

Material Summary: Clustering

1. Clustering Analysis

1.1 Clustering Analysis

- Unsupervised learning
 - Discover hidden structures in data where we don't know the right answer upfront
- **Clustering**
 - Find a natural grouping in data such that items in the same cluster are more similar to each other than those from different clusters
- Some applications
 - Medicine: classify different types of tissues
 - Marketing: group similar products
 - Image segmentation, object recognition
 - Social network analysis
 - Crime analysis: "hot" spatial areas; similar crimes

1.2 k-Means Clustering

- Prototype-based algorithm
 - Each cluster is represented by a "prototype" data point
 - As opposed to density-based
- Algorithm
 - Select k – number of clusters
 - Randomly pick k samples as "initial centroids"
 - Repeat until convergence (change in centers < tolerance):
 - Assign each sample to the nearest (using a distance metric) centroid
 - Move the centroids to the center of the newly created clusters
- Advantages: fast, simple, good for "spherical" clusters
- Disadvantages: we need to specify k , doesn't work too well with overlapping or hierarchical clusters
- Objective function

$$J = \sum_{i=0}^n \min_{\mu_j \in C} ||x_i - \mu_j||^2$$

- Within-cluster sum of squared errors (Euclidean distances)
- Generating clusters
 - We can use scikit-learn to generate "blobs" (clusters)

```
from sklearn.datasets import make_blobs
attributes, clusters = make_blobs(cluster_std = 1)
plt.scatter(attributes[:, 0], attributes[:, 1], c = clusters)
```

- Other options – "circles", "moons", etc.

```
from sklearn.datasets import make_blobs, make_circles,
    make_regression, make_s_curve, make_swiss_roll
# The last two are 3D - you can either plot them
# or see their projections using the x, y and z axes
```

1.3 Example: k-Means Clustering

- Generate several datasets
 - Blobs, circles, moons
- Apply k-means clustering to each dataset

- Display the original clusters and the clustering results
 - How do they differ? Can you explain why?

```
from sklearn.cluster import KMeans

attributes, clusters = make_blobs()
k_means = KMeans(3, init = "random")
assigned = k_means.fit_predict(attributes)
# Original, generated clusters
plt.scatter(attributes[:, 0], attributes[:, 1], c = clusters)
plt.show()
# Assigned clusters
plt.scatter(attributes[:, 0], attributes[:, 1], c = assigned)
plt.show()
```

1.4 k-Means++

- Random initial seed \Rightarrow may lead to poor performance
 - If the initial points aren't placed well enough or if the clusters are too "mixed"
- k-Means++ uses centers which are far away from each other
 - Instead of random initialization
- Algorithm
 - Choose the first centroid uniformly at random
 - To choose the next centroids, use a weighted probability distribution
 - Based on all currently selected centroids
 - Further away \Rightarrow greater probability
 - After all centroids have been initialized, proceed as usual

1.5 Example: k-Means++

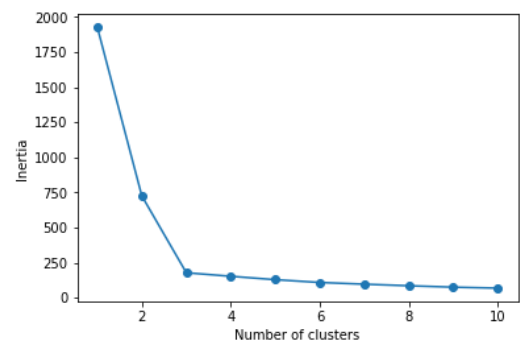
- Generate several datasets
 - Blobs, circles, moons
- Apply k-Means++

```
attributes, clusters = make_blobs()
k_means = KMeans(3, init = "k-means++")
assigned = k_means.fit_predict(attributes)
```

- Compare the performance of k-means++ versus k-means on blobs that are "close" to each other
 - Generate blobs with a higher standard deviation
 - Plot the centroids
 - * Count the misclassified points for the two algorithms

1.6 Finding an Optimal Number of Clusters

- Elbow method – graphical
- Inertia is a measure of clustering quality
 - Like grid search, initialize KMeans with a range of k values
 - Fit and calculate the inertia (given by default in scikit-learn)
 - Plot inertia vs. number of clusters
 - Find the "elbow point" of the plot – optimal
 - Inertia always decreases but some models overfit the data



```

inertias = []
for i in range(1, 11):
    km = KMeans(n_clusters = i)
    km.fit(attributes)
    inertias.append(km.inertia_)
plt.plot(range(1,11), inertias, marker = "o")
plt.xlabel("Number of clusters")
plt.ylabel("Inertia")
plt.show()

```

1.7 Evaluating Clustering Quality

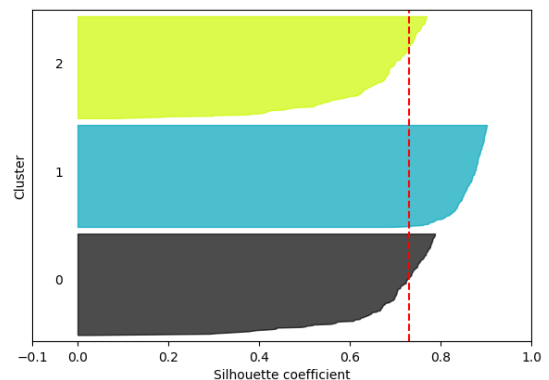
- Silhouette analysis – graphical
 - Use cluster cohesion (within-cluster distance) and cluster separation (between-cluster distance) to calculate the silhouette coefficient for each observation
 - $[-1; 1]$, 0 if the two distances are equal, 1 – ideal, -1 – worst
- Usage – import from scikit-learn and plot ([example](#))

```

from sklearn.metrics import silhouette_samples

```

- Interpretation
 - Each color is a separate cluster
 - All silhouettes should be close to 1 (these are similar to bar charts)
 - All silhouettes have a similar "depth" and "width" (if they don't \Rightarrow suboptimal clustering)



1.8 Hierarchical Clustering

- Another prototype-based clustering
- Advantage – allows us to plot dendrograms
 - Visualizations of binary hierarchical clustering
 - Allow us to interpret the results
- Algorithm
 - Compute the distance matrix (distances between any two points)
 - Start with each point at its own cluster
 - Repeat until only one cluster is left:
 - Merge the two closest clusters and update the distances
 - Update the distance matrix
 - In scikit-learn, linkage describes the selected [distance metric](#)

```

from sklearn.cluster import AgglomerativeClustering
aggl = AgglomerativeClustering(n_clusters = 3, linkage = "complete")

```

1.9 Example: Hierarchical Clustering

- Perform hierarchical (agglomerative) clustering on several datasets and visualize the results
 - Blobs, moons, nested circles
 - Try different linkage strategies
- Plot a dendrogram
 - `linkage()` calculates a distance matrix

- `dendrogram()` creates the plot (x-axis: ID, y-axis: distance)

```
from scipy.cluster.hierarchy import linkage, dendrogram
row_clusters = linkage(attributes, method = "complete", metric = "euclidean")
dendrogram(row_clusters, labels = clusters)
plt.show()

# Show only the last 10 merged clusters
dendrogram(row_clusters, labels = clusters, truncate_mode = "lastp", p = 10)
```

1.10 DBSCAN

- **Density-based Spatial Clustering of Applications with Noise**
- Label all points as "core points" or "noise points"
 - Core point: has at least m points within radius ε
- Use core points to create clusters
- DBSCAN vs. k-means
 - Noise points are not assigned to any cluster
 - Does not assume spherical shape
- Disadvantages: "curse of dimensionality", the hyperparameters need to be optimized
- Usage – like every other clustering algorithm

```
from sklearn.cluster import DBSCAN
db = DBSCAN(eps = 0.2, min_samples = 5, metric = "euclidean")
```

1.11 Example: DBSCAN

- Compare the performance of DBSCAN to the other clustering algorithms
 - Use the previous datasets and results
- Which algorithm performs best on which dataset?
- [Clustering comparison in scikit-learn](#)
- Conclusion
 - It's not always obvious which algorithm performs best on a dataset
 - Especially if data comes in many dimensions
 - A successful clustering depends on the algorithm and its hyperparameters
 - We need to choose an appropriate distance metric
 - We need some domain knowledge

1.12 Clustering and Classification

- Sometimes, classification tasks can be reduced to clustering tasks
 - Most trivially: just ignore the labels
 - Assumption: The data is easily (e.g., linearly) separable
 - The opposite is also true: we may be able to find a function that assigns a label to each data point
 - This is exactly what clustering does
- We can apply and compare both classification and clustering algorithms, metrics and tools to the same task
 - Even in ensembles
 - [Example](#)