Lecture 6: Unsupervised Learning

COMS W4995 Applied Machine Learning (Fall 2021)

Shouvik Mani

Columbia University

Announcements

- Homework 2 is due tomorrow 10/21 at 11:59pm
- Midterm on 10/27 in class

Outline

Unsupervised learning

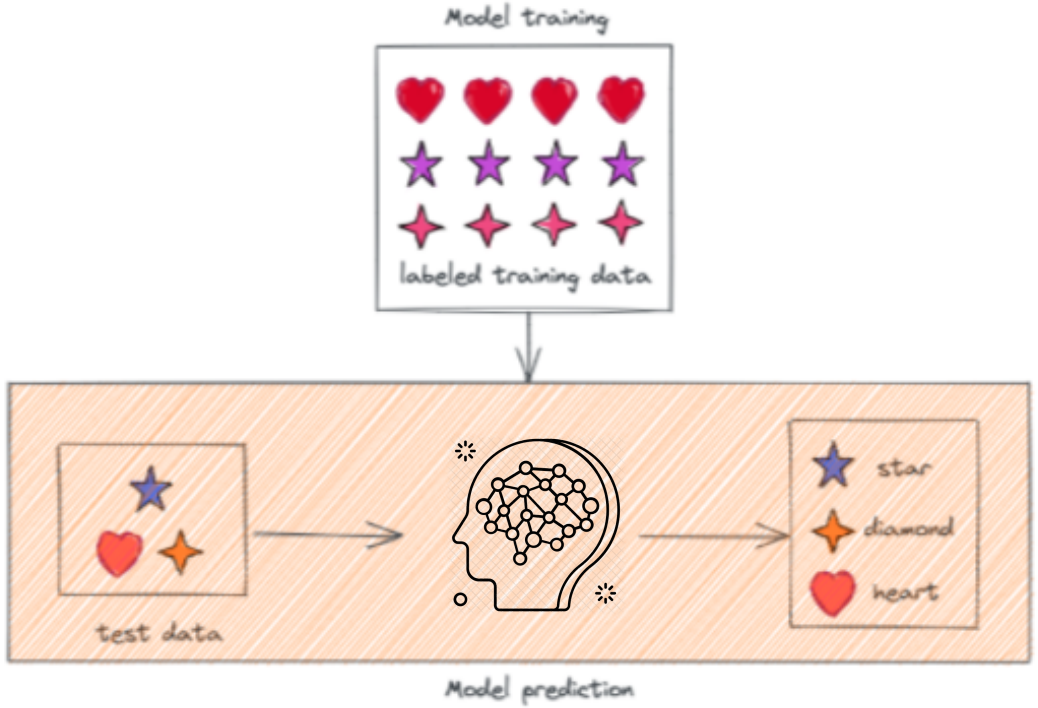
Clustering

Dimensionality Reduction

Unsupervised learning

In the unsupervised learning paradigm, we have features X, but **no labels y**. The goal is to **learn some structure in X** in order to better understand the data.

Supervised learning

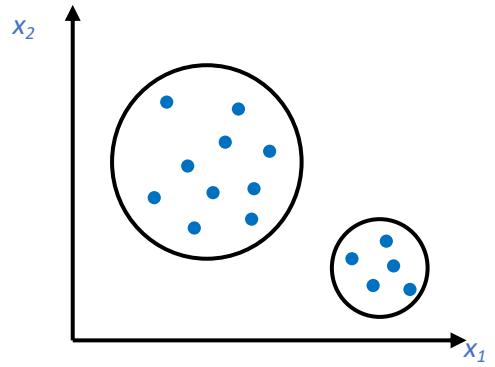


Unsupervised learning

Types of unsupervised learning problems

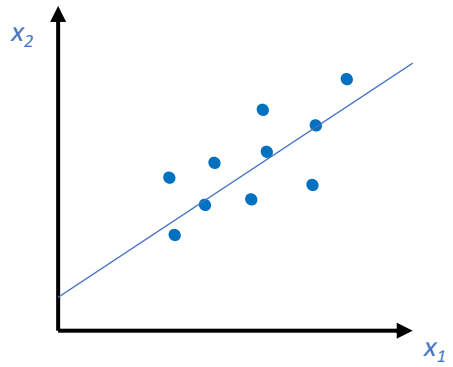
Three common unsupervised learning problems are: clustering, dimensionality reduction, and anomaly detection.

Clustering



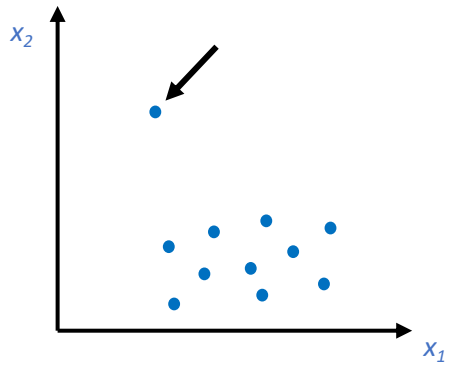
Identify groups of similar data points.

Dimensionality reduction



Reduce dataset to a lower-dimensional feature space.

Anomaly detection



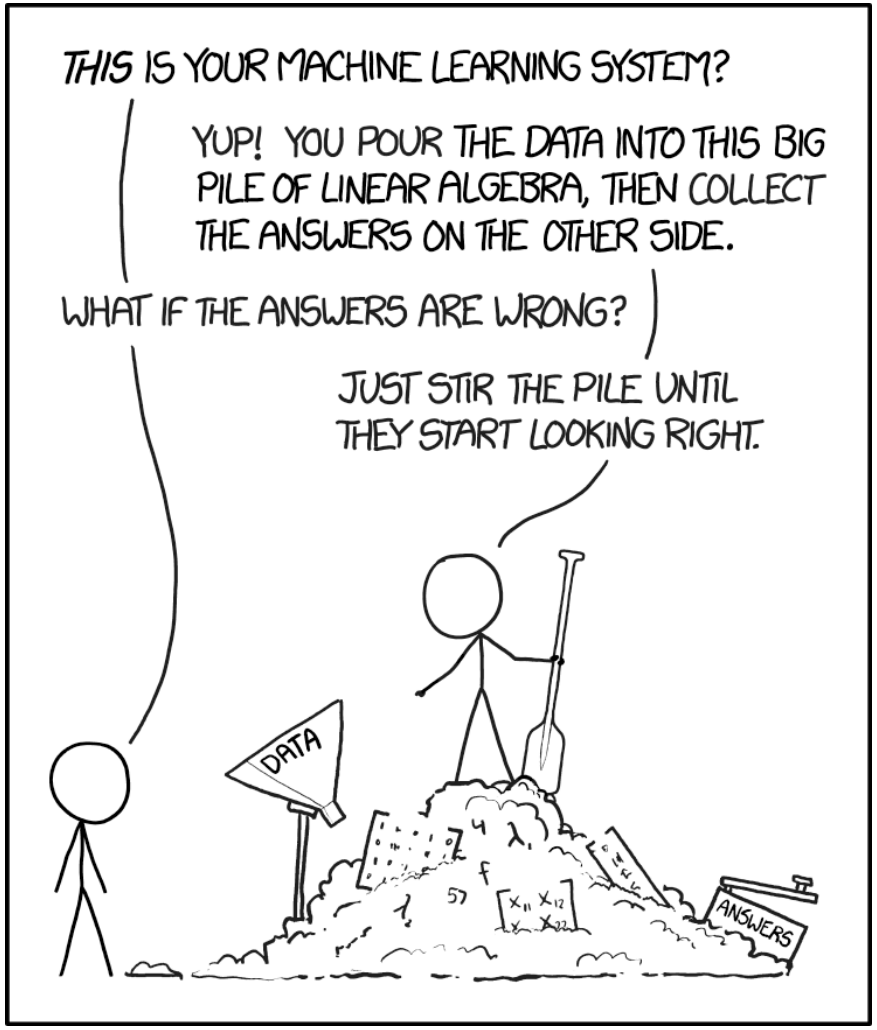
Identify outlier data points based on distribution of X.

The challenge in unsupervised learning

Because we have no labels in unsupervised learning, it is often difficult to evaluate whether our model is correct/useful.

It is often necessary to use domain expertise to understand if the model makes sense, and tune the model accordingly. For example:

- Choosing number of clusters
- Defining a distance metric
- Understanding what constitutes an anomaly



Outline

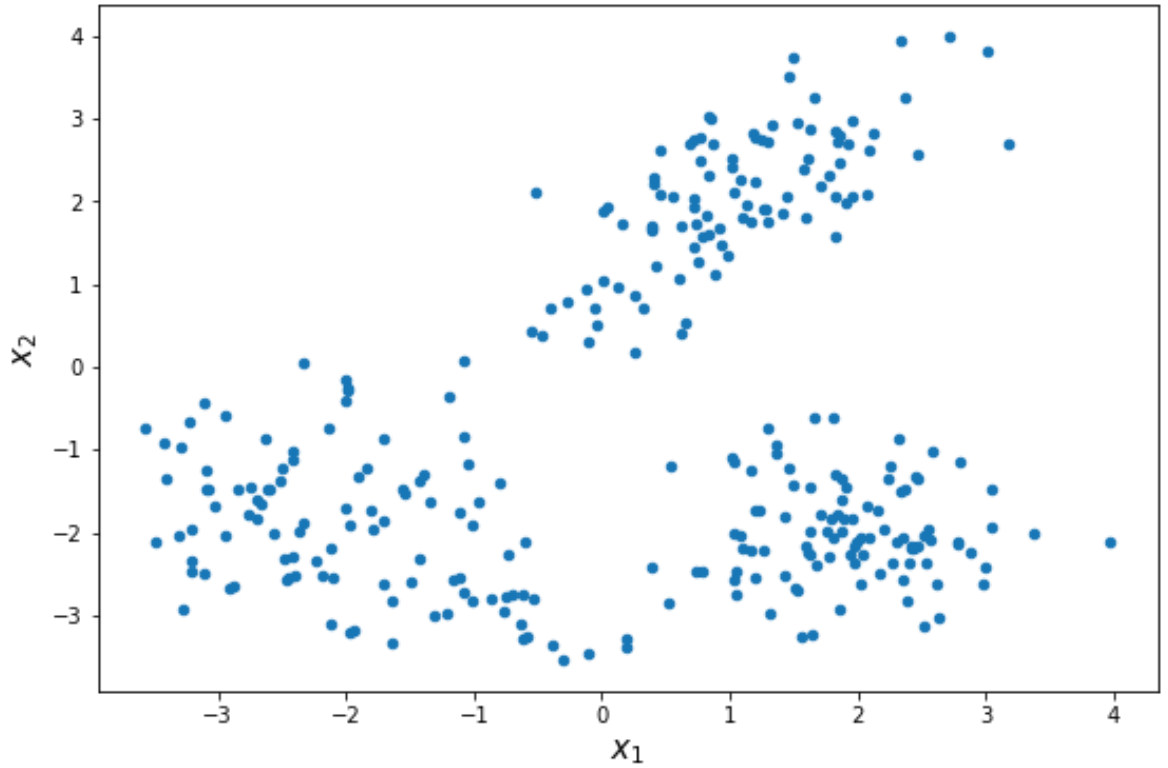
Unsupervised learning

Clustering

Dimensionality Reduction

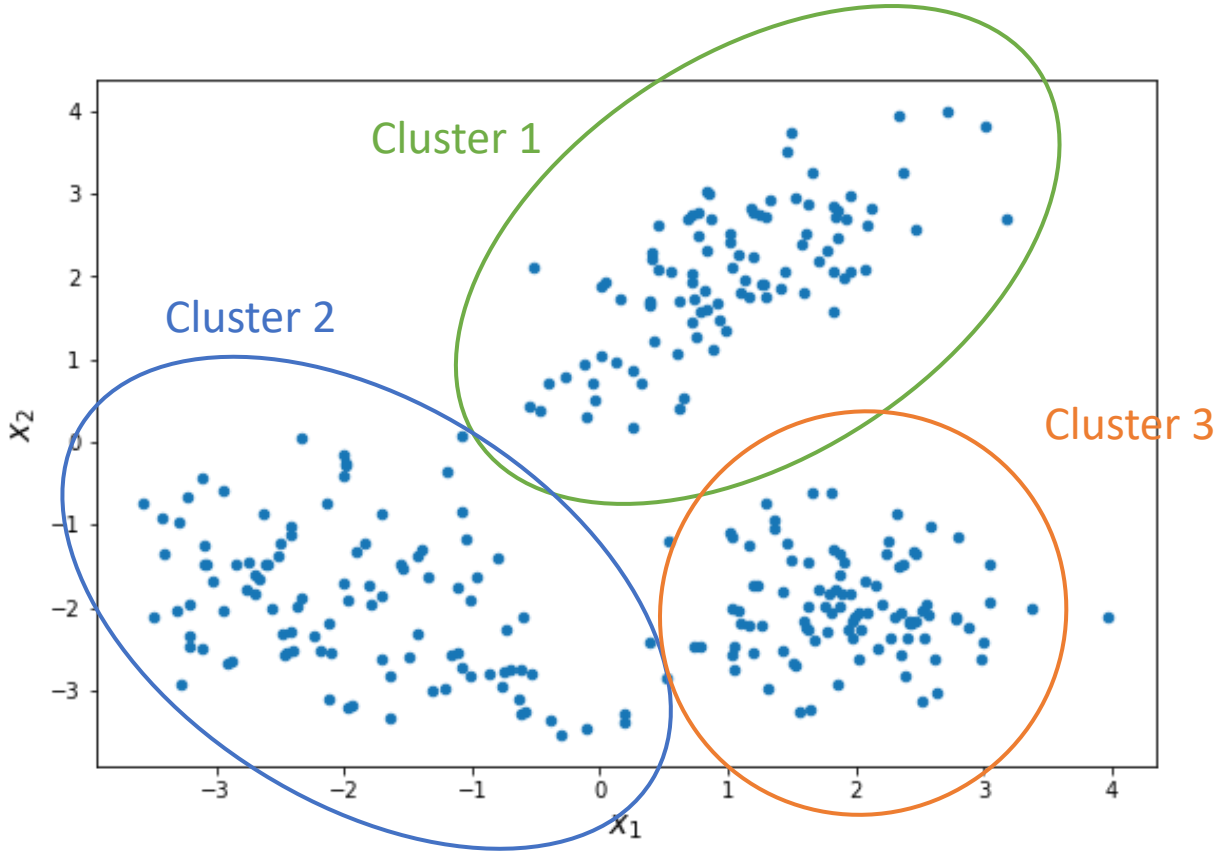
Clustering

Clustering is the problem of organizing data points into groups (clusters), such that data points within each cluster are *similar* to each other.



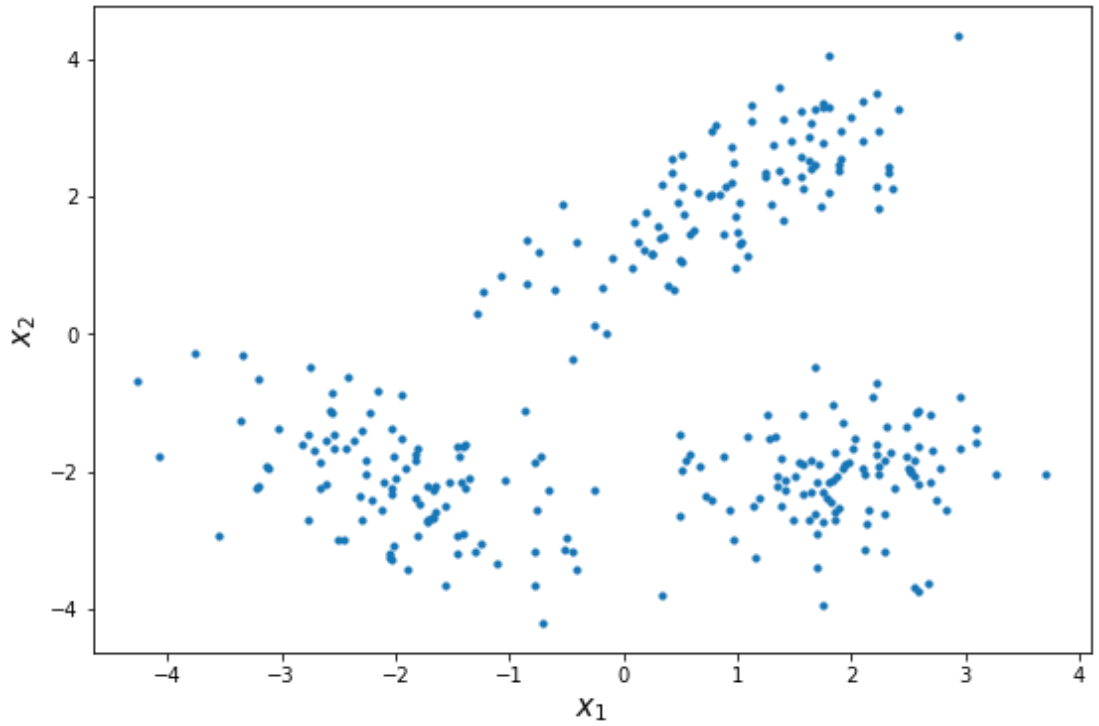
Clustering

Clustering is the problem of organizing data points into groups (clusters), such that data points within each cluster are *similar* to each other.



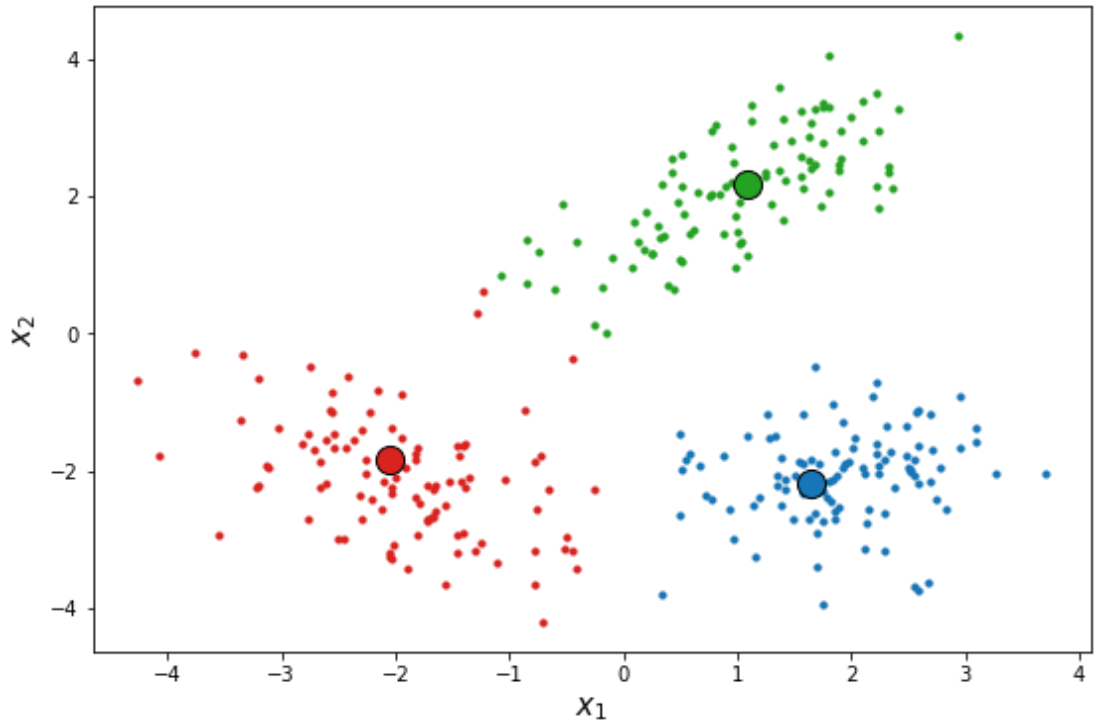
K-means clustering

The k-means algorithm groups data points into k clusters, based on the distance between each data point and it's closest cluster center.



K-means clustering

The k-means algorithm groups data points into k clusters, based on the distance between each data point and it's closest cluster center.



K-means clustering

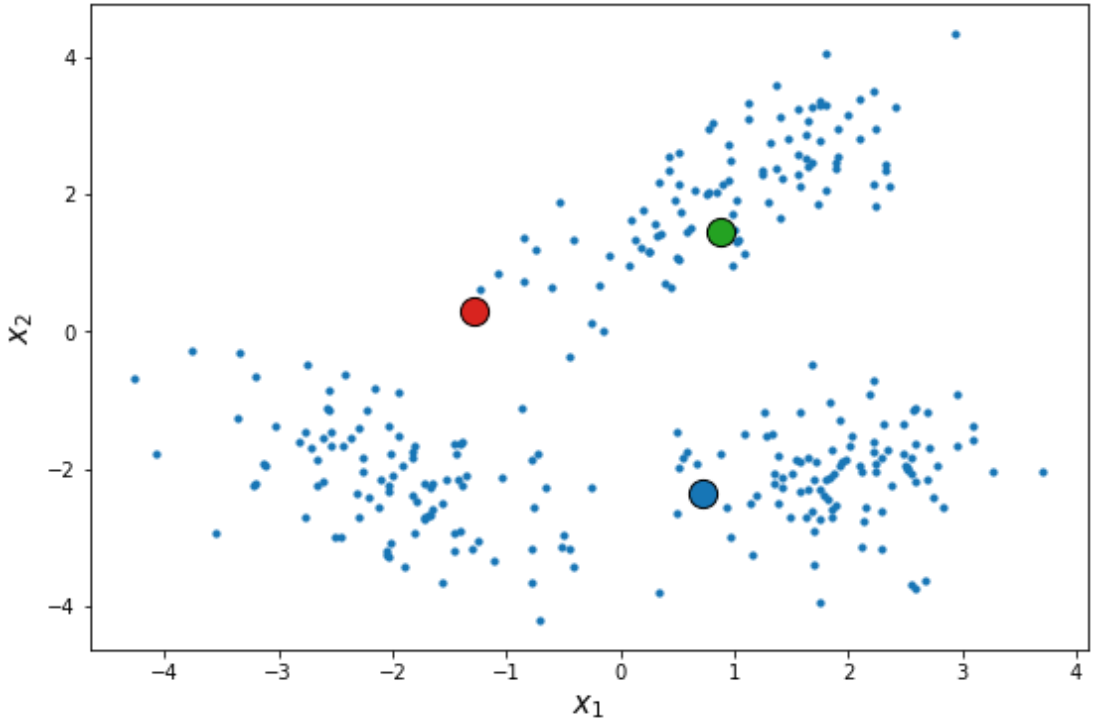
K-means algorithm

Given: number of clusters k,

data points $x^{(i)}$ for $i = 1, \dots, m$

Initialize cluster centers randomly:

$$\mu^{(j)} = Random\left(x^{(1)}, \dots, x^{(m)}\right), \quad j = 1, \dots, k$$



K-means clustering

K-means algorithm

Given: number of clusters k,

data points $x^{(i)}$ for $i = 1, \dots, m$

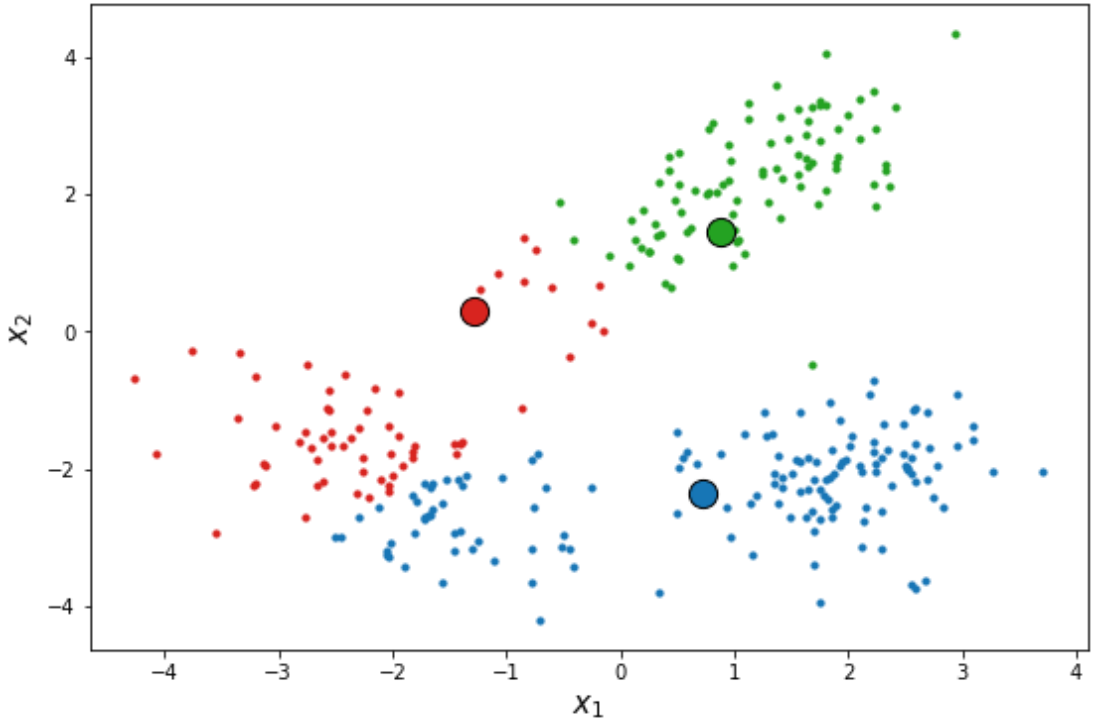
Initialize cluster centers randomly:

$$\mu^{(j)} = Random\left(x^{(1)}, \dots, x^{(m)}\right), \quad j = 1, \dots, k$$

Repeat until convergence:

1. Assign points $\boldsymbol{x^{(1)}, \dots, x^{(m)}}$ to nearest cluster center in $\boldsymbol{\mu^{(1)}, \dots, \mu^{(k)}}$

$$y^{(i)} = \underset{j}{argmin} \; || \mu^{(j)} - x^{(i)} ||_2 , \qquad i = 1, \dots, m$$



K-means clustering

K-means algorithm

Given: number of clusters k,

data points $x^{(i)}$ for $i = 1, \dots, m$

Initialize cluster centers randomly:

$$\mu^{(j)} = Random\left(x^{(1)}, \dots, x^{(m)}\right), \quad j = 1, \dots, k$$

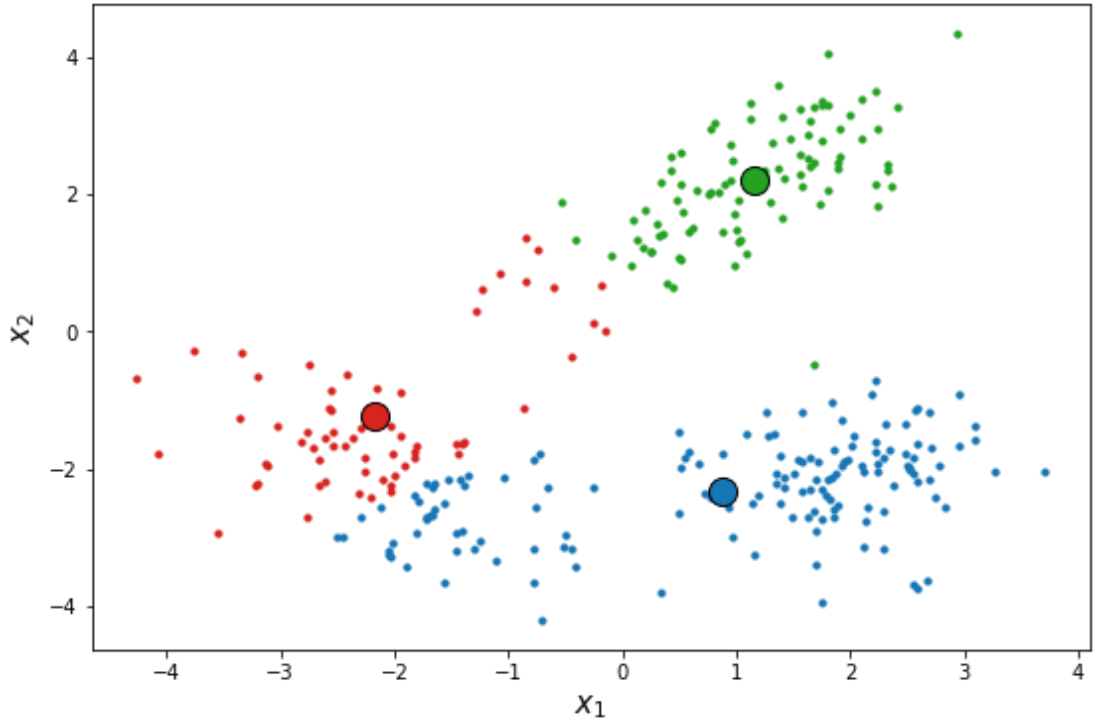
Repeat until convergence:

1. Assign points $x^{(1)}, \dots, x^{(m)}$ to nearest cluster center in $\mu^{(1)}, \dots, \mu^{(k)}$

$$y^{(i)} = \underset{j}{argmin} \; || \mu^{(j)} - x^{(i)} ||_2 , \qquad i = 1, \dots, m$$

2. Re-compute each cluster center as mean of all points assigned to it

$$\mu^{(j)} = mean\left(\{x^{(i)} \mid y^{(i)} = j \}\right), \qquad j = 1, \dots, k$$



K-means clustering

K-means algorithm

Given: number of clusters k,

data points $x^{(i)}$ for $i = 1, \dots, m$

Initialize cluster centers randomly:

$$\mu^{(j)} = Random\left(x^{(1)}, \dots, x^{(m)}\right), \quad j = 1, \dots, k$$

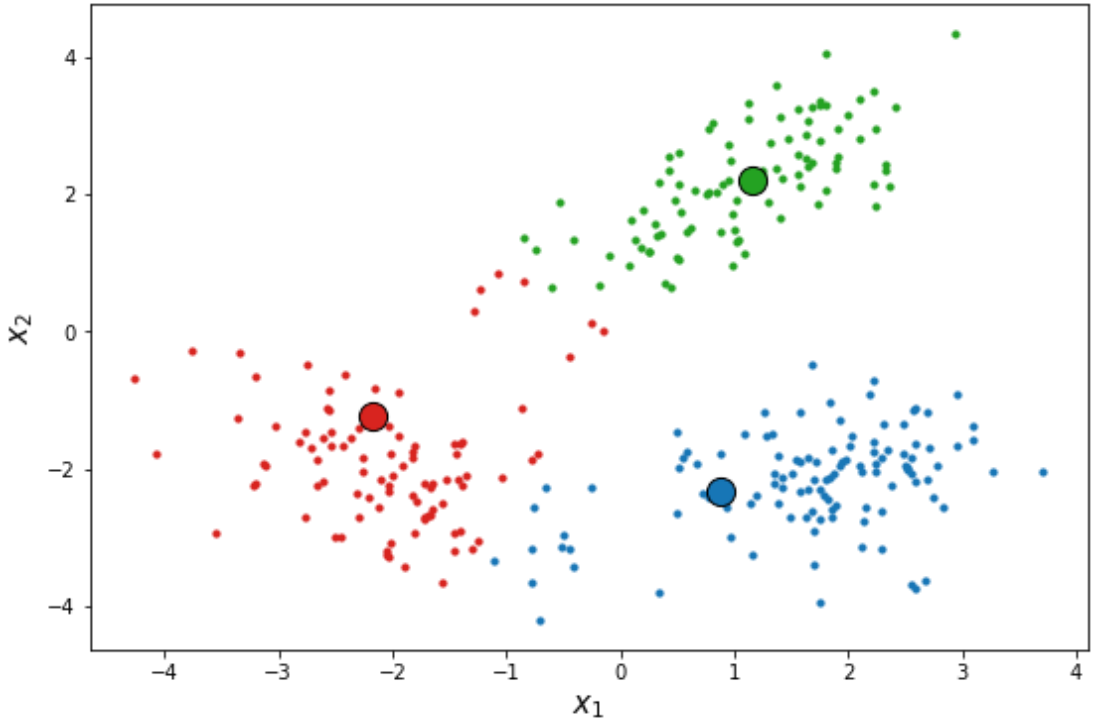
Repeat until convergence:

1. Assign points $\boldsymbol{x^{(1)}, \dots, x^{(m)}}$ to nearest cluster center in $\boldsymbol{\mu^{(1)}, \dots, \mu^{(k)}}$

$$y^{(i)} = \underset{j}{argmin} \; || \mu^{(j)} - x^{(i)} ||_2 , \qquad i = 1, \dots, m$$

2. Re-compute each cluster center as mean of all points assigned to it

$$\mu^{(j)} = mean\left(\{x^{(i)} \mid y^{(i)} = j \}\right), \qquad j = 1, \dots, k$$



K-means clustering

K-means algorithm

Given: number of clusters k,

data points $x^{(i)}$ for $i = 1, \dots, m$

Initialize cluster centers randomly:

$$\mu^{(j)} = Random\left(x^{(1)}, \dots, x^{(m)}\right), \quad j = 1, \dots, k$$

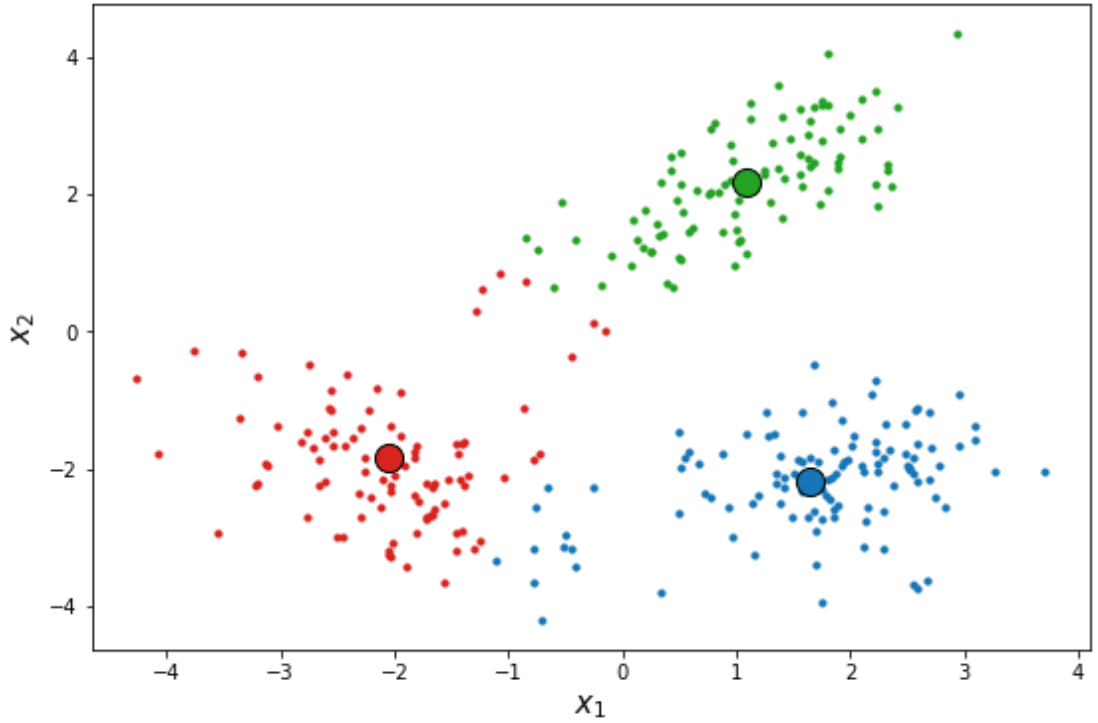
Repeat until convergence:

1. Assign points $x^{(1)}, \dots, x^{(m)}$ to nearest cluster center in $\mu^{(1)}, \dots, \mu^{(k)}$

$$y^{(i)} = \underset{j}{argmin} \; || \mu^{(j)} - x^{(i)} ||_2 , \qquad i = 1, \dots, m$$

2. Re-compute each cluster center as mean of all points assigned to it

$$\mu^{(j)} = mean\left(\{x^{(i)} \mid y^{(i)} = j \}\right), \qquad j = 1, \dots, k$$



K-means clustering

K-means algorithm

Given: number of clusters k,

data points $x^{(i)}$ for $i = 1, \dots, m$

Initialize cluster centers randomly:

$$\mu^{(j)} = Random\left(x^{(1)}, \dots, x^{(m)}\right), \quad j = 1, \dots, k$$

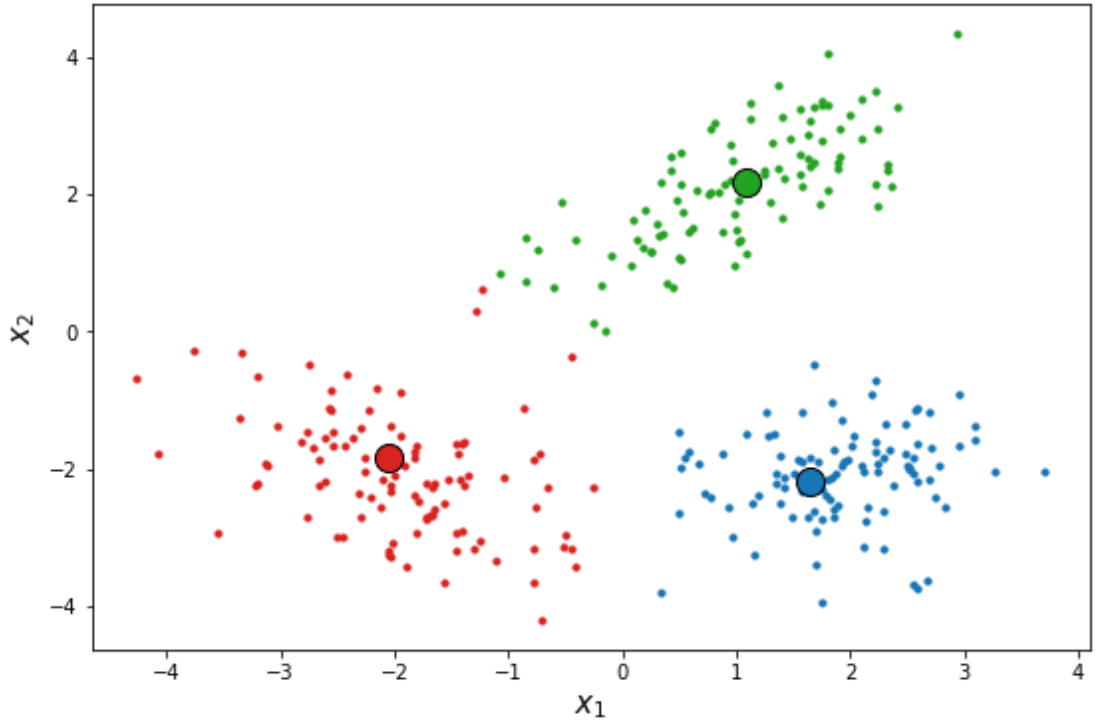
Repeat until convergence:

1. Assign points $\boldsymbol{x^{(1)}, \dots, x^{(m)}}$ to nearest cluster center in $\boldsymbol{\mu^{(1)}, \dots, \mu^{(k)}}$

$$y^{(i)} = \underset{j}{argmin} \; || \mu^{(j)} - x^{(i)} ||_2 , \qquad i = 1, \dots, m$$

2. Re-compute each cluster center as mean of all points assigned to it

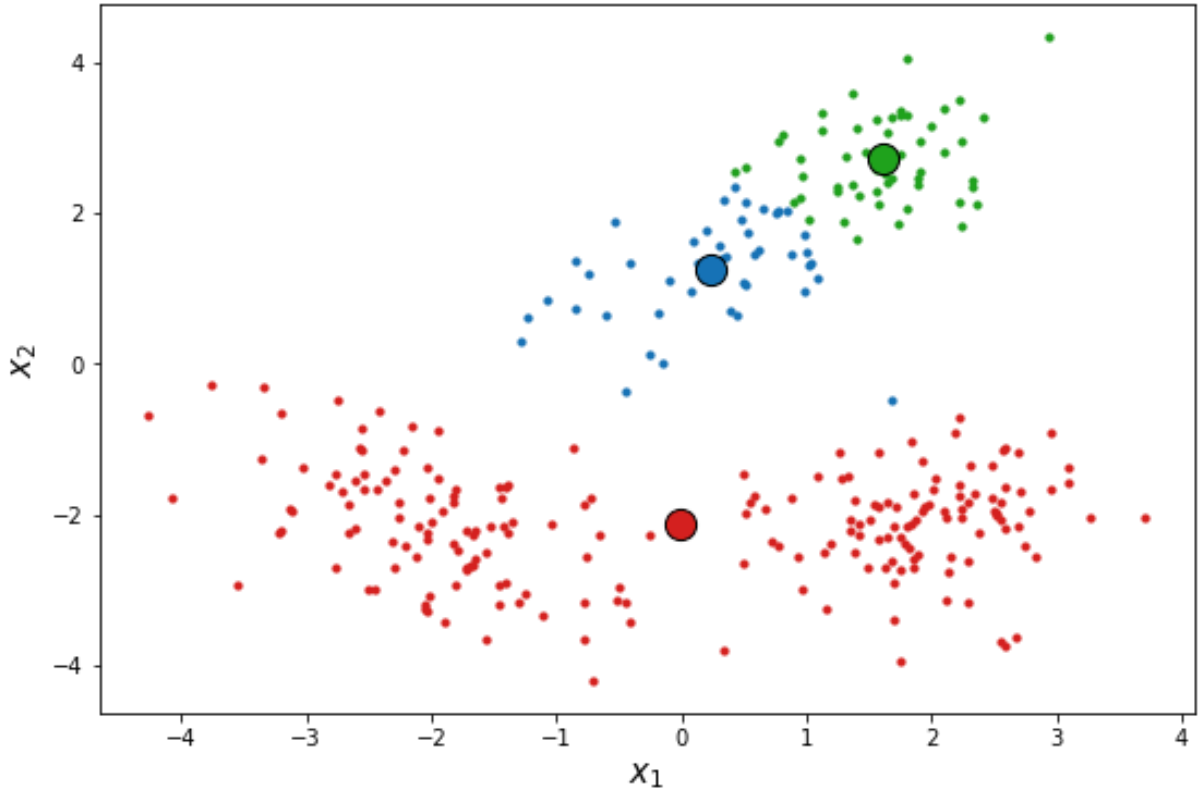
$$\mu^{(j)} = mean\left(\{x^{(i)} \mid y^{(i)} = j \}\right), \qquad j = 1, \dots, k$$



Problem: suboptimal clusters

K-means will produce different clusters (local optima) for different initializations of the cluster centers $\mu^{(1)}, \ldots, \mu^{(k)}$.

Due to this, it's possible that k-means finds a poor clustering of the data (shown on right).



Solutions:

- Run k-means with multiple (random) initializations of clusters centers. Take the clustering with the lowest loss.
- Use k-means++ algorithm, which "spreads out" initial cluster centers over the feature space.

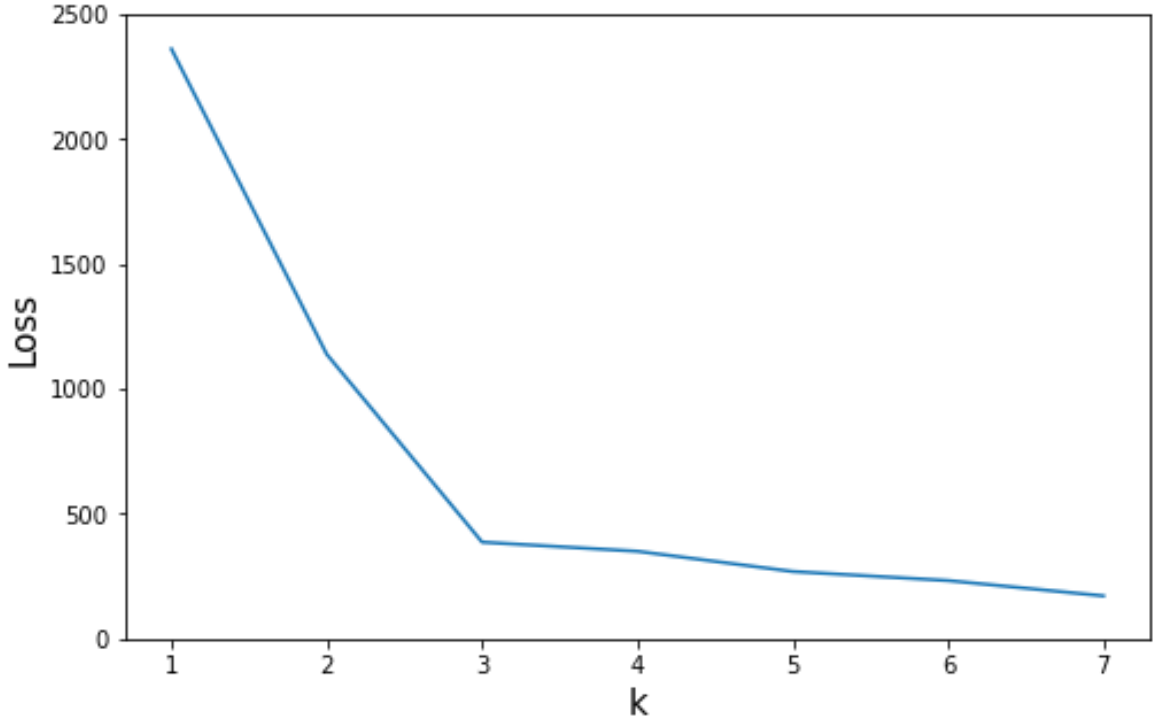
Choosing k in k-means

How do you choose the number of clusters k?

One option is to rely on **domain knowledge**: are there known clusters in the domain that you wish to discover?

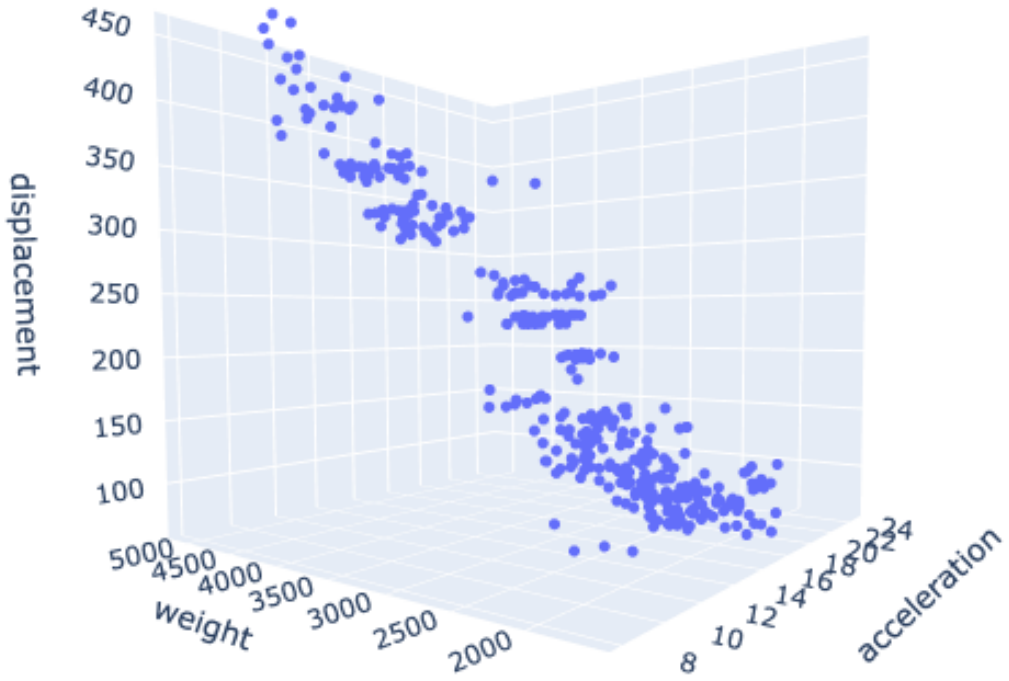
If not, a common approach is to plot the loss function of k-means over increasing k, and choose k to be the point where the loss looks reasonable.

$$loss = \sum_{i=1}^{m} || \mu^{(y^{(i)})} - x^{(i)} ||_2$$



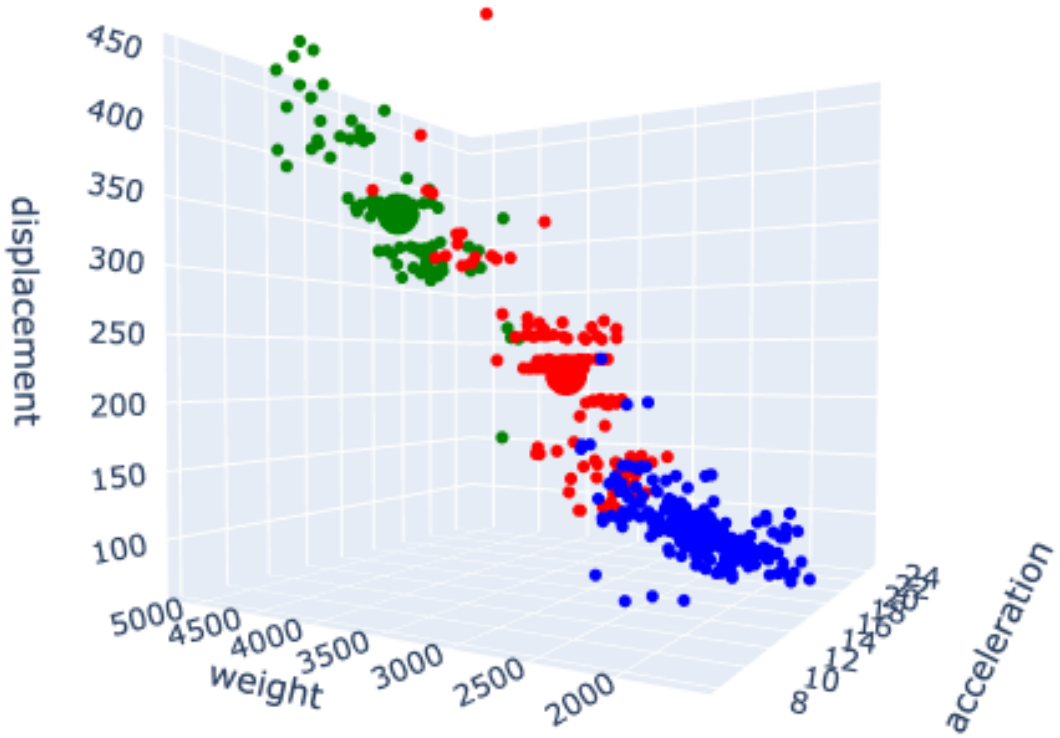
Application of k-means

Let’s apply k-means to the Auto MPG dataset from Homework 2. Plotting the displacement vs weight vs acceleration, there seem to be 3 clusters of cars present.



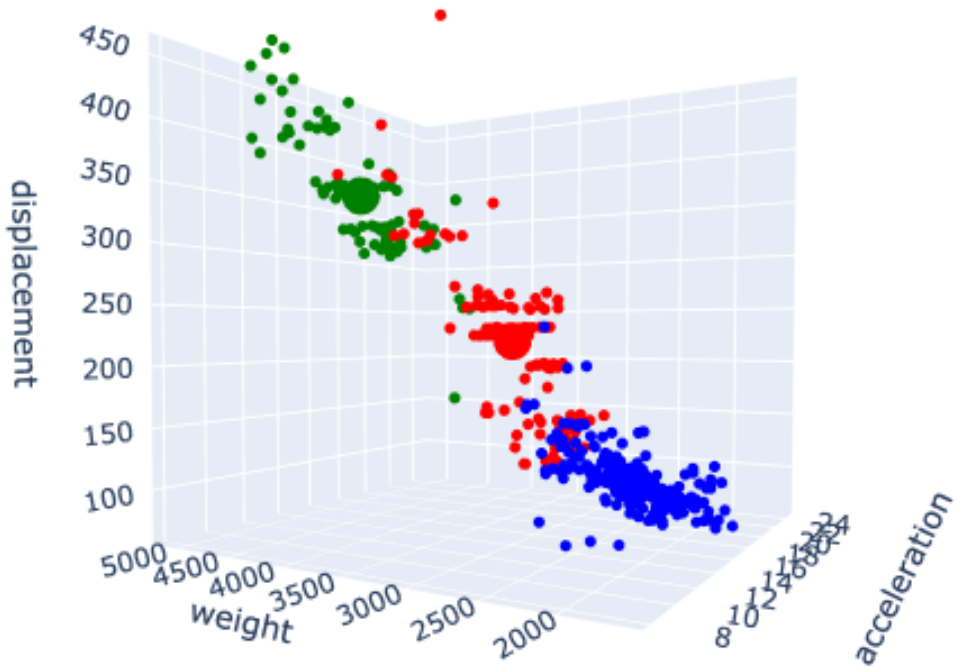
Application of k-means

K-means with $k = 3$ finds the 3 clusters.



Application of k-means

Insight: the clusters roughly correspond to the car type (large sedan vs pickup truck vs small sedan). Note that *nothing* about the car type was given to the algorithm – it "discovered" these clusters on its own.



Examples of cars in each cluster:

	Cluster 1	Cluster 2	Cluster 3
0	dodge st. regis	dodge charger 2.2	buick century limited
1	buick estate wagon (sw)	vw pickup	oldsmobile cutlass ciera (diesel)
2	ford country squire (sw)	dodge rampage	ford granada l
3	chrysler lebaron town @ country (sw)	ford ranger	chevrolet camaro
4	cadillac eldorado	chevy s-10	ford mustang gl









Outline

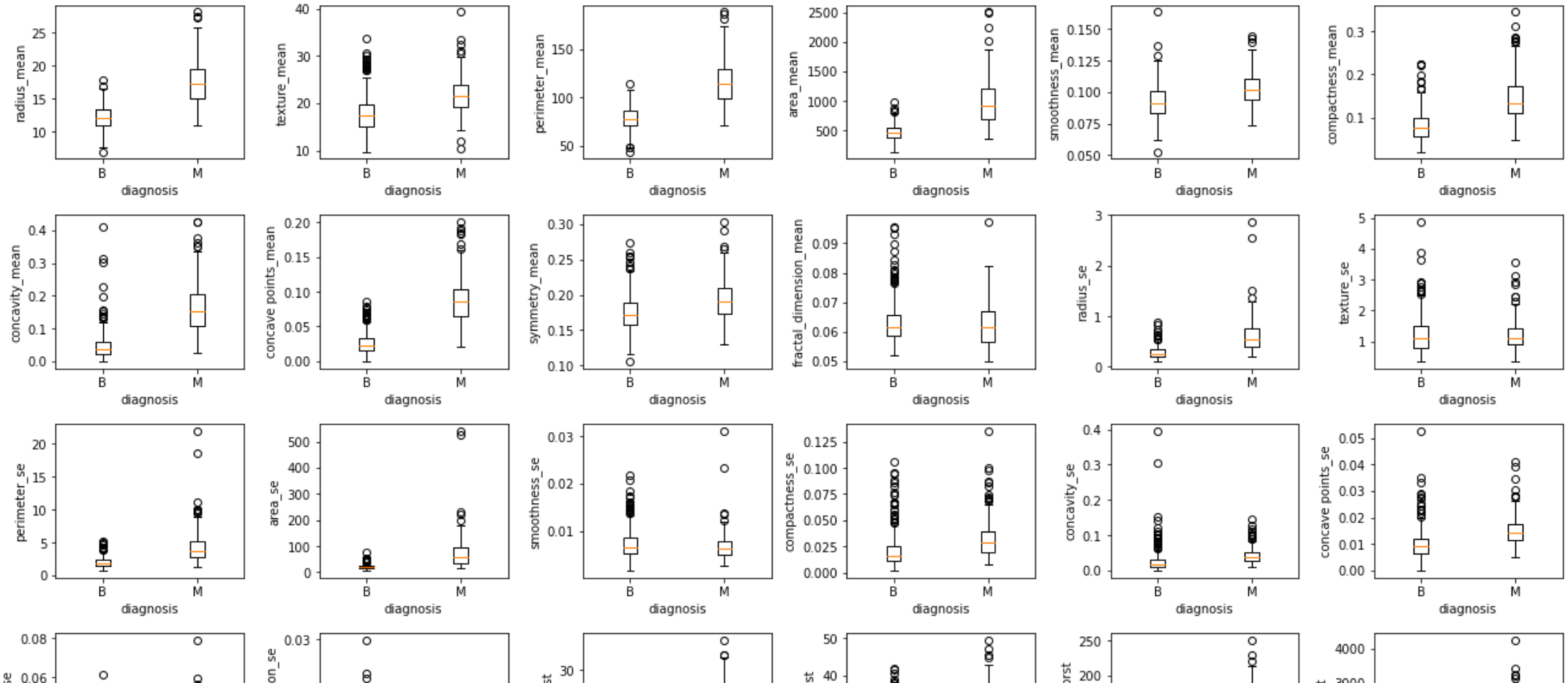
Unsupervised learning

Clustering

Dimensionality Reduction

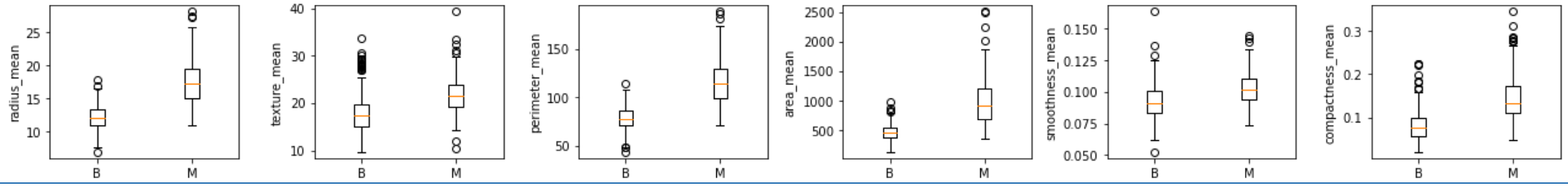
Motivation: data visualization

Data visualization in high dimensions is challenging. Recall the breast cancer dataset in Homework 2, where you made box plots of the features of the cells vs the diagnosis (benign or malignant).



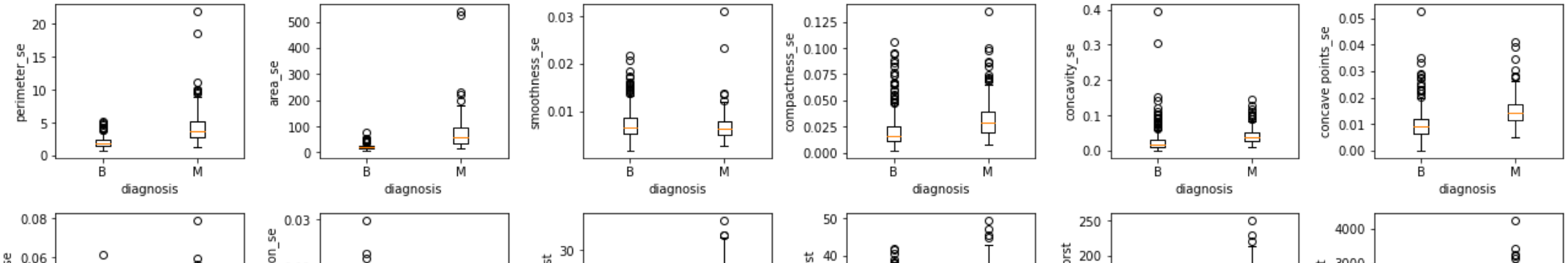
Motivation: data visualization

Data visualization in high dimensions is challenging. Recall the breast cancer dataset in Homework 2, where you made box plots of the features of the cells vs the diagnosis (benign or malignant).



That's a lot of plots!

Ideally we want a **low-dimensional representation** of the data which captures as much information as possible.

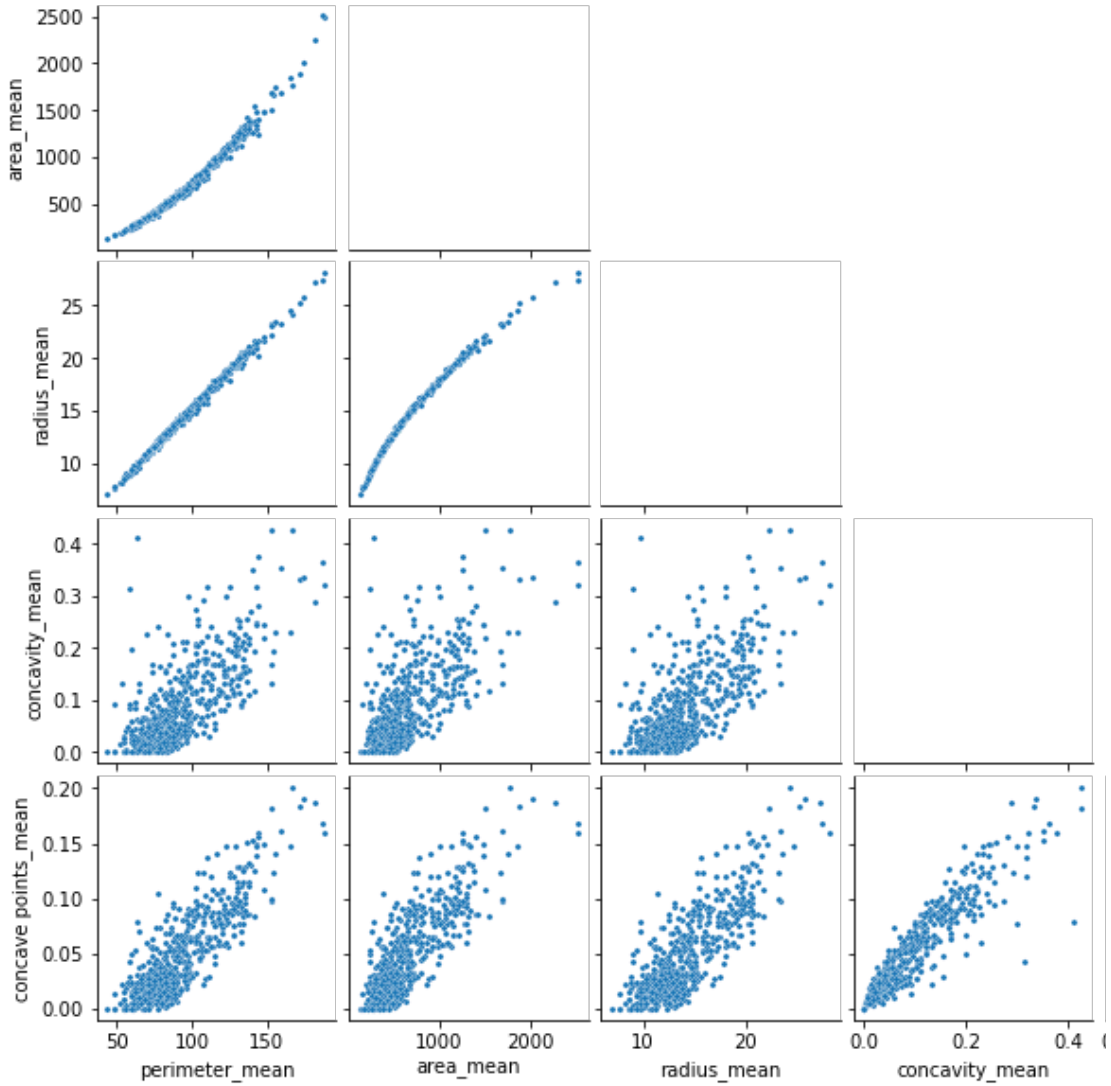


Motivation: data visualization

Observation: many features in the breast cancer dataset are highly correlated with each other.

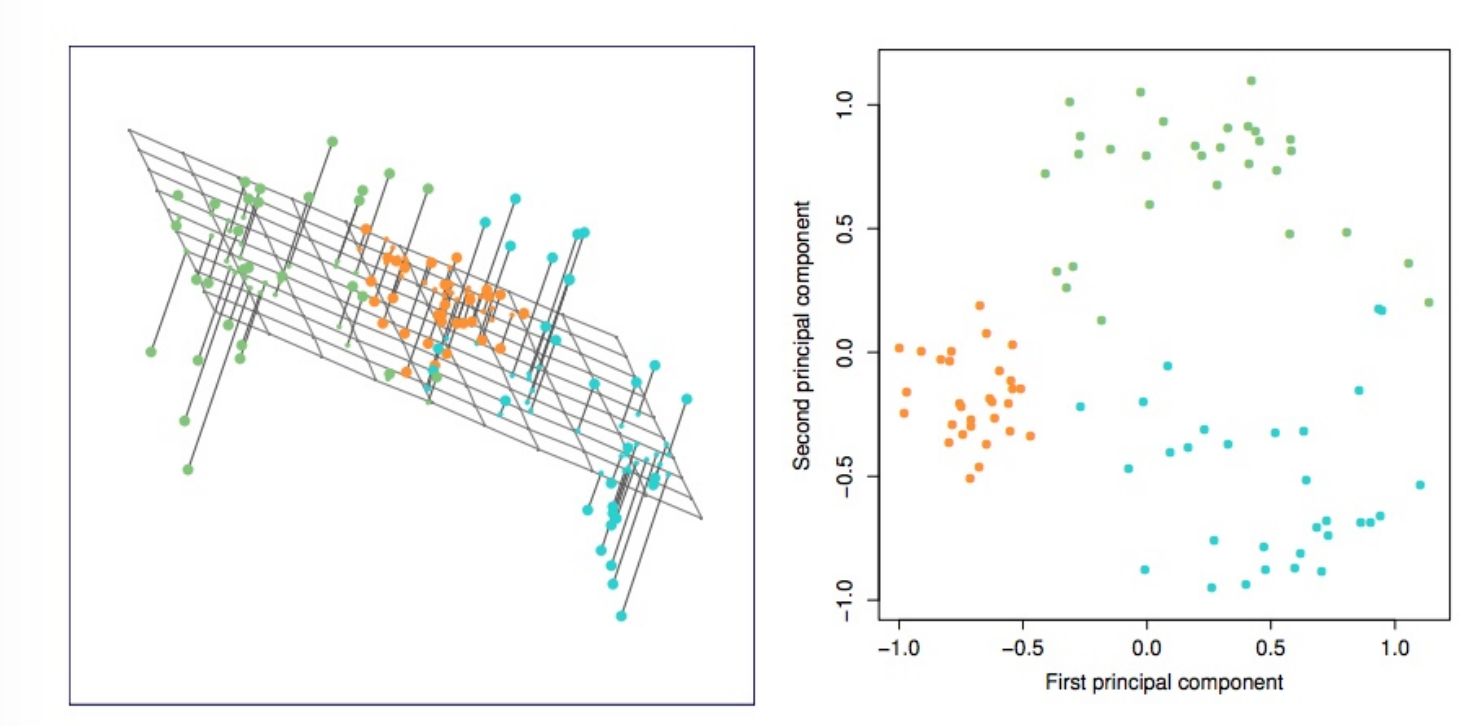
Idea: we can exploit this redundancy in the features to “combine” features which are highly correlated with each other.

This is the key idea behind **principal component analysis (PCA)**, a widely-used algorithm for dimensionality reduction.



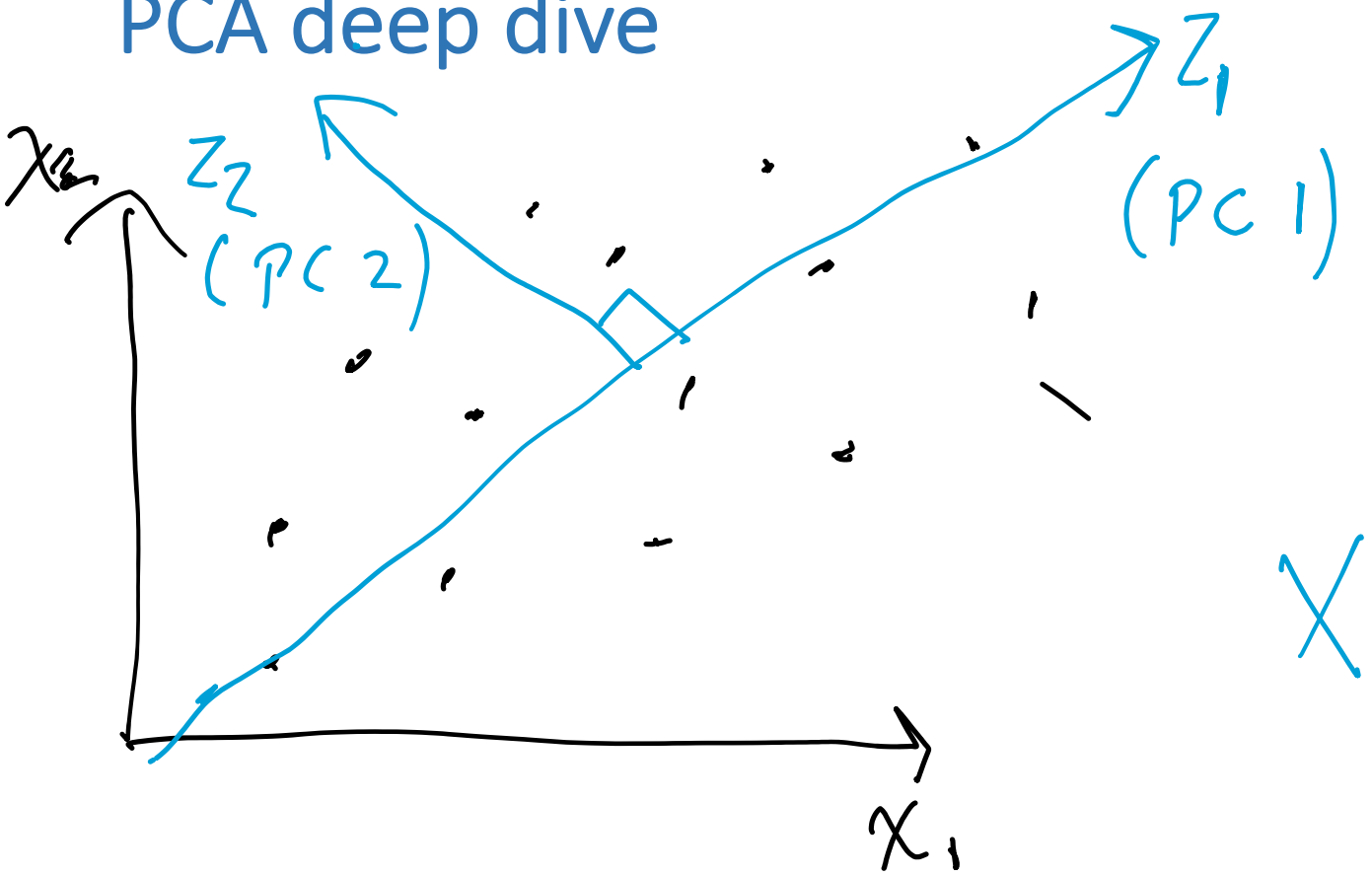
Principal component analysis (PCA)

Principal component analysis (PCA) projects data to a lower-dimensional linear subspace, in a way that preserves the axes of highest variance in the data.



Source: An Introduction to Statistical Learning, Witten et al.

PCA deep dive



$$Z_1 = \phi_{11} X_1 + \phi_{21} X_2$$

$$Z_2 = \phi_{12} X_1 + \phi_{22} X_2$$

$$X_1 = [x_{11}, x_{21}, \dots x_{n1}]^T$$

$$\max_{\phi_{11}, \phi_{21}, \dots} \left\{ \frac{1}{m} \sum_{i=1}^{m} \left(\sum_{j=1}^{n} \phi_{j1} x_{ij} \right)^2 \right\} \quad \text{s.t} \sum_{j=1}^{n} \phi_{j1}^2 = 1$$

Application of PCA

Let's use PCA to reduce the dimensionality of the breast cancer dataset.

An **important pre-processing step before applying PCA** is to standardize features to have zero mean and unit variance. Otherwise, features with high variance will dominate the principal component directions.

```
# 4. Standardize the columns in the feature matrices
scaler = StandardScaler()
cancer_X_train = scaler.fit_transform(cancer_X_train)   # Fit and transform scalar on cancer_X_train
cancer_X_val = scaler.transform(cancer_X_val)           # Transform cancer_X_val
cancer_X_test = scaler.transform(cancer_X_test)         # Transform cancer_X_test

# 5. Add a column of ones to the feature matrices
cancer_X_train = np.hstack([np.ones((cancer_X_train.shape[0], 1)), cancer_X_train])
cancer_X_val = np.hstack([np.ones((cancer_X_val.shape[0], 1)), cancer_X_val])
cancer_X_test = np.hstack([np.ones((cancer_X_test.shape[0], 1)), cancer_X_test])

# Verify that columns (other than the ones column) have 0 mean, 1 variance
print(cancer_X_train.mean(axis=0), '\n', cancer_X_train.std(axis=0))
```

executed in 51ms, finished 15:29:55 2021-10-20

```
[ 1.00000000e+00  1.12259501e-15  1.32836069e-16  3.90694320e-16
 -5.15716502e-16  1.07050244e-15 -3.38601744e-17 -1.25022182e-16
  2.08370304e-16 -8.36085844e-16  1.14603667e-15  2.00556417e-16
 -5.20925759e-18  3.12555456e-17  1.14603667e-16  1.04185152e-16
 -1.17208296e-16 -5.73018335e-17 -2.03161046e-16  5.46972047e-17
  5.46972047e-17  7.55342351e-17 -5.18321131e-16 -2.39625849e-16
 -7.29296063e-17  3.20369342e-16 -5.20925759e-17 -6.77203487e-17
 -1.04185152e-17  5.59995191e-17 -4.34973009e-16]
 [0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1.]
```

Application of PCA

We perform PCA (with 5 PCs here) and look at the principal component directions (i.e. loading vectors). These are the directions of highest variance in the data.

```
from sklearn.decomposition import PCA

pca = PCA(n_components=5)
cancer_X_train_pc = pca.fit_transform(cancer_X_train)
```

executed in 8ms, finished 15:32:09 2021-10-20

```
principal_components_df = pd.DataFrame(np.abs(pca.components_.T[:11, :]),  # magnitude of PCs
                                       columns=[f'PC {i}' for i in range(1, 6)],
                                       index=['intercept'] + list(cancer_X.columns)[:10])
principal_components_df.style.background_gradient(cmap="Reds")
```

executed in 33ms, finished 17:08:24 2021-10-20

	PC 1	PC 2	PC 3	PC 4	PC 5
intercept	0.00	0.00	0.00	0.00	0.00
radius_mean	0.22	0.23	0.01	0.05	0.02
texture_mean	0.10	0.06	0.02	0.60	0.04
perimeter_mean	0.23	0.22	0.01	0.05	0.02
area_mean	0.22	0.23	0.03	0.04	0.00
smoothness_mean	0.14	0.19	0.02	0.11	0.40
compactness_mean	0.24	0.14	0.08	0.04	0.04
concavity_mean	0.26	0.06	0.02	0.03	0.09
concave points_mean	0.26	0.04	0.01	0.05	0.06
symmetry_mean	0.15	0.19	0.00	0.04	0.25
fractal_dimension_mean	0.06	0.36	0.01	0.04	0.07

Note that the first principal component PC 1 places large weights on radius, perimeter, and area, which we saw were highly correlated with each other.

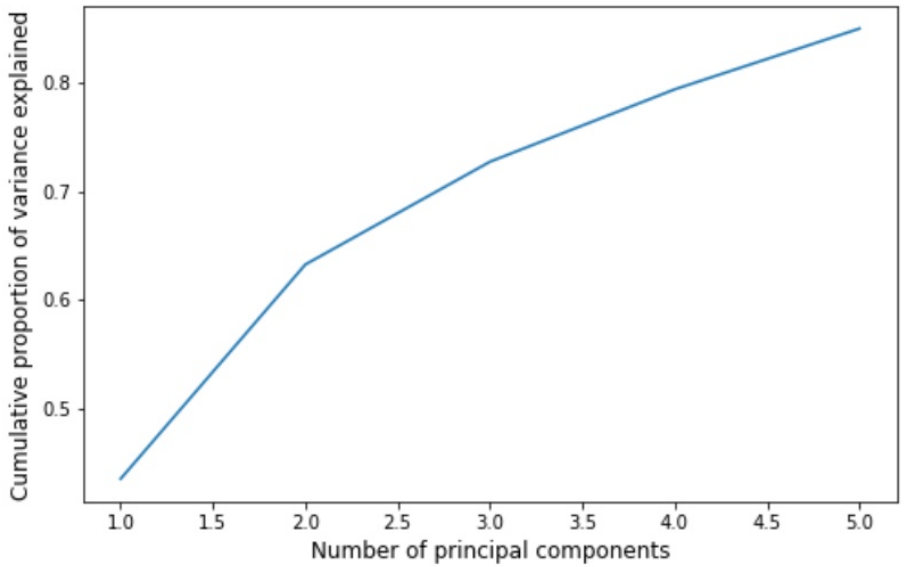
Application of PCA

The proportion of variance explained (PVE) tells us how much of the variance in the dataset is explained by each principal component. It appears that just the first two principal components together explain about 65% of the variance in this dataset.

```
plt.figure(figsize=(8, 5))
plt.plot(range(1, len(pca.explained_variance_ratio_)+1),
         pca.explained_variance_ratio_.cumsum())
plt.xlabel('Number of principal components', fontsize=12)
plt.ylabel('Cumulative proportion of variance explained', fontsize=12)
```

executed in 297ms, finished 17:08:25 2021-10-20

```
Text(0, 0.5, 'Cumulative proportion of variance explained')
```



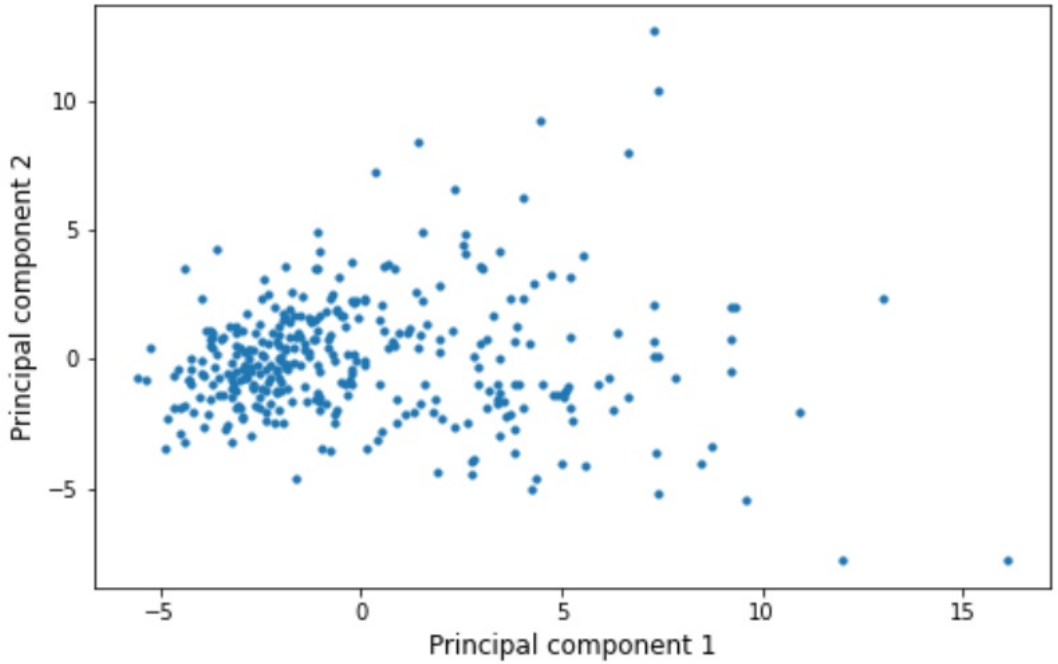
Application of PCA

Let’s visualize the data projected onto the first two principal components.

```
plt.figure(figsize=(8, 5))
plt.scatter(cancer_X_train_pc[:, 0], cancer_X_train_pc[:, 1], s=10)
plt.xlabel('Principal component 1', fontsize=12)
plt.ylabel('Principal component 2', fontsize=12)
```

executed in 202ms, finished 17:08:27 2021-10-20

```
Text(0, 0.5, 'Principal component 2')
```



Application of PCA

Let's color the projected data points by their diagnosis (benign or malignant).

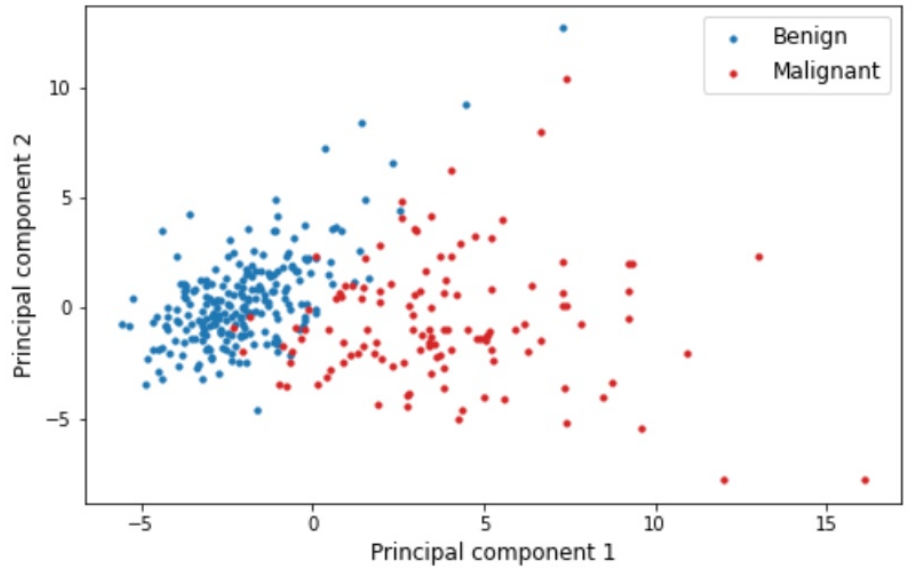
Amazingly, just the first two principal components offer a linear decision boundary to predict the diagnosis.

Idea: use the first two principal components (instead of the 31 original features) to predict diagnosis. This will result in a simpler, low variance model.

```
plt.figure(figsize=(8, 5))
colors = ['tab:blue', 'tab:red']
for class_ in [0, 1]:
    plt.scatter(cancer_X_train_pc[cancer_y_train.flatten()==class_, 0],
                cancer_X_train_pc[cancer_y_train.flatten()==class_, 1],
                label='Benign' if class_==0 else 'Malignant', s=10, c=colors[class_])
plt.legend(loc='upper right', fontsize=12)
plt.xlabel('Principal component 1', fontsize=12)
plt.ylabel('Principal component 2', fontsize=12)
```

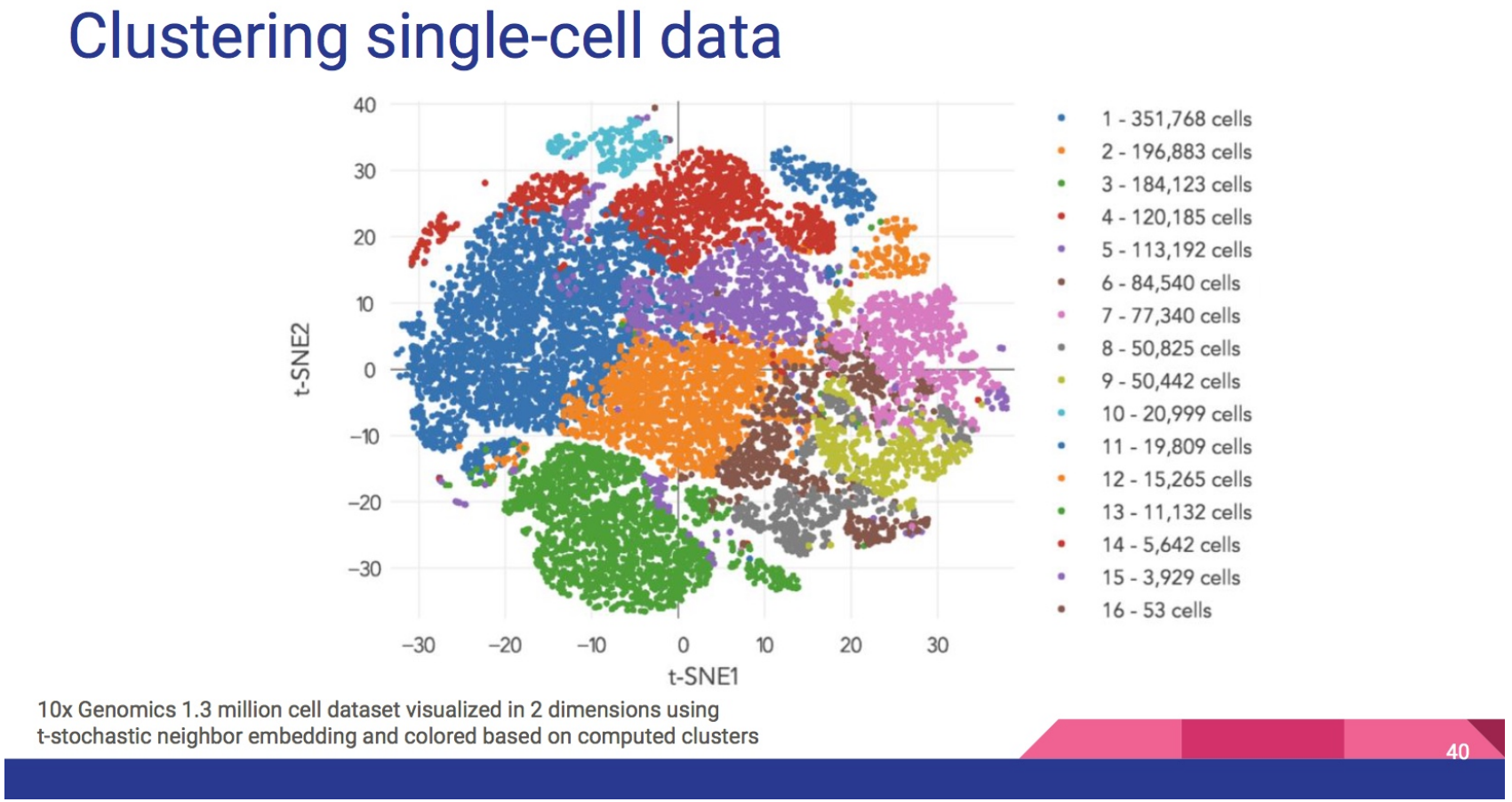
executed in 245ms, finished 17:08:28 2021-10-20

```
Text(0, 0.5, 'Principal component 2')
```



Non-linear dimensionality reduction: comp bio application

PCA is widely used in practice, but some problems are not well-suited to linear dimensionality reduction, such as problems in computational genomics. Here, t-SNE is used to visualize single-cell data.



Source: Elham Azizi, BMEN 4480: Statistical Machine Learning for Genomics