INTRODUCTION

To Data Science

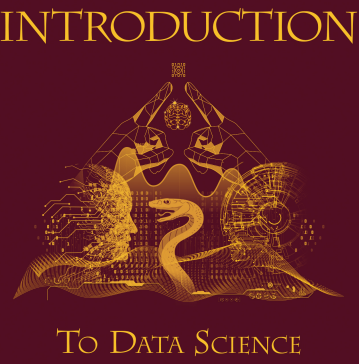
Introduction to Data Science

Unsupervised Learning: Cluster Analysis - Class 15

Giora Simchoni

gsimchoni@gmail.com and add #intro2ds in subject

Stat. and OR Department, TAU



Intro. to Unsupervised Learning

INTRODUCTION

To Data Science

From Supervised to Unsupervised

- Recall: each observation is made of a vector $x \in \mathcal{X}$ (for example) and a scalar
- Our goal is to build a model of the relationship between and :
- IID assumption: each pair is drawn indepednently from some distribution
- A modeling approach takes as input and outputs a *prediction model*
- In prediction: we get a new value and predict .
- How good is our prediction? We typically define a loss function and the quality of the model is

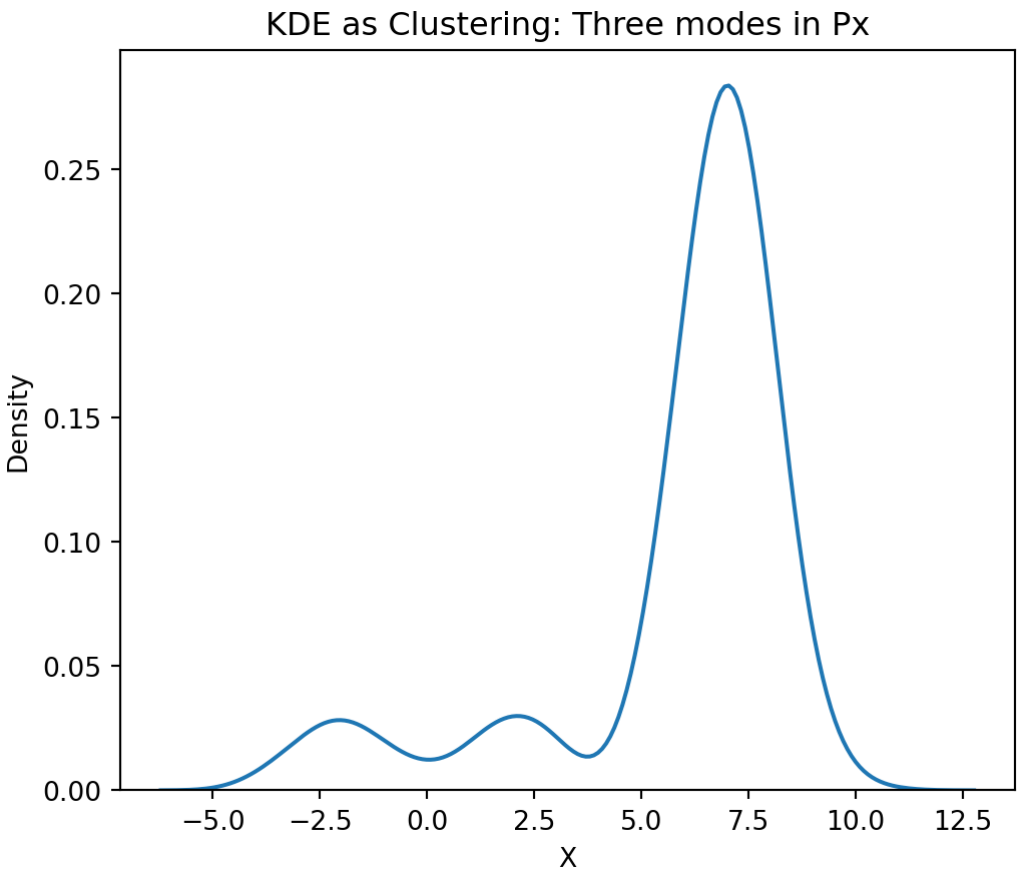
What if there is no ?

Unsupervised Learning

- Now: each observation is made of a vector (for example)
- IID assumption: each observation is drawn indepednently from some distribution
- Our goal is to *learn* distrubution (or properties of it)
- “without a supervisor”

- Example: Clustering = Finding modes of with high density
  - If we do find them, maybe can be represented by a mixture of simpler densities?
- This isn’t new, is it?

KDE as unsupervised learning



Will typically work for , above that: “curse of dimensionality”

Cluster Analysis

Group a set of observations into subsets, clusters, s.t. those within each cluster are more closely related to one another than observations assigned to different clusters

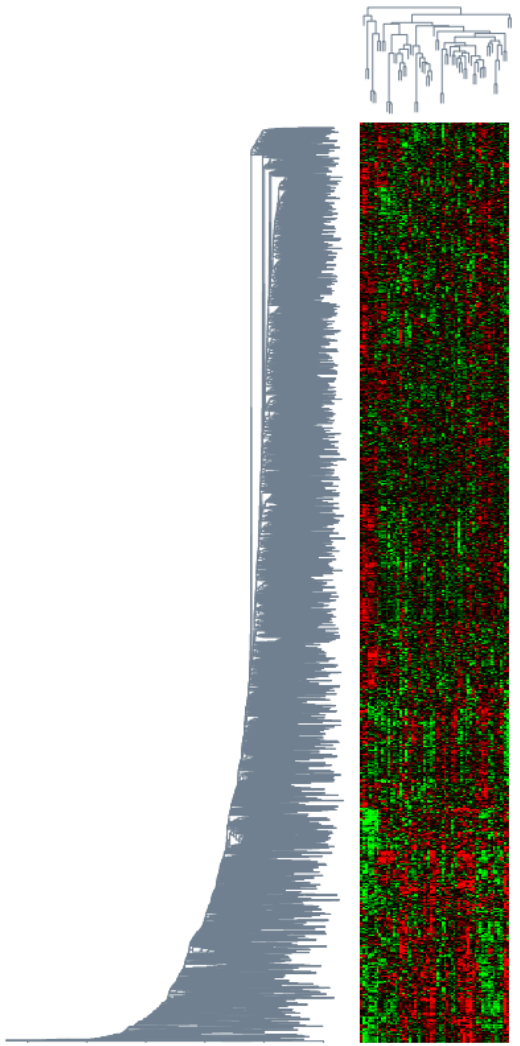
What for?

- EDA, Feature Engineering: interesting groups in the data
- Segmentation: customers, products, distribution centers location, software
- Hierarchy: diseases, evolution
- Deduplication
- Anomaly Detection

Many, many algorithms:

- Partition clustering: K-means
- Hierarchical clustering: Agglomerative
- Density-based clustering: DBSCAN

Example: Microarray Clustering



source

Unsupervised Learning Main Drawback

- Unless there is “ground truth”, no clear measure of success (as opposed to)
- Many times involves scrutinizing results and interpretation
- Not for the faint of heart

K-means Clustering

INTRODUCTION

To Data Science

How to evaluate a partition?

- Assume clusters are given
- is some function assigning cluster to observation
- is a distance metric for pair , e.g. Euclidean
- We wish to minimize the extent to which observations assigned to the same cluster tend to be close to one another

- The “within cluster” scatter/loss:
- equivalent to maximizing
- Can we go over all possible to find the global minimum?

Road to K-means

- Euclidean distance:
- Can show that:
- being the mean in cluster , and number of observations in cluster

- But for any set of observations , which would minimize ?
- Thus, the final goal of K-means:

K-means

0. Start with initial guess for

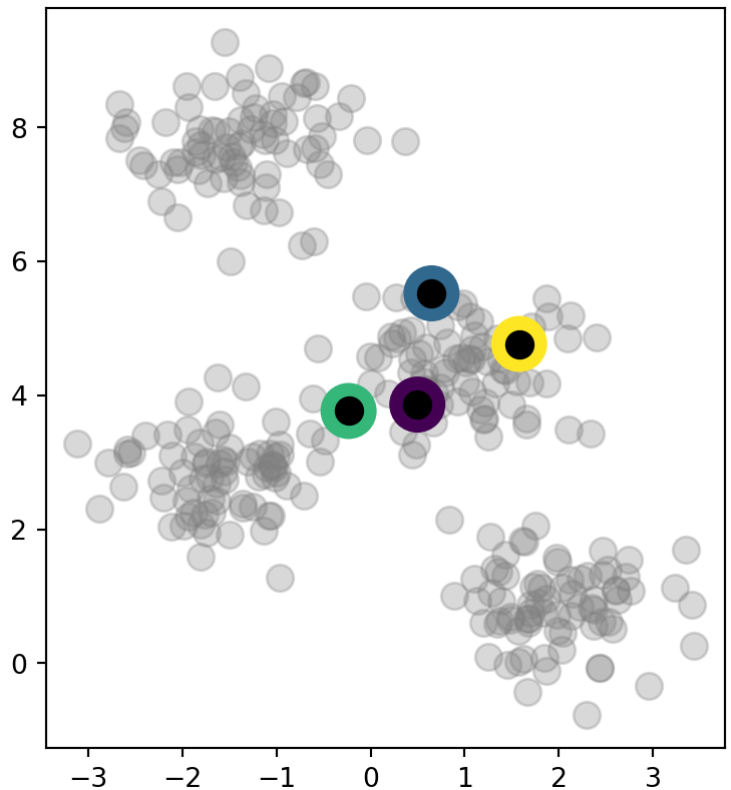
1. Assign each observation to the closest cluster mean. That is:

2. Update means . That is the centroids:

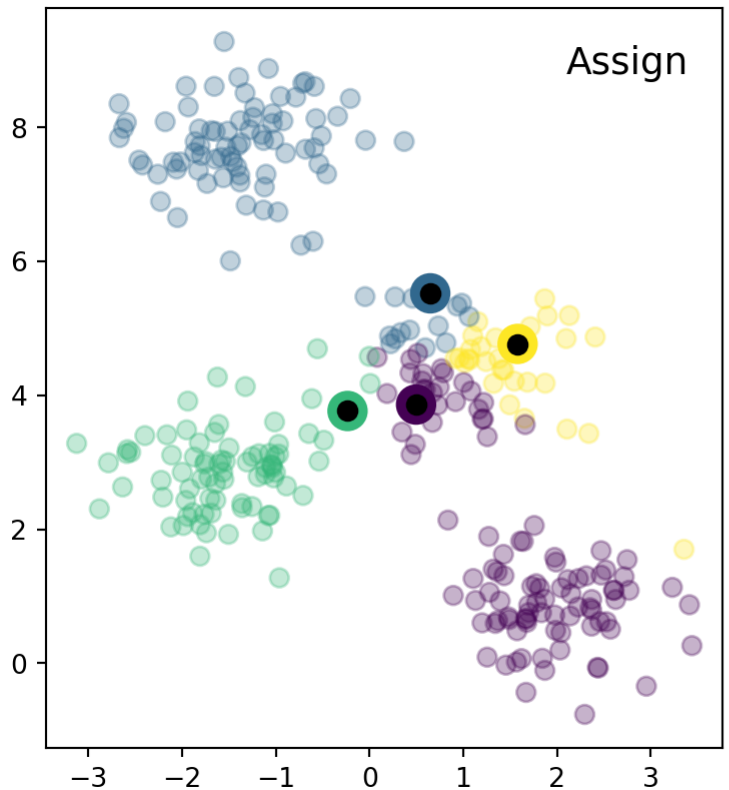
3. Repeat 1 and 2 until doesn’t change

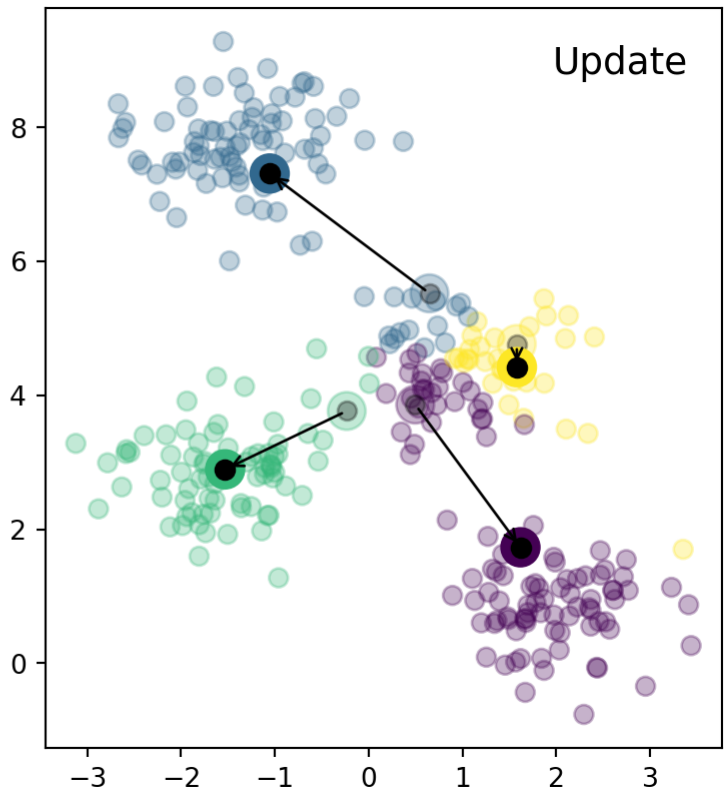
- Convergence is guaranteed (steps 1 and 2 can only reduce)
- Global optimum is NOT guaranteed
- Can try many different initial starting points

K-means Demo: Initial Guess

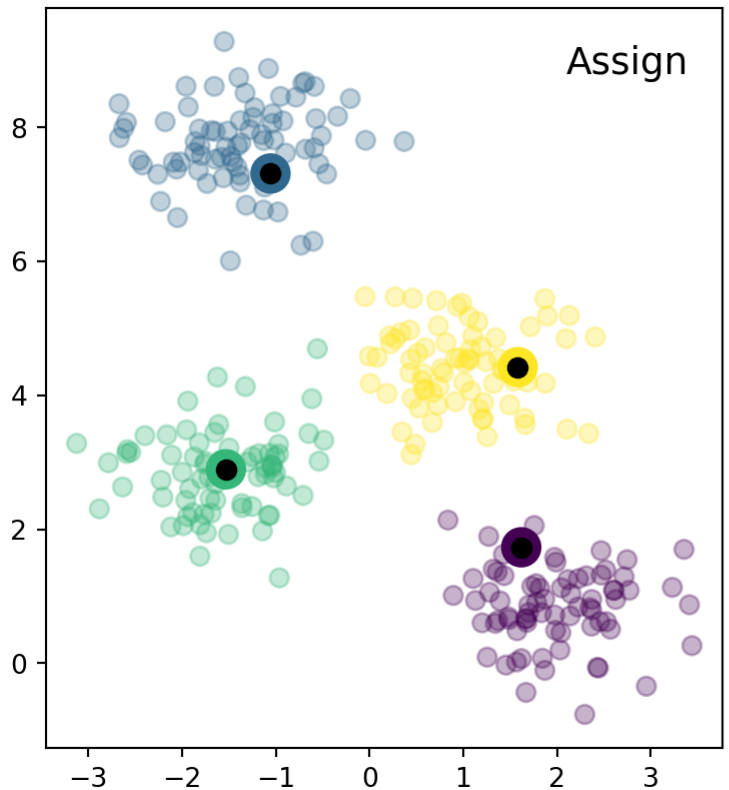


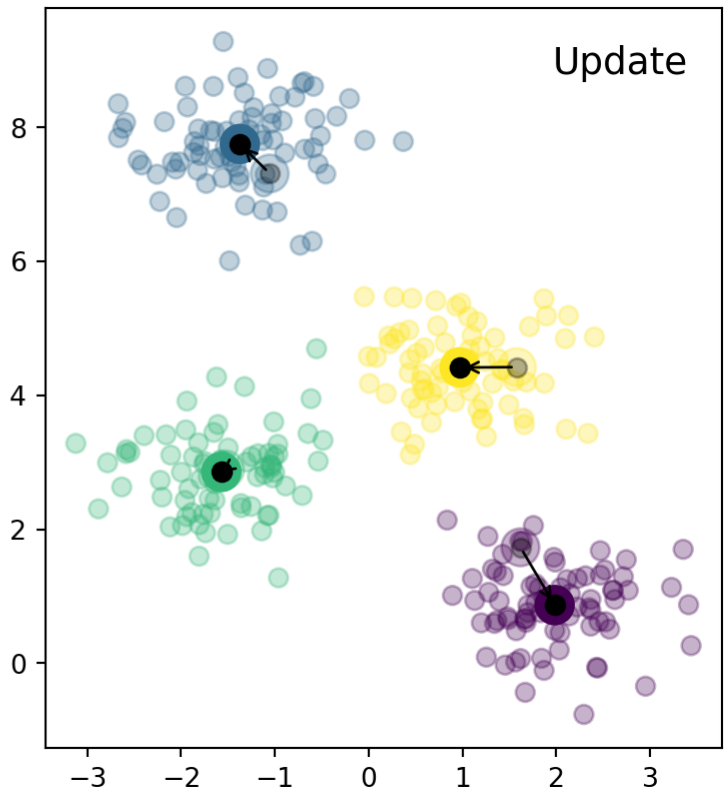
K-means Demo: Iteration 1



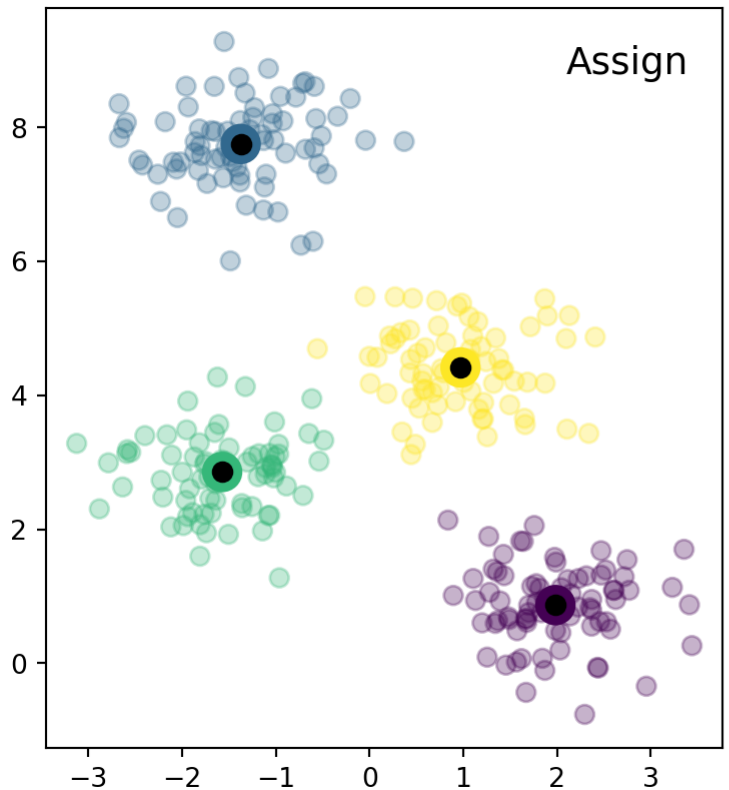


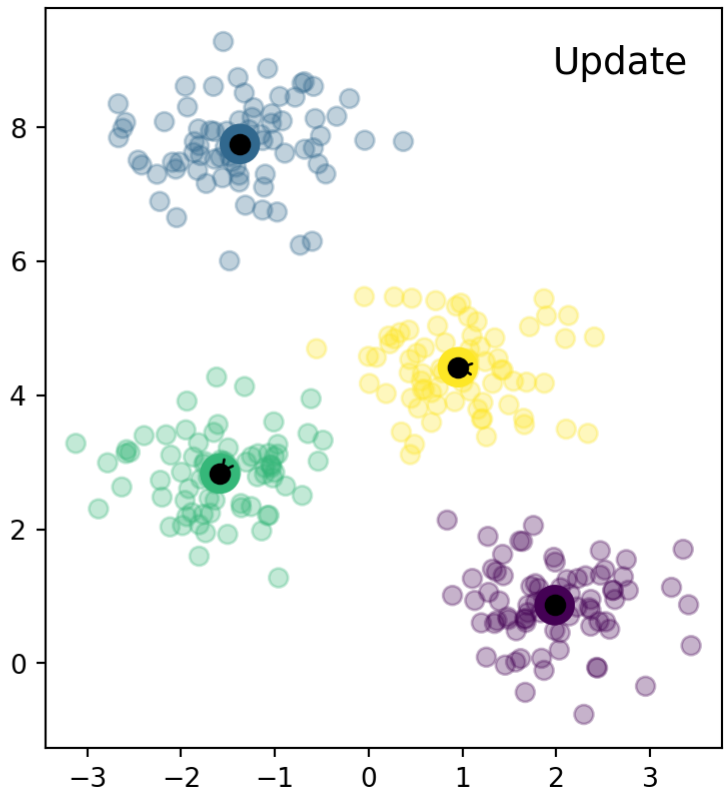
K-means Demo: Iteration 2





K-means Demo: Iteration 3





K-means on Netflix

```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=2, random_state=0)
_ = kmeans.fit(NE_Xtr)
```

What did we get?

```
print(kmeans.cluster_centers_.shape)
```

```
(2, 14)
```

```
print(kmeans.labels_[:10])
```

```
[1 0 0 1 1 1 0 0 0 1]
```

```
print(f'{kmeans.inertia_:.2f}')
```

```
68827.01
```

Can easily “predict”:

```
test_labels = kmeans.predict(NE_Xte)
test_labels[:10]
```

```
array([0, 1, 0, 0, 1, 0, 1, 0, 0, 0])
```

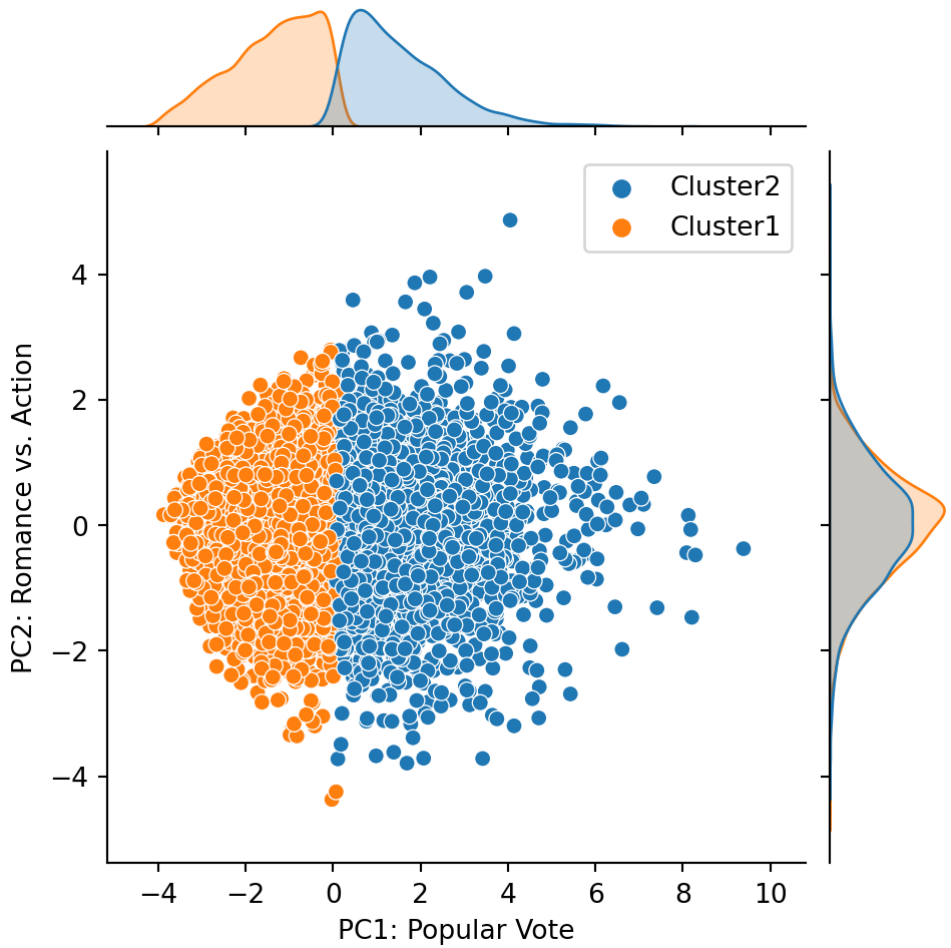
K-means on Netflix: the Centroids

```
pd.DataFrame({'title': kmeans.feature_names_in_,
    'mean_score': NE_Xtr.mean(axis = 0),
    'm_1': kmeans.cluster_centers_[0],
    'm_2': kmeans.cluster_centers_[1]}).set_index('title').head(8).round(2)
```

	mean_score	m_1	m_2
title			
Independence Day	4.14	4.48	3.78
The Patriot	4.11	4.46	3.72
The Day After Tomorrow	3.70	4.15	3.20
Pirates of the Caribbean	4.35	4.57	4.11
Pretty Woman	4.08	4.39	3.73
Forrest Gump	4.51	4.67	4.34
The Green Mile	4.46	4.69	4.20
Con Air	3.72	4.13	3.27

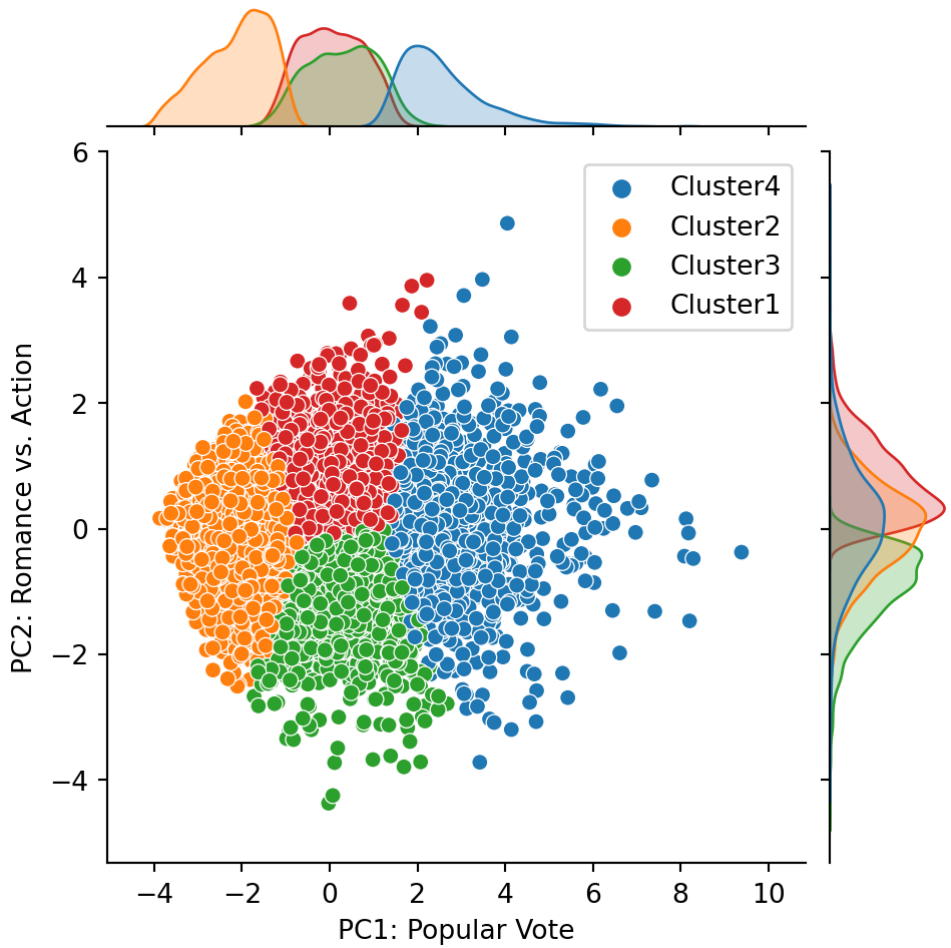
K-means on Netflix: “discrete” first PC!

▶ Code



K-means on Netflix: higher

▶ Code



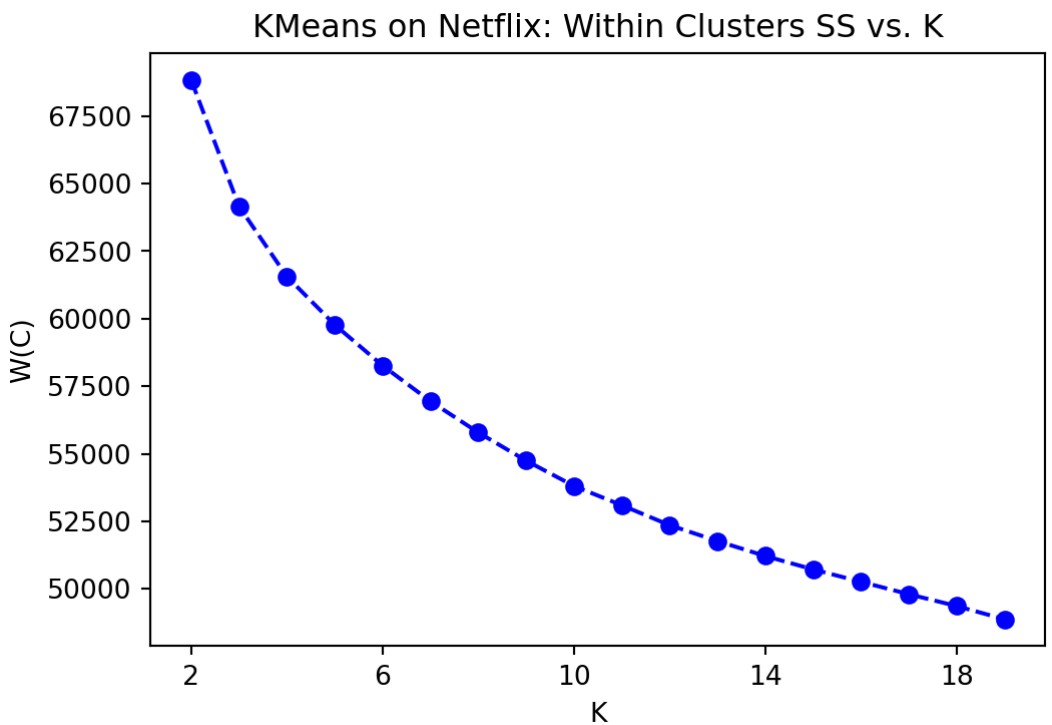
K-means Issues

INTRODUCTION

TO DATA SCIENCE

How to choose ?

```
W_C = []
for K in range(2, 20):
    kmeans = KMeans(n_clusters=K)
    kmeans.fit(NE_Xtr)
    W_C.append(kmeans.inertia_)
```



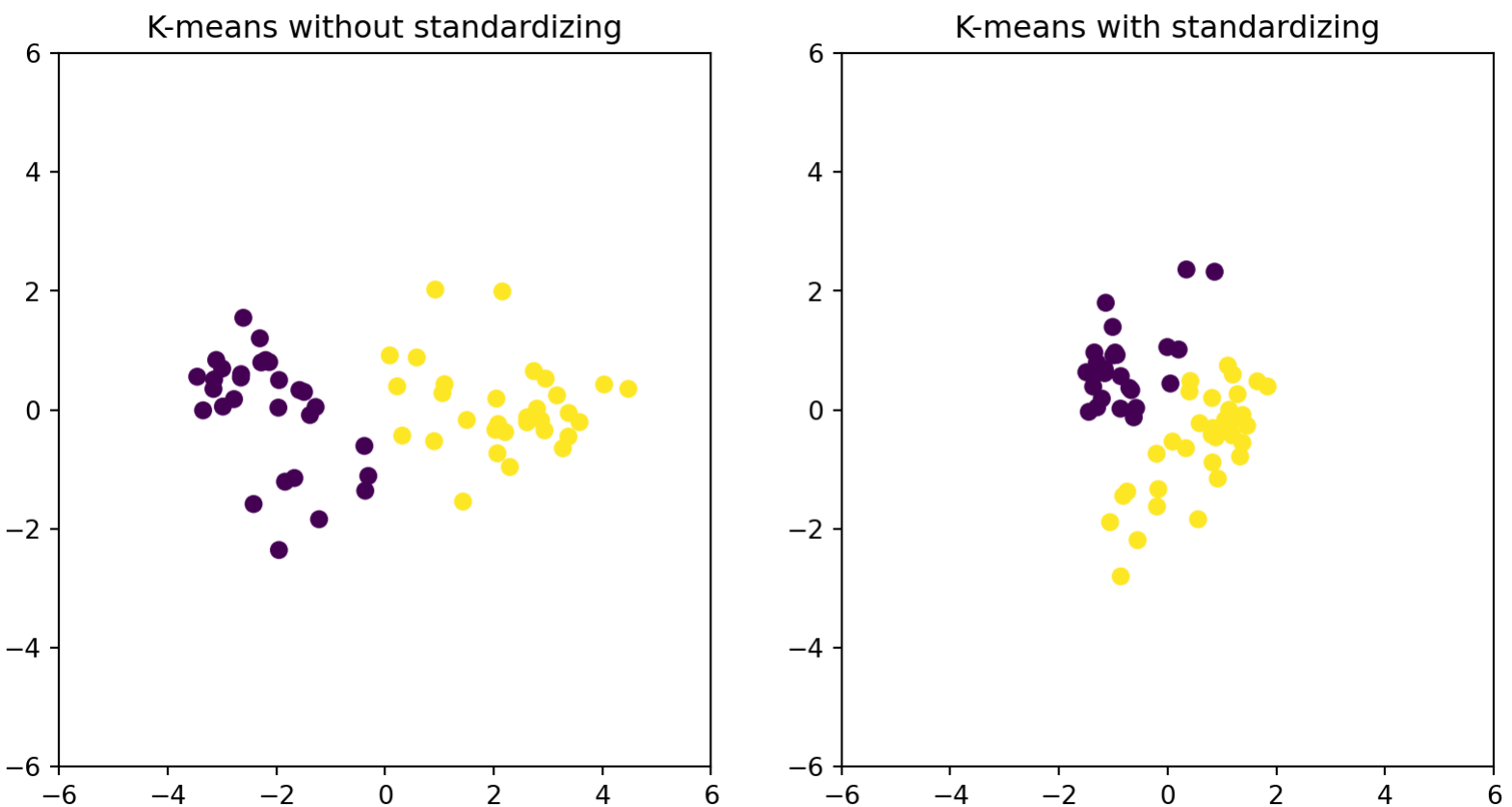
The “elbow” method won’t always work, there are others.

Should you always standardize?

As with KNN, K-means would be higly influenced by a feature with high variance.

But what if *that* feature is important for clustering?

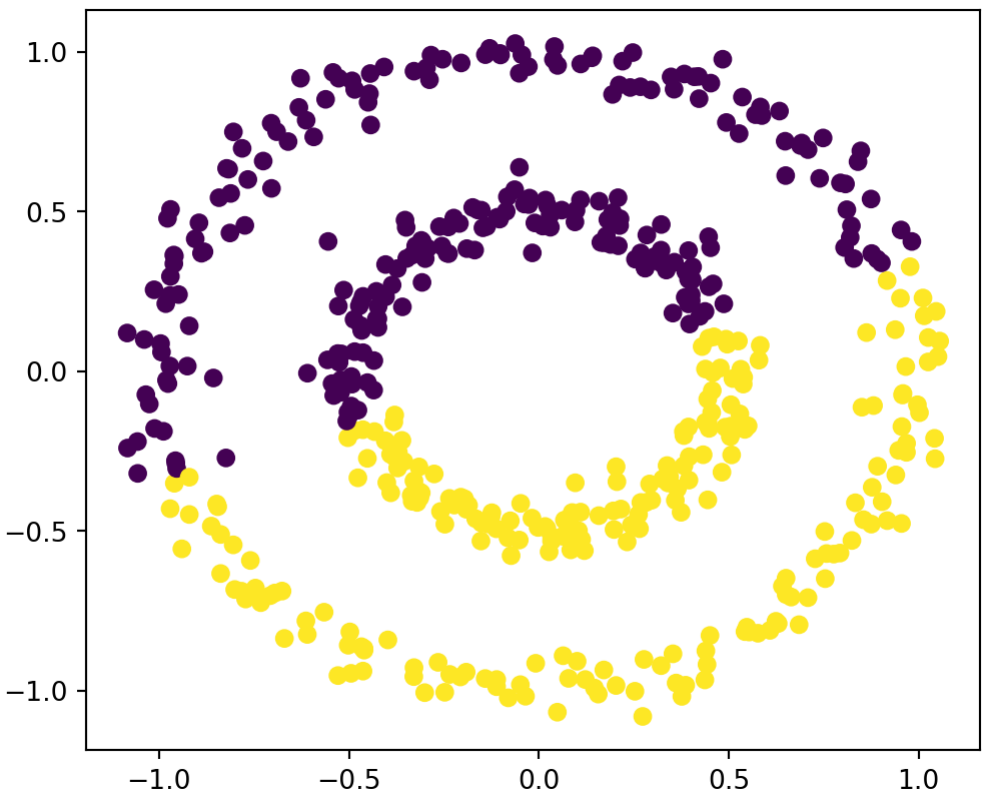
▶ Code



K-means failures (I)

Prefers separable spherical clusters (Gaussian).

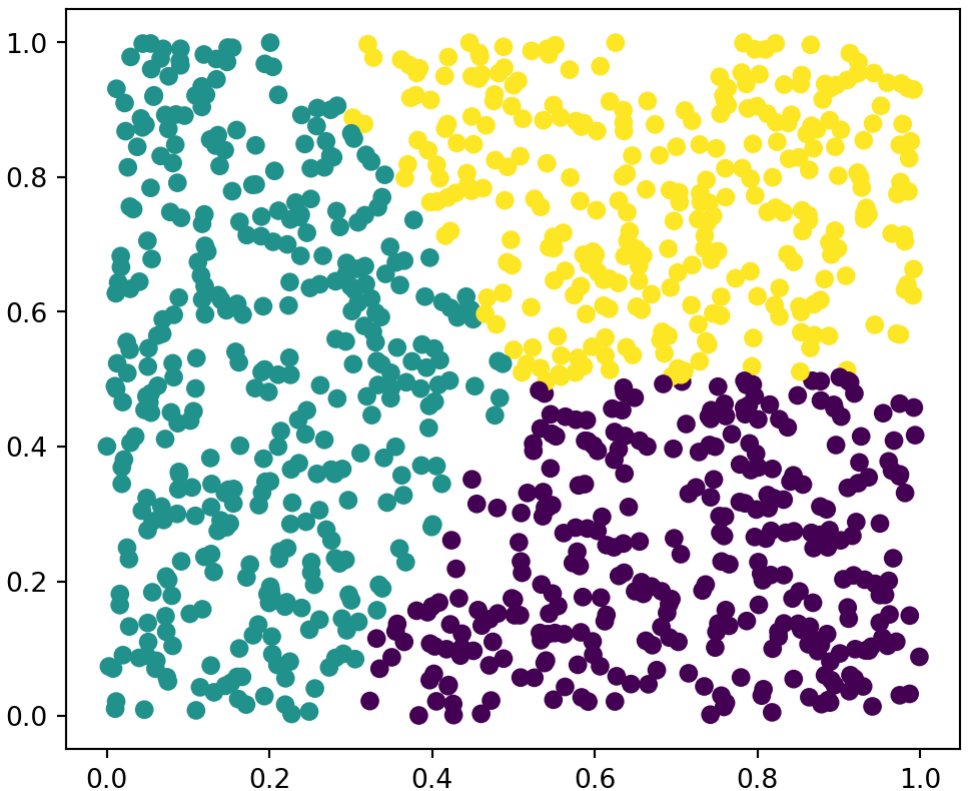
▶ Code



K-means failures (II)

Always specify .

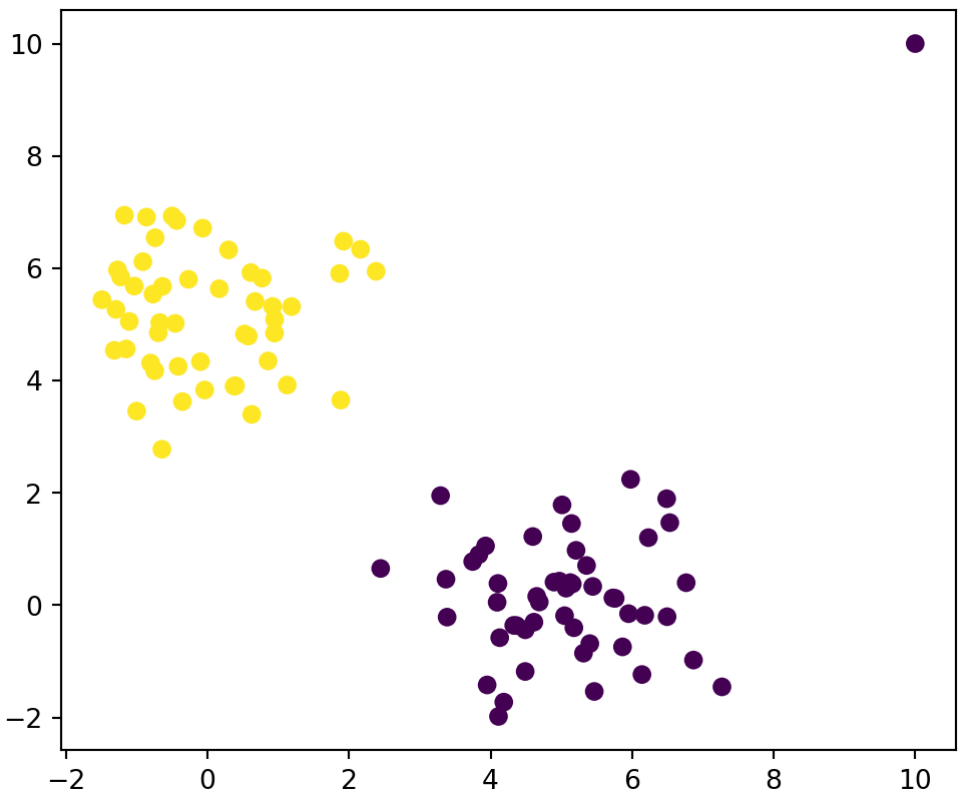
▶ Code



K-means failures (III)

No concept of outliers.

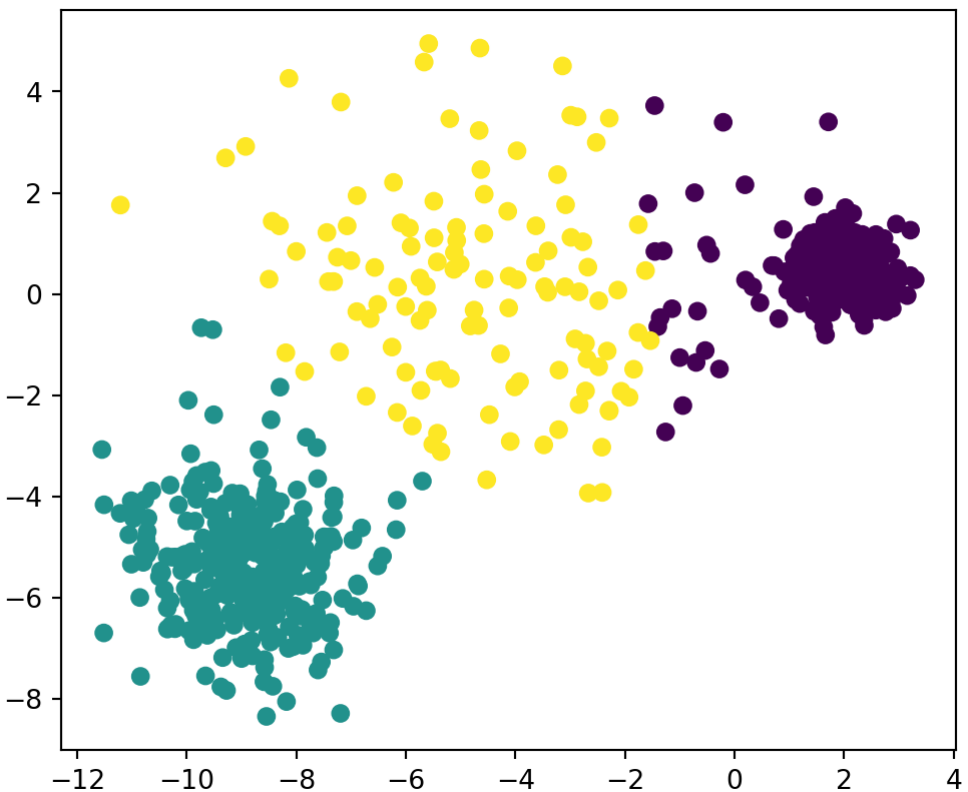
▶ Code



K-means failures (IV)

Bad with unequal densities, unequal cluster sizes.

▶ Code



DBSCAN

INTRODUCTION

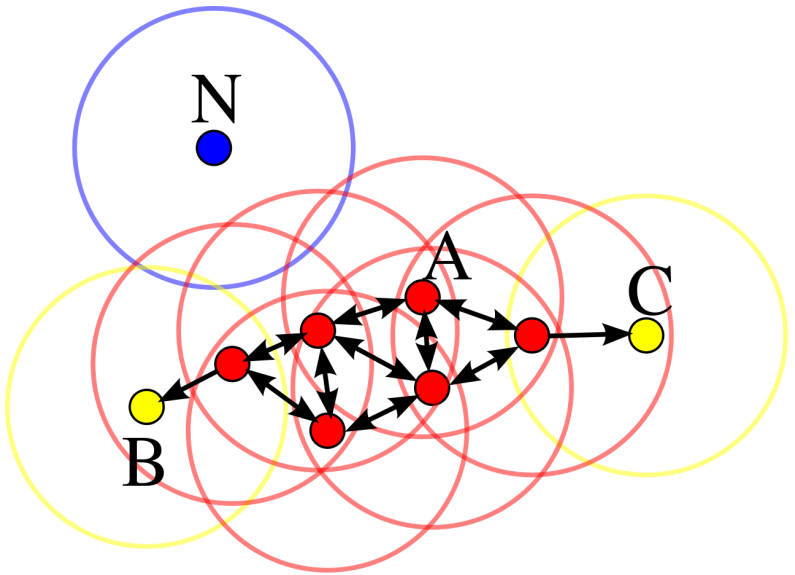
To Data Science

Density Based Clustering

- Back to premise, data comes from a distribution:
- Estimating is hard
- Find high-density regions through *connected components*
- Immediate Pros:
  - No specifying
  - Outliers
  - More complex clustering structures

DBSCAN

Density-Based Spatial Clustering of Applications with Noise



- 3 types of points: core, border, noise
- Core points are high-density points (parameters: (radius),)
- Connect core points into clusters
- Assign border points to clusters
- All else: noise

DBSCAN - Abstract

ALGORITHM 2: Abstract DBSCAN Algorithm

```
1 Compute neighbors of each point and identify core points      // Identify core points
2 Join neighboring core points into clusters                    // Assign core points
3 foreach non-core point do
4 |   Add to a neighboring core point if possible               // Assign border points
5 |   Otherwise, add to noise                                   // Assign noise points
```

Nice visualiztion

DBSCAN - Actual

ALGORITHM 1: Pseudocode of Original Sequential DBSCAN Algorithm

Input: *DB*: Database
Input: ε: Radius
Input: *minPts*: Density threshold
Input: *dist*: Distance function
Data: *label*: Point labels, initially *undefined*

```
1  foreach point p in database DB do                          // Iterate over every point
2      if label(p) ≠ undefined then continue                  // Skip processed points
3      Neighbors N ← RangeQuery(DB, dist, p, ε)               // Find initial neighbors
4      if |N| < minPts then                                   // Non-core points are noise
5          label(p) ← Noise
6          continue
7      c ← next cluster label                                 // Start a new cluster
8      label(p) ← c
9      Seed set S ← N \ {p}                                   // Expand neighborhood
10     foreach q in S do
11         if label(q) = Noise then label(q) ← c
12         if label(q) ≠ undefined then continue
13         Neighbors N ← RangeQuery(DB, dist, q, ε)
14         label(q) ← c
15         if |N| < minPts then continue                      // Core-point check
16         S ← S ∪ N
```

DBSCAN on Netflix

```
from sklearn.cluster import DBSCAN

dbscan = DBSCAN(eps=1.0, min_samples=10, metric='euclidean')
_ = dbscan.fit(NE_Xtr)
```

What did we get?

```
print(dbscan.core_sample_indices_.shape)
```

```
(305,)
```

```
print(dbscan.labels_[:10])
clusters, counts = np.unique(dbscan.labels_, return_counts=True)
d = dict(zip(clusters, counts))
print(d)
print(f'no. of noise points: {np.sum(dbscan.labels_ == -1)}')
```

```
[-1 -1 -1 -1 -1 -1  0 -1  0 -1]
{-1: 7390, 0: 494, 1: 109, 2: 7}
no. of noise points: 7390
```

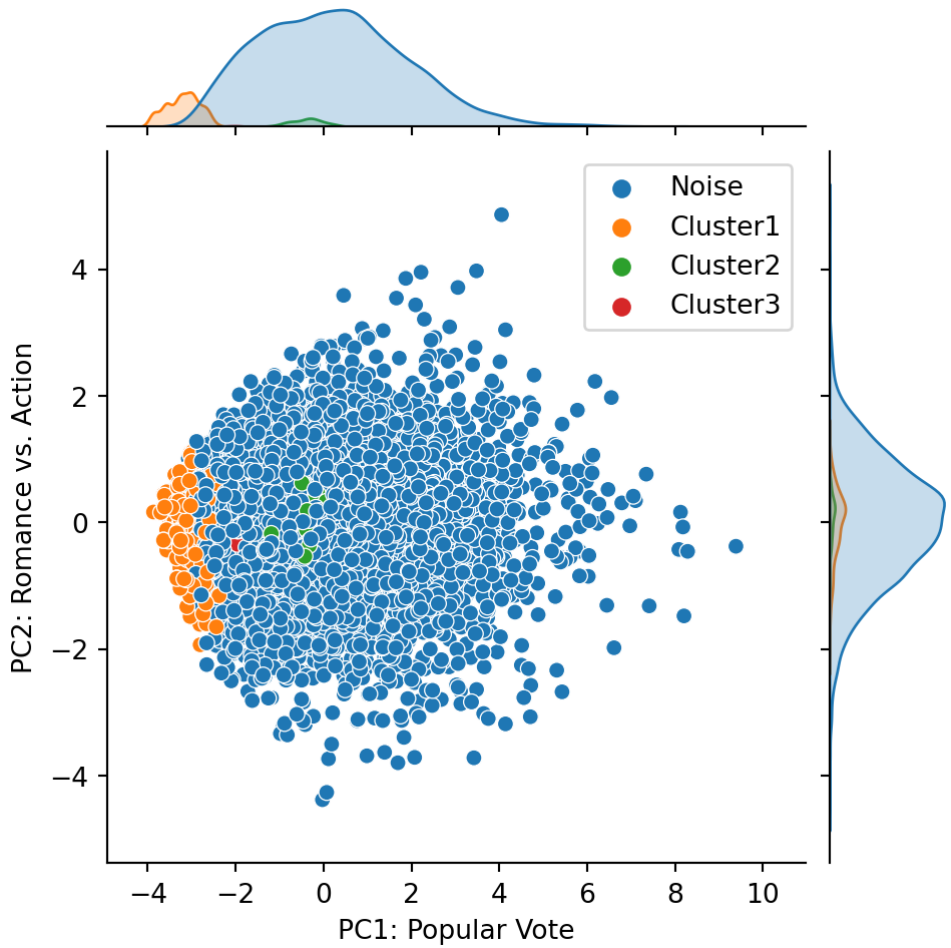
No concept of prediction!

```
test_labels = dbscan.predict(NE_Xte)
```

```
AttributeError: 'DBSCAN' object has no attribute 'predict'
```

DBSCAN on Netflix: Weak relation to PCA?

▶ Code



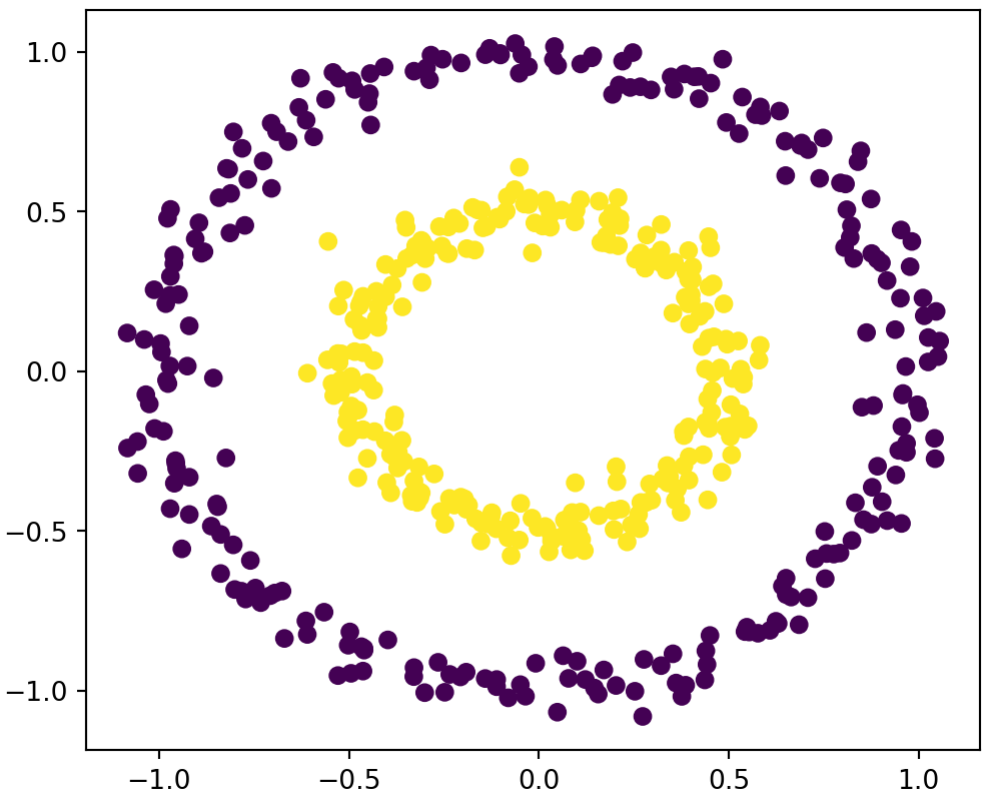
DBSCAN on K-means Failures

INTRODUCTION

TO DATA SCIENCE

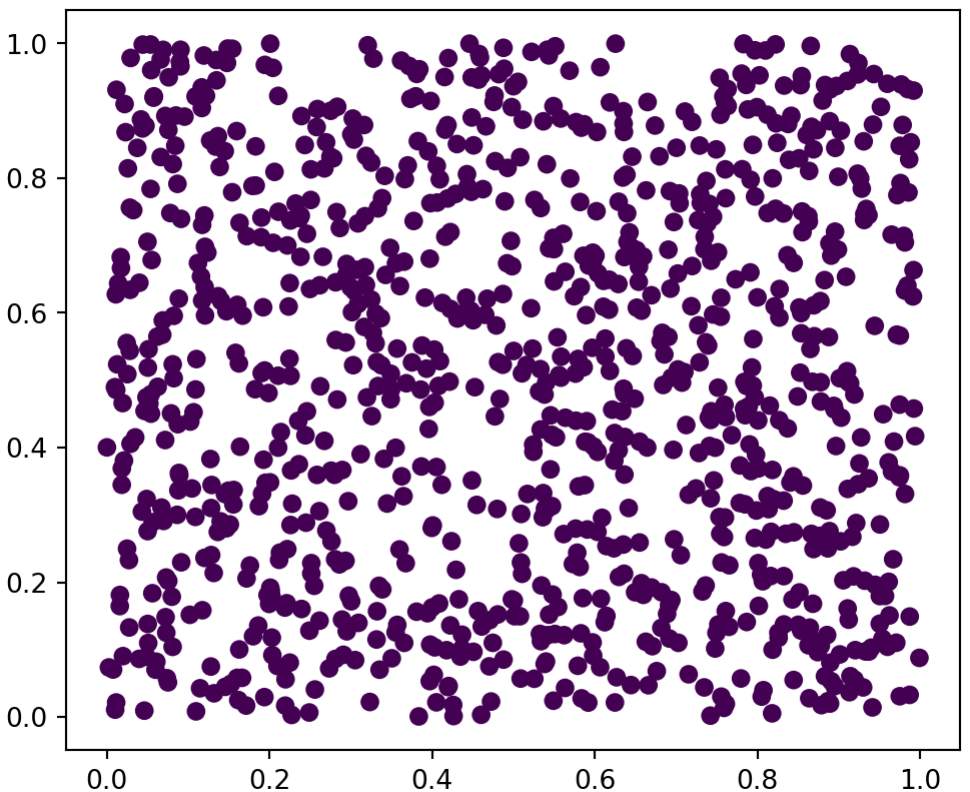
DBSCAN on K-means Failures (I)

▶ Code



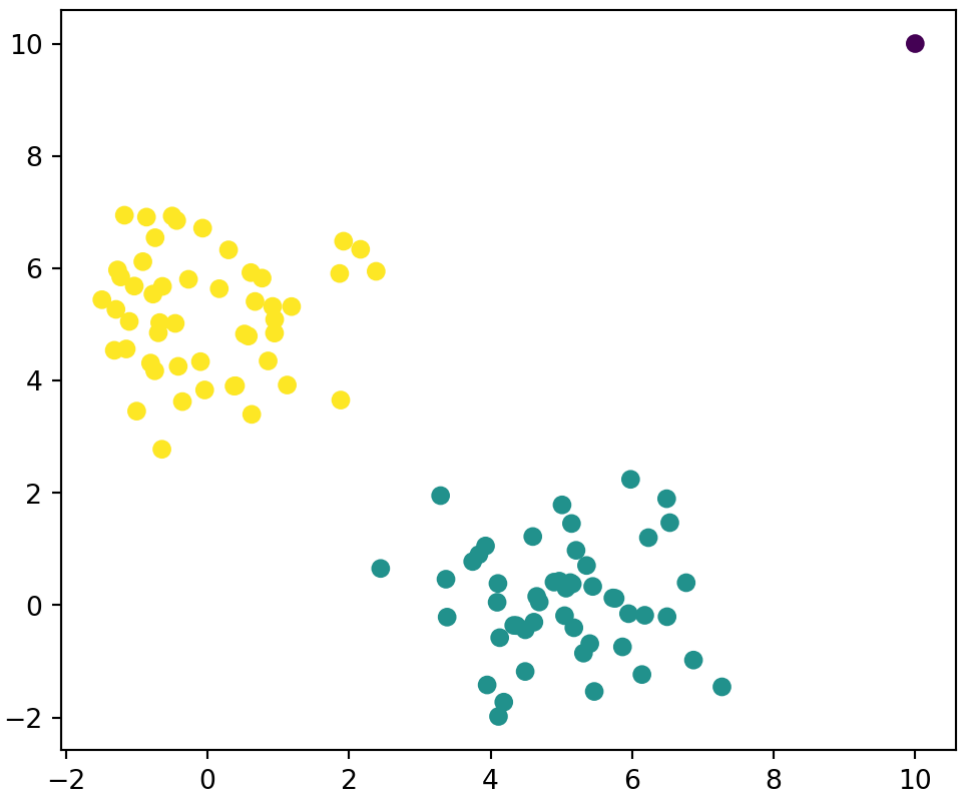
DBSCAN on K-means Failures (II)

▶ Code



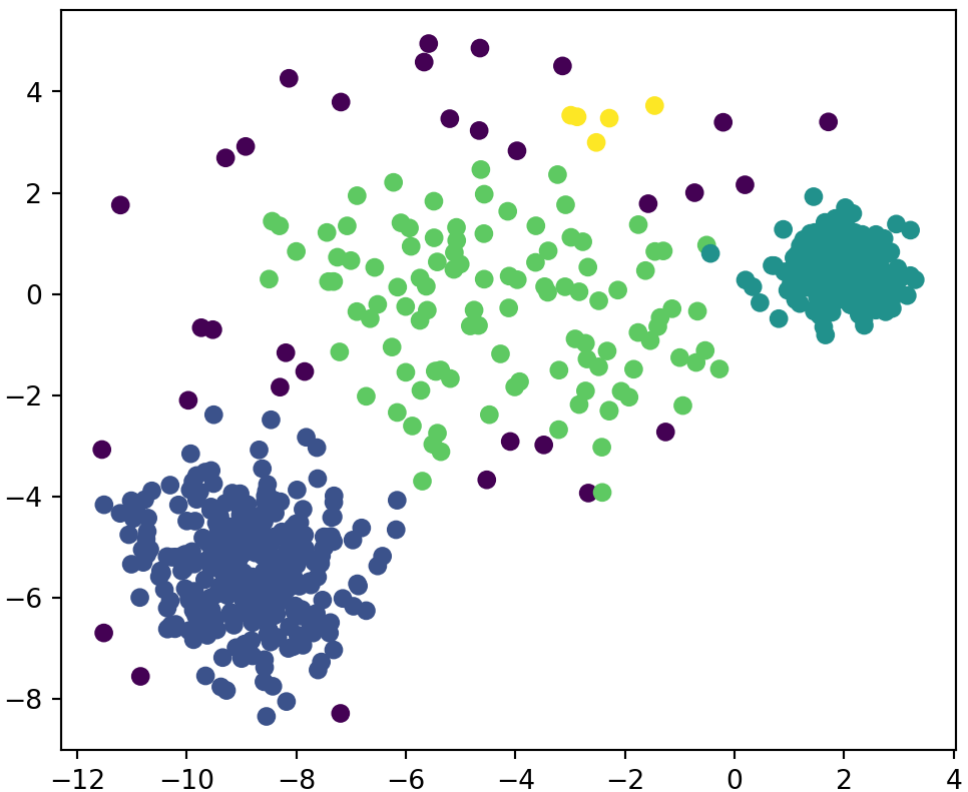
DBSCAN on K-means Failures (III)

▶ Code



DBSCAN on K-means Failures (IV)

▶ Code



Method	Pros	Cons
K-means	Faster, Scalable, Simple Related to other methods (PCA, EM, GMM)	Need Separable spherical clusters No outliers Bad with unequal densities Only Euclidean distance
DBSCAN	Complex structures No need of Any distance metric Outliers	Slower Very sensitive to Bad with unequal densities

Clustering after Dimensionality Reduction

INTRODUCTION

TO DATA SCIENCE

The FNIST Dataset

```
from tensorflow.keras.datasets import fashion_mnist
from sklearn.decomposition import PCA

(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()
X_train = X_train.astype(np.float32) / 255
X_test = X_test.astype(np.float32) / 255

print(X_train.shape)
print(y_train.shape)
```

```
(60000, 28, 28)
(60000,)
```

```
y_train[:10]
```

```
array([9, 0, 0, 3, 0, 2, 7, 2, 5, 5], dtype=uint8)
```

▶ Code

K-means on FNIST

```
X_train_flat = X_train.reshape((X_train.shape[0], -1))
print(X_train_flat.shape)
```

```
(60000, 784)
```

```
kmeans = KMeans(n_clusters=10, random_state=0)
kmeans.fit(X_train_flat)

print(pd.crosstab(y_train, kmeans.labels_).rename(index=fnist_dict))
```

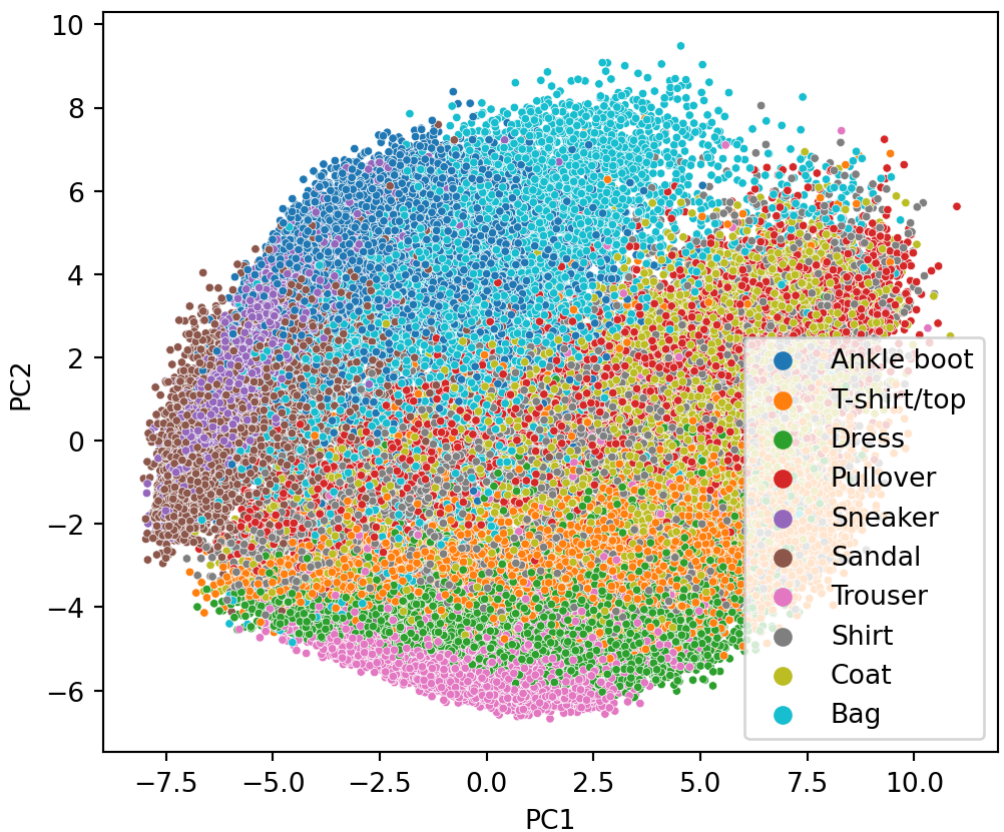
col_0	0	1	2	3	4	5	6	7	8	9
row_0										
T-shirt/top	201	23	3404	0	593	2	167	1583	27	0
Trouser	5413	3	236	0	156	0	63	129	0	0
Pullover	9	27	115	0	515	1	3519	1787	27	0
Dress	3209	5	1684	0	522	0	49	524	7	0
Coat	154	15	873	0	252	0	3596	1081	29	0
Sandal	1	4	2	480	3766	1444	0	29	13	261
Shirt	62	62	1053	0	774	6	1954	2071	17	1
Sneaker	0	1	0	785	508	4680	0	0	0	26
Bag	28	2201	22	66	494	237	269	228	2449	6
Ankle boot	2	4	2	2926	170	164	1	38	0	2693

Metrics for evaluating clustering vs. ground truth: Rand index, Mutual information, V-measure…

Recall: PCA might discover clusters

```
X_train_flat_centered = X_train_flat - X_train_flat.mean(axis=0)
pca = PCA(n_components = 2)
pca.fit(X_train_flat_centered)
T = pca.transform(X_train_flat_centered)
```

▶ Code



Does it improve clustering?

```
kmeans = KMeans(n_clusters=10, random_state=0)
kmeans.fit(T)

print(pd.crosstab(y_train, kmeans.labels_).rename(index=fnist_dict))
```

col_0	0	1	2	3	4	5	6	7	8	9
row_0										
T-shirt/top	216	73	371	16	2887	5	1204	205	9	1014
Trouser	26	5	49	3	165	1	1557	47	0	4147
Pullover	2286	216	1393	49	121	6	334	1560	24	11
Dress	31	9	149	1	1363	0	1662	69	0	2716
Coat	2225	88	698	48	773	5	291	1639	3	230
Sandal	0	3661	133	13	1	281	28	8	1874	1
Shirt	1256	251	1314	52	795	14	830	1130	35	323
Sneaker	0	2422	2	13	0	472	0	1	3090	0
Bag	256	164	1151	2481	16	1008	244	287	382	11
Ankle boot	2	61	248	1166	0	3451	6	151	914	1

Increasing latent dimension

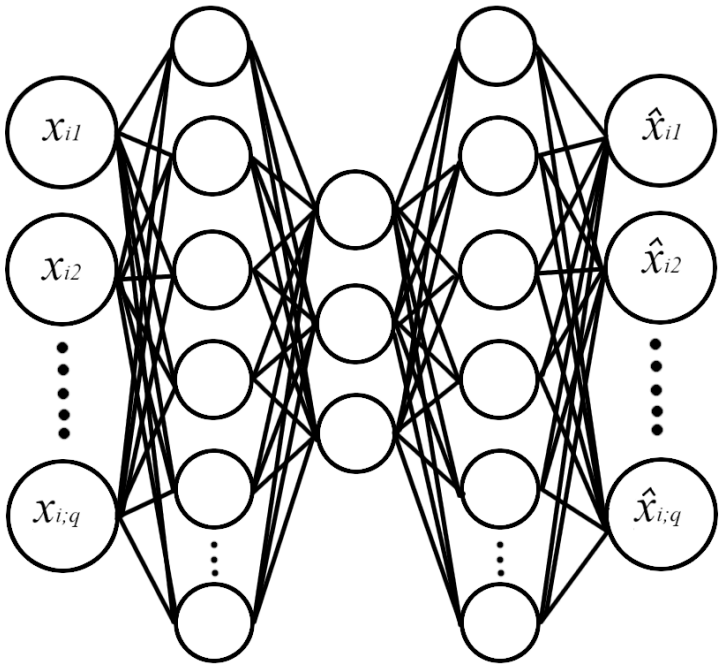
```
pca = PCA(n_components = 30)
pca.fit(X_train_flat_centered)
T = pca.transform(X_train_flat_centered)

kmeans = KMeans(n_clusters=10, random_state=0)
kmeans.fit(T)

print(pd.crosstab(y_train, kmeans.labels_).rename(index=fnist_dict))
```

col_0	0	1	2	3	4	5	6	7	8	9
row_0										
T-shirt/top	200	596	167	22	0	3403	2	0	31	1579
Trouser	5411	158	63	3	0	237	0	0	0	128
Pullover	9	518	3517	29	1	116	1	0	27	1782
Dress	3203	525	49	5	0	1685	0	0	7	526
Coat	153	254	3600	16	0	874	0	0	29	1074
Sandal	1	3751	0	4	264	2	1448	488	13	29
Shirt	60	777	1953	64	1	1056	7	0	17	2065
Sneaker	0	509	0	1	23	0	4703	764	0	0
Bag	26	493	266	2215	6	22	240	66	2438	228
Ankle boot	3	168	1	4	2639	2	176	2971	0	36

Autoencoders



- A branch of unsupervised learning: Representation Learning
- How do we learn representations useful for downstream tasks (e.g. clustering)
- Most basic AE: Encode via **encoder** network to lower dimension , decode with **decoder** network back to dimension , such that:
- hopefully *represents* the data well

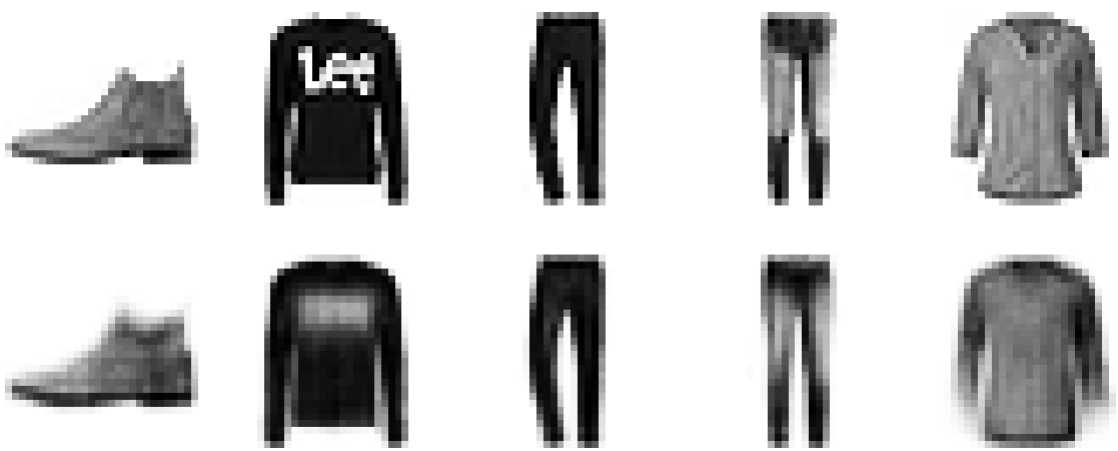
AE with Keras

```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, Flatten, Reshape

stacked_encoder = Sequential([
    Flatten(input_shape=[28, 28]),
    Dense(100, activation="relu"),
    Dense(30, activation="relu"),
])
stacked_decoder = Sequential([
    Dense(100, activation="relu", input_shape=[30]),
    Dense(28 * 28, activation="sigmoid"), # make output 0-1
    Reshape([28, 28])
])
stacked_ae = Sequential([stacked_encoder, stacked_decoder])
stacked_ae.compile(loss='mse', optimizer='adam')
history = stacked_ae.fit(X_train, X_train, epochs=20, verbose=0)
```

Can it reconstruct?

```
def show_reconstructions(sae, images, n_images=5):
  reconstructions = sae.predict(images[:n_images], verbose=0)
  fig = plt.figure(figsize=(n_images * 1.5, 3))
  for image_index in range(n_images):
      plt.subplot(2, n_images, 1 + image_index)
      plt.imshow(images[image_index], cmap='binary')
      plt.axis('off')
      plt.subplot(2, n_images, 1 + n_images + image_index)
      plt.imshow(reconstructions[image_index], cmap='binary')
      plt.axis('off')
show_reconstructions(stacked_ae, X_test)
plt.show()
```



Does it improve clustering?

```
T = stacked_encoder.predict(X_train, verbose=0)

print(T.shape)
```

```
(60000, 30)
```

```
kmeans = KMeans(n_clusters=10, random_state=0)
kmeans.fit(T)

print(pd.crosstab(y_train, kmeans.labels_).rename(index=fnist_dict))
```

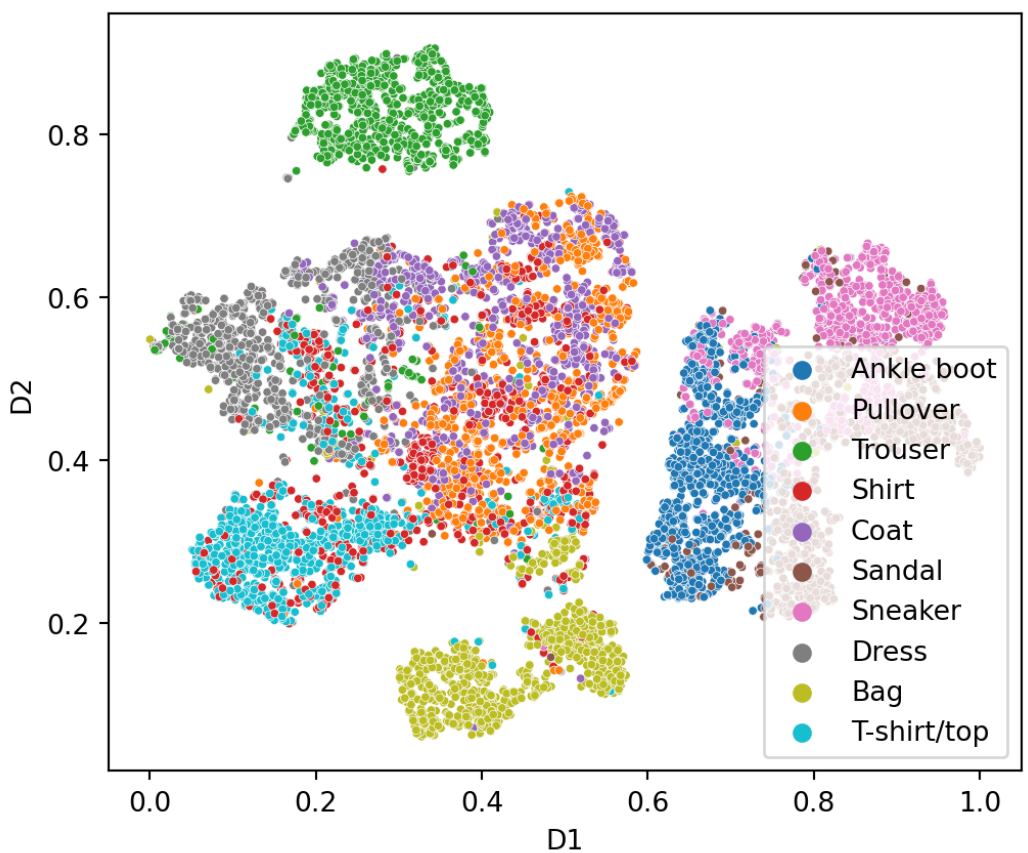
```
col_0           0     1     2     3     4     5     6     7     8     9
row_0
T-shirt/top  1815  3316    83     1    30     6     0   126     0   623
Trouser       404     4    49     0     1  4556     0     6     0   980
Pullover     1930    33  3401     0    27     0     1   511     0    97
Dress         931   111    89     0     3   721     0    11     0  4134
Coat         1085     5  3616     0    17     8     0   235     0  1034
Sandal        848     0     0  3032    83     0  1247     2   788     0
Shirt        2695   779  1662     0    55     5     0   356     0   448
Sneaker       215     0     0  4551    59     0    35     0  1140     0
Bag           952    12    45    62  2331     1     5  2240     3   349
Ankle boot    192     0     1   116    58     0  2750     0  2881     2
```

```
from sklearn.manifold import TSNE

T_te = stacked_encoder.predict(X_test, verbose=False)

tsne = TSNE(init='pca', learning_rate='auto')
X_test_2D = tsne.fit_transform(T_te)
X_test_2D = (X_test_2D - X_test_2D.min()) / (X_test_2D.max() - X_test_2D.min())
```

▶ Code



Intro to Data Science