AP5

March 19, 2025

0.1 # Anvendt Programmering 5

0.2 Machine Learning Basics with Scikit-Learn and Python

1 Introduction

- Understand basic machine learning concepts
- Learn how to use scikit-learn for machine learning tasks
- Unsupervised learning
  - K Means Clustering
- Supervised Learning
  - K Nearest Neighbors
  - Support Vector Machines

---

2 Setting Up Your Environment

- Install packages using pip:

```
pip install jupyter
pip install scikit-learn
pip install matplotlib
pip install seaborn
pip install pandas
pip install seaborn
```

[1]:
```
# In Jupyter, you can write this, inside a code-block, and it will download
# It might need to restart the kernel
%pip install scikit-learn matplotlib seaborn pandas jupyter
```

```
Requirement already satisfied: scikit-learn in ./.venv/lib/python3.13/site-
packages (1.6.1)
Requirement already satisfied: matplotlib in ./.venv/lib/python3.13/site-
packages (3.10.1)
Requirement already satisfied: seaborn in ./.venv/lib/python3.13/site-packages
(0.13.2)
Requirement already satisfied: pandas in ./.venv/lib/python3.13/site-packages
(2.2.3)
Requirement already satisfied: jupyter in ./.venv/lib/python3.13/site-packages
```

```
(1.1.1)
Requirement already satisfied: numpy>=1.19.5 in ./.venv/lib/python3.13/site-
packages (from scikit-learn) (2.2.4)
Requirement already satisfied: scipy>=1.6.0 in ./.venv/lib/python3.13/site-
packages (from scikit-learn) (1.15.2)
Requirement already satisfied: joblib>=1.2.0 in ./.venv/lib/python3.13/site-
packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in
./.venv/lib/python3.13/site-packages (from scikit-learn) (3.6.0)
Requirement already satisfied: contourpy>=1.0.1 in ./.venv/lib/python3.13/site-
packages (from matplotlib) (1.3.1)
Requirement already satisfied: cycler>=0.10 in ./.venv/lib/python3.13/site-
packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in ./.venv/lib/python3.13/site-
packages (from matplotlib) (4.56.0)
Requirement already satisfied: kiwisolver>=1.3.1 in ./.venv/lib/python3.13/site-
packages (from matplotlib) (1.4.8)
Requirement already satisfied: packaging>=20.0 in ./.venv/lib/python3.13/site-
packages (from matplotlib) (24.2)
Requirement already satisfied: pillow>=8 in ./.venv/lib/python3.13/site-packages
(from matplotlib) (11.1.0)
Requirement already satisfied: pyparsing>=2.3.1 in ./.venv/lib/python3.13/site-
packages (from matplotlib) (3.2.1)
Requirement already satisfied: python-dateutil>=2.7 in
./.venv/lib/python3.13/site-packages (from matplotlib) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in ./.venv/lib/python3.13/site-
packages (from pandas) (2025.1)
Requirement already satisfied: tzdata>=2022.7 in ./.venv/lib/python3.13/site-
packages (from pandas) (2025.1)
Requirement already satisfied: notebook in ./.venv/lib/python3.13/site-packages
(from jupyter) (7.3.3)
Requirement already satisfied: jupyter-console in ./.venv/lib/python3.13/site-
packages (from jupyter) (6.6.3)
Requirement already satisfied: nbconvert in ./.venv/lib/python3.13/site-packages
(from jupyter) (7.16.6)
Requirement already satisfied: ipykernel in ./.venv/lib/python3.13/site-packages
(from jupyter) (6.29.5)
Requirement already satisfied: ipywidgets in ./.venv/lib/python3.13/site-
packages (from jupyter) (8.1.5)
Requirement already satisfied: jupyterlab in ./.venv/lib/python3.13/site-
packages (from jupyter) (4.3.6)
Requirement already satisfied: six>=1.5 in ./.venv/lib/python3.13/site-packages
(from python-dateutil>=2.7->matplotlib) (1.17.0)
Requirement already satisfied: comm>=0.1.1 in ./.venv/lib/python3.13/site-
packages (from ipykernel->jupyter) (0.2.2)
Requirement already satisfied: debugpy>=1.6.5 in ./.venv/lib/python3.13/site-
packages (from ipykernel->jupyter) (1.8.13)
Requirement already satisfied: ipython>=7.23.1 in ./.venv/lib/python3.13/site-
```

```
packages (from ipykernel->jupyter) (9.0.2)
Requirement already satisfied: jupyter-client>=6.1.12 in
./.venv/lib/python3.13/site-packages (from ipykernel->jupyter) (8.6.3)
Requirement already satisfied: jupyter-core!=5.0.*,>=4.12 in
./.venv/lib/python3.13/site-packages (from ipykernel->jupyter) (5.7.2)
Requirement already satisfied: matplotlib-inline>=0.1 in
./.venv/lib/python3.13/site-packages (from ipykernel->jupyter) (0.1.7)
Requirement already satisfied: nest-asyncio in ./.venv/lib/python3.13/site-
packages (from ipykernel->jupyter) (1.6.0)
Requirement already satisfied: psutil in ./.venv/lib/python3.13/site-packages
(from ipykernel->jupyter) (7.0.0)
Requirement already satisfied: pyzmq>=24 in ./.venv/lib/python3.13/site-packages
(from ipykernel->jupyter) (26.3.0)
Requirement already satisfied: tornado>=6.1 in ./.venv/lib/python3.13/site-
packages (from ipykernel->jupyter) (6.4.2)
Requirement already satisfied: traitlets>=5.4.0 in ./.venv/lib/python3.13/site-
packages (from ipykernel->jupyter) (5.14.3)
Requirement already satisfied: widgetsnbextension~=4.0.12 in
./.venv/lib/python3.13/site-packages (from ipywidgets->jupyter) (4.0.13)
Requirement already satisfied: jupyterlab-widgets~=3.0.12 in
./.venv/lib/python3.13/site-packages (from ipywidgets->jupyter) (3.0.13)
Requirement already satisfied: prompt-toolkit>=3.0.30 in
./.venv/lib/python3.13/site-packages (from jupyter-console->jupyter) (3.0.50)
Requirement already satisfied: pygments in ./.venv/lib/python3.13/site-packages
(from jupyter-console->jupyter) (2.19.1)
Requirement already satisfied: async-lru>=1.0.0 in ./.venv/lib/python3.13/site-
packages (from jupyterlab->jupyter) (2.0.5)
Requirement already satisfied: httpx>=0.25.0 in ./.venv/lib/python3.13/site-
packages (from jupyterlab->jupyter) (0.28.1)
Requirement already satisfied: jinja2>=3.0.3 in ./.venv/lib/python3.13/site-
packages (from jupyterlab->jupyter) (3.1.6)
Requirement already satisfied: jupyter-lsp>=2.0.0 in
./.venv/lib/python3.13/site-packages (from jupyterlab->jupyter) (2.2.5)
Requirement already satisfied: jupyter-server<3,>=2.4.0 in
./.venv/lib/python3.13/site-packages (from jupyterlab->jupyter) (2.15.0)
Requirement already satisfied: jupyterlab-server<3,>=2.27.1 in
./.venv/lib/python3.13/site-packages (from jupyterlab->jupyter) (2.27.3)
Requirement already satisfied: notebook-shim>=0.2 in
./.venv/lib/python3.13/site-packages (from jupyterlab->jupyter) (0.2.4)
Requirement already satisfied: setuptools>=40.8.0 in
./.venv/lib/python3.13/site-packages (from jupyterlab->jupyter) (76.0.0)
Requirement already satisfied: beautifulsoup4 in ./.venv/lib/python3.13/site-
packages (from nbconvert->jupyter) (4.13.3)
Requirement already satisfied: bleach!=5.0.0 in ./.venv/lib/python3.13/site-
packages (from bleach[css]!=5.0.0->nbconvert->jupyter) (6.2.0)
Requirement already satisfied: defusedxml in ./.venv/lib/python3.13/site-
packages (from nbconvert->jupyter) (0.7.1)
Requirement already satisfied: jupyterlab-pygments in
```

```
./.venv/lib/python3.13/site-packages (from nbconvert->jupyter) (0.3.0)
Requirement already satisfied: markupsafe>=2.0 in ./.venv/lib/python3.13/site-
packages (from nbconvert->jupyter) (3.0.2)
Requirement already satisfied: mistune<4,>=2.0.3 in ./.venv/lib/python3.13/site-
packages (from nbconvert->jupyter) (3.1.2)
Requirement already satisfied: nbclient>=0.5.0 in ./.venv/lib/python3.13/site-
packages (from nbconvert->jupyter) (0.10.2)
Requirement already satisfied: nbformat>=5.7 in ./.venv/lib/python3.13/site-
packages (from nbconvert->jupyter) (5.10.4)
Requirement already satisfied: pandocfilters>=1.4.1 in
./.venv/lib/python3.13/site-packages (from nbconvert->jupyter) (1.5.1)
Requirement already satisfied: webencodings in ./.venv/lib/python3.13/site-
packages (from bleach!=5.0.0->bleach[css]!=5.0.0->nbconvert->jupyter) (0.5.1)
Requirement already satisfied: tinycss2<1.5,>=1.1.0 in
./.venv/lib/python3.13/site-packages (from
bleach[css]!=5.0.0->nbconvert->jupyter) (1.4.0)
Requirement already satisfied: anyio in ./.venv/lib/python3.13/site-packages
(from httpx>=0.25.0->jupyterlab->jupyter) (4.9.0)
Requirement already satisfied: certifi in ./.venv/lib/python3.13/site-packages
(from httpx>=0.25.0->jupyterlab->jupyter) (2025.1.31)
Requirement already satisfied: httpcore==1.* in ./.venv/lib/python3.13/site-
packages (from httpx>=0.25.0->jupyterlab->jupyter) (1.0.7)
Requirement already satisfied: idna in ./.venv/lib/python3.13/site-packages
(from httpx>=0.25.0->jupyterlab->jupyter) (3.10)
Requirement already satisfied: h11<0.15,>=0.13 in ./.venv/lib/python3.13/site-
packages (from httpcore==1.*->httpx>=0.25.0->jupyterlab->jupyter) (0.14.0)
Requirement already satisfied: decorator in ./.venv/lib/python3.13/site-packages
(from ipython>=7.23.1->ipykernel->jupyter) (5.2.1)
Requirement already satisfied: ipython-pygments-lexers in
./.venv/lib/python3.13/site-packages (from ipython>=7.23.1->ipykernel->jupyter)
(1.1.1)
Requirement already satisfied: jedi>=0.16 in ./.venv/lib/python3.13/site-
packages (from ipython>=7.23.1->ipykernel->jupyter) (0.19.2)
Requirement already satisfied: pexpect>4.3 in ./.venv/lib/python3.13/site-
packages (from ipython>=7.23.1->ipykernel->jupyter) (4.9.0)
Requirement already satisfied: stack_data in ./.venv/lib/python3.13/site-
packages (from ipython>=7.23.1->ipykernel->jupyter) (0.6.3)
Requirement already satisfied: platformdirs>=2.5 in ./.venv/lib/python3.13/site-
packages (from jupyter-core!=5.0.*,>=4.12->ipykernel->jupyter) (4.3.6)
Requirement already satisfied: argon2-cffi>=21.1 in ./.venv/lib/python3.13/site-
packages (from jupyter-server<3,>=2.4.0->jupyterlab->jupyter) (23.1.0)
Requirement already satisfied: jupyter-events>=0.11.0 in
./.venv/lib/python3.13/site-packages (from jupyter-
server<3,>=2.4.0->jupyterlab->jupyter) (0.12.0)
Requirement already satisfied: jupyter-server-terminals>=0.4.4 in
./.venv/lib/python3.13/site-packages (from jupyter-
server<3,>=2.4.0->jupyterlab->jupyter) (0.5.3)
Requirement already satisfied: overrides>=5.0 in ./.venv/lib/python3.13/site-
```

```
packages (from jupyter-server<3,>=2.4.0->jupyterlab->jupyter) (7.7.0)
Requirement already satisfied: prometheus-client>=0.9 in
./.venv/lib/python3.13/site-packages (from jupyter-
server<3,>=2.4.0->jupyterlab->jupyter) (0.21.1)
Requirement already satisfied: send2trash>=1.8.2 in ./.venv/lib/python3.13/site-
packages (from jupyter-server<3,>=2.4.0->jupyterlab->jupyter) (1.8.3)
Requirement already satisfied: terminado>=0.8.3 in ./.venv/lib/python3.13/site-
packages (from jupyter-server<3,>=2.4.0->jupyterlab->jupyter) (0.18.1)
Requirement already satisfied: websocket-client>=1.7 in
./.venv/lib/python3.13/site-packages (from jupyter-
server<3,>=2.4.0->jupyterlab->jupyter) (1.8.0)
Requirement already satisfied: babel>=2.10 in ./.venv/lib/python3.13/site-
packages (from jupyterlab-server<3,>=2.27.1->jupyterlab->jupyter) (2.17.0)
Requirement already satisfied: json5>=0.9.0 in ./.venv/lib/python3.13/site-
packages (from jupyterlab-server<3,>=2.27.1->jupyterlab->jupyter) (0.10.0)
Requirement already satisfied: jsonschema>=4.18.0 in
./.venv/lib/python3.13/site-packages (from jupyterlab-
server<3,>=2.27.1->jupyterlab->jupyter) (4.23.0)
Requirement already satisfied: requests>=2.31 in ./.venv/lib/python3.13/site-
packages (from jupyterlab-server<3,>=2.27.1->jupyterlab->jupyter) (2.32.3)
Requirement already satisfied: fastjsonschema>=2.15 in
./.venv/lib/python3.13/site-packages (from nbformat>=5.7->nbconvert->jupyter)
(2.21.1)
Requirement already satisfied: wcwidth in ./.venv/lib/python3.13/site-packages
(from prompt-toolkit>=3.0.30->jupyter-console->jupyter) (0.2.13)
Requirement already satisfied: soupsieve>1.2 in ./.venv/lib/python3.13/site-
packages (from beautifulsoup4->nbconvert->jupyter) (2.6)
Requirement already satisfied: typing-extensions>=4.0.0 in
./.venv/lib/python3.13/site-packages (from beautifulsoup4->nbconvert->jupyter)
(4.12.2)
Requirement already satisfied: sniffio>=1.1 in ./.venv/lib/python3.13/site-
packages (from anyio->httpx>=0.25.0->jupyterlab->jupyter) (1.3.1)
Requirement already satisfied: argon2-cffi-bindings in
./.venv/lib/python3.13/site-packages (from argon2-cffi>=21.1->jupyter-
server<3,>=2.4.0->jupyterlab->jupyter) (21.2.0)
Requirement already satisfied: parso<0.9.0,>=0.8.4 in
./.venv/lib/python3.13/site-packages (from
jedi>=0.16->ipython>=7.23.1->ipykernel->jupyter) (0.8.4)
Requirement already satisfied: attrs>=22.2.0 in ./.venv/lib/python3.13/site-
packages (from jsonschema>=4.18.0->jupyterlab-
server<3,>=2.27.1->jupyterlab->jupyter) (25.3.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in
./.venv/lib/python3.13/site-packages (from jsonschema>=4.18.0->jupyterlab-
server<3,>=2.27.1->jupyterlab->jupyter) (2024.10.1)
Requirement already satisfied: referencing>=0.28.4 in
./.venv/lib/python3.13/site-packages (from jsonschema>=4.18.0->jupyterlab-
server<3,>=2.27.1->jupyterlab->jupyter) (0.36.2)
Requirement already satisfied: rpds-py>=0.7.1 in ./.venv/lib/python3.13/site-
```

```
packages (from jsonschema>=4.18.0->jupyterlab-
server<3,>=2.27.1->jupyterlab->jupyter) (0.23.1)
Requirement already satisfied: python-json-logger>=2.0.4 in
./.venv/lib/python3.13/site-packages (from jupyter-events>=0.11.0->jupyter-
server<3,>=2.4.0->jupyterlab->jupyter) (3.3.0)
Requirement already satisfied: pyyaml>=5.3 in ./.venv/lib/python3.13/site-
packages (from jupyter-events>=0.11.0->jupyter-
server<3,>=2.4.0->jupyterlab->jupyter) (6.0.2)
Requirement already satisfied: rfc3339-validator in ./.venv/lib/python3.13/site-
packages (from jupyter-events>=0.11.0->jupyter-
server<3,>=2.4.0->jupyterlab->jupyter) (0.1.4)
Requirement already satisfied: rfc3986-validator>=0.1.1 in
./.venv/lib/python3.13/site-packages (from jupyter-events>=0.11.0->jupyter-
server<3,>=2.4.0->jupyterlab->jupyter) (0.1.1)
Requirement already satisfied: ptyprocess>=0.5 in ./.venv/lib/python3.13/site-
packages (from pexpect>4.3->ipython>=7.23.1->ipykernel->jupyter) (0.7.0)
Requirement already satisfied: charset-normalizer<4,>=2 in
./.venv/lib/python3.13/site-packages (from requests>=2.31->jupyterlab-
server<3,>=2.27.1->jupyterlab->jupyter) (3.4.1)
Requirement already satisfied: urllib3<3,>=1.21.1 in
./.venv/lib/python3.13/site-packages (from requests>=2.31->jupyterlab-
server<3,>=2.27.1->jupyterlab->jupyter) (2.3.0)
Requirement already satisfied: executing>=1.2.0 in ./.venv/lib/python3.13/site-
packages (from stack_data->ipython>=7.23.1->ipykernel->jupyter) (2.2.0)
Requirement already satisfied: asttokens>=2.1.0 in ./.venv/lib/python3.13/site-
packages (from stack_data->ipython>=7.23.1->ipykernel->jupyter) (3.0.0)
Requirement already satisfied: pure-eval in ./.venv/lib/python3.13/site-packages
(from stack_data->ipython>=7.23.1->ipykernel->jupyter) (0.2.3)
Requirement already satisfied: fqdn in ./.venv/lib/python3.13/site-packages
(from jsonschema[format-nongpl]>=4.18.0->jupyter-events>=0.11.0->jupyter-
server<3,>=2.4.0->jupyterlab->jupyter) (1.5.1)
Requirement already satisfied: isoduration in ./.venv/lib/python3.13/site-
packages (from jsonschema[format-nongpl]>=4.18.0->jupyter-
events>=0.11.0->jupyter-server<3,>=2.4.0->jupyterlab->jupyter) (20.11.0)
Requirement already satisfied: jsonpointer>1.13 in ./.venv/lib/python3.13/site-
packages (from jsonschema[format-nongpl]>=4.18.0->jupyter-
events>=0.11.0->jupyter-server<3,>=2.4.0->jupyterlab->jupyter) (3.0.0)
Requirement already satisfied: uri-template in ./.venv/lib/python3.13/site-
packages (from jsonschema[format-nongpl]>=4.18.0->jupyter-
events>=0.11.0->jupyter-server<3,>=2.4.0->jupyterlab->jupyter) (1.3.0)
Requirement already satisfied: webcolors>=24.6.0 in ./.venv/lib/python3.13/site-
packages (from jsonschema[format-nongpl]>=4.18.0->jupyter-
events>=0.11.0->jupyter-server<3,>=2.4.0->jupyterlab->jupyter) (24.11.1)
Requirement already satisfied: cffi>=1.0.1 in ./.venv/lib/python3.13/site-
packages (from argon2-cffi-bindings->argon2-cffi>=21.1->jupyter-
server<3,>=2.4.0->jupyterlab->jupyter) (1.17.1)
Requirement already satisfied: pycparser in ./.venv/lib/python3.13/site-packages
(from cffi>=1.0.1->argon2-cffi-bindings->argon2-cffi>=21.1->jupyter-
```

```
server<3,>=2.4.0->jupyterlab->jupyter) (2.22)
Requirement already satisfied: arrow>=0.15.0 in ./.venv/lib/python3.13/site-
packages (from isoduration->jsonschema[format-nongpl]>=4.18.0->jupyter-
events>=0.11.0->jupyter-server<3,>=2.4.0->jupyterlab->jupyter) (1.3.0)
Requirement already satisfied: types-python-dateutil>=2.8.10 in
./.venv/lib/python3.13/site-packages (from
arrow>=0.15.0->isoduration->jsonschema[format-nongpl]>=4.18.0->jupyter-
events>=0.11.0->jupyter-server<3,>=2.4.0->jupyterlab->jupyter) (2.9.0.20241206)
Note: you may need to restart the kernel to use updated packages.
```

3 Understanding the Basics

- **Supervised Learning**: Training a model on labeled data (e.g., classification, regression).
- **Unsupervised Learning**: Training a model on unlabeled data (e.g., clustering, dimensionality reduction).

---

4 Unsupervised Learning

A type of machine learning where the algorithm learns patterns from unlabeled data. - **Key Methods**: - Clustering - Dimensionality Reduction - **Applications**: - Customer segmentation - Anomaly detection - Image compression

5 What is K-means Clustering?

K-means Clustering: A method to partition data into K clusters, where each data point belongs to the cluster with the nearest mean.

- **Steps**:
  1. **Cluster Assignment Step (E-Step)** Each data point x_i is assigned to the nearest cluster centroid:
  $$C_k = \{x_i \mid \arg\min_K \|x_i - \mu_k\|^2\}$$
  2. **Centroid Update Step (M-Step)**: The centroid of each cluster is updated as the mean of all points assigned to it:
  $$\mu_k = \frac{1}{|C_k|} \sum_{x_i \in C_k} x_i$$
  3. **Objective Function (Minimization of Within-Cluster Variance)**: The K-Means algorithm minimizes the following objective function:
  $$J = \sum_{j=1}^{K} \sum_{x_i \in C_k} \|x_i - \mu_k\|^2$$
  4. **Continue until Convergence**: The algorithm iterates between the assignment step and the update step until convergence (i.e., when centroids no longer change significantly or a stopping criterion is met).

6 Visualizing K-means Clustering

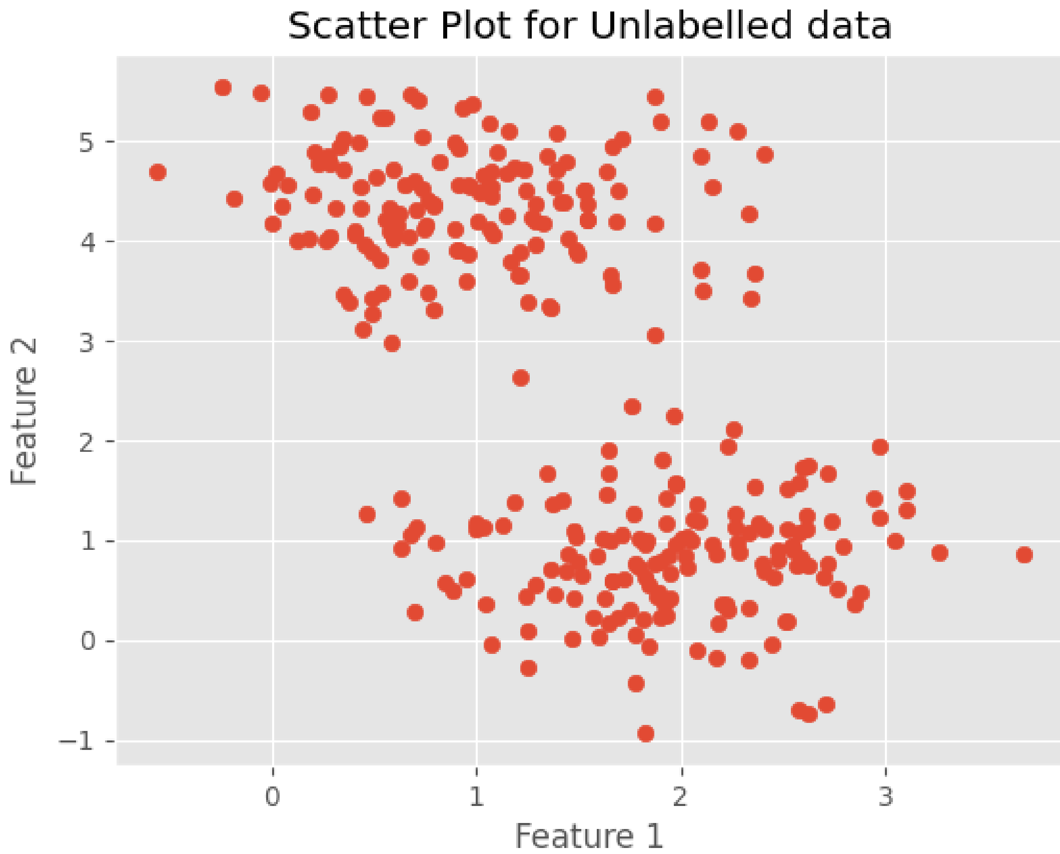
- Data points grouped into 4 clusters.
- Feature is just a measurement of a specific sample

[2]:
```python
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
import pandas as pd

plt.style.use("ggplot")

N_CLUSTERS = 2
# Generate sample data
X, _ = make_blobs(n_samples=300, centers=N_CLUSTERS, cluster_std=0.60,
 ↪random_state=0)
df = pd.DataFrame(X, columns=["x", "y"])

df.head()
```

[2]:

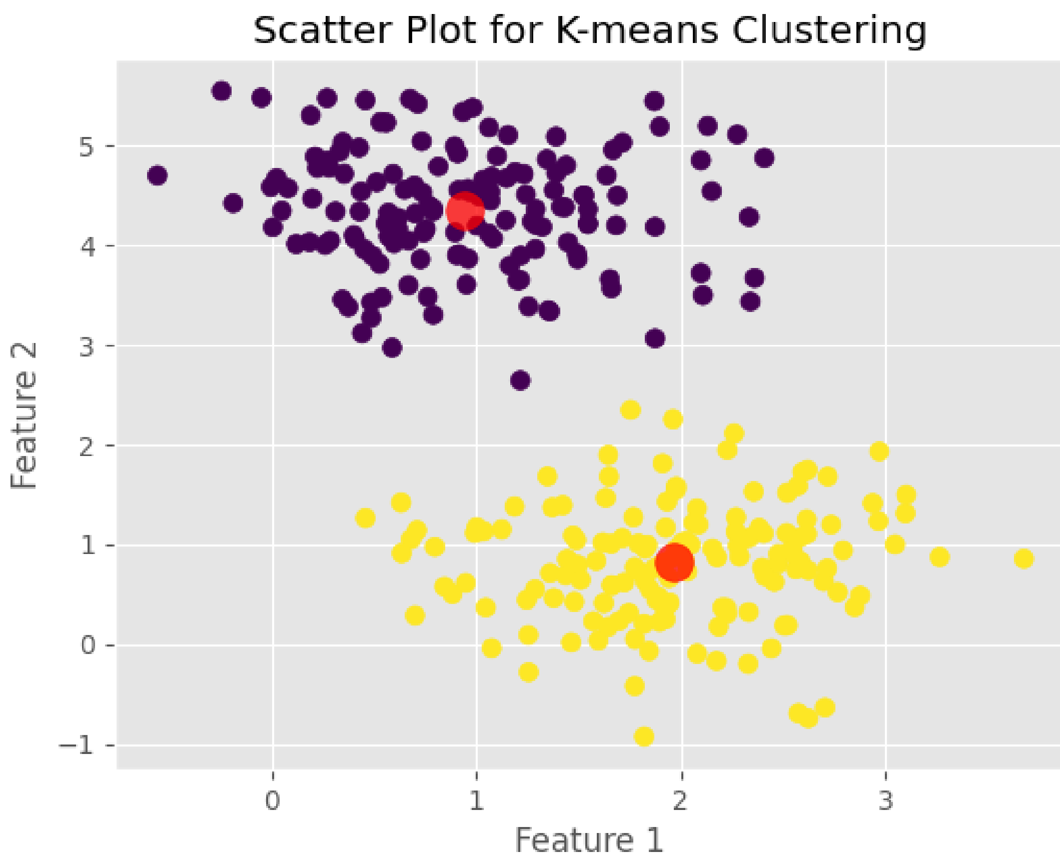
	x	y
0	2.406157	4.870475
1	2.580767	0.828599
2	1.062696	5.176351
3	2.548219	0.900839
4	1.390161	5.084895

[3]:
```python
# Visualization of the data
plt.scatter(X[:, 0], X[:, 1])
plt.title("Scatter Plot for Unlabelled data")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```



6.1 K-means Clustering

[4]:
```
from sklearn.cluster import KMeans

# Apply K-means clustering
kmeans = KMeans(n_clusters=N_CLUSTERS)
kmeans.fit(X)
y_kmeans = kmeans.predict(X)

# Visualization of the Clusters
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=50, cmap="viridis")
centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c="red", s=200, alpha=0.75)
plt.title("Scatter Plot for K-means Clustering")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```



7 Breast Cancer Dataset

- https://scikit-learn.org/stable/datasets/toy_dataset.html

[5]:
```
from sklearn.datasets import load_breast_cancer
import pandas as pd

# Load the Breast Cancer dataset
data = load_breast_cancer()
X, y = data.data, data.target


# Display the dataframe
df_all = pd.DataFrame(X, columns=data.feature_names)

df_all["Target"] = y
df_all["Target"] = df_all["Target"].map({i: v for i, v in enumerate(data.
    ↪target_names)})
df_all.head()
```

[5]:

```
   mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
0        17.99         10.38          122.80     1001.0          0.11840
1        20.57         17.77          132.90     1326.0          0.08474
2        19.69         21.25          130.00     1203.0          0.10960
3        11.42         20.38           77.58      386.1          0.14250
4        20.29         14.34          135.10     1297.0          0.10030

   mean compactness  mean concavity  mean concave points  mean symmetry  \
0           0.27760          0.3001              0.14710         0.2419
1           0.07864          0.0869              0.07017         0.1812
2           0.15990          0.1974              0.12790         0.2069
3           0.28390          0.2414              0.10520         0.2597
4           0.13280          0.1980              0.10430         0.1809

   mean fractal dimension  …  worst texture  worst perimeter  worst area  \
0                 0.07871  …          17.33           184.60      2019.0
1                 0.05667  …          23.41           158.80      1956.0
2                 0.05999  …          25.53           152.50      1709.0
3                 0.09744  …          26.50            98.87       567.7
4                 0.05883  …          16.67           152.20      1575.0

   worst smoothness  worst compactness  worst concavity  worst concave points  \
0            0.1622             0.6656           0.7119                0.2654
1            0.1238             0.1866           0.2416                0.1860
2            0.1444             0.4245           0.4504                0.2430
3            0.2098             0.8663           0.6869                0.2575
4            0.1374             0.2050           0.4000                0.1625

   worst symmetry  worst fractal dimension     Target
0          0.4601                  0.11890  malignant
1          0.2750                  0.08902  malignant
2          0.3613                  0.08758  malignant
3          0.6638                  0.17300  malignant
4          0.2364                  0.07678  malignant

[5 rows x 31 columns]
```

8 Splitting Data into training and test datasets

Improves evaluation of a model's performance on unseen data - Robust

Data can be split into Training, Test - Generally a good split is: - Training:80% - Test 20% - At some point you will also get to worry about validation, however, we will skip this for now!

8.1 Code

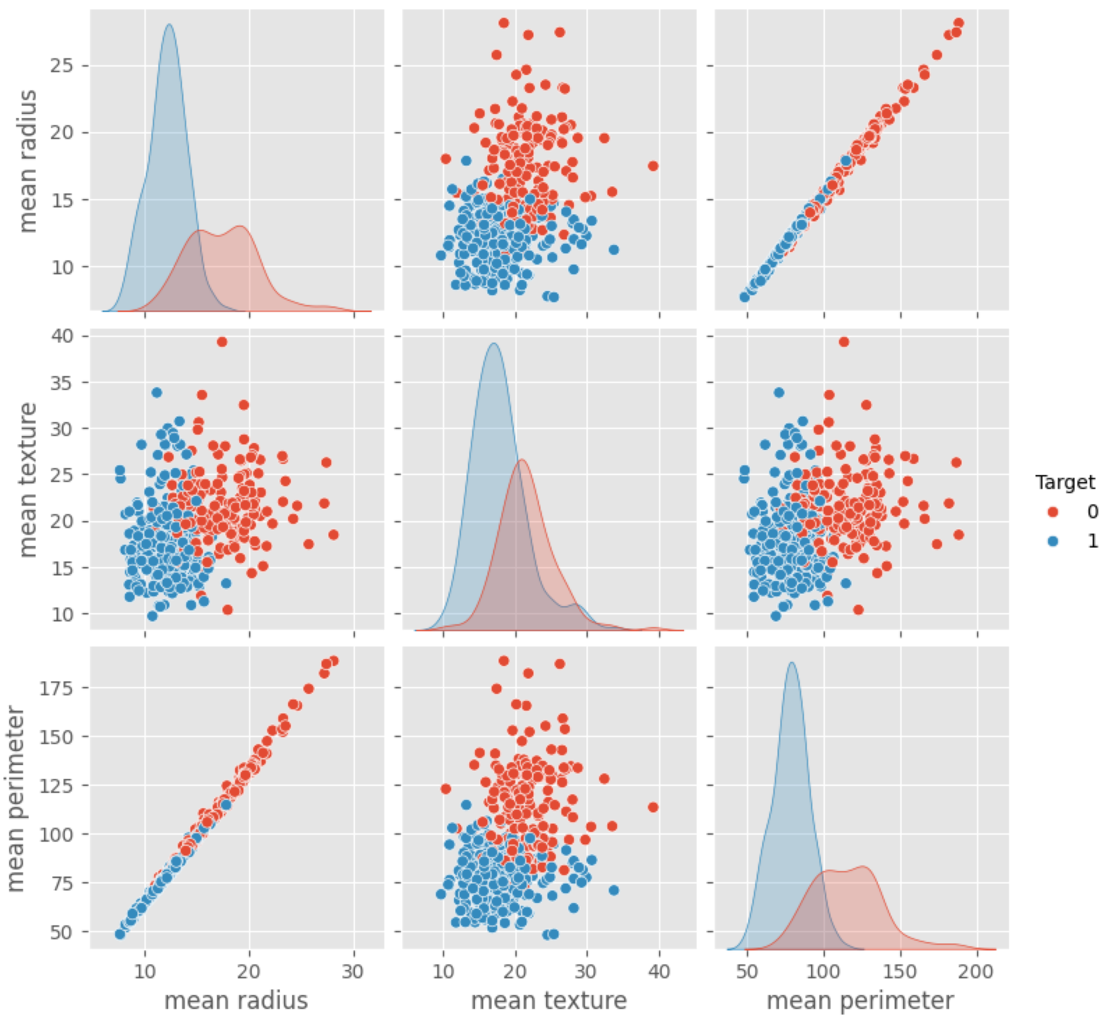
[6]:
```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
# Normalize X

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Display the dataframe
df_test = pd.DataFrame(X_test, columns=data.feature_names)

df_test["Target"] = y_test
df_test["Target"] = df_test["Target"].map({i: v for i, v in enumerate(data.
 ↪target_names)})

df_test.sample(10)
```

[6]:
```
    mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
61       19.690         21.25          130.00     1203.0          0.10960
32        9.742         19.12           61.93      289.7          0.10750
83       20.940         23.56          138.90     1364.0          0.10070
54       11.900         14.65           78.11      432.8          0.11520
31       12.060         12.74           76.84      448.6          0.09311
27       10.290         27.61           65.67      321.4          0.09030
7        17.570         15.05          115.00      955.1          0.09847
63       14.400         26.99           92.25      646.1          0.06995
82       14.480         21.46           94.25      648.2          0.09444
66       15.780         22.91          105.70      782.6          0.11550

    mean compactness  mean concavity  mean concave points  mean symmetry  \
61           0.15990        0.197400              0.12790         0.2069
32           0.08333        0.008934              0.01967         0.2538
83           0.16060        0.271200              0.13100         0.2205
54           0.12960        0.037100              0.03003         0.1995
31           0.05241        0.019720              0.01963         0.1590
27           0.07658        0.059990              0.02738         0.1593
7            0.11570        0.098750              0.07953         0.1739
63           0.05223        0.034760              0.01737         0.1707
82           0.09947        0.120400              0.04938         0.2075
66           0.17520        0.213300              0.09479         0.2096

    mean fractal dimension  …  worst texture  worst perimeter  worst area  \
```

```
61                 0.05999  ...         25.53          152.50      1709.0
32                 0.07029  ...         23.17           71.79       380.9
83                 0.05898  ...         27.00          165.30      2010.0
54                 0.07839  ...         16.51           86.26       509.6
31                 0.05907  ...         18.41           84.08       532.8
27                 0.06127  ...         34.91           69.57       357.6
7                  0.06149  ...         19.52          134.90      1227.0
63                 0.05433  ...         31.98          100.40       734.6
82                 0.05636  ...         29.25          108.40       808.9
66                 0.07331  ...         30.50          130.30      1272.0

    worst smoothness  worst compactness  worst concavity  \
61            0.1444             0.4245          0.45040
32            0.1398             0.1352          0.02085
83            0.1211             0.3172          0.69910
54            0.1424             0.2517          0.09420
31            0.1275             0.1232          0.08636
27            0.1384             0.1710          0.20000
7             0.1255             0.2812          0.24890
63            0.1017             0.1460          0.14720
82            0.1306             0.1976          0.33490
66            0.1855             0.4925          0.73560

    worst concave points  worst symmetry  worst fractal dimension     Target
61               0.24300          0.3613                  0.08758  malignant
32               0.04589          0.3196                  0.08009     benign
83               0.21050          0.3126                  0.07849  malignant
54               0.06042          0.2727                  0.10360     benign
31               0.07025          0.2514                  0.07898     benign
27               0.09127          0.2226                  0.08283     benign
7                0.14560          0.2756                  0.07919  malignant
63               0.05563          0.2345                  0.06464     benign
82               0.12250          0.3020                  0.06846  malignant
66               0.20340          0.3274                  0.12520  malignant

[10 rows x 31 columns]
```

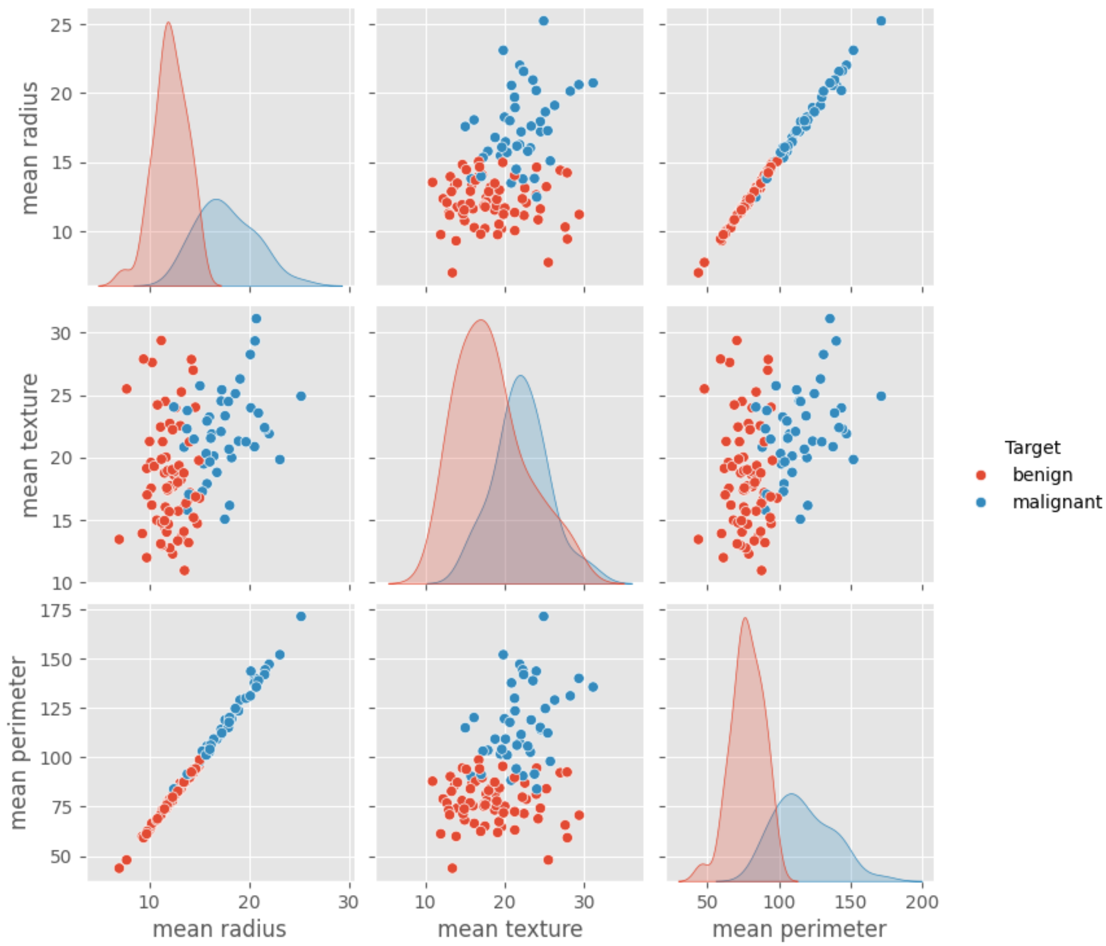
8.2 Visualize Breast Cancer Dataset - Train

[24]:
```
import seaborn as sns
df_tmp = pd.DataFrame(X_train, columns=data.feature_names)
df_tmp["Target"] = y_train
sns.pairplot(df_tmp, hue="Target", vars=data.feature_names[:3])
plt.show()
```



8.3 Visualize Breast Cancer Dataset - Test

[18]:
```
import seaborn as sns

sns.pairplot(df_test, hue="Target", vars=data.feature_names[:3])
plt.show()
```



9 Confusion Matrix

9.0.1 Confusion Matrix: Observed vs. Predicted

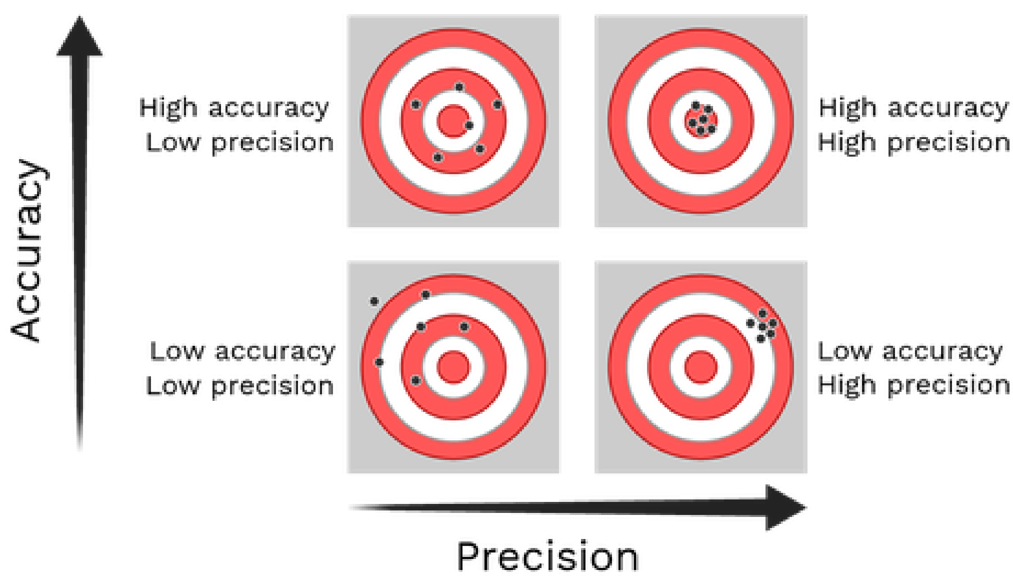
	Predicted Positive	**Predicted Negative**
Actual Positive	TP	FN
Actual Negative	FP	TN

9.0.2 Performance Metrics for a Diagnostic Test

Metric	Formula	Description
Sensitivity (True Positive Rate)	$\frac{TP}{TP+FN}$	Probability of correctly identifying a positive case

Metric	Formula	Description
Specificity (True Negative Rate)	$\frac{TN}{TN+FP}$	Probability of correctly identifying a negative case
Positive Predictive Value (PPV)	$\frac{TP}{TP+FP}$	Probability that a positive test result is a true positive
Negative Predictive Value (NPV)	$\frac{TN}{TN+FN}$	Probability that a negative test result is a true negative
Accuracy	$\frac{TP+TN}{TP+TN+FP+FN}$	Overall correctness of the test

9.1 Accuracy



10 Running and validating KMeans in python

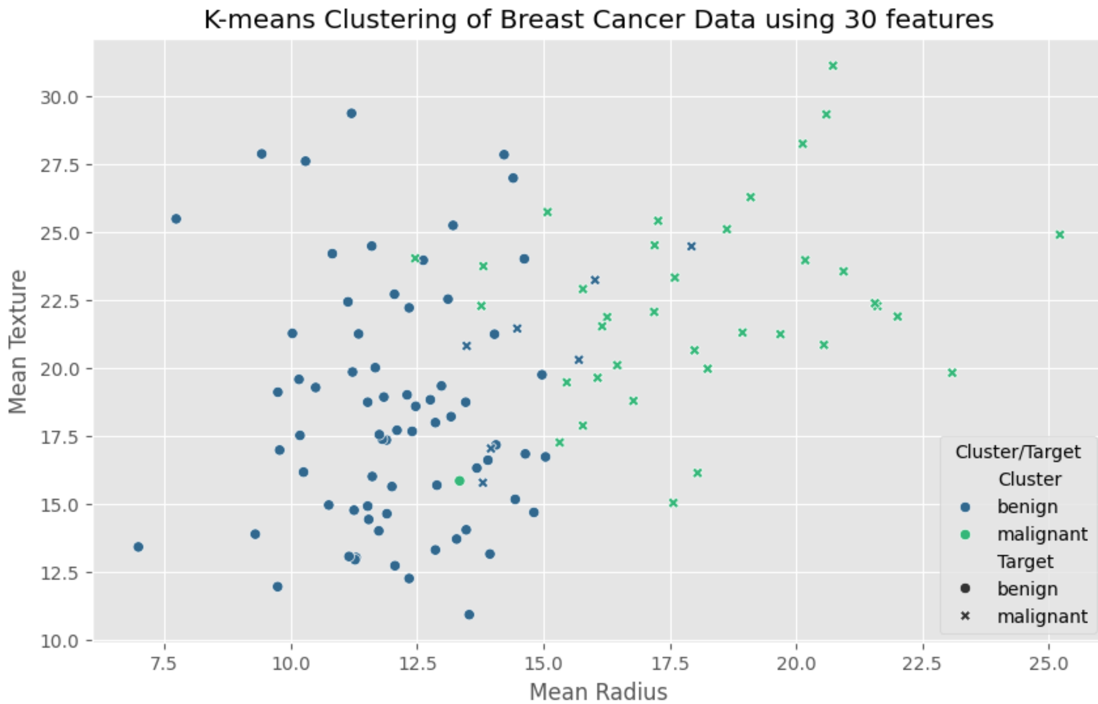
[9]:
```
from sklearn.metrics import confusion_matrix, accuracy_score

# Apply K-means clusteriang
kmeans = KMeans(n_clusters=N_CLUSTERS, random_state=42)
```

```
kmeans.fit(X_train_scaled)
y_kmeans = kmeans.predict(X_test_scaled)

# Map cluster labels to original labels (0: benign, 1: malignant)
mapping = (
    {0: 1, 1: 0}
    if confusion_matrix(y_test, y_kmeans)[0][0] < confusion_matrix(y_test,␣
↪y_kmeans)[1][0]
    else {0: 0, 1: 1}
)
y_kmeans_mapped = [mapping[label] for label in y_kmeans]

# Evaluate the clustering performance
accuracy_Kmeans = accuracy_score(y_test, y_kmeans_mapped)
conf_matrix_Kmeans = confusion_matrix(y_test, y_kmeans_mapped)

# Print the results
print(f"KMeans Accuracy using {len(data.feature_names)} features:␣
↪{accuracy_Kmeans * 100:.1f}%")
print(f"KMeans Confusion Matrix using {len(data.feature_names)} features:")
print(conf_matrix_Kmeans)
```

```
KMeans Accuracy using 30 features: 93.0%
KMeans Confusion Matrix using 30 features:
[[36  7]
 [ 1 70]]
```

11 Visualize the clusters

[10]:
```
df_test["Cluster"] = y_kmeans_mapped
df_test["Cluster"] = df_test["Cluster"].map({i: v for i, v in enumerate(data.
↪target_names)})



plt.figure(figsize=(10, 6))
sns.scatterplot(data=df_test, x="mean radius", y="mean texture", hue="Cluster",␣
↪style="Target", palette="viridis")
plt.title(f"K-means Clustering of Breast Cancer Data using {len(data.
↪feature_names)} features")
plt.xlabel("Mean Radius")
plt.ylabel("Mean Texture")
plt.legend(title="Cluster/Target")
plt.show()
```



12 Exercise

- What is the accuracy of KMeans if we use 1 feature?
- Use the following code as the starting point
- Data should be fitted using the training data, and verified using test data
- What will happen to the accuracy

[11]:
```
# get first feature of X, and do KMeans clustering
X_train_scaled1 = X_train_scaled[:, [0]]  # 1 feature only
X_test_scaled1 = X_test_scaled[:, [0]]  # 1 feature only
# y_test is the label variable, for the test set! the label doesn't change!
```

13 Answer

[12]:
```
# Apply K-means clustering and predict the labels
print("Applt K-means clustering after this print!!!!")
kmeans.fit(X_train_scaled1)
y_kmeans1 = kmeans.predict(X_test_scaled1)


# Map cluster labels to original labels (0: benign, 1: malignant)
```

```
mapping = (
    {0: 1, 1: 0}
    if confusion_matrix(y_test, y_kmeans1)[0][0] < confusion_matrix(y_test,␣
↪y_kmeans1)[1][0]
    else {0: 0, 1: 1}
)
y_kmeans1_mapped = [mapping[label] for label in y_kmeans1]

# Evaluate the clustering performance
print("Evaluate the clustering performance")
accuracy_Kmeans1 = accuracy_score(y_test, y_kmeans1_mapped)
conf_matrix_Kmeans1 = confusion_matrix(y_test, y_kmeans1_mapped)

# Print the results
print(f"KMeans Accuracy using {1} features: {accuracy_Kmeans1 * 100:.1f}%")
print(f"KMeans Confusion Matrix using {1} features:")
print(conf_matrix_Kmeans1)
```

```
Applt K-means clustering after this print!!!!
Evaluate the clustering performance
KMeans Accuracy using 1 features: 88.6%
KMeans Confusion Matrix using 1 features:
[[30 13]
 [ 0 71]]
```

14 Supervised Learning

Why Train the Model?

- Training the model involves learning patterns from the training data, which the model uses to make predictions.

14.1 K Nearest Neighbors

- Lazy Learner
- K: How many neighbors, should be considered, to find the closet fit
  - Basically, if K = 3, then we find the three closest samples to a given sample, and pick the majority

14.2

1. The k-nearest neighbor algorithm is imported from the scikit-learn package.
2. Create feature and target variables.
3. Split data into training and test data.
4. Generate a k-NN model using neighbors value.
5. Train or fit the data into the model.
6. Predict the future.

15 K Nearest Neighbors

1. **Distance Calculation**: To find the nearest neighbors, KNN typically uses **Euclidean distance** between a query point (x) and a training point (x_i):

$$d(x, x_i) = \sqrt{\sum_{j=1}^{n} (x_j - x_{ij})^2}$$

2. **Majority Voting for Classification**: For classification, the predicted class $\hat{y}$ is determined by majority voting among the K nearest neighbors:

$$\hat{y} = \arg\max_c \sum_{i \in N_K(x)} \mathbb{1}(y_i = c)$$

   - $N_k(x)$ represents the set of the k nearest neighbors of x,
   - y_i is the class label of neighbor x_i,
   - $\mathbb{1}(y_i = c)$ is an indicator function that equals 1 if $y_i = c$, otherwise 0,
   - $\arg\max_c$ selects the class with the highest count.

3. **Weighted Voting (Optional)**: A weight can be atributed to each sample. However, we skip this for now

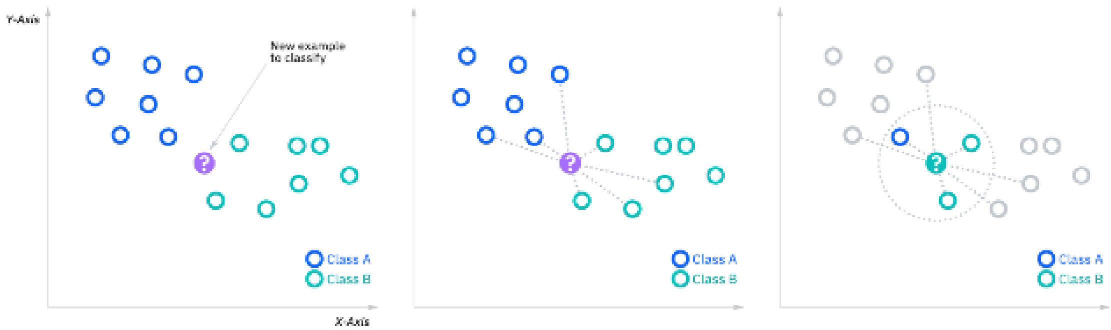
4. **Regression with KNN**: For regression, the predicted value $\hat{y}$ is the average of the target values of the k nearest neighbors

$$\hat{y} = \frac{1}{k} \sum_{i \in N_k(x)} y_i$$

   where:

   - y_i is the target value of neighbor x_i,
   - w_i is the weight based on distance.

16 K Nearest Neighbors



17 K Nearest Neighbors

[13]:
```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=7)

knn.fit(X_train_scaled, y_train)
y_knn = knn.predict(X_test_scaled)

# Calculate the accuracy of the model
# Evaluate the clustering performance
accuracy_knn = knn.score(X_test_scaled, y_test)
conf_matrix_knn = confusion_matrix(y_test, y_knn)

# Print the results
print(f"KNN Accuracy using {len(data.feature_names)} features: {accuracy_knn *
↪100:.1f}%")
print(f"KNN Confusion Matrix using {len(data.feature_names)} features:")
print(conf_matrix_knn)
```

```
KNN Accuracy using 30 features: 94.7%
KNN Confusion Matrix using 30 features:
[[40  3]
 [ 3 68]]
```

18 Exercise

- What is the accuracy of K Nearest Neighbors if we use 1, 7 nearest neighbors?
- Data should be fitted using the training data, and verified using test data
- What do you expect will happen to the accuracy
- How will the accuracy be compared to KMeans?
- Use the following code as the starting point

[14]:
```
# get first feature of X, and do KMeans clustering
#X_train_scaled
#X_test_scaled
# y_train, y_test are the label variables. the label doesn't change!


# Apply KNN  and predict the labels
print("Applt KNN clustering after this print!!!!")
```

```
Applt KNN clustering after this print!!!!
```

19 Answer

[15]:
```
# Apply KNN  and predict the labels
print("Applt KNN clustering after this print!!!!")

knn1 = KNeighborsClassifier(1)
knn1.fit(X_train_scaled, y_train)
y_knn1 = knn1.predict(X_test_scaled)
accuracy_knn1 = knn1.score(X_test_scaled, y_test)
conf_matrix_knn1 = confusion_matrix(y_test, y_knn1)


knn7 = KNeighborsClassifier(7)
knn7.fit(X_train_scaled, y_train)
y_knn7 = knn7.predict(X_test_scaled)
accuracy_knn7 = knn7.score(X_test_scaled, y_test)
conf_matrix_knn7 = confusion_matrix(y_test, y_knn7)

# Print the results
print(f"KNN, with 1 NN, Accuracy using {len(data.feature_names)} features:␣
 ↪{accuracy_knn1 * 100:.1f}%")
print(f"KNN, with 1 NN, Confusion Matrix using {len(data.feature_names)}␣
 ↪features:")
print(conf_matrix_knn1)
```

```python
# Print the results
print(f"KNN, with 7 NN, Accuracy using {len(data.feature_names)} features:␣
 ↪{accuracy_knn7 * 100:.1f}%")
print(f"KNN, with 7 NN, Confusion Matrix using {len(data.feature_names)}␣
 ↪features:")
print(conf_matrix_knn7)
```

```
Applt KNN clustering after this print!!!!
KNN, with 1 NN, Accuracy using 30 features: 93.9%
KNN, with 1 NN, Confusion Matrix using 30 features:
[[39  4]
 [ 3 68]]
KNN, with 7 NN, Accuracy using 30 features: 94.7%
KNN, with 7 NN, Confusion Matrix using 30 features:
[[40  3]
 [ 3 68]]
```

20 Support Vector Machines (SVM's)

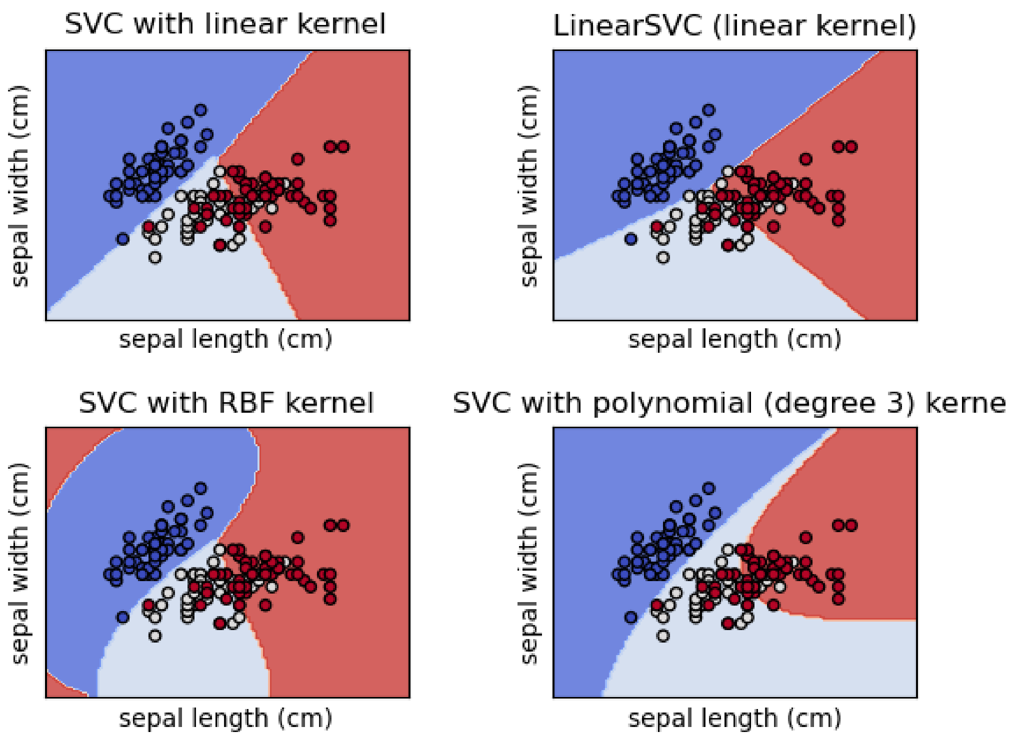
Support Vector Machine tries to find the best separating line between classes.

The advantages of support vector machines are: - Effective in high dimensional spaces. - Still effective in cases where number of dimensions is greater than the number of samples. - Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient. - Versatile: different Kernel functions can be specified for the decision function.

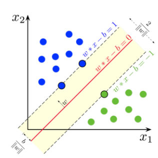
The disadvantages of support vector machines include: - If the number of features is much greater than the number of samples, avoid over-fitting in choosing Kernel functions and regularization term is crucial. - SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation (see Scores and probabilities, below).

20.1 How SVM Works

- SVM finds the best hyperplane that separates the data into different classes while maximizing the margin.



21 Support Vector Machine (SVM)



22 Example using Breast Cancer

[16]:
```python
from sklearn import svm

clf = svm.SVC()
clf.fit(X_train_scaled, y_train)

y_clf = clf.predict(X_test_scaled)
accuracy_clf = accuracy_score(y_test,y_clf)
conf_matrix_clf = confusion_matrix(y_test,y_clf)


# Print the results
```

```
print(f"SVM Accuracy using {len(data.feature_names)} features: {accuracy_knn1 *␣
↪100:.1f}%")
print(f"SVM Confusion Matrix using {len(data.feature_names)} features:")
print(conf_matrix_knn1)
```

```
SVM Accuracy using 30 features: 93.9%
SVM Confusion Matrix using 30 features:
[[39  4]
 [ 3 68]]
```

23 Scaling, Why?

Machine Learning Models Are Sensitive to Feature Magnitudes

- Many models use distance-based calculations (e.g., SVM, KNN, K-Means, PCA).
- Features with different scales (e.g., height in cm vs. income in dollars) can distort results.

Problems Without Scaling

- Some features dominate due to larger numerical values.
- Slower convergence in optimization, especially for SVM.
- Poor model performance and incorrect classifications.

Common Scaling Methods

- **Standardization:**
$$X' = \frac{X - \mu}{\sigma}$$
(zero mean, unit variance).
- **Min-Max Scaling:**
$$X' = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}}$$
(scales values between 0 and 1).

Scaling ensures optimal model performance and accurate results.

---

24 Exercises

24.0.1 Exercise 1 - Classification with Breast Cancer Dataset

Objective: Train a classifier on the Breast Cancer dataset and evaluate its performance. Steps: 1. Load the Breast Cancer dataset. 2. Split the data into training and testing sets. 3. Train a Decision Tree classifier using one feature. 4. Train a Decision Tree classifier using multiple features. 5. Make predictions and evaluate accuracy.

25 Answers

25.0.1 Answer 1 - Classification with Breast Cancer Dataset

[17]:
```python
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Load data
data = load_breast_cancer()
X, y = data.data, data.target

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
 ↪random_state=42)

# Train model with one feature
X_train_one_feature = X_train[:, [0]]  # Using 'mean radius'
X_test_one_feature = X_test[:, [0]]
model = DecisionTreeClassifier()
model.fit(X_train_one_feature, y_train)

# Make predictions with one feature
y_pred_one_feature = model.predict(X_test_one_feature)

# Evaluate model with one feature
accuracy_one_feature = accuracy_score(y_test, y_pred_one_feature)
print(f"Accuracy with one feature: {accuracy_one_feature}")

# Train model with multiple features
model.fit(X_train, y_train)

# Make predictions with multiple features
y_pred = model.predict(X_test)

# Evaluate model with multiple features
accuracy_Kmeans = accuracy_score(y_test, y_pred)
print(f"Accuracy with multiple features: {accuracy_Kmeans}")
```

```
Accuracy with one feature: 0.8508771929824561
Accuracy with multiple features: 0.9473684210526315
```