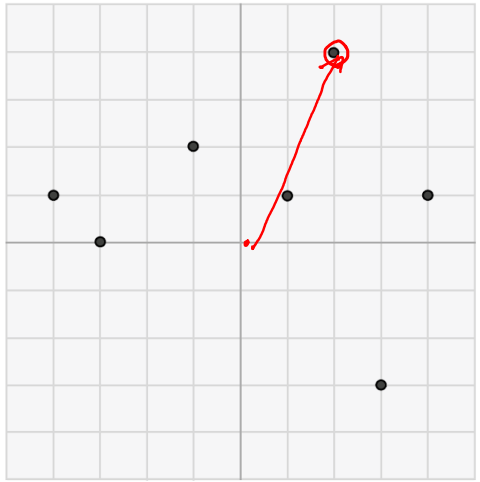
Perfectly representing data via a fixed spanning set

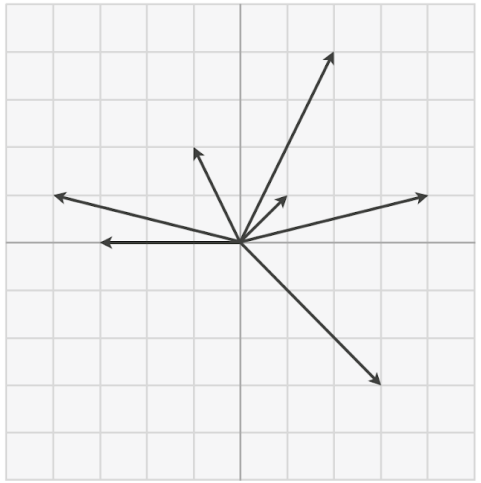
- Remember when thinking about data points in a multi-dimensional *vector space*, we can think of them simultaneously as 'dots' (as shown in the left panel) or as 'arrows' (as in the middle panel below).
- When thinking about *spanning sets of vectors* it is often helpful to visualize points in the same space using both of these conventions - i.e., some as 'dots' and others as 'arrows' as shown in the right panel.

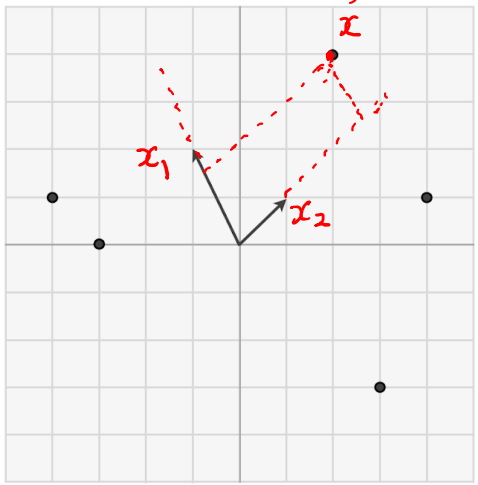
$\vec{x} = w_1\vec{x}_1 + w_2\vec{x}_2$

$\{\vec{x}_1, \vec{x}_2\}$ basis

2D







- Those vectors drawn as arrows are particular points - often called a *basis* or *spanning set of vectors* - over which we aim to efficiently represent every other point in the space.
- Those other vectors shown as dots are those points we wish to represent over our basis / spanning set.

$$\bar{x}_i = x_i - \frac{1}{P}\sum_{i=1}^{P} x_i$$

normalization / standardization $= \frac{x - \mu}{\sigma} \sim G(0,1)$

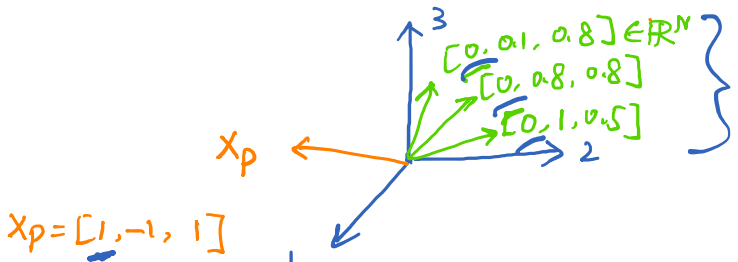
- Let's suppose our set of input points - our *dataset* - is written as $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_P\}$ and lives in N dimensions (that is, each point is N dimensional).
- Moreover throughout we will assume that our dataset has been *mean-centered* - a simple and completely reversible operation that involves subtracting off the mean of the dataset along each input dimension - so that it straddles the origin.
- This sort of normalization is almost always done in practice, and is computationally inexpensive to execute.

- In order for our basis / spanning set to be capable of perfectly representing all P of our points it too must live in the same N dimensional space.
- For a candidate basis / spanning set $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_N$ to be capable of perfectly representing such generic N dimensional (input) data means...
- ...that for each data point a set of weights exists so that, in particular linear combination, our basis set can match each data point as

vector $c_n \in \mathbb{R}^N$

$$\sum_{n=1}^{N} \mathbf{c}_n w_{n,p} = \mathbf{x}_p \in \mathbb{R}^N \qquad p = 1 \ldots P.$$

coefficient



- Technically speaking in order for this possibility to exist our spanning set must be *linearly independent*
- that is, the spanning vectors do not 'overlap', they point in completely different directions in the space.

$\vec{x}_p \in \mathbb{R}^N$: $\vec{c}_1 \cdots \vec{c}_N$ (2N)

$\vec{x}_p = \sum_{n=1}^{N} c_n \cdot w_{n,p}$

$\vec{x}_p = \sum_{n=1}^{2N} c_n \cdot w_{n,p}$

& at least N vectors are linearly independent

[given any set of N vectors, we can find $\{w_{np}\}_{n=1..N}$ that can represent $\vec{x}_p$.] False

As the simplest example imagine our spanning set was the set of N *standard basis vectors*. The n^{th} element of a standard basis takes the form of vectors that consist entirely of zeros, except for a 1 in its n^{th} slot

(n^{th} element of the standard basis) $\mathbf{c}_n = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$. n th

$$\vec{x}_p = \sum_{n=1}^{N} c_n \cdot x_{n,p}$$

$$x_{n,p} = \vec{x}_p[n-1]$$

- To represent a data point $\mathbf{x}_p$ over the standard basis is a trivial affair, and one can easily check that the perfect weights must be defined as

$$w_{n,\,p} = x_{n,\,p}$$

- i.e., each weight is simply equal to the value of the data point we aim to represent.
- For most any other spanning set, however these weights must be solved for numerically, which we frame in terms of a cost function minimization after the examples below.

- If indeed the spanning set of vectors is linearly independent, then having properly tuned the weights of point $\mathbf{x}_p$, denoted as the N length vector

$$\vec{\mathbf{w}}_p = \begin{bmatrix} w_{1,p} \\ w_{2,p} \\ \vdots \\ w_{N,p} \end{bmatrix}$$

$$\vec{x}_p = \sum_{n=1}^{N} c_n \cdot w_{n,p}$$

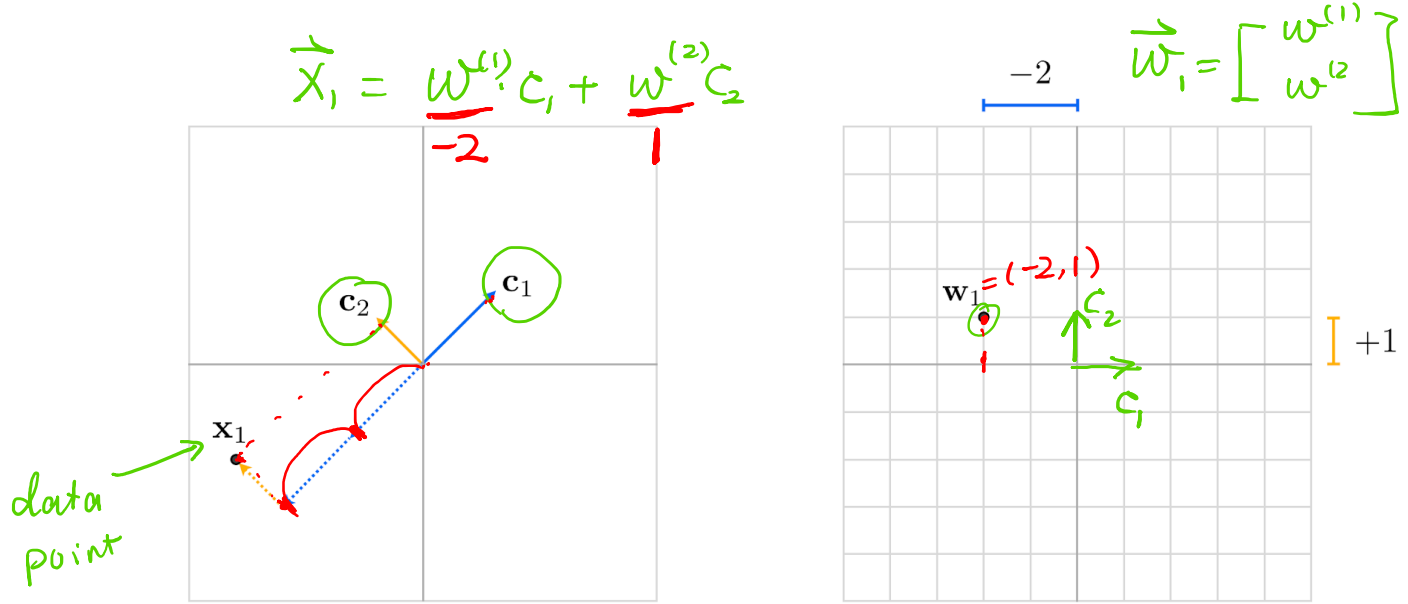
vector

- This weight set provides the new representation of $\mathbf{x}_p$ with respect to the spanning set.

$\vec{x}_p$ original $\quad c_n = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 8 \end{bmatrix} \Rightarrow \vec{w}_p \quad \vec{x}_p = \{c_n\} \begin{bmatrix} 1.0 \\ 2.0 \\ 3.3 \\ \vdots \end{bmatrix} \cdot \vec{w}_p$

basis

- For example, in the trivial case where our spanning set consists of the standard basis our representation or feature vector $\mathbf{w}_p = \mathbf{x}_p$ is the data point itself!
- Otherwise $\mathbf{w}_p$ - typically referred to as the new ***encoding*** of $\mathbf{x}_p$ in the *transformed feature space* whose coordinate axes are defined by the spanning set - is a vector that differs from original data point.



(left) A 2-d point, $\mathbf{x}_1$, shown in the space spanned by vectors $\mathbf{c}_1$ and $\mathbf{c}_2$. (right) Representation, also known as the *encoding*, of $\mathbf{x}_1$ in the transformed feature space.

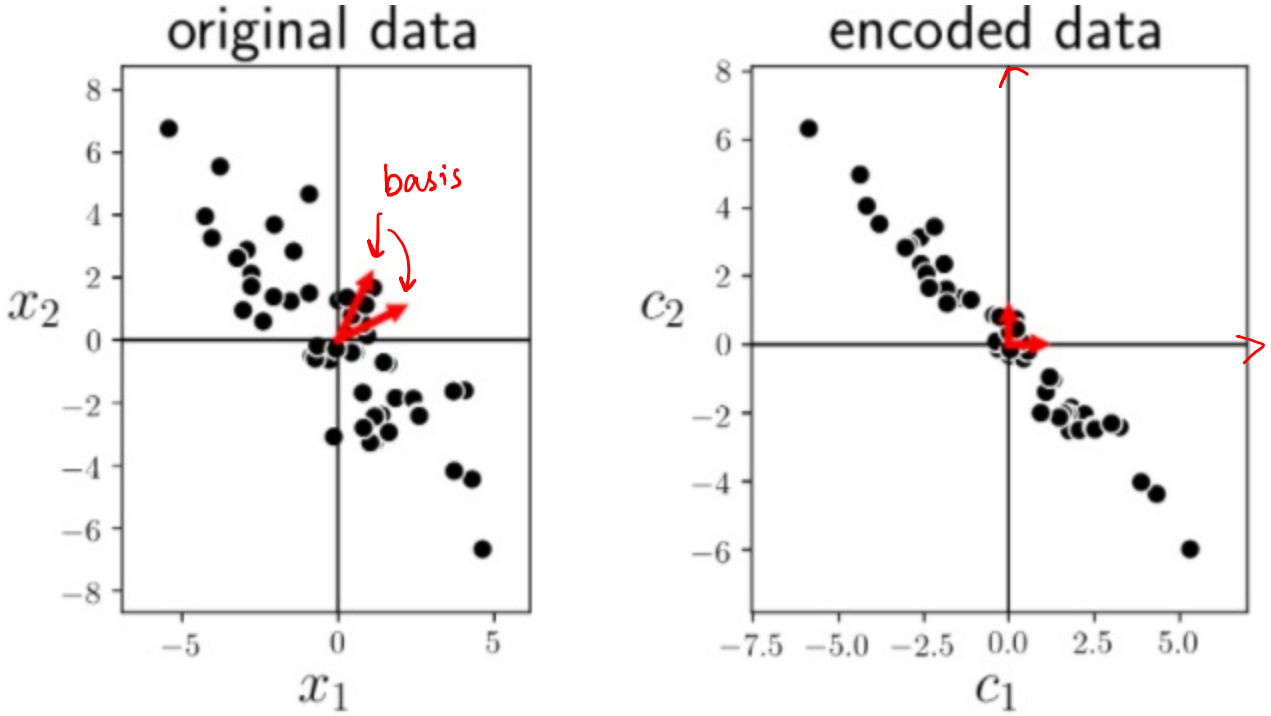
Perfect representation with complete spanning set in N=2 dimensions

- Given two spanning vectors which are *linearly independent*.

$$\mathbf{c}_1 = \begin{bmatrix} 2 \\ 1 \end{bmatrix} \qquad \mathbf{c}_2 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}.$$

- That is they do not overlap completely and point in different directions in the space, we can perfectly represent any point in the space using some linear combination of them as $w_{1,p}\mathbf{c}_1 + w_{2,p}\mathbf{c}_2 = \mathbf{x}_p$

- Here the weights $w_{1,p}$ and $w_{2,p}$ are unique to each point $\mathbf{x}_p$.


original data
basis
x_2
x_1
8
6
4
2
0
−2
−4
−6
−8
−5
0
5
encoded data
c_2
c_1
8
6
4
2
0
−2
−4
−6
−7.5
−5.0
−2.5
0.0
2.5
5.0

- One way we can solve for the proper weights is by setting up / minimizing an appropriate cost function.
- It is natural (as in our development of e.g., linear regression) to square the difference as

$$g(w_{p,n}) = \left(\sum_{n=1}^{N} \mathbf{c}_n w_{p,n} - \mathbf{x}_p\right)^2$$

known

- We then tune the corresponding weights to make this squared error as small as possible.

- So for each data point $\mathbf{x}_p$ we have a cost function

$$g\left(w_{1,p},\ldots,w_{N,p}\right)=\left(\sum_{n=1}^{N}\mathbf{c}_n w_{n,p}-\mathbf{x}_p\right)^2 \qquad p=1\ldots P$$

- When minimized appropriately provides us with a proper set of parameters for representing $\mathbf{x}_p$ over the spanning set.

- It is perhaps easiest to see this by writing the above cost function more compactly as

$$h\left(\mathbf{w}_p\right) = \|\mathbf{C}\mathbf{w}_p - \mathbf{x}_p\|_2^2.$$

- Here the $N \times N$ matrix $\mathbf{C}$ is formed by stacking the spanning set vectors column-wise, and the $N \times 1$ vectors $\mathbf{w}_p$ as

$$\mathbf{C} = \begin{bmatrix} | & | & \cdots & | \\ \mathbf{c}_1 & \mathbf{c}_2 & & \mathbf{c}_N \\ | & | & \cdots & | \end{bmatrix} \qquad \mathbf{w}_p = \begin{bmatrix} w_{p,1} \\ w_{p,2} \\ \vdots \\ w_{p,N} \end{bmatrix}$$

- Formally then our optimization problem to recover all of the proper weight vectors can be expressed as

$$\underset{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_P}{\text{minimize}} \ \frac{1}{P} \sum_{p=1}^{P} \left\| \mathbf{C}\mathbf{w}_p - \mathbf{x}_p \right\|_2^2.$$

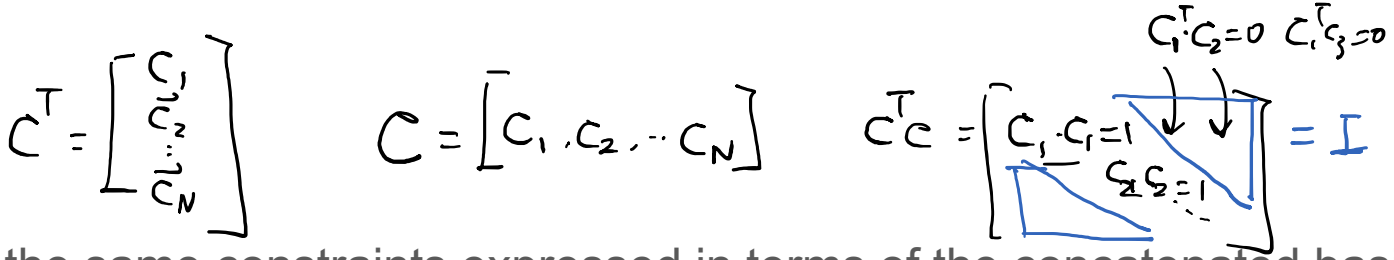
Perfect representation using a fixed orthonormal spanning set

A very special kind of basis / spanning set one often sees when representing data points is one whose elements are perpendicular and unit length.

Such a spanning set $\mathbf{c}_1, \mathbf{c}_2, \ldots \mathbf{c}_N$ is one that spans an entire space, but whose elements are perpendicular have length one.

Algebraically this is written as

$$\begin{cases} \mathbf{c}_n^T \mathbf{c}_m = 0 \\ \|\mathbf{c}_n\|_2^2 = 1 \quad \text{for } n = 1 \ldots N. \end{cases}$$



- Note that the same constraints expressed in terms of the concatenated basis matrix $\mathbf{C}$ can be written compactly as

$$\mathbf{C}^T\mathbf{C} = \mathbf{I}_{N\times N}.$$

- Often such a spanning sets are likewise referred to as *orthonormal* or - sometimes - *orthogonal*
- Technically speaking the latter term should only be applied when a set of vectors consists of perpendicular elements that are not necessarily unit-length, it is unfortunately often used to refer to unit-length perpendicular vectors as well.

Imperfectly representing data using a fixed spanning set

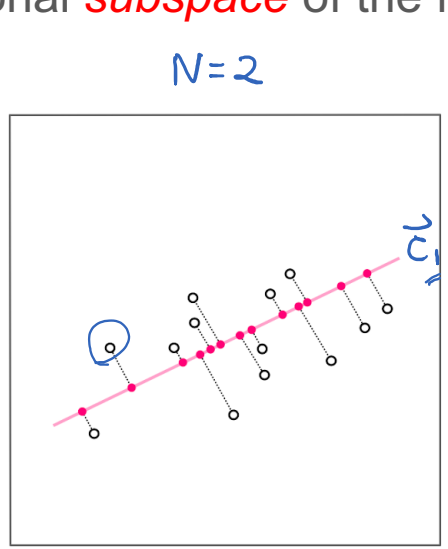
- Above we reviewed two important facts that we require for a spanning set / basis in order for it to be capable of perfectly representing points in a generic N dimensional space:
- The set of vectors are ***linearly independent***, that is they point in different directions in the space
- **There are at least N spanning vectors**

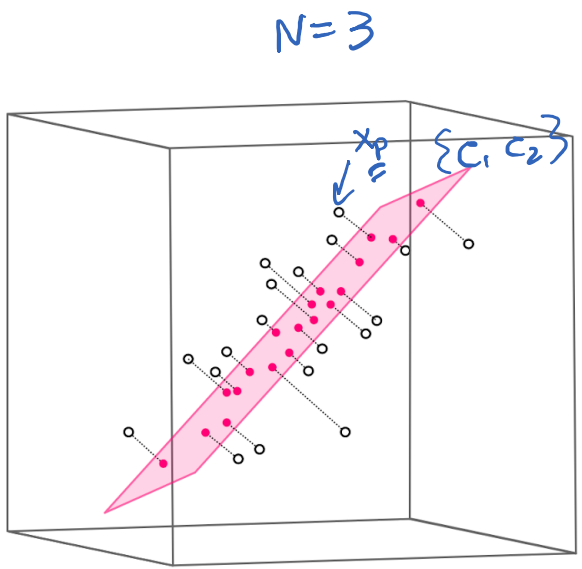
basis

$x_p \in \mathbb{R}^N$ $\quad \vec{C} = [\vec{c}_1 \dots \vec{c}_K]$

- What happens when we relax the second condition and suppose we more generally have $K \leq N$ spanning vectors / basis elements?
- Unsurprisingly we can no longer perfectly represent a generic set of P points in the space.
- In N dimensions this set of K vectors can - at best - span a K dimensional subspace.

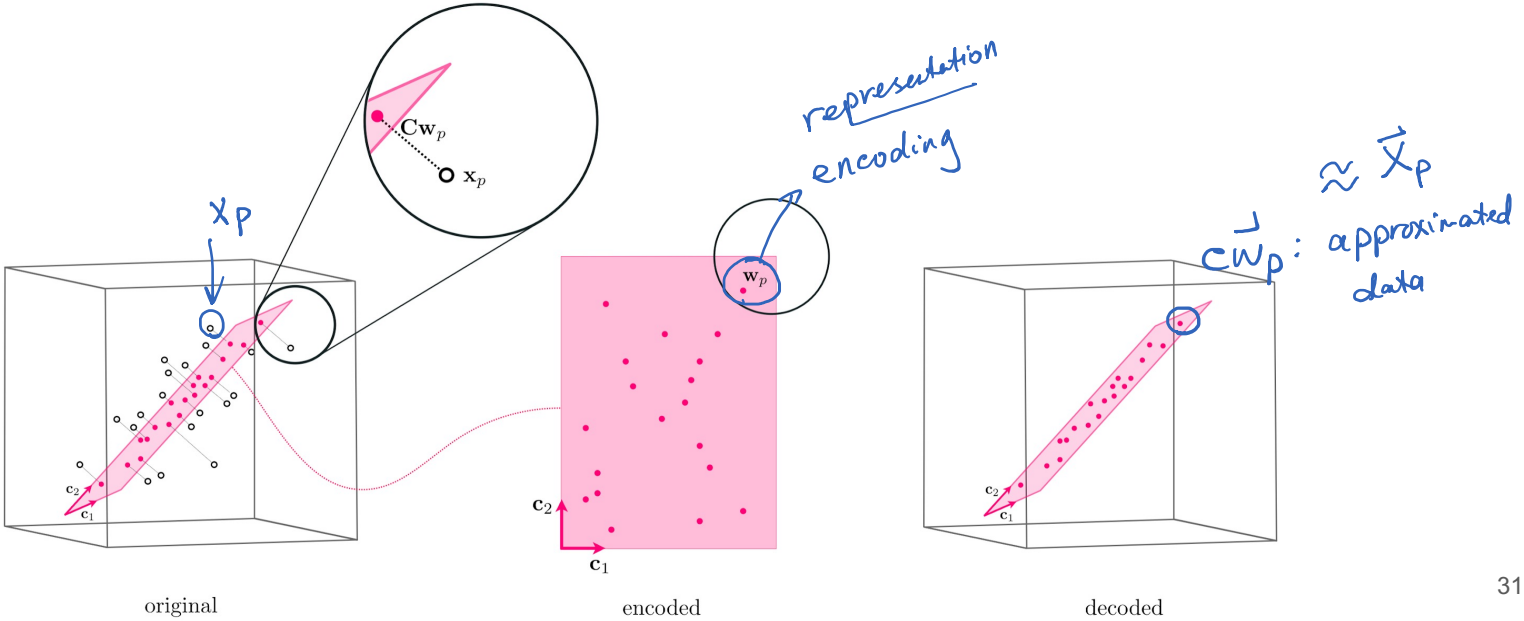
- For example as illustrated in the figure below, if N=3 the best any set of K=2 spanning vectors can do is span a hyperplane.
- A set consisting of a single K = 1 spanning vector can only span a line.
- In both instances we clearly will not be able to perfectly represent all possible points in a space, since our reach in each case is restricted to a lower dimensional *subspace* of the full space.





$\vec{x}_p \approx C\vec{w}_p$

- The representation over the subspace spanned by $\mathbf{C}$ - is $\mathbf{C}\mathbf{w}_p$
- This is called the *projection*
- The weight vector $\mathbf{w}_p$ is the *encoding* of a point $\mathbf{x}_p$ in the (sub)space spanned by K spanning vectors $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_K$



$C^T C = I_{N \times N}$

Imperfect representation using a fixed orthonormal spanning set

$C = [\vec{c}_1, \vec{c}_2 \dots \vec{c}_N] \Rightarrow \vec{x}_p = C \cdot \vec{w}_p$ + C orthonormal $\Rightarrow C^T \cdot \vec{x}_p = C^T \cdot C \cdot \vec{w}_p$

basis

encoding

$C^T \vec{x}_p = I \cdot \vec{w}_p$

- If our spanning set / basis of K elements is orthonormal, each weight vector encoding $\mathbf{w}_p$ is:

another basis

$$\mathbf{w}_p = \mathbf{C}^T \mathbf{x}_p \qquad p = 1 \dots P$$

encoding of x_p

encoding

- since we will have $\mathbf{C}^T \mathbf{C} = \mathbf{I}_{K \times K}$.

"decoding" $w_p \rightarrow x_p$: $\vec{x}_p = C \cdot \vec{w}_p$

if $C \in \mathbb{R}^{N \times N}$, orthonormal

$$\min \; g(C) = \| C C^T x_p - x_p \|_2$$

PCA:

$C \in \mathbb{R}^{K \times K}$

orthonormal

$C = [C_1, C_2, \cdots C_K]$

point to largest variance

- Our representation is given simply in terms of the incomplete spanning set and datapoints as

$$\mathbf{C}\,\mathbf{C}^T\mathbf{x}_p \approx \mathbf{x}_p \qquad p = 1 \ldots P.$$

- The above formula is called autoencoder formula.

encode decode ~

$x_p \rightarrow w_p \rightarrow \tilde{x}_p$

- In this set of lectures, **our aim is to *learn* the best basis for a given set of data the assumption of orthogonality**.

C:

image → encode → $\mathbb{R}^d$ → decode → ~image

8.3 The Linear Autoencoder and Principal Component Analysis

- The most fundamental unsupervised learning method is known as Principal Component Analysis or PCA for short.

W

- Learn the proper weights to best represent input data over a given fixed spanning set
- Learn a proper spanning set as well.

C

$$g(C, W) = \|C \cdot C^T x - x\|_2^2$$

$$\text{s.t. } W = C^T X$$

$$C^T C = I$$

Learning proper spanning sets

- Given a fixed basis or spanning set of $K \leq N$ vectors $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_K$ - we can represent a set of P mean-centered points $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_P$ as

$$\sum_{n=1}^{K} \mathbf{c}_n w_{n,p} \approx \mathbf{x}_p \qquad\qquad p = 1 \ldots P$$

- Here we tune the weights $w_{p,n}$ by minimizing a Least Squares cost function

$$g\left(\mathbf{w}_1, \ldots, \mathbf{w}_P\right) = \frac{1}{P} \sum_{p=1}^{P} \left\| \mathbf{C}\mathbf{w}_p - \mathbf{x}_p \right\|_2^2.$$

- In general when we used a full set of *linearly independent* K = N fixed basis vectors, we can learn corresponding weights that drive this cost function to zero, and give us strict equality in the desired equation.

- **Here we will *learn* a proper basis (along with the weights).**
- This slight thematic twist - where we *learn* the basis along with the proper weights - is called *Principal Component Analysis*.

$$g\left(\mathbf{w}_1, \ldots, \mathbf{w}_P, \mathbf{C}\right) = \frac{1}{P}\sum_{p=1}^{P}\left\|\mathbf{C}\mathbf{w}_p - \mathbf{x}_p\right\|_2^2.$$

basis

- This cost function can be properly minimized using any number of standard approaches like e.g., gradient descent or coordinate / block-coordinate descent.

- if our ***K*** spanning vectors concatenated column-wise to form the spanning matrix $\mathbf{C}$ are **orthonormal**, then the encoding of each $\mathbf{x}_p$ may be written simply as $\mathbf{w}_p = \mathbf{C}^T\mathbf{x}_p$
- If we plug in this simple solution for $\mathbf{w}_p$ into the pth summand of the Least Squares cost above, we get a cost that is a function of $\mathbf{C}$

$$g(\mathbf{C}) = \frac{1}{P}\sum_{p=1}^{P}\left\|\mathbf{C}\,\mathbf{C}^T\mathbf{x}_p - \mathbf{x}_p\right\|_2^2.$$

w_p : encoding

decode

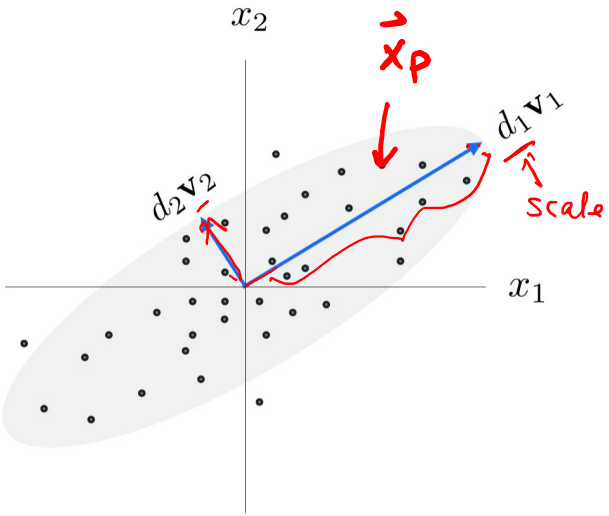
It is often referred to as the linear Autoencoder

Principal Component Analysis

$C = [\vec{c}_1, \vec{c}_2, \ldots \vec{c}_K]$ $\vec{c}_K$: pricipal component

- The linear Autoencoder cost may have many minimizers, of which the set of principal components is a particularly important one.
- The spanning set of principal components always provide a consistent skeleton for a dataset, with its members pointing in the dataset's ***largest directions of orthogonal variance***.
- Employing this particular solution to the linear Autoencoder is often referred to as Principal Component Analysis, or PCA for short, in practice.

- A scaled version of the first principal component of this dataset points in the direction in which the dataset is most spread out, also called its largest direction of variance.
- A scaled version of the second principal component points in the next most important direction in which the dataset is spread out that is orthogonal to the first.

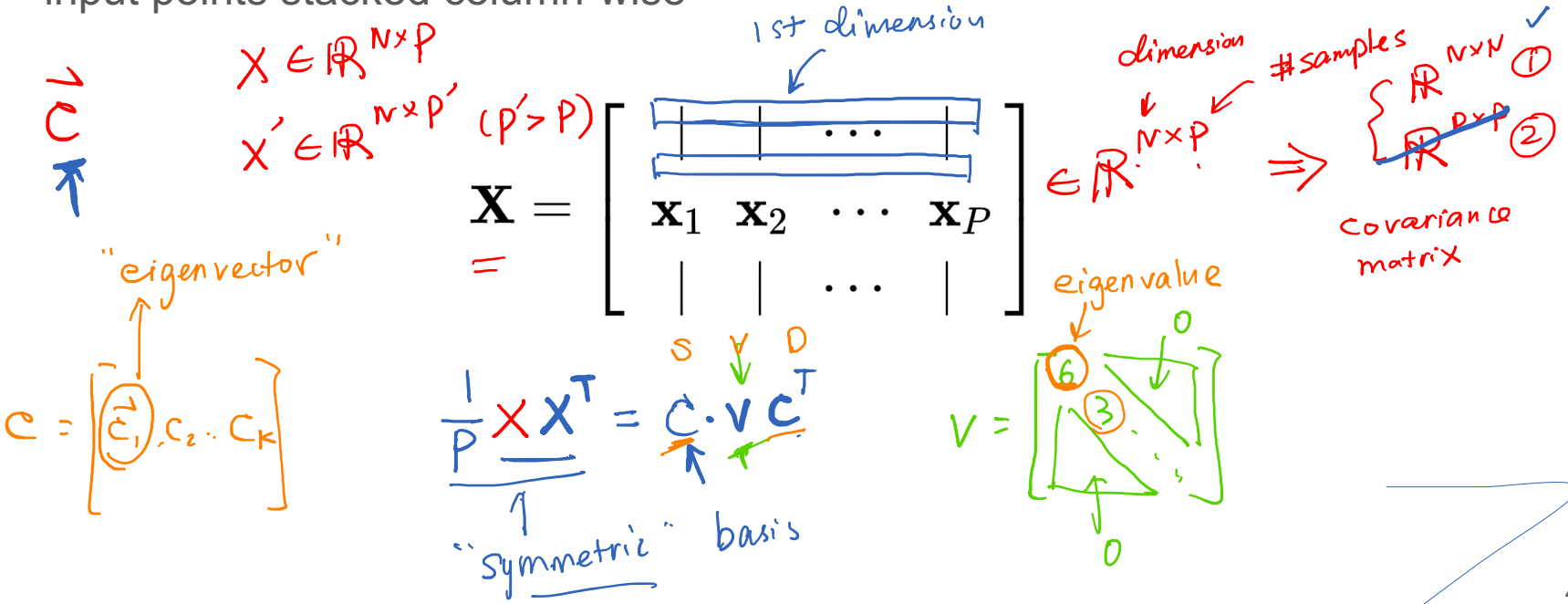


N=2

$\vec{C} = [\vec{v}_1 \; \vec{v}_2] \in \mathbb{R}^{2\times 2}$

"basis"

- This special orthonormal minimizer of the linear Autoencoder is given by the eigenvectors of the so-called covariance matrix of the data.
- Denoting by $\mathbf{X}$ the $N \times P$ data matrix consisting of our P mean-centered input points stacked column-wise



- The orthogonal basis provided by this special solution (called the *Principal Components* of a dataset) can be computed (as a minimum of the Autoencoder cost function) as the *eigenvectors* of the corresponding correlation matrix of this data

$$\text{covariance matrix of } \mathbf{X} := \frac{1}{P}\mathbf{X}\,\mathbf{X}^T$$

- Denoting the eigenvector/value decomposition of the covariance matrix $\frac{1}{P}\mathbf{X}\,\mathbf{X}^T$ is given as

$$\frac{1}{P}\mathbf{X}\,\mathbf{X}^T = \mathbf{V}\,\mathbf{D}\,\mathbf{V}^T$$

- then above the orthonormal basis we recover is given precisely by the eigenvectors above, i.e., $\mathbf{C} = \mathbf{V}$
- Again, these are referred to in the jargon of machine learning as the *principal components* of the data.
- Moreover, the variance in each (principal component) direction is given precisely by the corresponding eigenvalue in $\mathbf{D}$.

Python Implementation of PCA

- *First we center the data*

```
def center(X):
    '''
    A function for normalizing each feaure dimension of an input array, mean-centering
    and division by its standard deviation

    '''
    X_means = np.mean(X,axis=1)[:,np.newaxis]
    X_normalized = X - X_means

    return X_normalized
```

- *Next, we compute the principal components of the mean-centered data.*

```
def compute_pcs(X,lam):
    '''
    A function for computing the principal components of an input data matrix.  Both
    principal components and variance parameters (eigenvectors and eigenvalues of XX^T)
    are returned
    '''
    # create the correlation matrix
    P = float(X.shape[1])
    Cov = 1/P*np.dot(X,X.T) + lam*np.eye(X.shape[0])

    # use numpy function to compute eigenvalues / vectors of correlation matrix
    D,V = np.linalg.eigh(Cov)
    return D,V
```

numerical stability

$\lambda \cdot I$

eigen value

C

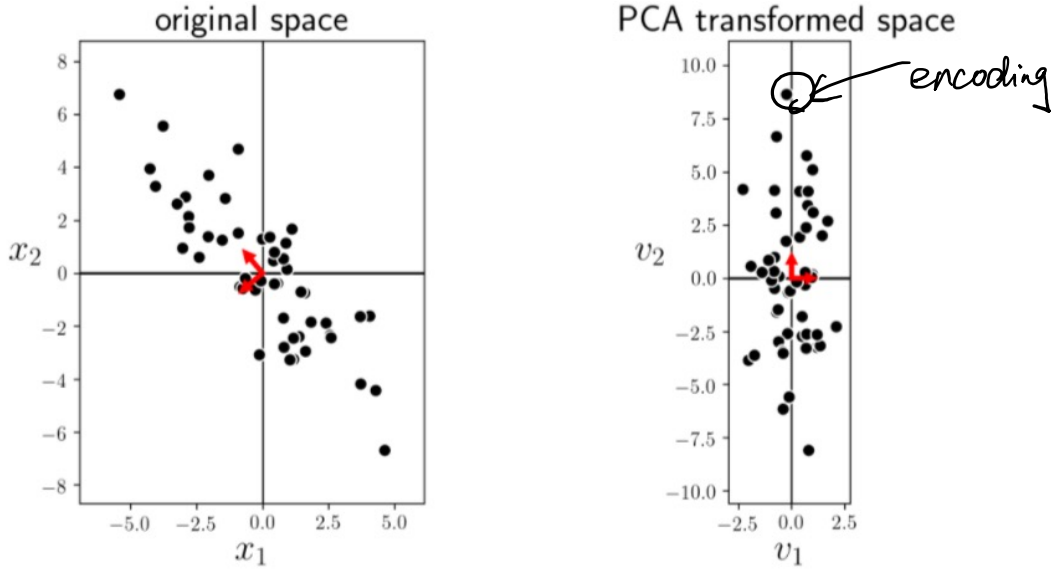
- *Put it together*

```
def pca_transform_data(X,**kwargs):
    '''
    A function for producing the full PCA transformation on an input dataset X.
    '''
    # user-determined number of principal components to keep, and regularizer penalty param
    num_components = X.shape[0]
    if 'num_components' in kwargs:
        num_components = kwargs['num_components']
    lam = 10**(-7)
    if 'lam' in kwargs:
        lam = kwargs['lam']

    # compute principal components
    D,V = compute_pcs(X,lam)
    V = V[:,-num_components:]
    D = D[-num_components:]

    # compute transformed data for PC space: V^T X
    W = np.dot(V.T,X)
    return W,V
```

- In the left panel we show the mean-centered data along with its two principal components (pointing in the two orthogonal directions of greatest variance in the dataset) shown as red arrows.
- In the right panel we show the encoded version of the data in a space where the principal components are in line with the coordinate axes.



Recap

$\vec{x}_p \in \mathbb{R}^N$ P: sampler $\Rightarrow$ $X \in \mathbb{R}^{N \times P}$

usage: ① dimension reduction $x^p \in \mathbb{R}^N \Rightarrow w^p \in \mathbb{R}^K$, $K < N$

$f(x^{pT}\theta) = \hat{y}$

② $k = 2, 3$

③ fill in missing values

basis / a spanning set of vectors: $\{\vec{c}_n\}_{n=1,\dots,N}$, $\vec{c}_n \in \mathbb{R}^N$ $\Rightarrow$ $C \in \mathbb{R}^{N \times N}$

linearly independent $\Rightarrow$ span a space, N dimension

encoding: $\vec{x}_p = C \cdot \vec{w}_p$

$\vec{w}_p$: representation {encoding

$C \in \mathbb{R}^{N \times N}$: feature space; $\vec{w}_p \in \mathbb{R}^N$: transformed feature space

Orthonormal: $\vec{c}_i^T \vec{c}_j = 0$ if $i \neq j$, $i, j \in \{1 \dots N\}$ $\Rightarrow$ $C^T C = I_{N \times N}$

decoding (C is orthonormal): $\vec{w}_p = C^T \vec{x}_p$

$(\vec{x}_p = C \cdot \vec{w}_p) \Rightarrow C^T \vec{x}_p = C^T C \vec{w}_p \Rightarrow C^T \vec{x}_p = \vec{w}_p$ (C^T, C^T; $C^T C = I_{N \times N}$)

Autoencoder: $C(C^T \vec{x}_p) = \vec{x}_p$

$C \in \mathbb{R}^{N \times K}$, $K \leq N$ $\Rightarrow$ $C(C^T \vec{x}_p) \approx \vec{x}_p$

PCA: $C^*, \{\vec{w}_p^*\} = \arg\min \sum_p \| C\vec{w}_p - \vec{x}_p \|_2^2$ s.t. $C^T C = I$

$X \in \mathbb{R}^{N \times P}$

① $X^T X \in \mathbb{R}^{P \times P}$

② $X X^T \in \mathbb{R}^{N \times N}$

Numerical Solution

$\frac{1}{P} X X^T = V D V^T$

$C^* = V$ eigenvector

D: eigenvalue

True/False Statement

Given the above context, give me 10 True/False statements.
Report whether the statement is True or False in the following format: statement - [True/False].
Explain the reason why a statement is correct or wrong .

- PCA, as an unsupervised learning technique, can be used to reduce the dimension of a dataset by reducing the number of inputs.
- Spanning vectors that match the data dimension and are linearly independent can perfectly represent all data points.
- An orthonormal basis consists of vectors that have varying lengths and are orthogonal to each other.
- The principal components point in random directions in a dataset.
- Using PCA for dimension reduction always results in better classification results
- Principal components are derived from the eigendecomposition of a dataset's covariance matrix.

$C^* \in \mathbb{R}^{N \times K}$: principal components = eigen vectors

basis = { }$_{1 \cdots K}$

$\frac{1}{p} X X^T = V D C^T$

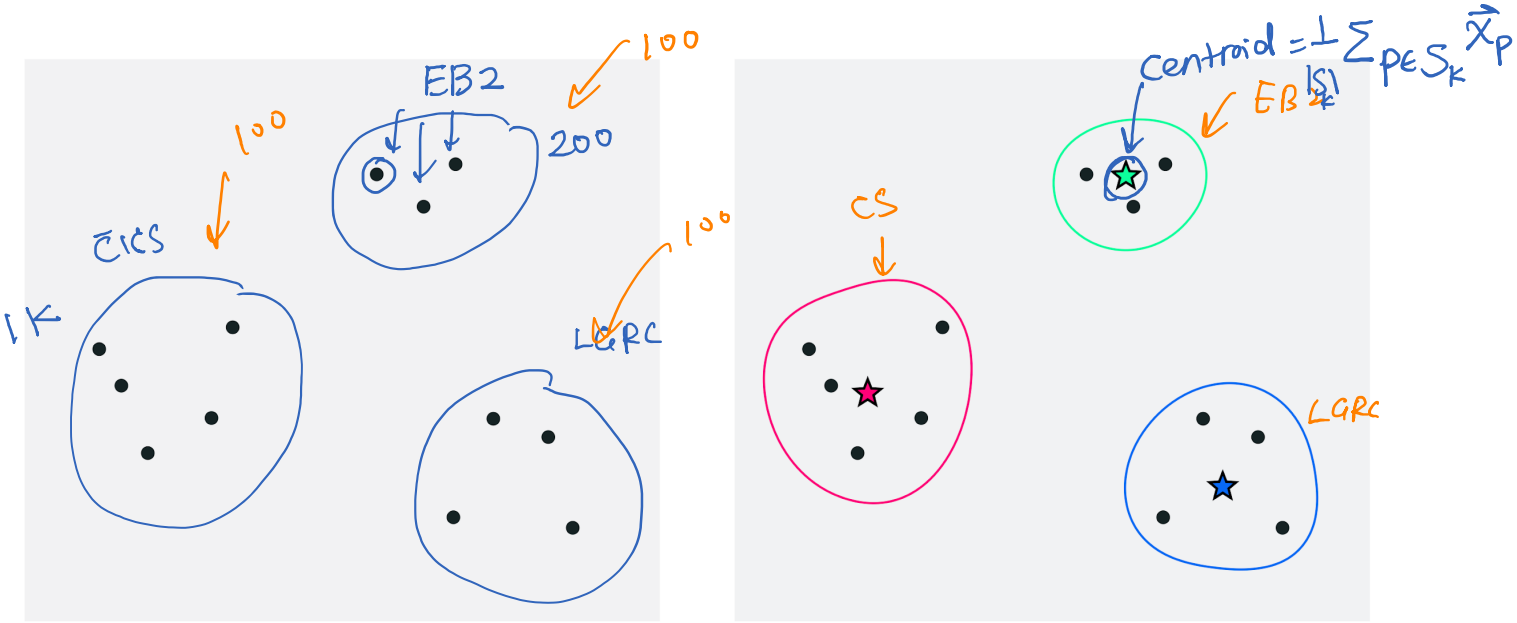
$D = \begin{bmatrix} 2 & & & \\ & 1.2 & & \\ & & 0.8 & \\ & & & \cdots \\ & & & 0 \end{bmatrix}$ close to zero

- PCA, as an unsupervised learning technique, can be used to reduce the dimension of a dataset by reducing the number of inputs. – [FLASE]
    - Explanation: Should be the number of features instead of inputs.
- Spanning vectors that match the data dimension and are linearly independent can perfectly represent all data points. – [TRUE]
- An orthonormal basis consists of vectors that have varying lengths and are orthogonal to each other. – [FALSE]
    - **Correct Statement:** An orthonormal basis consists of vectors that have unit length and are orthogonal to each other.
- The principal components point in random directions in a dataset. - [False]
    - **Correct Statement:** The principal components point in the dataset's largest directions of orthogonal variance.

- Using PCA for dimension reduction always results in better classification results. - [False]
  - Correct Statement: PCA can potentially destroy inter-class separation, so it doesn't always guarantee better classification results.
- Principal components are derived from the eigendecomposition of a dataset's covariance matrix. - [True]

8.5 K-means Clustering

- The focus of this Section - the *K-means algorithm* - is an elementary example of another set of unsupervised learning methods called *clustering algorithms*.
- These algorithms are designed to (properly) reduce the number of points in a dataset, which we refer to as the *data dimension* of a dataset...
- ...and in doing so help us understand the structure of our data.



(left) A 2-dimensional toy dataset with P=10 data points.

(right) The data shown naturally clustered into K=3 clusters. Each cluster center - also called a centroid - is marked by a star symbol colored to match its cluster boundary.

$$S_k = \{p \mid \text{if } \mathbf{x}_p \text{ belongs to the } k^{th} \text{ cluster}\}.$$

- Algebraically, each centroid is represented as:

$$\mathbf{c}_k = \frac{1}{|S_k|} \sum_{p \in S_k} \mathbf{x}_p.$$

- This formula confirms the intuition that each centroid represents a chunk of the data - the average of those points belonging to each cluster.
- Next we can state mathematically an obvious and implicit fact about the clustering scenario above: that each point belongs to the cluster whose centroid it is closest to.

- To express this algebraically for a given point $\mathbf{x}_p$ is simply say that the point must belong to the cluster where the distance to the centroid $\|\mathbf{x}_p - \mathbf{c}_k\|_2$ is minimal.
- In other words, the point $\mathbf{x}_p$ belongs to or is *assigned* to cluster $k^\star$ if

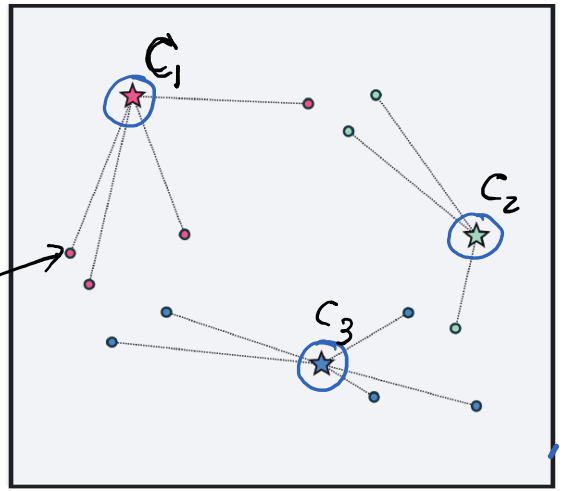
$$a_p = \underset{k=1,\ldots,K}{\operatorname{argmin}} \|\mathbf{x}_p - \mathbf{c}_k\|_2$$

- In the jargon of machine learning these are called *cluster assignments*.

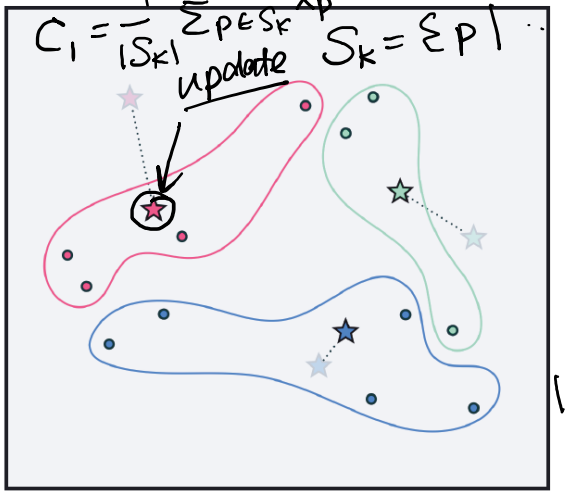
- We do not want to have to visually identify clusters in a dataset ourselves...
- ...and in any being constrained three dimensions if the dimension of of a dataset is greater than N=3 there is no way for us humans to do this anyway.
- Instead we want an algorithm that will do this for us automatically.

- Thankfully we can do this rather easily using the framework detailed above for mathematically describing clusters...
- ...the resulting algorithm being called the *K-means clustering algorithm*.
- As with many of the algorithms we have seen K-means is an *iterative method*, meaning that we will refine the ideal location for our cluster centroids / cluster assignments over a number of update steps.

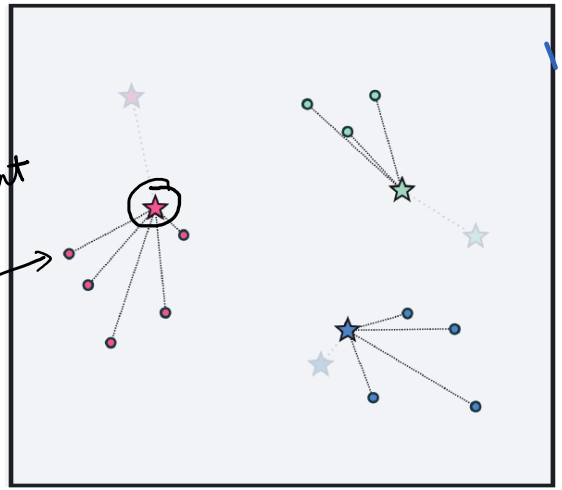
$\min_k \| x_p - c_k \|_2$

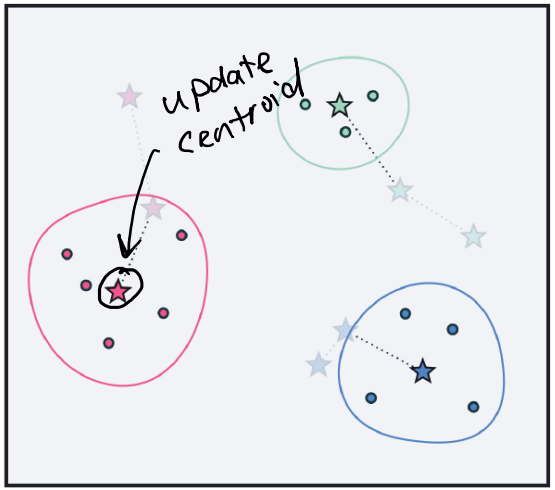


$c_1 = \frac{1}{|S_k|} \sum_{p \in S_k} x_p \quad S_k = \{ p \mid \ldots \quad k \}$



① $\| C^{(t-1)} - C^{(t)} \| < \text{abs}$





② $t < 100$

The K-means algorithm

input: dataset $\mathbf{x}_1, \ldots, \mathbf{x}_P$, initializations for centroids $\mathbf{c}_1, \ldots, \mathbf{c}_K$, and maximum number of iterations J

`for` $j = 1, \ldots, J$

    # Update cluster assignments

    for $p = 1, \ldots, P$

        $a_p = \underset{k=1,\ldots,K}{\operatorname{argmin}} \|\mathbf{c}_k - \mathbf{x}_p\|_2$

    end for

    # Update centroid locations

    for $k = 1, \ldots, K$

        denote S_k the index set of points $\mathbf{x}_p$ currently assigned to the k^{th} cluster

        update $\mathbf{c}_k$ via $\mathbf{c}_k = \frac{1}{|S_k|} \sum_{p \in S_k} \mathbf{x}_p$

    end for

end for

Update cluster assignments using final centroids

for $p = 1, \ldots, P$

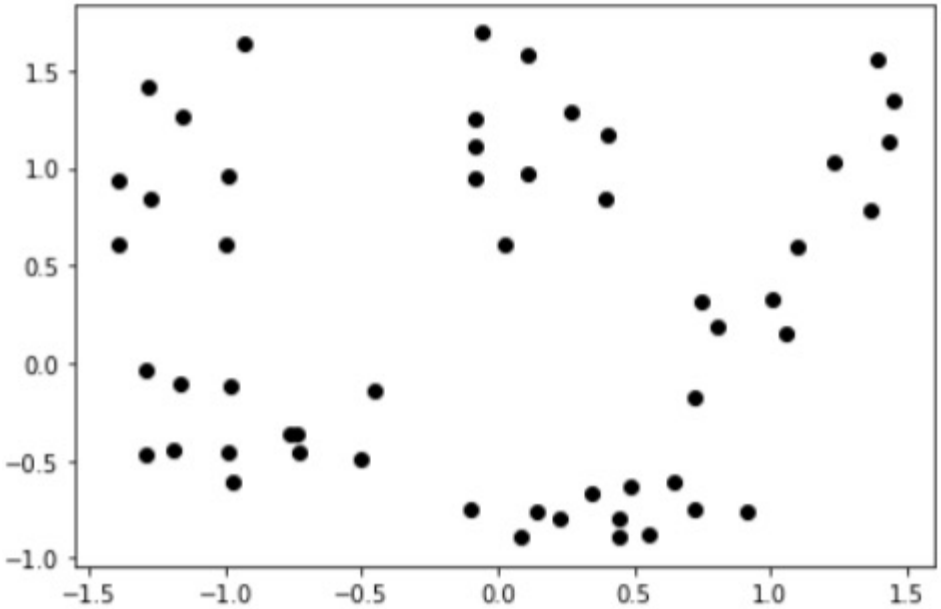
    $a_p = \underset{k=1,\ldots,K}{\operatorname{argmin}} \|\mathbf{c}_k - \mathbf{x}_p\|_2$

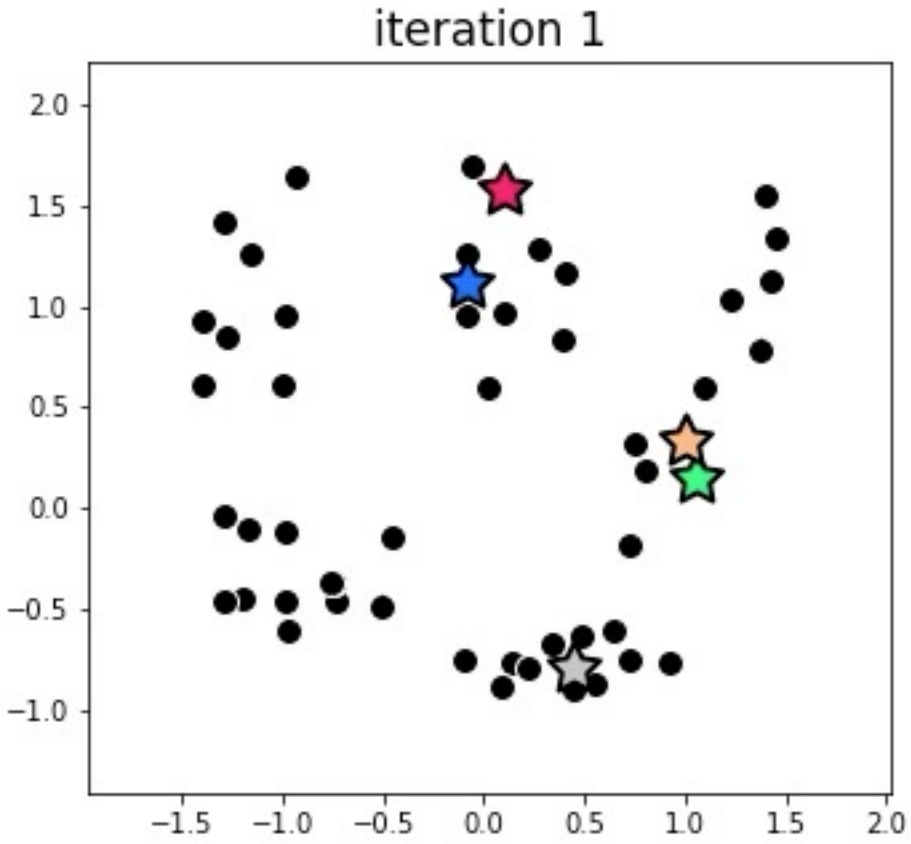
end for

output: optimal centroids and assignments

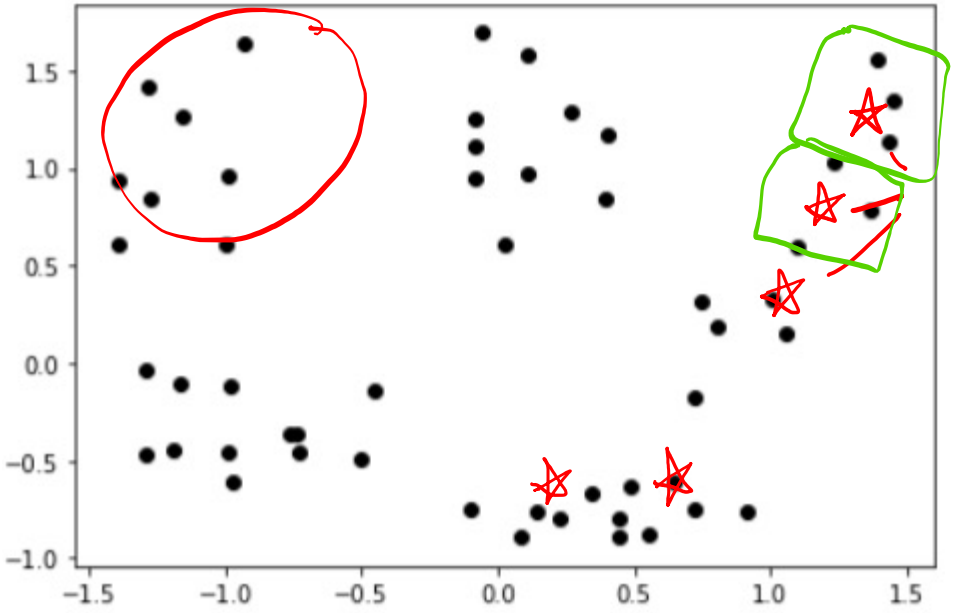
optional

Q: How many clusters does the dataset have?




iteration 1
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

Q: What are the potential problems with k-means clustering?

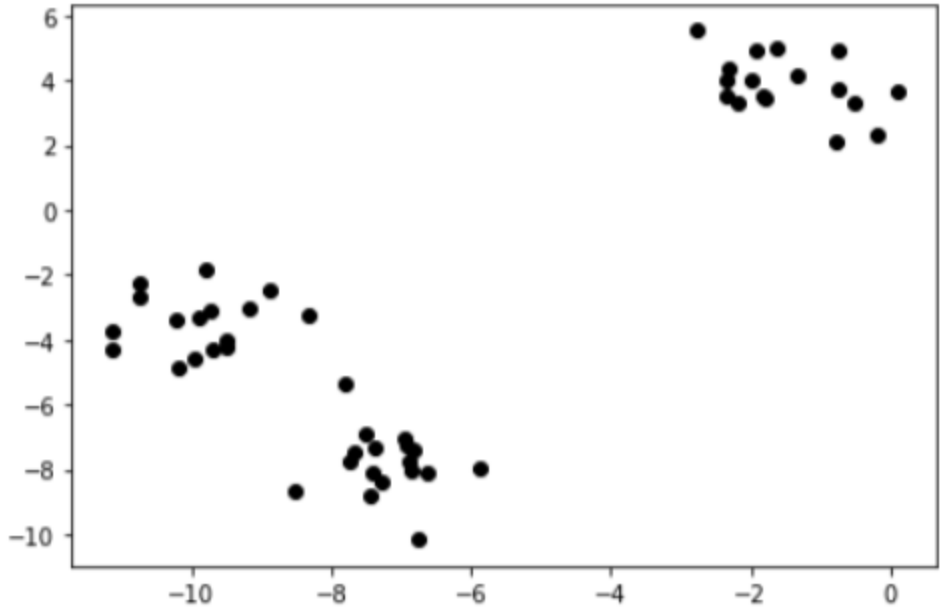


① k ?

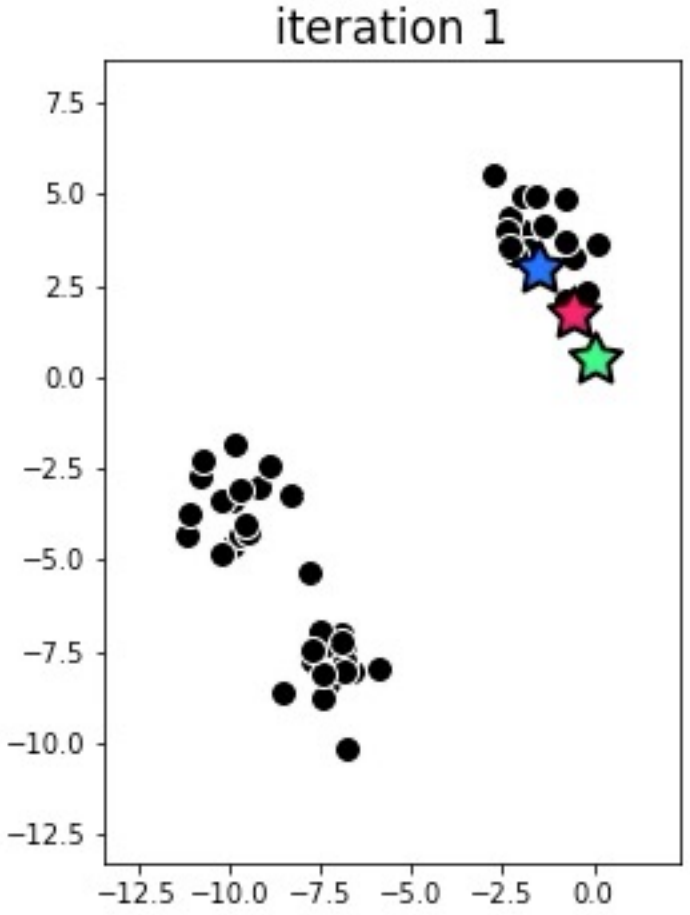
② initialization

Example: K-means failures due to a poor choice of initialization

- Generally speaking, a good initializations is one where the initial centroids are spread evenly throughout the distribution of the data.
- Conversely, a poor initialization is typically one where all of the initial centroids are bunched up together in a small region of the space.
- This can make it difficult for the centroids to spread out effectively, leading to less than optimal clusterings or even 'empty clusters' (i.e., those to which no points are finally assigned).

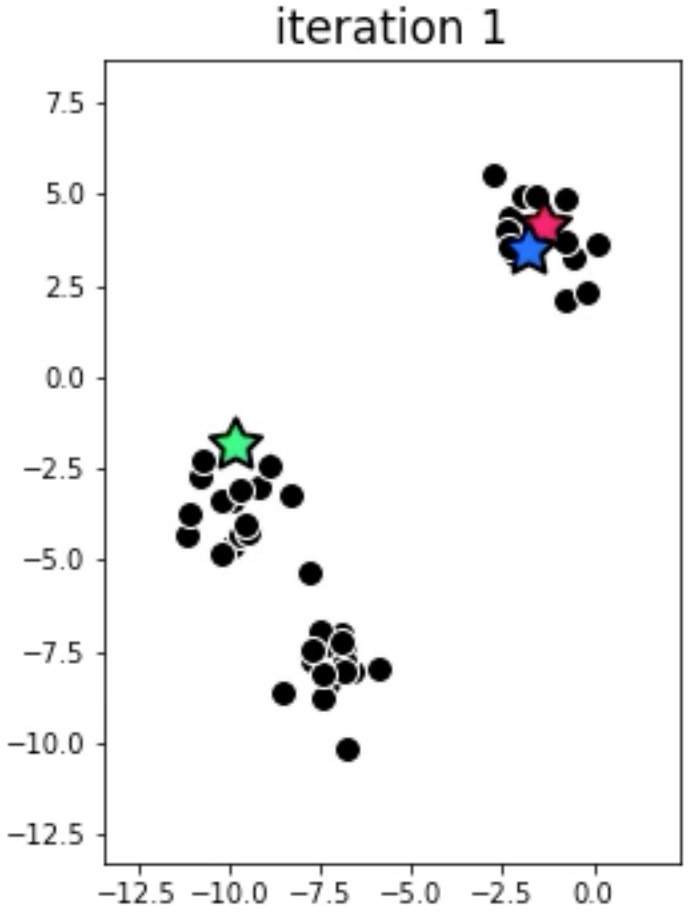

6
4
2
0
−2
−4
−6
−8
−10
−10
−8
−6
−4
−2
0

- The first and most fundamental issue is that of *empty clusters*, meaning clusters with no points assigned to them.
- It is indeed possible, for some unfortunate initialization choices, for clusters to end up being empty when running K-means.
- This is not an overly common occurrence, but is still worth noting.


iteration 1
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−12.5
−12.5 −10.0 −7.5 −5.0 −2.5 0.0

- Notice here how - due to the centroid initializations - the blue cluster very quickly becomes empty (every point is closer to one of the other two centroids) and never recovers any assigned points.
- This problem can be easily detected in practice (simply count up the number of assignments at the end of a run of K-means), and **re-running with a different initialization is typically the anecdote**.

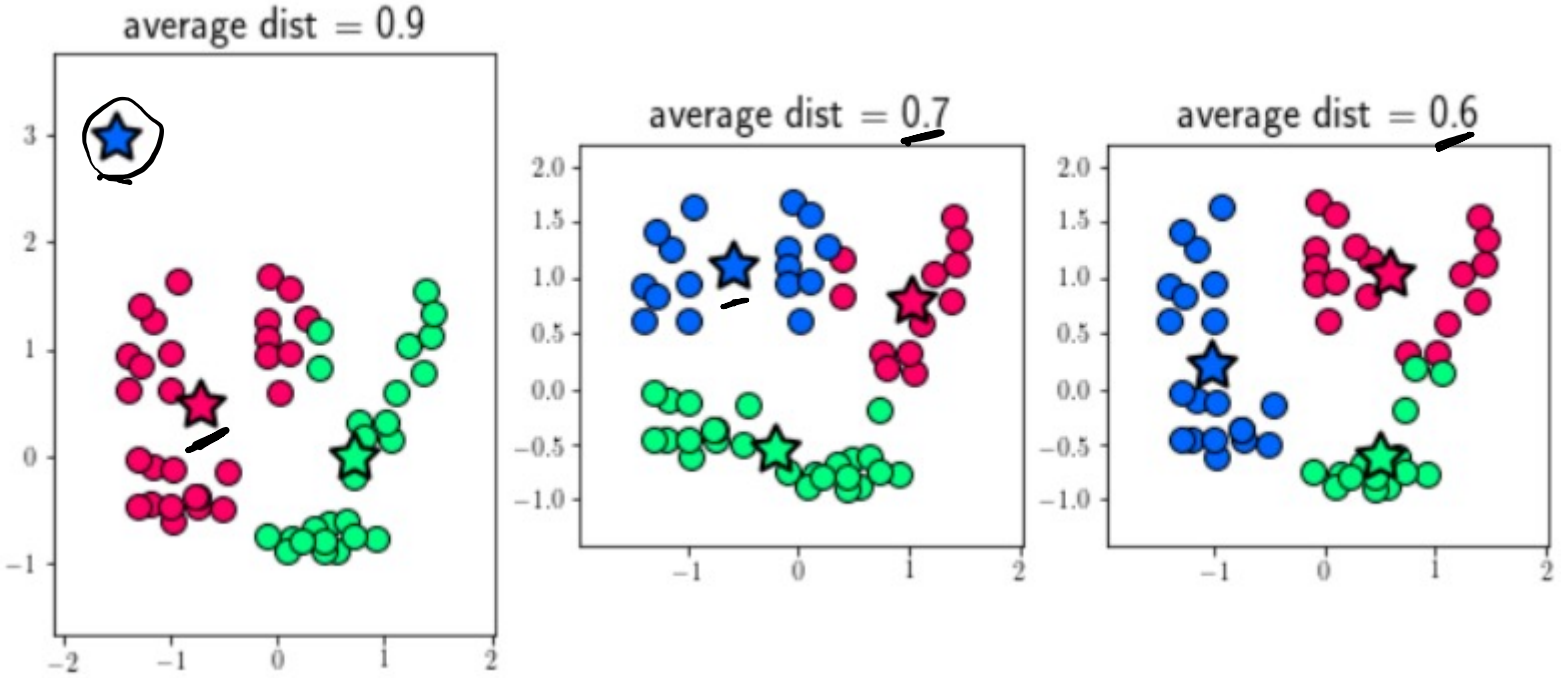
- The second issue to keep note of with K-means is the *sub-optimal clustering*.
- This means that although the proper number of clusters K was chosen the final clustering itself fails to fully capture the cluster-structure of the data.
- The animation below illustrates this failure on the same dataset used above.
- Here we can see that - due to the initialization - the single cluster in the top of the image is cleaved in two, while the two lower clusters are identified a single massive cluster.


iteration 1
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−12.5
−12.5
−10.0
−7.5
−5.0
−2.5
0.0

- Once again the best antidote for this issue is to simply re-run the algorithm with a different initialization.
- To determine the best clustering from set of runs we can use the *average distance of each point to its cluster centroid* - called the average intra-cluster distance - as an objective measure to rank the runs.
- Denoting $\mathbf{c}_{k_p}$ the final cluster centroid of the p^{th} point $\mathbf{x}_p$, then the average distance from each point to its respective centroid can be written as

$$\text{average intra-cluster distance} = \frac{1}{P}\sum_{p=1}^{P}\left\|\mathbf{x}_p - \mathbf{c}_{k_p}\right\|_2.$$

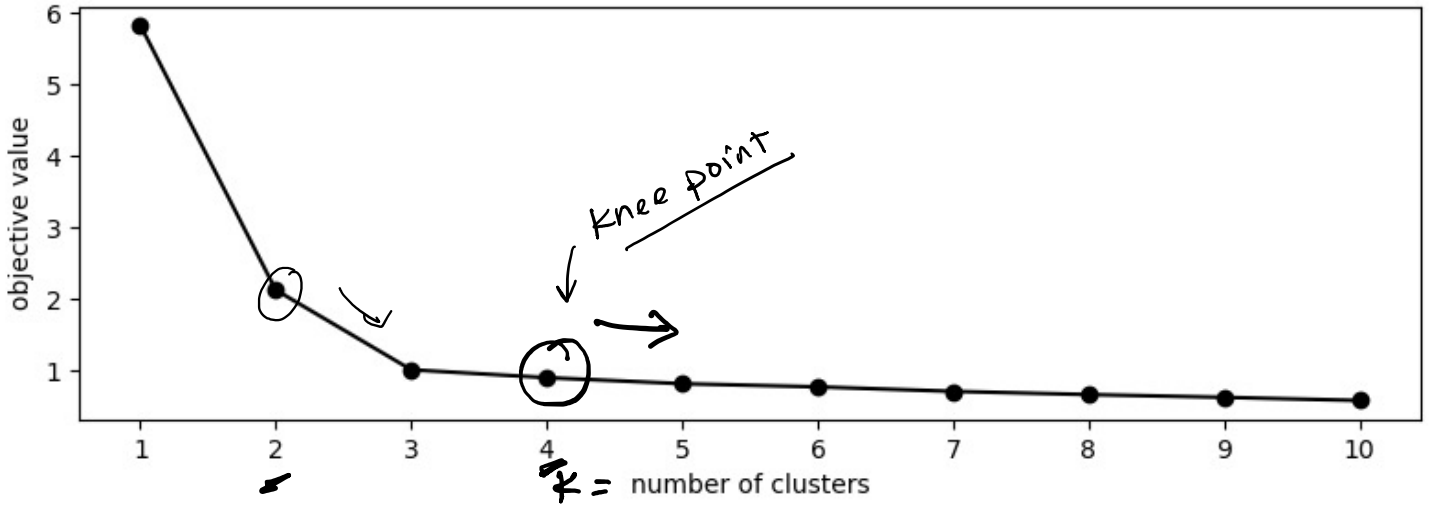
- Computing this for each run of K-means we choose the final clustering that achieves the ***smallest*** such value as the best clustering arrangement.



Example: Choosing the ideal number of clusters K

- To determine the optimal setting of the parameter K - i.e., the number of clusters in which to cluster the data - **we typically must try a range of different values for K**, run the K-means algorithm in each case, and compare the results using the average intra-cluster distance.

Scree plot



- As one should expect, the intra-cluster distance decreases **monotonically** as we increase K.
- Notice, however that the scree plot above has an ***elbow*** at K=3, meaning that increasing the number of clusters from 3 to 4 and onwards reduces the objective value by very little.
- Because of this **we can argue that K=3 is a good choice for the number of clusters for this particular dataset** (as we saw in the prior example, it is indeed the optimal number) since any fewer clusters and the intra-cluster distance is comparatively large, while adding additional clusters does not decrease the total intra-cluster distance too much.

Fun Stuff

Table 8.1 Common matrix factorization problems $\mathbf{CW} \approx \mathbf{X}$ subject to possible constraints on $\mathbf{C}$ and $\mathbf{W}$.

Matrix factorization problem	Constraints on C and W
Principal Component Analysis	**C** is orthonormal
Recommender systems	No constraint on **C** or **W**; **X** is only partially known
K-means clustering	Each column of **W** is a standard basis vector
Sparse dictionary learning	Each column of **W** is sparse
Nonnegative matrix factorization	Both **C** and **W** are nonnegative

Summary

- Background: Fixed Spanning Sets, Orthonormality, and Projections
- The Linear Autoencoder and Principal Component Analysis
  - Reduce the feature dimension N to the number of principle components
- K-means Clustering
  - Reduce the data dimension P to a set of centroids
  - Sensitive to initialization
  - Sensitive to the choice of K