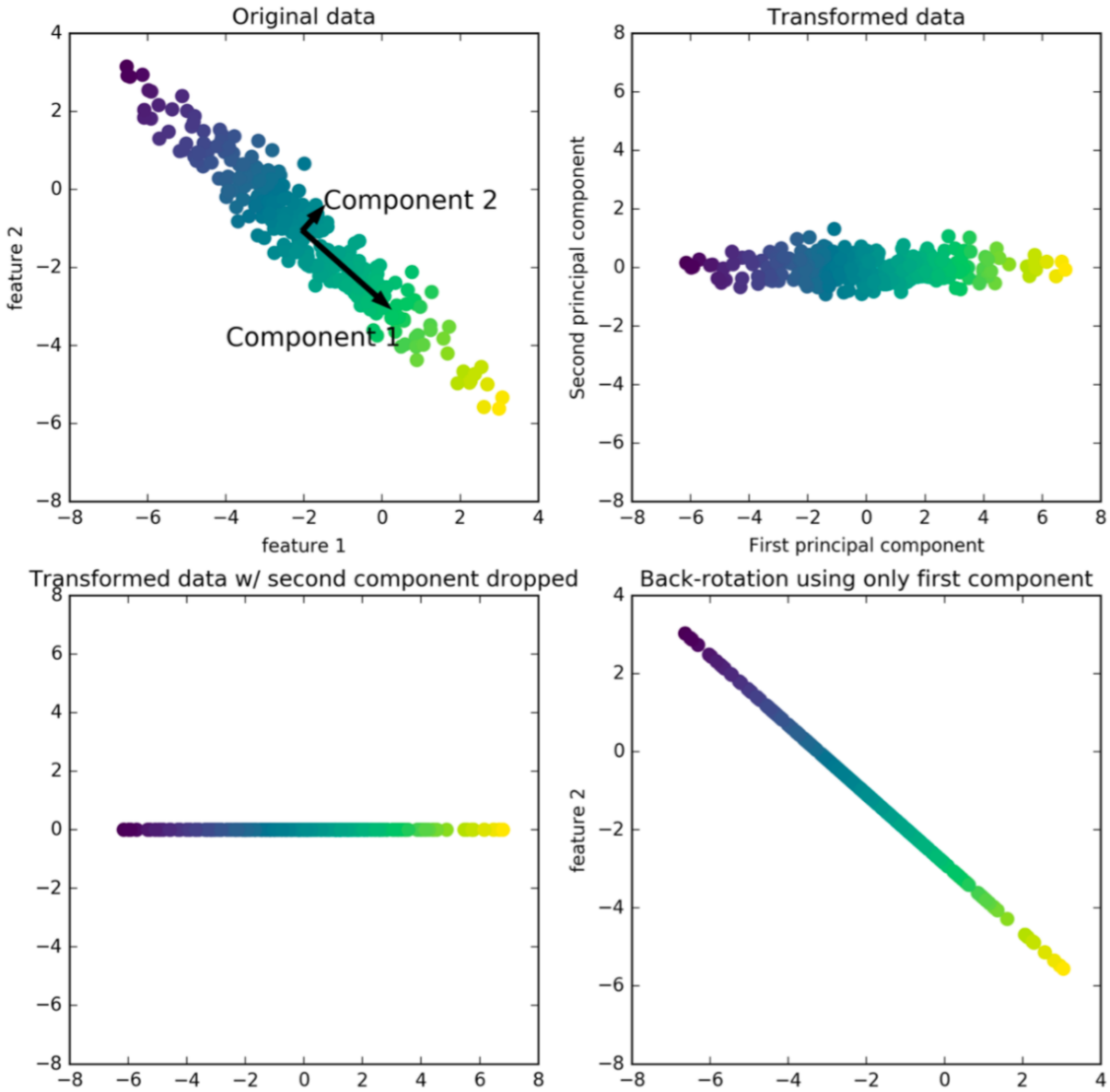
# Module-5.1-PCA and Clustering

Presented by Yasin Ceran

# Table of Contents

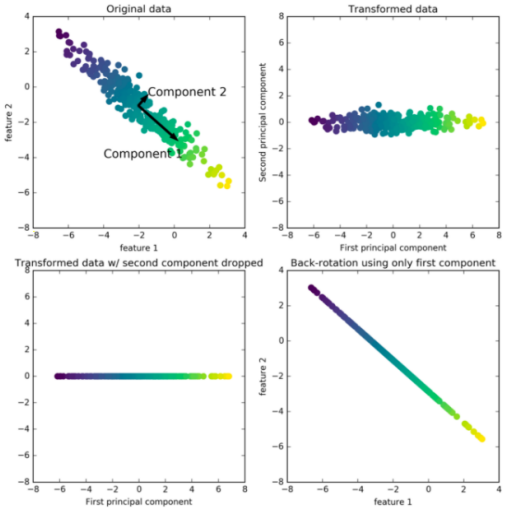
# Introduction to PCA



# PCA Objective

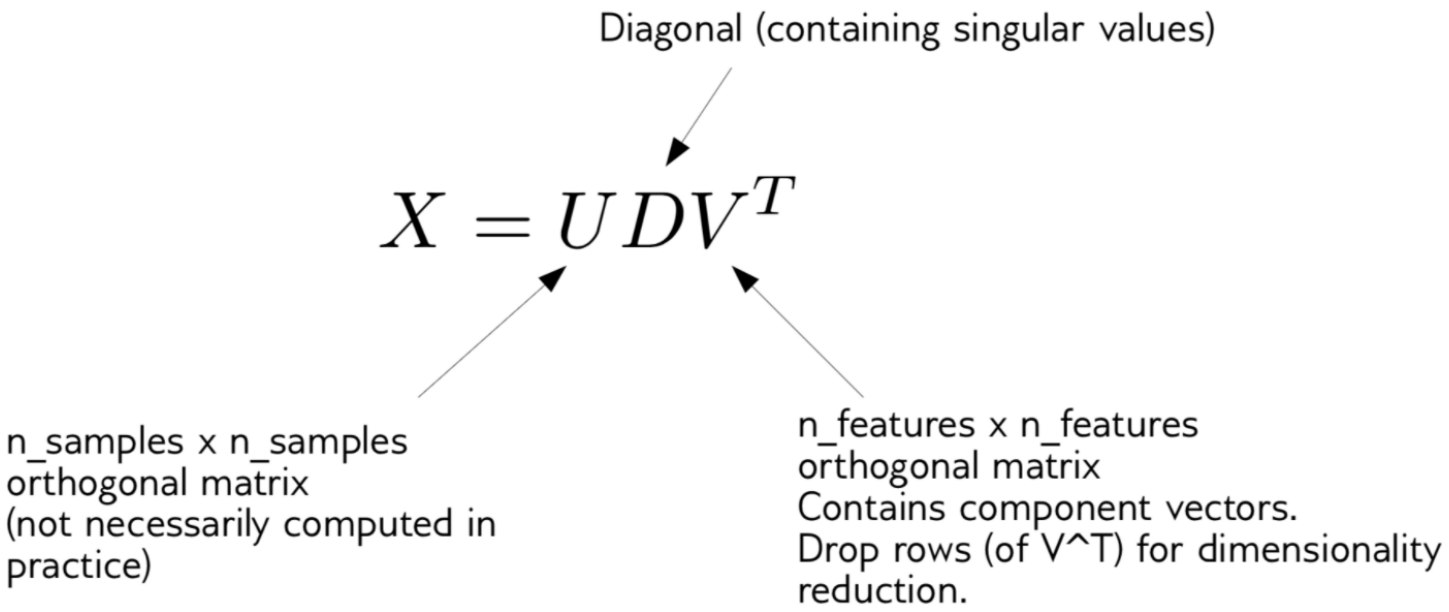
$$\max_{u_1 \in R^p, \|u_1\|=1} \text{var}(Xu_1)$$

$$\max_{u_1 \in R^p, \|u_1\|=1} u_1^T \text{cov}(X) u_1$$



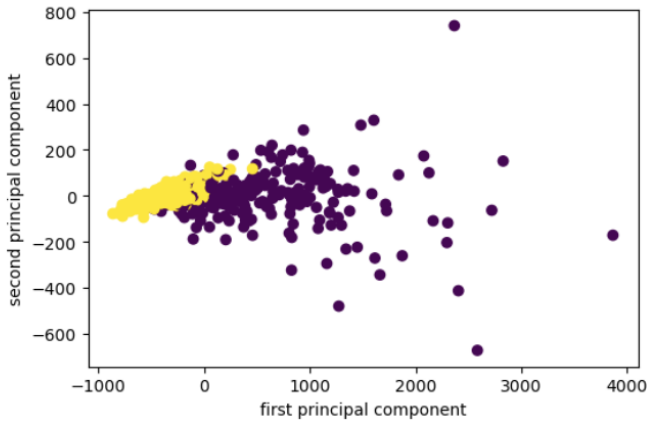
# PCA Computation

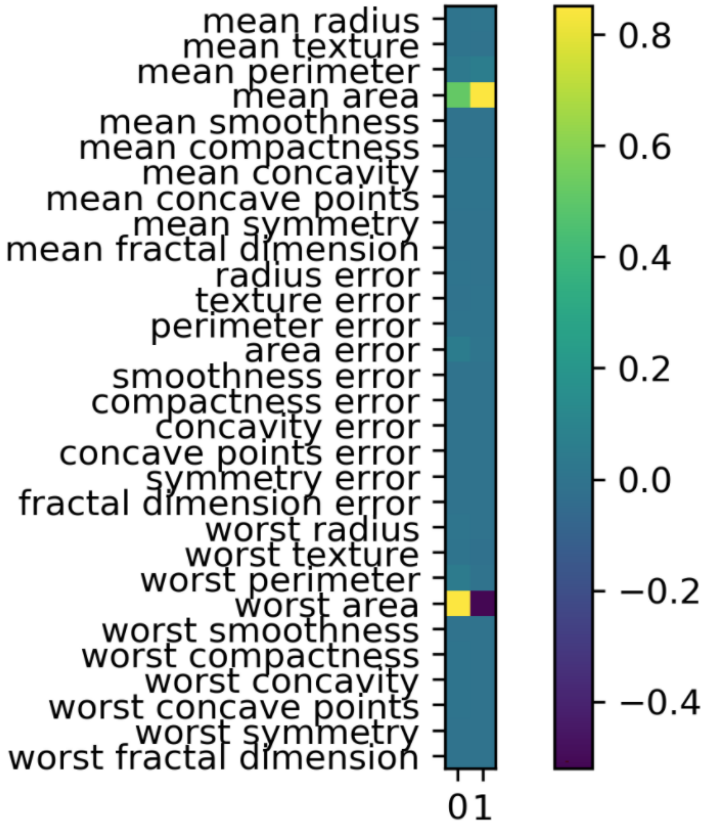
- Center X (subtract mean).
- In practice: Also scale to unit variance.
- Compute singular value decomposition:



# PCA for Visualisation

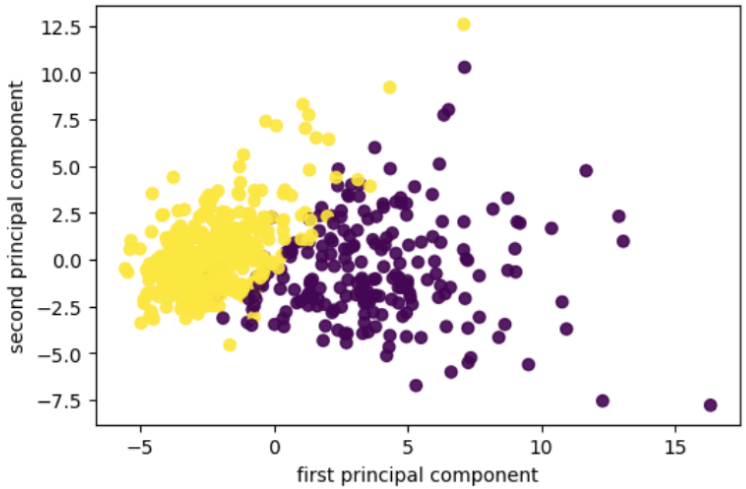
```
from sklearn.decomposition import PCA
print(cancer.data.shape)
pca = PCA(n_components=2)
X_pca = pca.fit_transform(cancer.data)
print(X_pca.shape)
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=cancer.target)
plt.xlabel("first principal component")
plt.ylabel("second principal component")
components = pca.components_
plt.imshow(components.T)
plt.yticks(range(len(cancer.feature_names)), cancer.feature_names)
plt.colorbar()
```



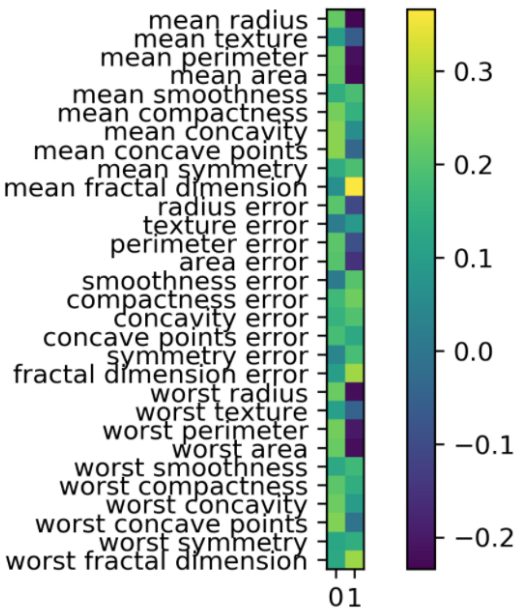
# Scaling!

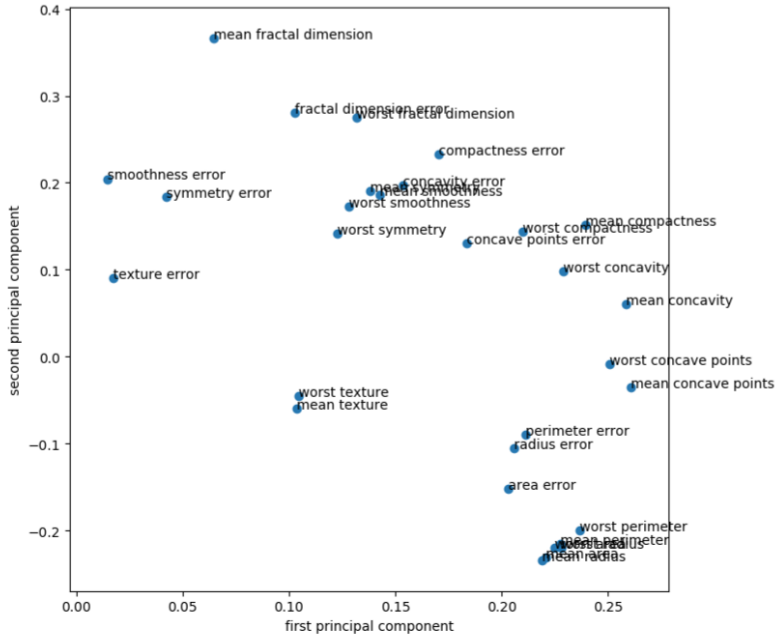
```
pca_scaled = make_pipeline(StandardScaler(), PCA(n_components=2))
X_pca_scaled = pca_scaled.fit_transform(cancer.data)
plt.scatter(X_pca_scaled[:, 0], X_pca_scaled[:, 1], c=cancer.target, alpha=.9)
plt.xlabel("first principal component")
plt.ylabel("second principal component")
```



# PCA Components

```
components = pca_scaled.named_steps['pca'].components_
plt.imshow(components.T)
plt.yticks(range(len(cancer.feature_names)), cancer.feature_names)
plt.colorbar()
```



# PCA for regularization

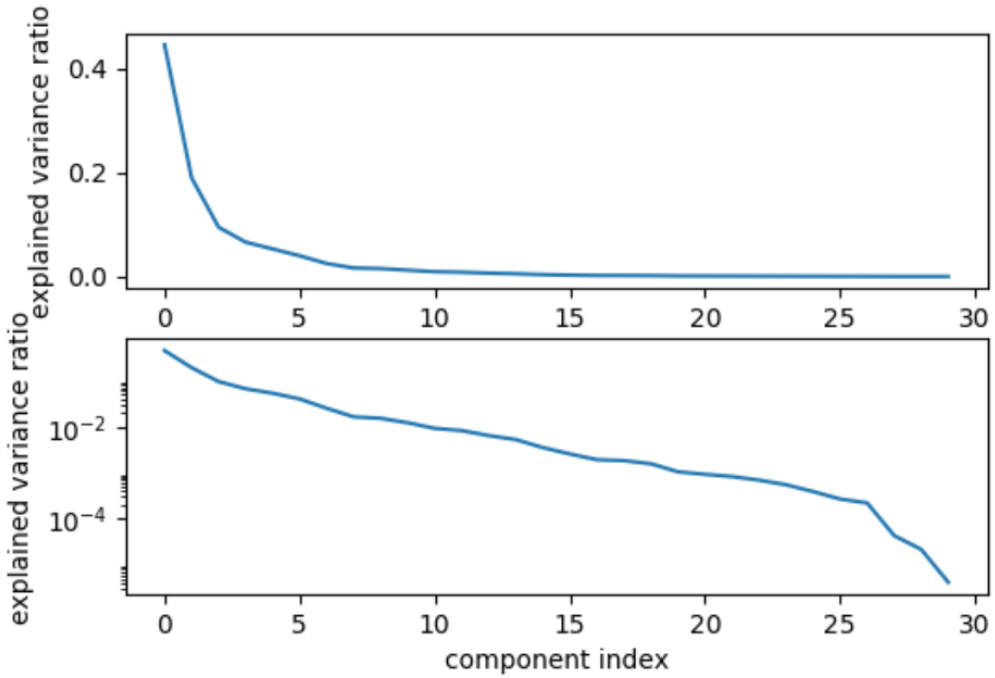
```
from sklearn.linear_model import LogisticRegression
X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target, stratify=cancer.target, random_state=0)
lr = LogisticRegression(C=10000).fit(X_train, y_train)
print(lr.score(X_train, y_train))
print(lr.score(X_test, y_test))
```

```
0.993
0.944
```

```
pca_lr = make_pipeline(StandardScaler(), PCA(n_components=2), LogisticRegression(C=10000))
pca_lr.fit(X_train, y_train)
print(pca_lr.score(X_train, y_train))
print(pca_lr.score(X_test, y_test))
```

```
0.961
0.923
```

# Variance Covered



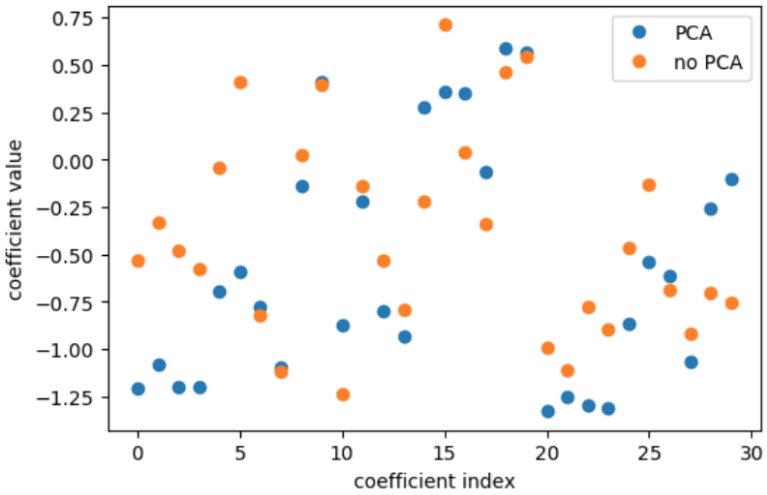

```
pca_lr = make_pipeline(StandardScaler(), PCA(n_components=6), LogisticRegression(C=10000))
pca_lr.fit(X_train, y_train)
print(pca_lr.score(X_train, y_train))
print(pca_lr.score(X_test, y_test))
```

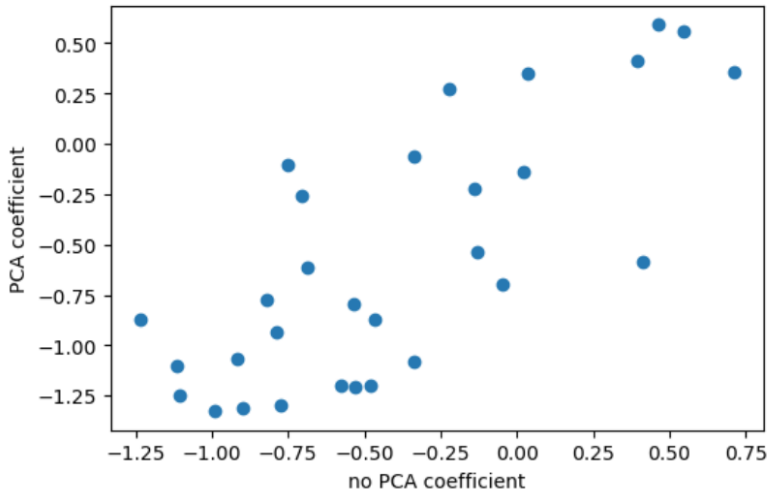
```
0.981
0.958
```

# Interpreting coefficients

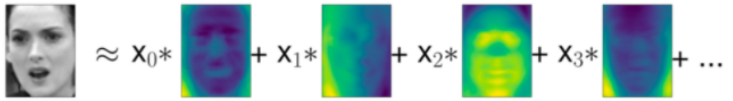
```
pca = pca_lr.named_steps['pca']
lr = pca_lr.named_steps['logisticregression']
coef_pca = pca.inverse_transform(lr.coef_)
```

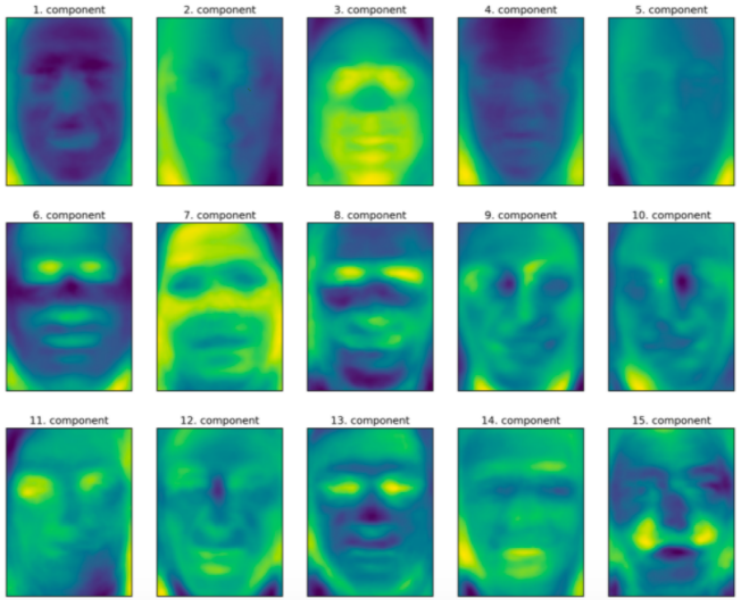
Comparing PCA + Logreg vs plain Logreg:



# PCA for feature extraction


Remember:
Signs don't mean anything!

# Table of Contents

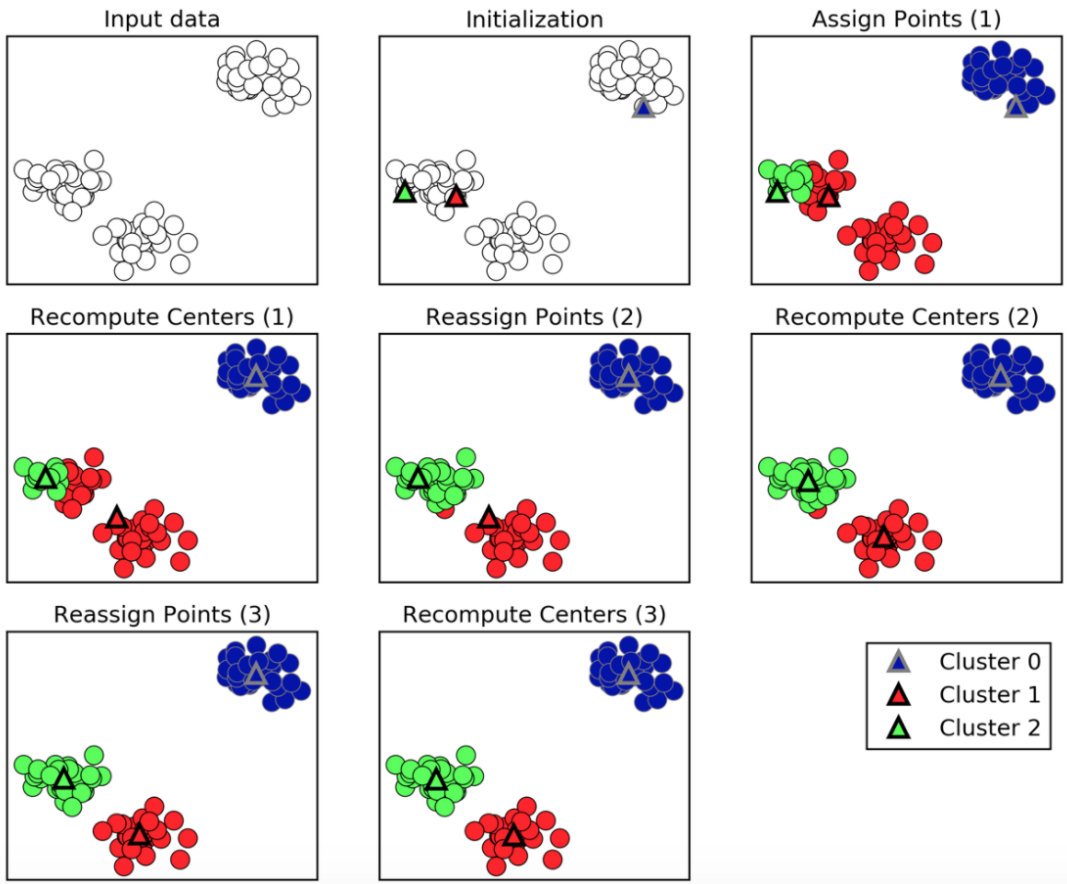
# Clustering

1. Partition data into groups (clusters)
2. Points within a cluster should be “similar”.
3. Points in different cluster should be “different”.

# Goals of Clustering

- Data Exploration
  - Are there coherent groups?
  - How many groups are there?
- Data Partitioning
  - Divide data by group before further processing
- Unsupervised feature extraction
  - Derive features from clusters or cluster distances

# K-Means Algorithm



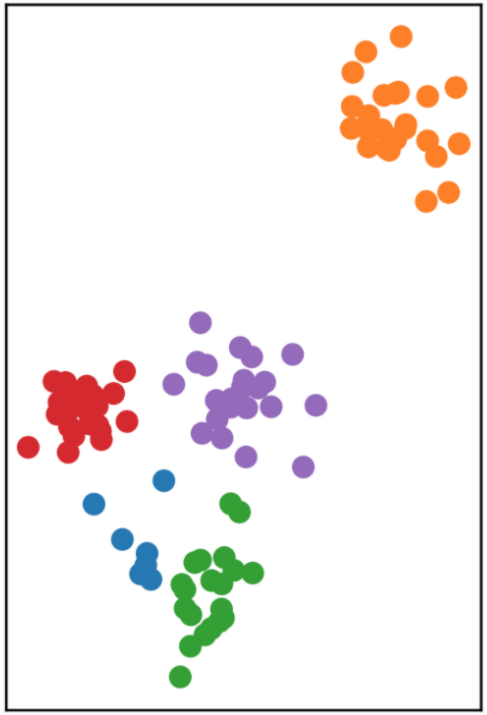

- Pick number of clusters k.
- Pick k random points as "cluster center"
- While cluster centers change:
  1. Assign each data point to it's closest cluster center
  2. Recompute cluster centers as the mean of the assigned points.

# k-Means API

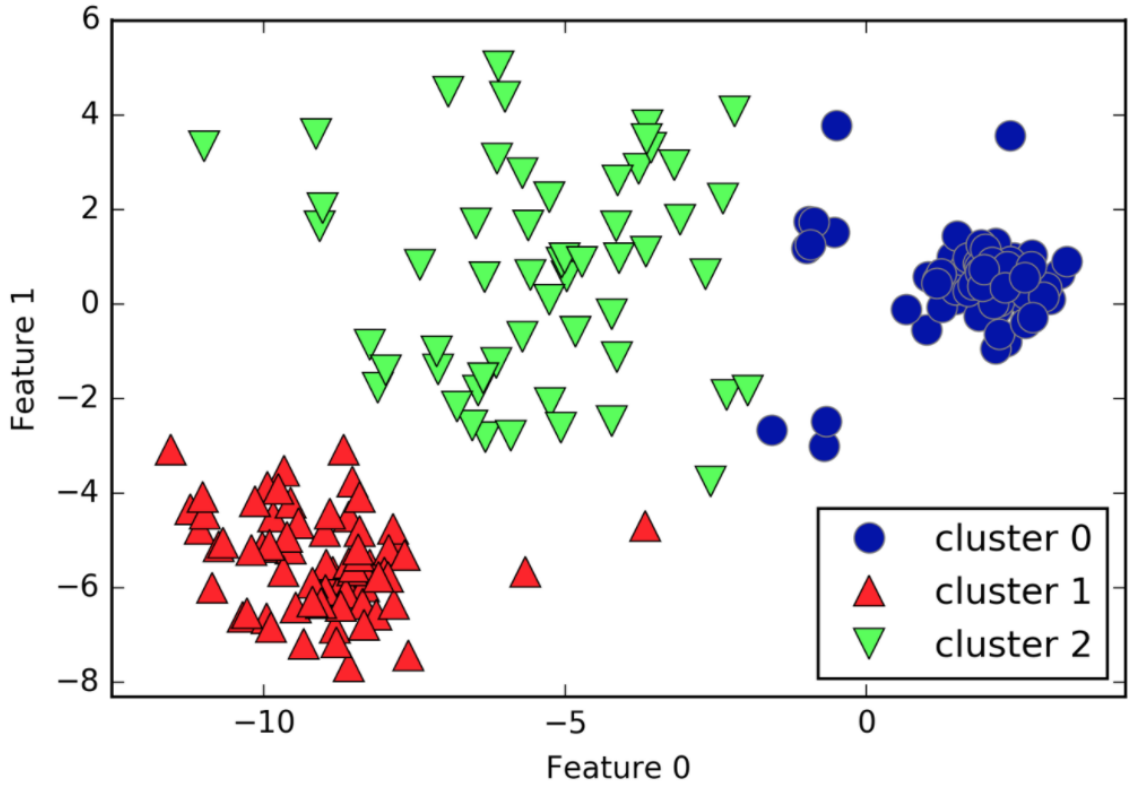
```
X, y = make_blobs(centers=4, random_state=1)

km = KMeans(n_clusters=5, random_state=0)
km.fit(X)
print(km.cluster_centers_.shape)
print(km.labels_.shape)
# predict is the same as labels_ on training data
# but can be applied to new data
print(km.predict(X).shape)
plt.scatter(X[:, 0], X[:, 1], c=km.labels_)
```

(5, 2)
(100,)
(100,)

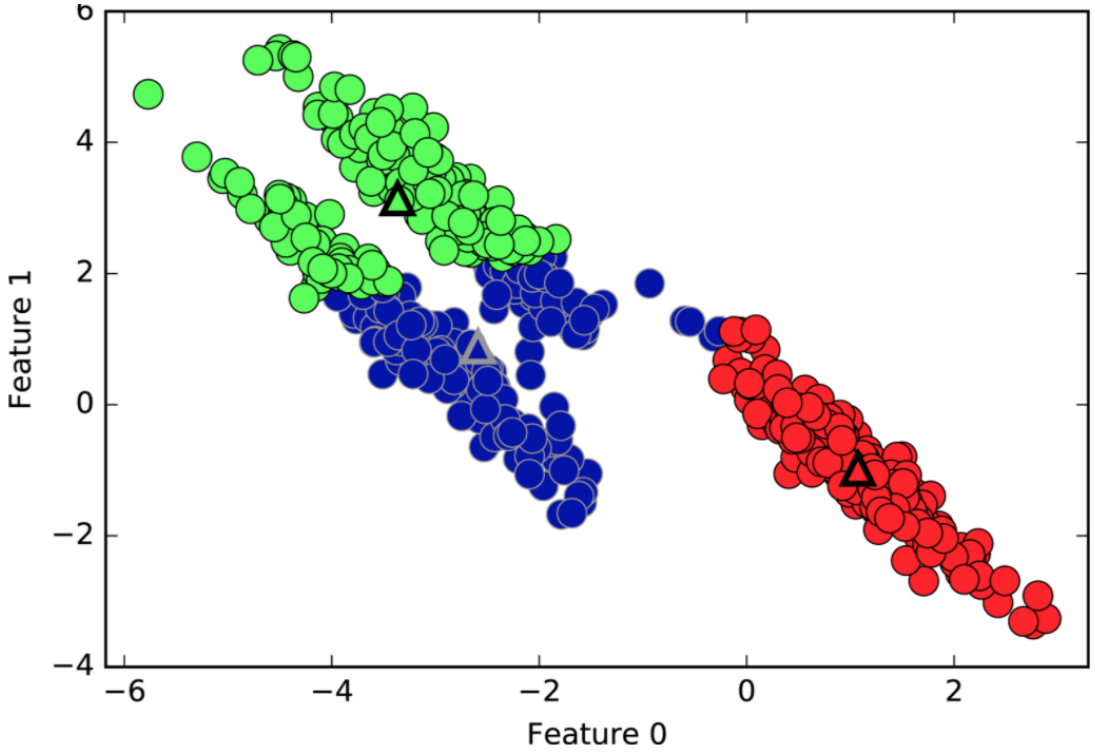


# Limitations of K-Means (1)



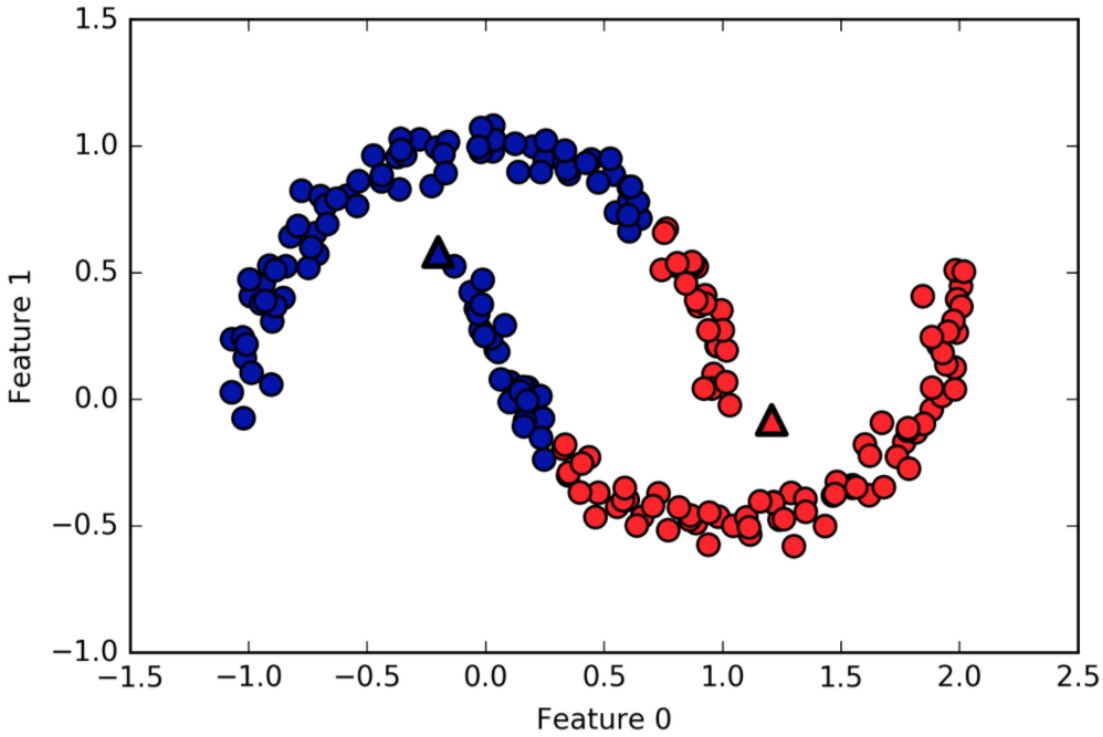

- Cluster boundaries equidistant to centers

# Limitations of K-Means (2)




- Can't model covariances well

# Limitations of K-Means (3)



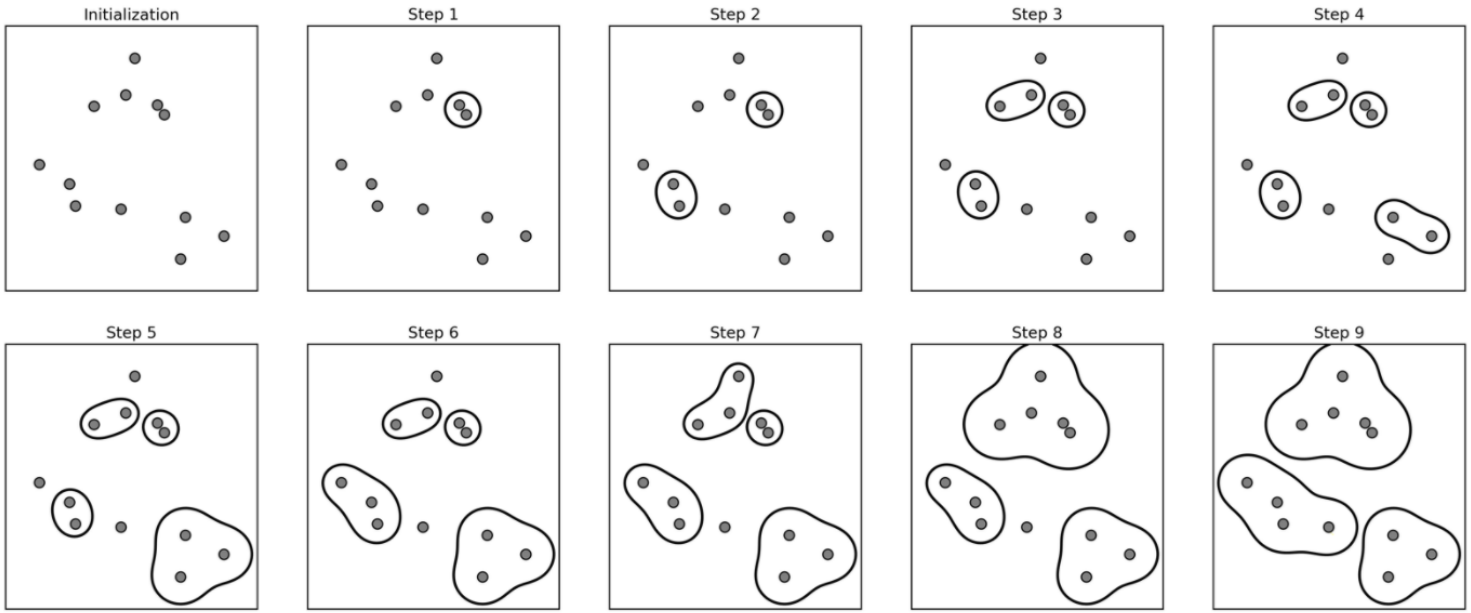

- Only simple cluster shapes

# Feature Extraction using K-Means

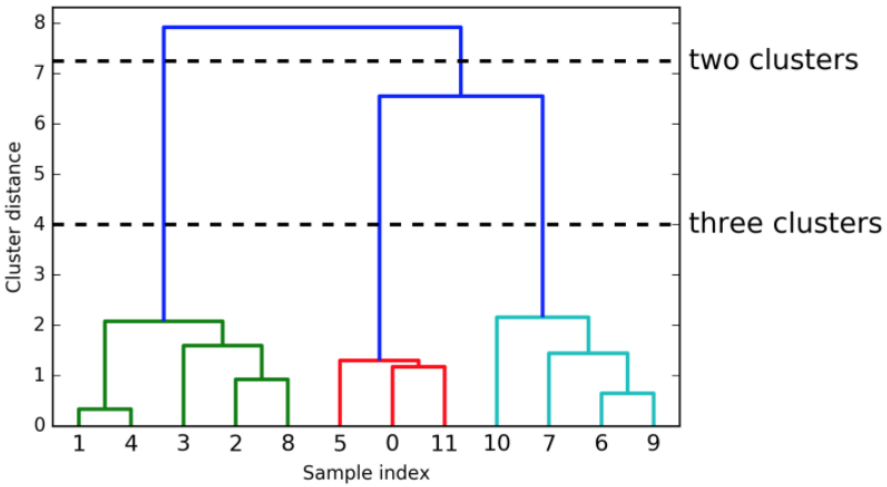
- Cluster membership $\longrightarrow$ categorical feature
- Cluster distanced $\longrightarrow$ continuous feature

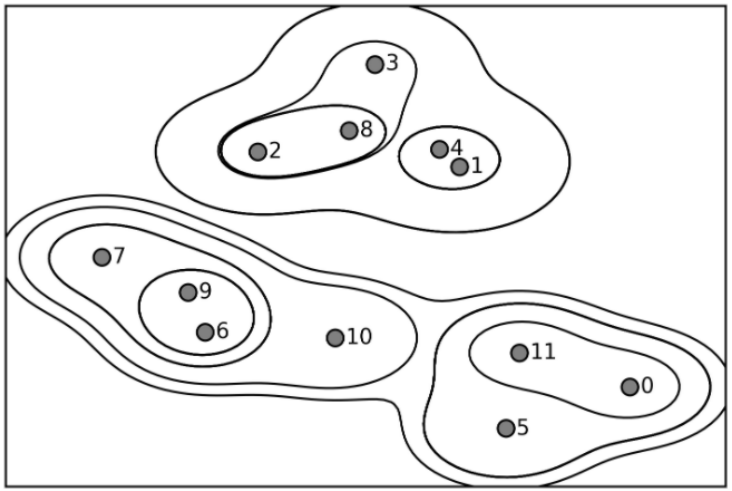
# Agglomerative Clustering

- Start with all points in their own cluster.
- Greedily merge the two most similar clusters.



# Dendograms

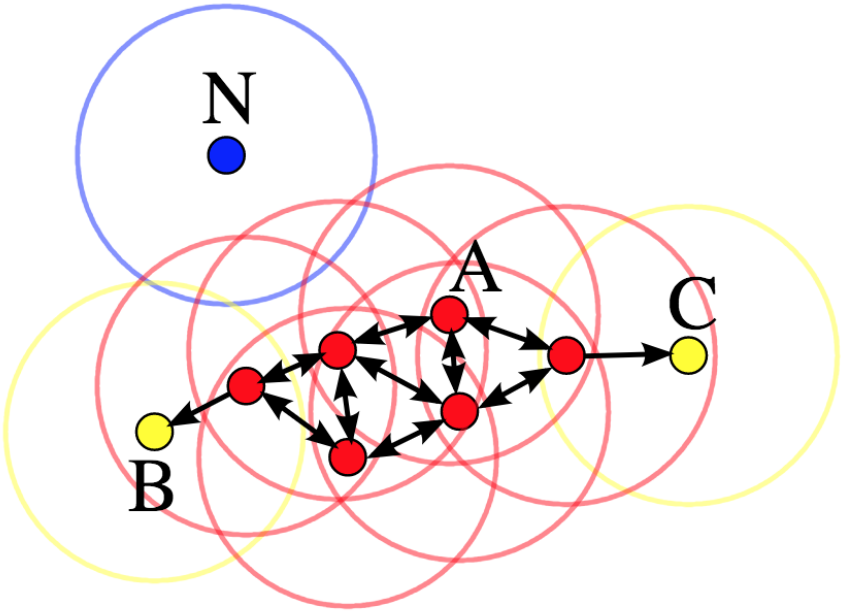

two clusters
three clusters
Cluster distance
Sample index
0
1
2
3
4
5
6
7
8
1 4 3 2 8 5 0 11 10 7 6 9


3
2
8
4
1
7
9
6
10
11
0
5

# Merging (Linkage) Criteria

- Complete Linkage
  - Smallest maximum distance
- Average Linkage
  - Smallest average distance between all pairs in the clusters
- Single Linkage
  - Smallest minimum distance
- Ward (default in sklearn)
  - Smallest increase in within-cluster variance
  - Leads to more equally sized clusters.

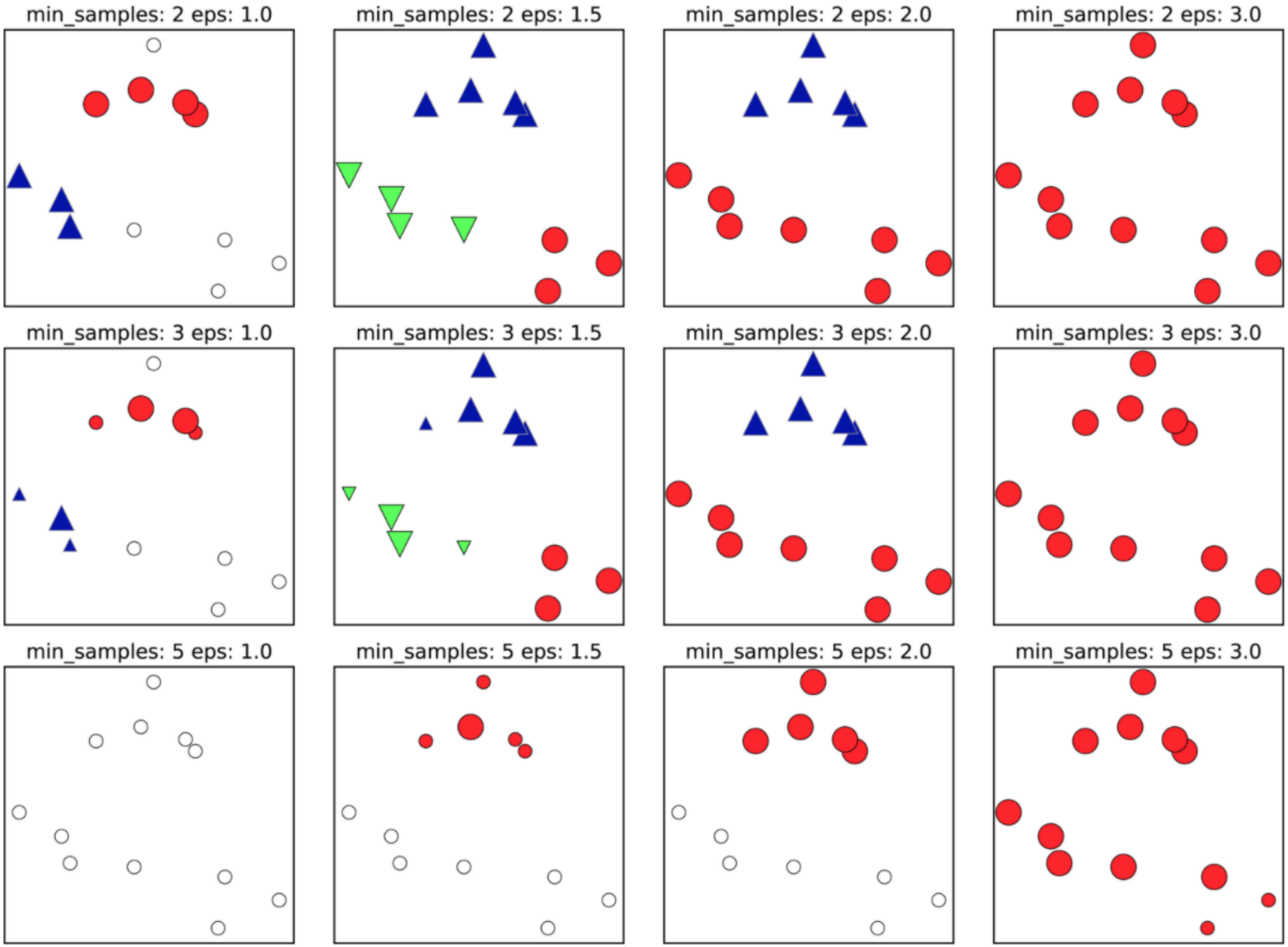
# DBSCAN - Algoritm




- eps: neighborhood radius
- min_samples: 4

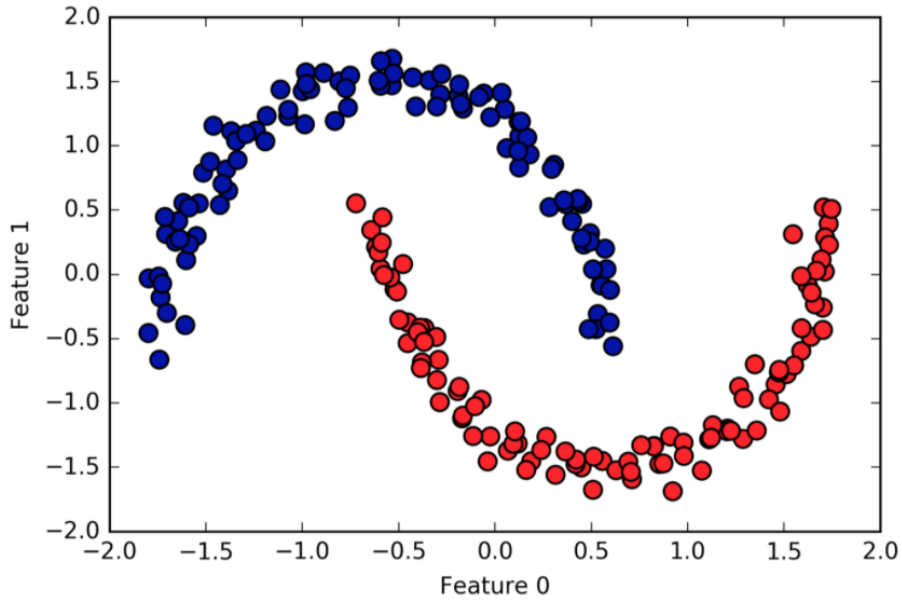
- A: Core
- B, C: not core
- N: noise

# DBSCAN - Illustration



# DBSCAN - Pros and Cons

- Can learn arbitrary cluster shapes
- Can detect outliers
- Needs two parameters to adjust



# Summary

- PCA: reduces the number of dimensions
- K-Means: Classic, simple. Only convex cluster shapes, determined by cluster centers.
- Agglomerative - Can produce hierarchy.
- DBSCAN-Arbitrary cluster shapes, can detect outliers.