Assignment List 7: Advanced Data Clustering

Machine Learning – University of Wrocław

Winter Semester 2024/2025

Note: Each task is worth 2 points. Solutions should include Python code and brief comments explaining key implementation decisions.

Task 1: Semi-supervised Learning with Clustering (2 pts)

Scenario: The company "WroMed" collects medical patient data, but labeling diseases by doctors is expensive and time-consuming. You have access to a large unlabeled dataset and a small subset (5%) with labels. Your task is to use clustering to propagate labels to the remaining data.

Data: Use the `digits` dataset from `sklearn.datasets` (1797 8x8 digit images, 10 classes).

Instructions:

(a) **Simulating missing labels:** Randomly hide 95% of labels (mark as -1). Keep only 5% of the data as "labeled".

(b) **Clustering + propagation:** Perform K-Means clustering ($k = 10$). For each cluster, find the most frequent label among labeled points and assign it to all points in the cluster.

(c) **Comparing methods:** Implement two propagation strategies:

- **Voting:** Most frequent label in the cluster (part b)
- **Nearest centroid:** For clusters without labels, assign the label of the nearest cluster that has labels

(d) **Evaluation:** Calculate accuracy on **unlabeled** data. Compare with:

- k-NN ($k = 5$) trained only on labeled data
- Fully supervised classifier (100% labels) – upper bound

How does accuracy change depending on the % of available labels (1%, 5%, 10%, 20%)?

Task 2: Anomalies in a Sensor Network (2 pts)

Scenario: The company "WroExpress" monitors a fleet of 200 trams using temperature, vibration, and speed sensors. After a series of failures, your task is to detect anomalous tram behavior before complete breakdown.

Data: Use synthetic data generated in the starter code or your own measurements simulating sensor behavior.

Instructions:

(a) **K-Means method:** Implement anomaly detection based on distance from centroids. A point is an anomaly if its distance from the nearest centroid exceeds a threshold (e.g., 95th percentile).

(b) **DBSCAN method:** Use points marked as noise (label = -1) by DBSCAN as anomaly candidates. Tune the `eps` and `min_samples` parameters.

(c) **LOF method:** Apply Local Outlier Factor and compare results with the previous methods.

(d) **Comparative analysis:** For each method, calculate precision, recall, and F1-score (assuming you know the true anomalies from the test data). Which method best detects anomalies in this scenario? Justify your answer.

Starter code:

```
import numpy as np
from sklearn.cluster import KMeans, DBSCAN
from sklearn.neighbors import LocalOutlierFactor

def generate_sensor_data(n_normal=180, n_anomaly=20, random_state=42):
    """Generuje dane czujnikow z anomaliami."""
    np.random.seed(random_state)

    # Normalne tramwaje - 3 klastry operacyjne
    normal = np.vstack([
        np.random.randn(60, 3) * 0.5 + [50, 10, 30],  # temp, vibration,
            speed
        np.random.randn(60, 3) * 0.5 + [55, 12, 35],
        np.random.randn(60, 3) * 0.5 + [48, 8, 28],
    ])

    # Anomalie - nietypowe zachowania
    anomalies = np.vstack([
        np.random.randn(10, 3) * 0.3 + [80, 25, 15],  # przegrzanie
        np.random.randn(10, 3) * 0.3 + [30, 30, 5],   # awaria silnika
    ])

    X = np.vstack([normal, anomalies])
    y_true = np.array([0]*n_normal + [1]*n_anomaly)  # 0=normal, 1=
       anomaly

    return X, y_true

def detect_anomalies_kmeans(X, k=3, percentile=95):
    """Wykrywanie anomalii metoda K-Means."""
    # TODO: Zaimplementuj
    pass

def detect_anomalies_dbscan(X, eps=5, min_samples=5):
    """Wykrywanie anomalii metoda DBSCAN."""
    # TODO: Zaimplementuj
    pass
```

Task 3: GPS Trajectory Clustering (2 pts)

Scenario: The city of Wrocław analyzes cyclist mobility patterns based on data from the "WroVelo" system. Trajectories have different lengths and speeds, so standard Euclidean distance doesn't work.

Data: Use synthetic trajectories generated in the starter code or public GPS datasets (e.g., GeoLife, OpenStreetMap traces).

Instructions:

(a) **DTW implementation:** Implement the Dynamic Time Warping algorithm to calculate the distance between two trajectories. You can use the `dtaidistance` library or write your own implementation.

(b) **Distance matrix:** Calculate the DTW distance matrix for all trajectory pairs. Pay attention to computational complexity.

(c) **Hierarchical clustering (Ward):** Apply Ward's method with a precomputed distance matrix. Visualize the dendrogram and choose an appropriate number of clusters.

(d) **Interpretation:** For each cluster, draw representative trajectories. What mobility patterns did you discover? (e.g., commute routes, recreational, tourist)

Starter code:

```
import numpy as np
from scipy.cluster.hierarchy import linkage, dendrogram, fcluster
from scipy.spatial.distance import squareform
import matplotlib.pyplot as plt

def generate_trajectories(n_trajectories=50, random_state=42):
    """Generuje syntetyczne trajektorie GPS."""
    np.random.seed(random_state)
    trajectories = []

    # Typ 1: Trasa do centrum (z roznych stron)
    for _ in range(15):
        length = np.random.randint(20, 40)
        start = np.random.randn(2) * 5 + [10, 10]
        end = np.array([0, 0])  # centrum
        t = np.linspace(0, 1, length).reshape(-1, 1)
        traj = start + t * (end - start) + np.random.randn(length, 2) *
            0.3
        trajectories.append(traj)

    # Typ 2: Trasa rekreacyjna (petla)
    for _ in range(15):
        length = np.random.randint(30, 50)
        t = np.linspace(0, 2*np.pi, length)
        r = 3 + np.random.rand() * 2
        traj = np.column_stack([r*np.cos(t), r*np.sin(t)])
        traj += np.random.randn(length, 2) * 0.2
        trajectories.append(traj)
```

```
    # Typ 3: Trasa wzdluz rzeki (liniowa)
    for _ in range(20):
        length = np.random.randint(25, 45)
        start_x = np.random.rand() * 10 - 5
        traj = np.column_stack([
            np.linspace(start_x, start_x + 8, length),
            np.sin(np.linspace(0, np.pi, length)) * 2
        ])
        traj += np.random.randn(length, 2) * 0.2
        trajectories.append(traj)

    return trajectories

def dtw_distance(traj1, traj2):
    """Oblicza odleglosc DTW miedzy dwiema trajektoriami."""
    # TODO: Zaimplementuj algorytm DTW
    pass

def compute_dtw_matrix(trajectories):
    """Oblicza macierz odleglosci DTW."""
    n = len(trajectories)
    D = np.zeros((n, n))
    # TODO: Wypelnij macierz (symetryczna!)
    return D
```

Task 4: WroShop - Large-Scale Data Processing (2 pts)

Scenario: The company "WroShop" runs an e-commerce platform in Wrocław. Their transaction data is growing exponentially and exceeds pandas capabilities. Your task is to learn tools for processing large data: Dask and PySpark.

Data:

- `data/wroshop_small.csv` – 100,000 transactions (single file)
- `data/wroshop_medium/` – 1,000,000 transactions (10 partitioned files)

Instructions:

(a) **Pandas baseline (100K):** Load the small data and perform operations: groupby by category, top 10 cities, filtering. Measure time and memory usage.

(b) **Pandas on larger data (1M):** Load all files from `wroshop_medium/` using `glob` and `pd.concat`. Repeat the operations. Observe the slowdown.

(c) **Dask DataFrame:** Use `dask.dataframe`:

```
import dask.dataframe as dd
ddf = dd.read_csv('data/wroshop_medium/*.csv')
# Operations are LAZY - execute .compute() at the end
result = ddf.groupby('category')['total_amount'].mean().compute()
```

Visualize the task graph. Compare time with pandas.

(d) **PySpark DataFrame:** Use `pyspark` (or Google Colab):

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("WroShop").getOrCreate()
df = spark.read.csv('data/wroshop_medium/', header=True,
    inferSchema=True)
df.groupBy('category').avg('total_amount').show()
```

Show the execution plan: `result.explain()`.

(e) **Comparison:** Create a comparative table of times. When should you use pandas/Dask/Spark?

Task 5: WroAI - Matrix Multiplication in ML Practice (2 pts)

Scenario: The startup "WroAI" builds machine learning systems. As an ML engineer, your task is to optimize key computational operations through efficient use of matrix multiplication instead of loops.

Instructions:

(a) **Softmax Attention:** Implement the attention mechanism from Transformers:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Compare the implementation with 3 nested loops vs the matrix version. Benchmark for sequence lengths 64, 256, 512.

(b) **Confusion matrix without loops:** Implement confusion matrix calculation using matrix multiplication. Benchmark for 10K, 100K samples and 10, 50, 100 classes.

(c) **Batch Normalization:** Implement batch normalization. Benchmark for batch=128, features=1024, 4096.

(d) **PageRank:** Implement the PageRank algorithm as a matrix iteration:

$$r_{t+1} = d \cdot M \cdot r_t + \frac{1-d}{n} \cdot \mathbf{1}$$

where M is the normalized transition matrix. Compare with the loop version. Visualize convergence.

Requirements: For each operation: (1) naive implementation, (2) matrix derivation, (3) benchmark, (4) correctness verification (`np.allclose`).

General Guidelines

- Recommended libraries: `numpy`, `scipy`, `scikit-learn`, `matplotlib`, `networkx` (for task 3), `dtaidistance` or `tslearn` (for task 3).
- Plots should have **titles, axis labels, and legends**.
- Code should be **readable** and contain comments explaining key steps.